TRƯỜNG ĐẠI HỌC ĐÀ LẠT
KHOA CÔNG NGHỆ THÔNG TIN

BÀI GIẢNG TÓM TẮT

LẬP TRÌNH MẠNG

Dành cho sinh viên ngành Công Nghệ Thông Tin

(Lưu hành nội bộ)

Đà Lạt 2009

MỤC LỤC

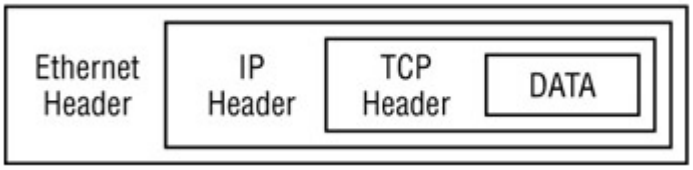
CHƯƠNG I: NHỮNG KIẾN THỨC CƠ BẢN VỀ LẬP TRÌNH MẠNG

I.1. Tổng quan

Internet Protocol (IP) là nền tảng của lập trình mạng. IP là phương tiện truyền tải dữ liệu giữa các hệ thống bất kể đó là hệ thống mạng cục bộ (LAN) hay hệ thống mạng diện rộng (WAN). Mặc dù lập trình viên mạng có thể chọn các giao thức khác để lập trình nhưng IP cung cấp các kỹ thuật mạnh nhất để gởi dữ liệu giữa các thiết bị, đặc biệt là thông qua mạng Internet.

Để hiểu rõ các khái niệm bên dưới lập trình mạng, chúng ta phải hiểu rõ giao thức IP, hiểu cách nó chuyển dữ liệu giữa các thiết bị mạng. Lập trình mạng dùng giao thức IP thường rất phức tạp. Có nhiều yếu tố cần quan tâm liên quan đến cách dữ liệu được gởi qua mạng: số lượng Client và Server, kiểu mạng, tắc nghẽn mạng, lỗi mạng,… Bởi vì các yếu tố này ảnh hưởng đến việc truyền dữ liệu từ thiết bị này đến thiết bị khác trên mạng do đó việc hiểu rõ chúng là vấn đề rất quan trọng để lập trình mạng được thành công.

Một gói dữ liệu mạng gồm nhiều tầng thông tin. Mỗi tầng thông tin chứa một dãy các byte được sắp đặt theo một trật tự đã được định sẵn. Hầu hết các gói dữ liệu dùng trong lập trình mạng đều chứa ba tầng thông tin cùng với dữ liệu được dùng để truyền tải giữa các thiết bị mạng. Hình sau mô tả hệ thống thứ bậc của một gói IP:

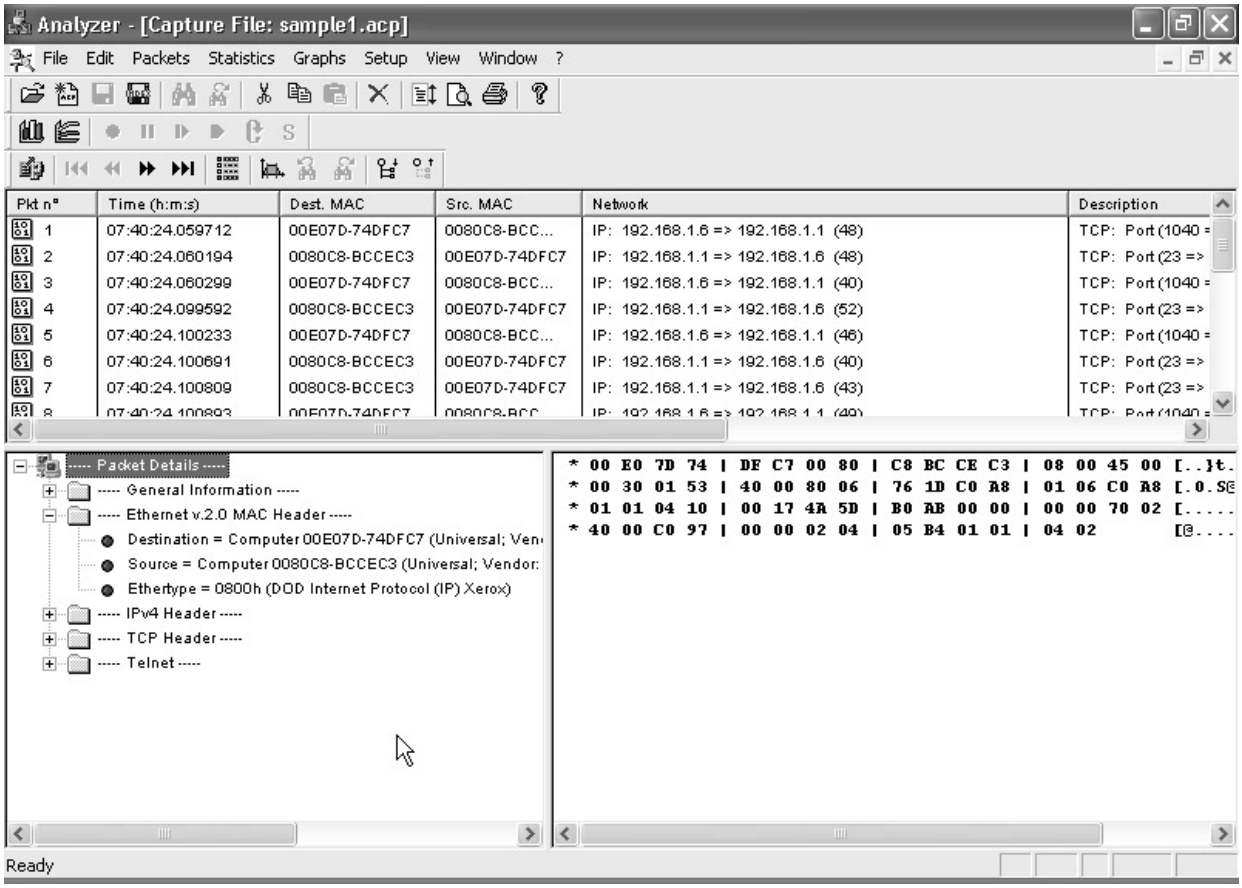


Hình I.1: Các tầng giao thức mạng trong các gói dữ liệu

I.1.1. Tầng Ethernet

Tầng đầu tiên của gói dữ liệu mạng được gọi là Ethernet Header, trong tầng này có ba gói giao thức Ethernet: Ethernet 802.2, Ethernet 802.3, và Ethernet phiên bản 2. Các giao thức Ethernet 802.2 và Ethernet 802.3 là các giao thức chuẩn của IEEE. Ethernet phiên bản 2 tuy không phải là giao thức chuẩn nhưng nó được sử dụng rộng

rãi trong mạng Ethernet. Hầu hết các thiết bị mạng kể cả hệ điều hành Windows mặc định dùng giao thức Ethernet phiên bản 2 để truyền tải các gói IP.



Hình I.2: Ethernet Header

Phần đầu của Ethernet phiên bản 2 là địa chỉ MAC (Media Access Card) dùng để xác định các thiết bị trên mạng cùng với số giao thức Ethernet xác định giao thức tầng tiếp theo chứa trong gói Ethernet. Mỗi gói Ethernet bao gồm:

- 6 byte địa chỉ MAC đích
- 6 byte địa chỉ MAC nguồn
- 2 byte xác định giao thức tầng kế tiếp
- Data payload từ 46 đến 1500 byte
- 4-byte checksum

I.1.2. Địa chỉ Ethernet

Địa chỉ Ethernet (địa chỉ MAC) là địa chỉ của các thiết bị, địa chỉ này được gán bởi các nhà sản xuất thiết bị mạng và nó không thay đổi được. Mỗi thiết bị trên mạng Ethernet phải có 1 địa chỉ MAC duy nhất. Địa chỉ MAC gồm 2 phần:

- 3 byte xác định nhà sản xuất
- 3 byte xác định số serial duy nhất của nhà sản xuất

Giản đồ địa chỉ Ethernet cho phép các địa chỉ broadcast và multicast. Đối với địa chỉ broadcast thì tất cả các bit của địa chỉ đích được gán bằng 1 (FFFFFFFFFFFF). Mỗi thiết bị mạng sẽ chấp nhận các gói có địa chỉ broadcast. Địa chỉ này hữu ích cho các giao thức phải gởi các gói truy vấn đến tất cả các thiết bị mạng. Địa chỉ multicast cũng là một loại địa chỉ đặc biệt của địa chỉ Ethernet, các địa chỉ multicast chỉ cho phép một số các thiết bị chấp nhận gói tin. Một số địa chỉ Ethernet multicast:

Địa Chỉ	**Mô Tả**
01-80-C2-00-00-00	Spanning tree (for bridges)
09-00-09-00-00-01	HP Probe
09-00-09-00-00-01	HP Probe
09-00-09-00-00-04	HP DTC
09-00-2B-00-00-00	DEC MUMPS
09-00-2B-00-00-01	DEC DSM/DTP
09-00-2B-00-00-02	DEC VAXELN
09-00-2B-00-00-03	DEC Lanbridge Traffic Monitor (LTM)
09-00-2B-00-00-04	DEC MAP End System Hello
09-00-2B-00-00-05	DEC MAP Intermediate System Hello
09-00-2B-00-00-06	DEC CSMA/CD Encryption
09-00-2B-00-00-07	DEC NetBios Emulator
09-00-2B-00-00-0F	DEC Local Area Transport (LAT)
09-00-2B-00-00-1x	DEC Experimental
09-00-2B-01-00-00	DEC LanBridge Copy packets (all bridges)
09-00-2B-02-00-00	DEC DNA Lev. 2 Routing Layer Routers
09-00-2B-02-01-00	DEC DNA Naming Service Advertisement
09-00-2B-02-01-01	DEC DNA Naming Service Solicitation
09-00-2B-02-01-02	DEC DNA Time Service
09-00-2B-03-xx-xx	DEC default filtering by bridges

Địa Chỉ	**Mô Tả**
09-00-2B-04-00-00	DEC Local Area System Transport (LAST)
09-00-2B-23-00-00	DEC Argonaut Console
09-00-4E-00-00-02	Novell IPX
09-00-77-00-00-01	Retix spanning tree bridges
09-00-7C-02-00-05	Vitalink diagnostics
09-00-7C-05-00-01	Vitalink gateway
0D-1E-15-BA-DD-06	HP
CF-00-00-00-00-00	Ethernet Configuration Test protocol (Loopback)

I.1.3. Ethernet Protocol Type

Một phần khác rất quan trọng của Ethernet Header là trường Protocol Type, trường này có kích thước hai byte. Sự khác nhau giữa gói tin Ethernet phiên bản 2 và Ethernet 802.2 và 802.3 xảy ra ở trường này. Các gói tin Ethernet 802.2 và 802.3 sử dụng trường này để cho biết kích thước của một gói tin Ethernet. Ethernet phiên bản 2 dùng trường này để định nghĩa giao thức tầng kế tiếp trong gói tin Ethernet. Một số giá trị của trường này:

Giá Trị	**Giao Thức**
0800	IP
0806	ARP
0BAD	Banyan VINES
8005	HP Probe
8035	Reverse ARP
809B	AppleTalk
80D5	IBM SNA
8137	Novell
8138	Novell
814C	Raw SNMP
86DD	IPv6
876B	TCP/IP compression

I.1.4. Data payload

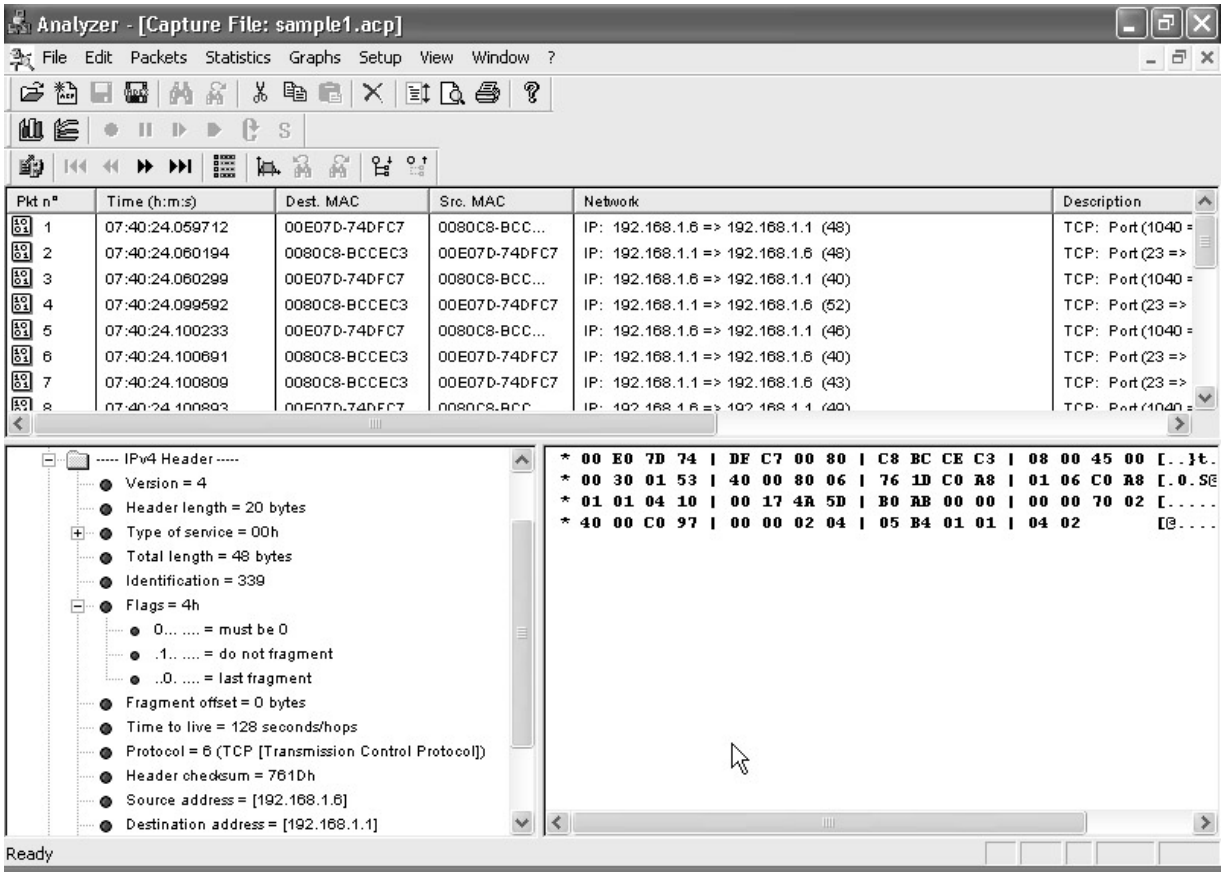
Data payload phải chứa tối thiểu 46 byte để đảm bảo gói Ethernet có chiều dài tối thiểu 64 byte. Nếu phần data chưa đủ 46 byte thì các ký tự đệm được thêm vào cho đủ. Kích thước của trường này từ 46 đến 1500 byte.

I.1.5. Checksum

Giá trị checksum cung cấp cơ chế kiểm tra lỗi cho dữ liệu, kích thước của trường này là 4 byte . Nếu gói tin bị hỏng trong lúc truyền, giá trị checksum sẽ bị tính toán sai và gói tin đó được đánh dấu là gói tin xấu.

I.2. Tầng IP

Tầng IP định nghĩa thêm nhiều trường thông tin của của giao thức Ethernet



Hình I.3: Thông tin tầng IP

Các trường trong tầng IP:

Trường	**Bit**	**Mô Tả**
Version	4	Phiên bản IP header (phiên bản hiện tại là 4)
Header Length	4	Chiều dài phần header của gói IP

Trường	**Bit**	**Mô Tả**
Type of Service	8	Kiểu chất lượng dịch vụ QoS (Quality of Service)
Total Length	16	Chiều dài của gói IP
Identification	16	Giá trị ID duy nhất xác định các gói IP
Flags	3	Cho biết gói IP có bị phân đoạn hay không hay còn các phân đoạn khác
Fragment offset	13	Vị trí của phân đoạn trong gói IP
Time to Live (TTL)	8	Thời gian tối đa gói tin được phép ở lại trên mạng (được tính bằng giây)
Protocol	8	Kiểu giao thức của tầng dữ liệu kế tiếp
Header Checksum	16	Checksum của dữ liệu gói IP header
Source Address	32	Địa chỉ IP của thiết bị gởi
Destination Address	32	Địa chỉ IP của thiết bị nhận
Options		Định nghĩa các đặc điểm của gói IP trong tươnglai

I.2.1. Trường địa chỉ

Địa chỉ Ethernet dùng để xác định các thiết bị trên mạng LAN nhưng nó không thể dùng để xác định địa chỉ của các thiết bị trên mạng ở xa. Để xác định các thiết bị trên các mạng khác nhau, địa chỉ IP được dùng. Một địa chỉ IP là một số 32 bit và địa chỉ IP được chia thành 4 lớp sau:

Lớp A	0.x.x.x–127.x.x.x
Lớp B	128.x.x.x–191.x.x.x
Lớp C	192.x.x.x–223.x.x.x
Lớp D	224.x.x.x–254.x.x.x

I.2.2. Các cờ phân đoạn

Một trong những phức tạp, rắc rối của gói IP là kích thước của chúng. Kích thước tối đa của gói IP có thể lên đến 65,536 byte. Đây là một lượng rất lớn dữ liệu cho một gói tin. Thực tế hầu hết các truyền tải dữ liệu ở cấp thấp như Ethernet không thể hỗ trợ một gói IP lớn (phần dữ liệu của Ethernet chỉ có thể tối đa 1500 byte). Để giải quyết vấn đề này, các gói IP dùng fragmentation (phân đoạn) để chia các gói IP thành các phần nhỏ hơn để truyền tải tới đích. Khi các mảnh được truyền tải tới đích, phần mềm của thiết bị nhận phải có cách để nhận ra các phân đoạn của gói tin và ráp chúng lại thành thành 1 gói IP.

Sự phân đoạn được thành lập nhờ vào việc sử dụng 3 trường của gói IP: fragmentation flags, fragment offset, và trường identification. Cờ phân đoạn bao gồm ba cờ một bit sau:

- Cờ reserved: giá trị zero
- Cờ Don't Fragment: cho biết gói IP không bị phân đoạn
- Cờ More Fragment: cho biết gói tin bị phân đoạn và còn các phân đoạn khác nữa

Trường IP Indentification xác định duy nhất định danh mỗi gói IP. Tất cả các phân đoạn của bất kỳ gói IP nào cũng đều có cùng số indentification. Số identification giúp cho phần mềm máy nhận biết được các phân đoạn nào thuộc gói IP nào và ráp lại cho đúng.

Trường fragment offset cho biết vị trí của phân đoạn trong gói tin ban đầu.

I.2.3. Trường Type of Service

Trường Type of Service xác định kiểu chất lượng dịch vụ QoS (Quality of Service) cho gói IP. Trường này được dùng để đánh dấu một gói IP có một độ ưu tiên nào đó chẳng hạn như được dùng để tăng độ ưu tiên của các dữ liệu cần thời gian thực như Video, Audio.

Trong hầu hết các truyền tải mạng, trường này được được thiết lập giá trị zero, cho biết đây là dữ liệu bình thường, tuy nhiên với các ứng dụng cần thời gian thực như Video hay Audio thì trường này sẽ được sử dụng để tăng độ ưu tiên cho gói dữ liệu. Trường này gồm tám bit và ý nghĩa các bit như sau:

- 3 bit được dùng làm trường ưu tiên
- 1 bit cho biết thời gian trễ là bình thường hay thấp
- 1 bit cho biết thông lượng bình thường hay cao
- 1 bit cho biết độ tin cậy bình thường hay cao
- 2 bit được dùng trong tương lai

I.2.4. Trường Protocol

Được dùng để xác định giao thức tầng tiếp theo trong gói IP, IANA định nghĩa 135 giá trị cho trường này có thể dùng trong gói IP nhưng chỉ có một số giá trị hay được dùng trong bảng sau:

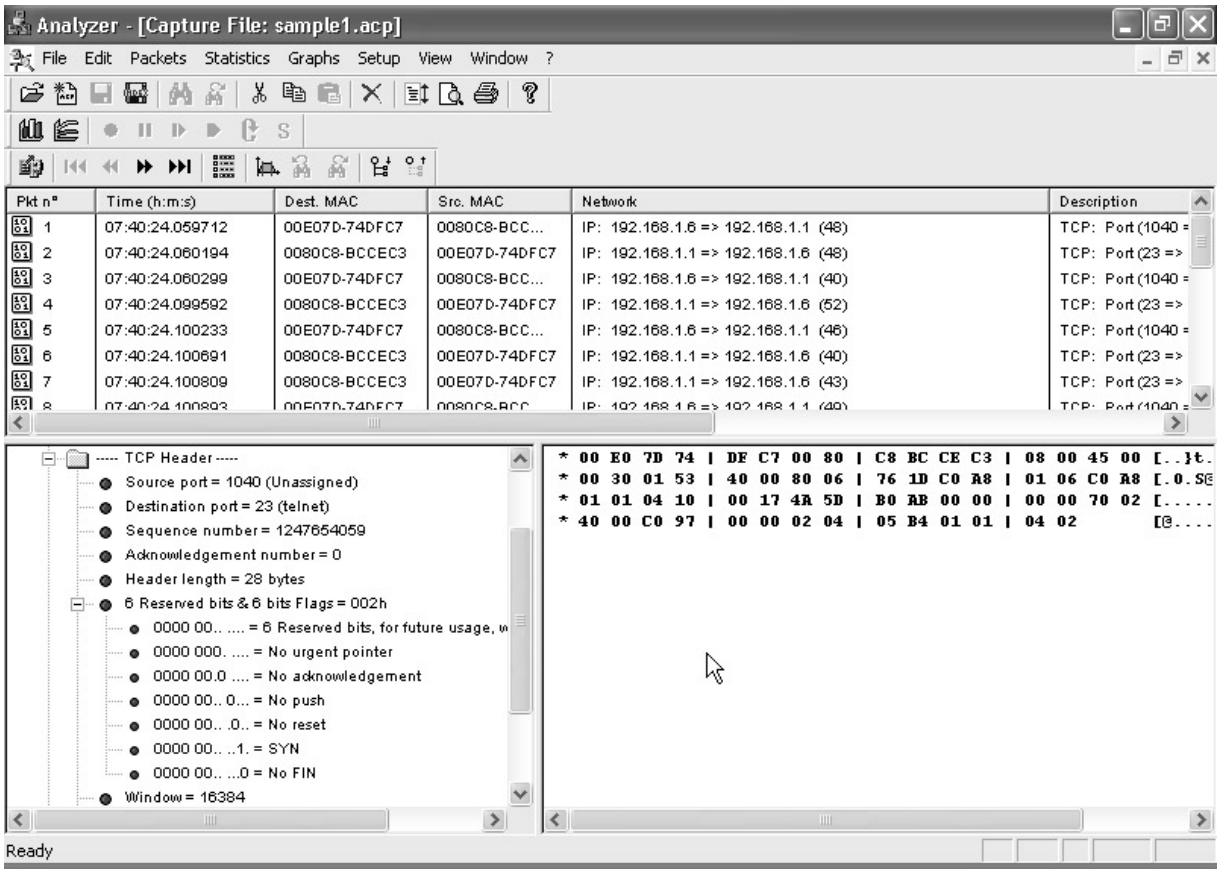
Giá Trị	**Giao Thức**
1	Internet Control Message (ICMP)
2	Internet Group Message (IGP)
6	Transmission Control (TCP)
8	Exterior Gateway (EGP)
9	Interior Gateway (Cisco IGP)
17	User Datagram (UDP)
88	Cisco EIGRP

Hai giao thức được dùng nhiều nhất trong lập trình mạng là TCP và UDP

I.3. Tầng TCP

Giao thức TCP (Transmission Control Protocol) là giao thức hướng kết nối, nó cho phép tạo ra kết nối điểm tới điểm giữa hai thiết bị mạng, thiết lập một đường nhất quán để truyền tải dữ liệu. TCP đảm bảo dữ liệu sẽ được chuyển tới thiết bị đích, nếu dữ liệu không tới được thiết bị đích thì thiết bị gởi sẽ nhận được thông báo lỗi.

Các nhà lập trình mạng phải hiểu cách hoạt động cơ bản của TCP và đặc biệt là phải hiểu cách TCP truyền tải dữ liệu giữ các thiết bị mạng. Hình sau cho thấy những trường của TCP Header. Những trường này chứa các thông tin cần thiết cho việc thực thi kết nối và truyền tải dữ liệu một cách tin tưởng.



Hình I.4: Các trường của TCP Header

Mỗi trường của TCP Header kết hợp với một chức năng đặc biệt của một phiên làm việc TCP. Có một số chức năng quan trọng sau:

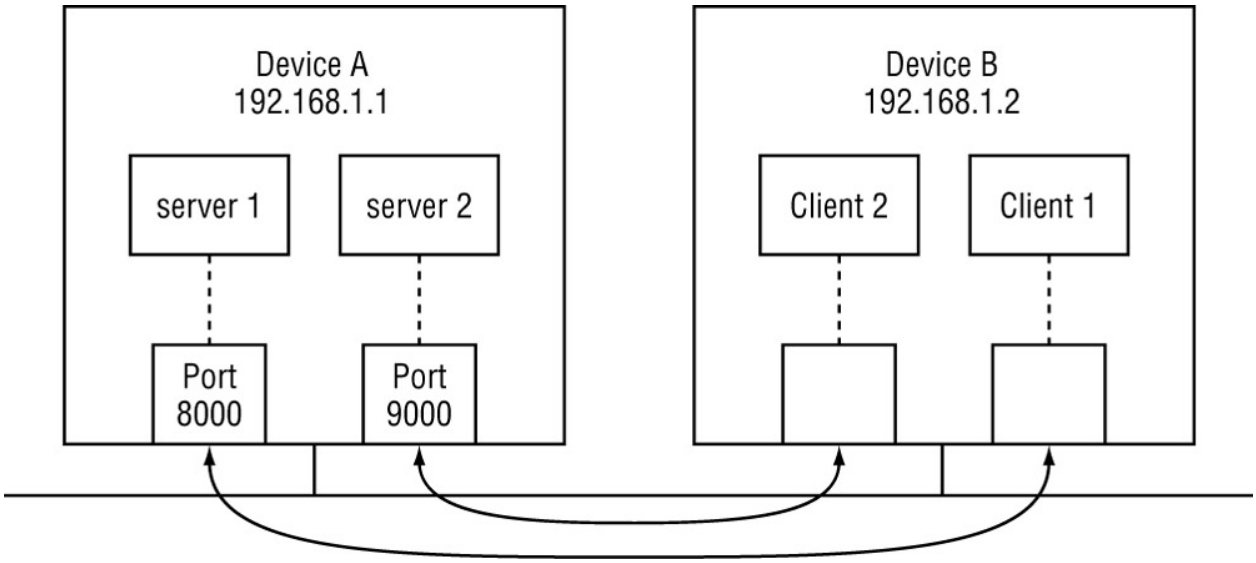
- Source port và Destination port: theo dõi các kết nối giữa các thiết bị
- Sequence và Acknowledgement number: theo dõi thứ tự các gói tin và truyền tải lại các gói tin bị mất
- Flag: mở và đóng kết nối giữa các thiết bị để truyền tải dữ liệu

I.3.1. TCP port

TCP sử dụng các port để xác định các kết nối TCP trên một thiết bị mạng. Để liên lạc với một ứng dụng chạy trên một thiết bị mạng ở xa ta cần phải biết hai thông tin :

- Địa chỉ IP của thiết bị ở xa
- TCP port được gán cho thiết bị ở xa

Để kết nối TCP được thành lập, thiết bị ở xa phải chấp nhận các gói tin truyền đến port đã được gán. Bởi vì có nhiều ứng dụng chạy trên một thiết bị sử dụng TCP do đó thiết bị phải cấp phát các cổng khác nhau cho các ứng dụng khác nhau.



Hình I.5: Kết nối TCP đơn giản

Trong hình trên thì thiết bị A đang chạy hai ứng dụng Server, hai ứng dụng này đang chờ các gói tin từ Client. Một ứng dụng được gán port 8000 và một ứng dụng được gán port 9000. Thiết bị mạng B muốn kết nối đến thiết bị mạng A thì nó phải được gán một TCP port còn trống từ hệ điều hành và port này sẽ được mở trong suốt phiên làm việc. Các port ở Client thường không quan trọng và có thể gán bất kỳ một port nào hợp lệ trên thiết bị.

Tổ hợp của một địa chỉ IP và một port là một IP endpoint. Một phiên làm việc TCP được định nghĩa là một sự kết hợp của một IP endpoint cục bộ và một IP endpoint ở xa. Một ứng dụng mạng có thể sử dụng cùng một IP endpoint cục bộ nhưng mỗi thiết bị ở xa phải sử dụng một địa chỉ IP hay port riêng.

IANA định nghĩa một danh sách các port TCP tiêu chuẩn được gán cho các ứng dụng đặc biệt:

Port	**Mô Tả**
7	Echo
13	Daytime
17	Quote of the day
20	FTP (data channel)
21	FTP (control channel)

Port	**Mô Tả**
22	SSH
23	Telnet
25	SMTP
37	Time
80	HTTP
110	POP3
119	NNTP
123	Network Time Protocol (NTP)
137	NETBIOS name service
138	NETBIOS datagram service
143	Internet Message Access Protocol (IMAP)
389	Lightweight Directory Access Protocol (LDAP)
443	Secure HTTP (HTTPS)
993	Secure IMAP
995	Secure POP3

Các port từ 0->1023 được gán cho các ứng dụng thông dụng do đó với các ứng dụng mà các lập trình viên tạo ra thì các port được gán phải từ 1024->65535.

I.3.2. Cơ chế đảm bảo độ tin cậy truyền tải các gói tin

Trường tiếp theo trong TCP Header sau port là số sequence và acknowledgement. Những giá trị này cho phép TCP theo dõi các gói tin và đảm bảo nó được nhận theo đúng thứ tự. Nếu bất kỳ gói tin nào bị lỗi, TCP sẽ yêu cầu truyền tải lại các gói tin bị lỗi và ráp chúng lại trước khi gởi gói tin cho ứng dụng.

Mỗi gói tin có một số duy nhất sequence cho một phiên làm việc TCP. Một số ngẫu nhiên được chọn cho gói tin đầu tiên được gởi đi trong phiên làm việc. Mỗi gói tin tiếp theo được gởi sẽ tăng số sequence bằng số byte dữ liệu TCP trong gói tin trước đó. Điều này đảm bảo mỗi gói tin được xác định duy nhất trong luồng dữ liệu TCP.

Thiết bị nhận sử dụng trường acknowledgement để hồi báo số sequence cuối cùng được nhận từ thiết bị gởi. Thiết bị nhận có thể nhận nhiều gói tin trước khi gởi lại một hồi báo. Số acknowledgement được trả về là số sequence cao nhất liền sau của dữ liệu được nhận. Kỹ thuật này được gọi là cửa sổ trượt. Các gói tin được nhận ngoài thứ

tự có thể được giữ trong bộ đệm và được đặt vào đúng thứ tự khi các gói tin khác đã được nhận thành công. Nếu một gói tin bị mất, thiết bị nhận sẽ thấy được số sequence bị lỗi và gởi một số acknowledgement thấp hơn để yêu cầu các gói tin bị lỗi. Nếu không có cửa sổ trượt mỗi gói tin sẽ phải hồi báo lại, làm tăng băng thông và độ trễ mạng.

I.3.3. Quá trình thành lập một phiên làm việc TCP

Quá trình làm thành lập một phiên làm việc TCP được thực hiện nhờ vào việc sử dụng các cờ (Flag):

Flag	**Mô Tả**
6 bit dành riêng	Dành riêng để sử dụng trong tương lai, giá trị luôn luôn là zero
1-bit URG flag	Đánh dấu gói tin là dữ liệu khẩn cấp
1-bit ACK flag	Hồi báo nhận một gói tin
1-bit PUSH flag	Cho biết dữ liệu được đẩy vào ứng dụng ngay lập tức
1-bit RESET flag	Thiết lập lại tình trạng khởi đầu kết nối TCP
1-bit SYN flag	Bắt đầu một phiên làm việc
1-bit FIN flag	Kết thúc một phiên làm việc

TCP sử dụng các tình trạng kết nối để quyết định tình trạng kết nối giữa các thiết bị. Một giao thức bắt tay đặc biệt được dùng để thành lập những kết nối này và theo dõi tình trạng kết nối trong suốt phiên làm việc. Một phiên làm việc TCP gồm ba pha sau:

- Mở bắt tay
- Duy trì phiên làm việc
- Đóng bắt tay

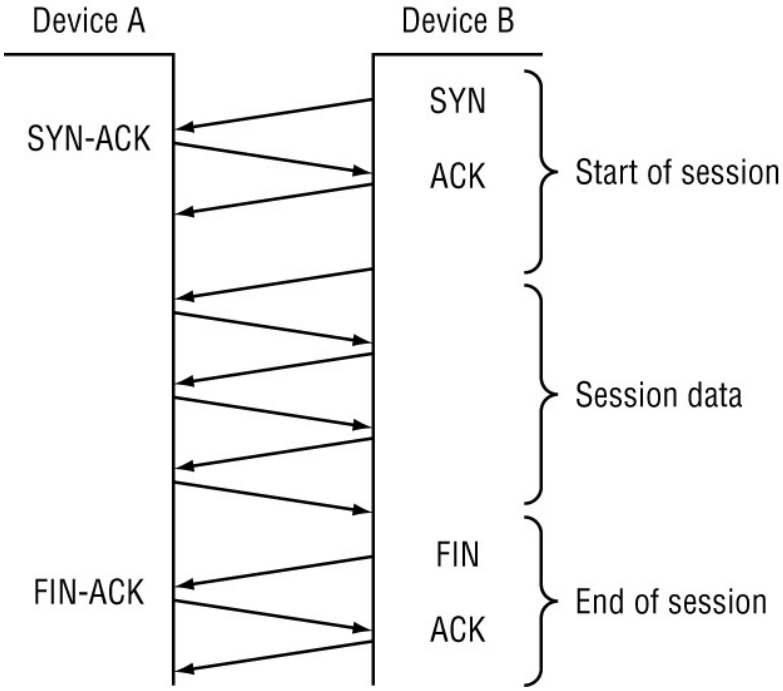
Mỗi pha yêu cầu các bit cờ được thiết lập trong một thứ tự nào đó. Quá trình mở bắt tay thường được gọi là ba cái bắt tay và nó yêu cầu ba bước để thành lập kết nối.

- Thiết bị gởi gởi cờ SYN cho biết bắt đầu phiên làm việc
- Thiết bị nhận gởi cả cờ SYN và cờ ACK trong cùng một gói tin cho biết nó chấp nhận bắt đầu phiên làm việc

- Thiết bị gởi gởi cờ ACK cho biết phiên làm việc đã mở và đã sẵng sàng cho việc gởi và nhận các gói tin.

Sau khi phiên làm việc được thành lập, cờ ACK sẽ được thiết lập trong các gói tin. Để đóng phiên làm việc, một quá trình bắt tay khác được thực hiện dùng cờ FIN:

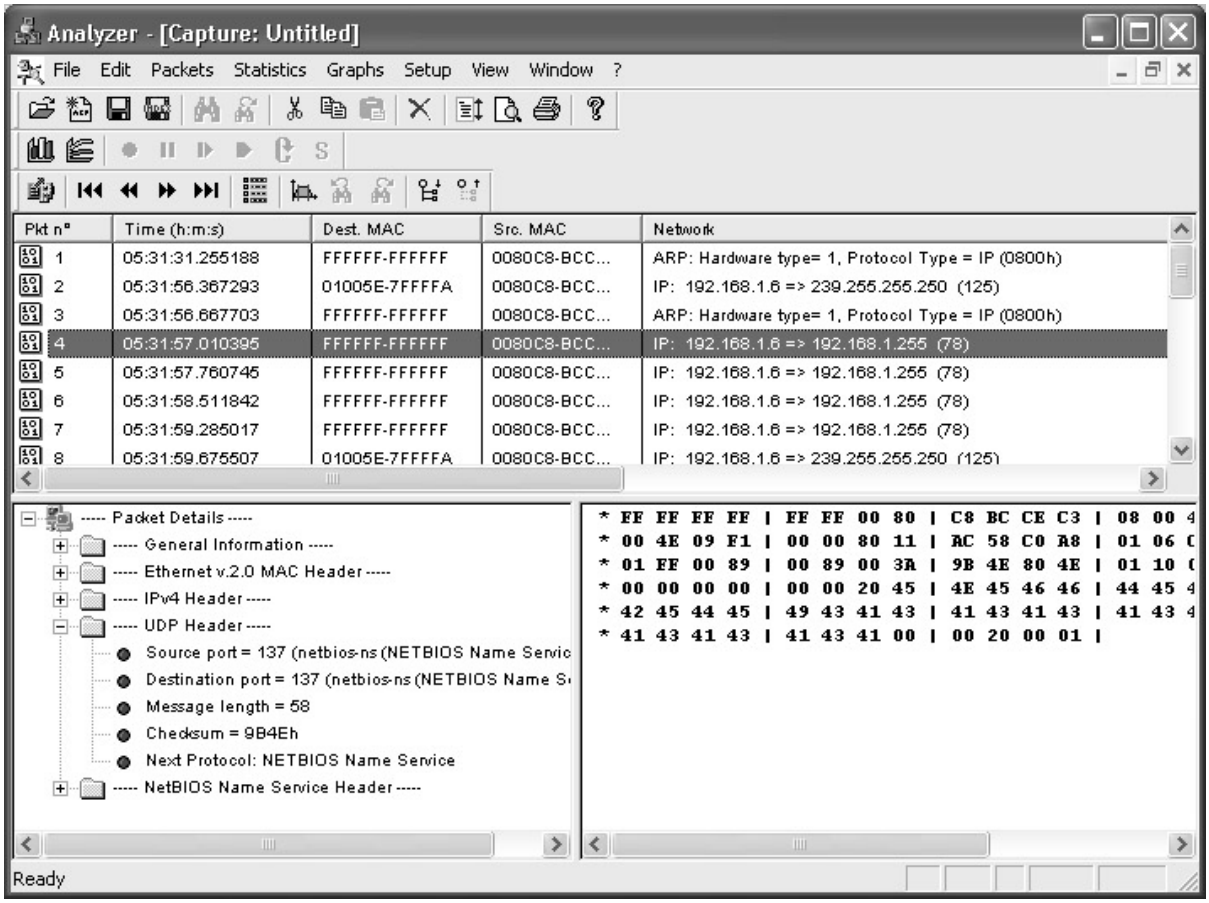
- Thiết bị khởi đầu đóng kết nối gởi cờ FIN
- Thiết bị bên kia gởi cờ FIN và ACK trong cùng một gói tin cho biết nó chấp nhận đóng kết nối
- Thiết bị khởi đầu đóng kết nối gởi cờ ACK để đóng kết nối



Hình I.6: Các bước bắt tay của giao thức TCP

I.4. Tầng UDP

User Datagram Protocol (UDP) là một giao thức phổ biến khác được dùng trong việc truyền tải dữ liệu của các gói IP. Không giống như TCP, UDP là giao thức phi nối kết. Mỗi phiên làm việc UDP không gì khác hơn là truyền tải một gói tin theo một hướng. Hình sau sẽ mô tả cấu trúc của một gói tin UDP



Hình I.7: UDP Header

UDP header gồm những trường sau:

- Source Port
- Destination Port
- Message Length
- Checksum
- Next Level Protocol

Cũng giống như TCP, UDP theo dõi các kết nối bằng cách sử dụng các port từ 1024->65536, các port UDP từ 0->1023 là các port dành riêng cho các ứng dụng phổ biến, một số dùng phổ biến như:

Port	**Mô Tả**
53	Domain Name System
69	Trivial File Transfer Protocol
111	Remote Procedure Call
137	NetBIOS name service
138	NetBIOS datagram

Port	**Mô Tả**
161	Simple Network Management Protocol

CHƯƠNG II: LẬP TRÌNH SOCKET HƯỚNG KẾT NỐI

II.1. Socket

Trong lập trình mạng dùng Socket, chúng ta không trực tiếp truy cập vào các thiết bị mạng để gởi và nhận dữ liệu. Thay vì vậy, một file mô tả trung gian được tạo ra để điều khiển việc lập trình. Các file mô tả dùng để tham chiếu đến các kết nối mạng được gọi là các Socket. Socket định nghĩa những đặc trưng sau:

- Một kết nối mạng hay một đường ống dẫn để truyền tải dữ liệu
- Một kiểu truyền thông như stream hay datagram
- Một giao thức như TCP hay UDP

Sau khi một Socket được tạo ra nó phải được gắn vào một địa chỉ mạng và một port trên hệ thống cục bộ hay ở xa. Một khi Socket đã được gắn vào các địa chỉ mạng và port, nó có thể được dùng để gởi và nhận dữ liệu trong mạng.

Trong .Net Framework lớp Socket hỗ trợ cho việc lập trình Socket. Phương thức tạo lập như sau:

```
Socket (AddressFamily, SocketType, ProtocolType)
```

Phương thức tạo lập của lớp Socket cần các đối số truyền vào sau:

+AddressFamily: họ địa chỉ được dùng, tham số này có thể có các giá trị sau:

AppleTalk	Địa chỉ AppleTalk
Atm	Native ATM services address.
Banyan	Địa chỉ Banyan
Ccitt	Địa chỉ cho giao thức CCITT, như là X25
Chaos	Địa chỉ cho giao thức MIT CHAOS
Cluster	Địa chỉ cho các sản phẩm cluster của Microsoft
DataKit	Địa chỉ cho giao thức Datakit
DataLink	Địa chỉ của giao thức tầng data-link
DecNet	Địa chỉ DECnet

Ecma	Địa chỉ ECMA (European Computer Manufacturers Association)
FireFox	Địa chỉ FireFox
HyperChannel	Địa chỉ NSC Hyperchannel
Ieee12844	Địa chỉ workgroup IEEE 1284.4
ImpLink	Địa chỉ ARPANET IMP
InterNetwork	Địa chỉ IP version 4
InterNetworkV6	Địa chỉ IP version 6
Ipx	Địa chỉ IPX hoặc SPX
Irda	Địa chỉ IrDA
Iso	Địa chỉ cho giao thức ISO
Lat	Địa chỉ LAT
Max	Địa chỉ MAX
NetBios	Địa chỉ NetBios
NetworkDesigners	Địa chỉ Network Designers
NS	Địa chỉ Xerox NS
Osi	Địa chỉ cho giao thức ISO
Pup	Địa chỉ cho giao thức PUP
Sna	Địa chỉ IBM SNA
Unix	Địa chỉ Unix
Unknown	Chưa biết họ địa chỉ
Unspecified	Chưa chỉ ra họ địa chỉ
VoiceView	Địa chỉ VoiceView

+SocketType: kiểu Socket, tham số này có thể có các giao thức sau:

Kiểu	Mô tả
Dgram	Được sử dụng trong các giao thức phi kết nối, không tin tưởng. Thông điệp có thể bị mất, bị trùng lặp hoặc có thể đến sai thứ tự.

	Dgram sử dụng giao thức **UDP** và họ địa chỉ InterNetwork.
Raw	Được sử trong các giao thức cấp thấp như Internet Control Message Protocol (Icmp) và Internet Group Management Protocol (Igmp). Ứng dụng phải cung cấp IP header khi gởi. Khi nhận sẽ nhận được IP header và các tùy chọn tương ứng.
Rdm	Được sử dụng trong các giao thức phi kết nối, hướng thông điệp, truyền thông điệp tin cậy, và biên của thông điệp được bảo vệ. Rdm (Reliably Delivered Messages) thông điệp đến không bị trùng lặp và đúng thứ tự. Hơn nữa, thiết bị nhận được thiết bị nếu thông điệp bị mất. Nếu khởi tạo **Socket** dùng Rdm, ta không cần yêu cầu kết nối tới host ở xa trước khi gởi và nhận dữ liệu.
Seqpacket	Cung cấp hướng kết nối và truyền 2 chiều các dòng byte một cách tin cậy . Seqpacket không trùng lập dữ liệu và bảo vệ biên dữ liệu. **Socket** kiểu **Seqpacket** truyền thông với 1 máy đơn và yêu cầu kết nối trước khi truyền dữ liệu.
Stream	Được sử dụng trong các giao thức hướng kết nối, không bị trùng lặp dữ liệu, không bảo vệ biên dữ liệu. **Socket** kiểu Stream chỉ truyền thông với một máy đơn và yêu cầu kết nối trước khi truyền dữ liệu. **Stream** dùng giao thức Transmission Control Protocol (**Tcp**) và họ địa chỉ **InterNetwork**
Unknown	Chưa biết kiểu **Socket**

+ProtocolType: kiểu giao thức, tham số này có thể có các giá trị sau:

ProtocolType	Mô tả
Ggp	Gateway To Gateway Protocol.
Icmp	Internet Control Message Protocol.
IcmpV6	Internet Control Message Protocol IPv6.
Idp	Internet Datagram Protocol.
Igmp	Internet Group Management Protocol.
IP	Internet Protocol.
IPSecAuthenticationHeader	IPv6 Authentication.
IPSecEncapsulatingSecurityPayload	IPv6 Encapsulating Security Payload header.

IPv4	Internet Protocol version 4.
IPv6	Internet Protocol version 6 (IPv6).
Ipx	Internet Packet Exchange Protocol.
ND	Net Disk Protocol (unofficial).
Pup	PARC Universal Packet Protocol.
Raw	Raw IP packet protocol.
Spx	Sequenced Packet Exchange protocol.
SpxII	Sequenced Packet Exchange version 2 protocol.
Tcp	Transmission Control Protocol.
Udp	User Datagram Protocol.
Unknown	Giao thức chưa biết
Unspecified	Giao thức chưa được chỉ ra

Ví dụ phương thức tạo lập của lớp Socket:

```
Socket sk = Socket(AddressFamily.InterNetwork, SocketType.Stream,
ProtocolType.Tcp);
```

II.2. IPAddress

IPAddress là một đối tượng dùng để mô tả một địa chỉ IP, đối tượng này có thể được sử dụng trong nhiều phương thức của Socket. Một số phương thức của lớp IPAddress

Phương Thức	**Mô Tả**
Equals	So sánh 2 địa chỉ IP
GetHashCode	Lấy giá trị has cho 1 đối tượng IPAddress
GetType	Trả về kiểu của một thể hiện địa chỉ IP
HostToNetworkOrder	Chuyển 1 địa chỉ IP từ host byte order thành network byte order
IsLoopBack	Cho biết địa chỉ IP có phải là địa chỉ LoopBack hay không
NetworkToHostOrder	Chuyển 1 địa chỉ IP từ network byte order thành host byte order

Phương Thức	**Mô Tả**
Parse	Chuyển 1 chuỗi thành 1 thể hiện IPAddress
ToString	Chuyển 1 đối tượng IPAddress thành một chuỗi

Phương thức Parse() thường được dùng để tạo ra 1 thể hiện của IPAddress:

```
IPAddress localIpAddress = IPAddress.Parse("127.0.0.1");
```

Lớp IPAddress cũng cung cấp 4 thuộc tính để mô tả các địa chỉ IP đặc biệt:

- Any: dùng để mô tả một địa chỉ IP bất kỳ của hệ thống.
- Broadcast: dùng để mô tả địa chỉ IP Broadcast cho mạng cục bộ
- Loopback: dùng để mô tả địa chỉ loopback của hệ thống
- None: không dùng địa chỉ IP

II.3. IPEndPoint

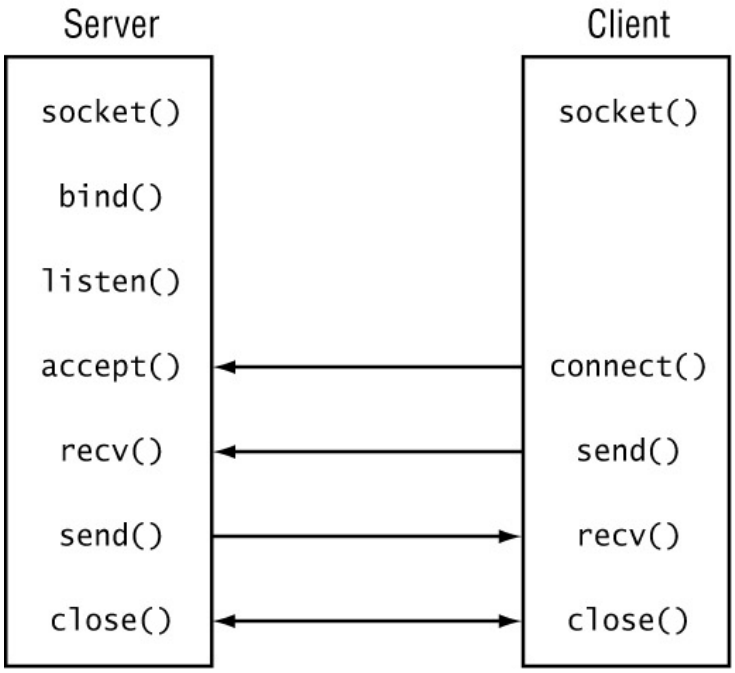
IPEndPoint là một đối tượng mô tả sự kết hợp của một địa chỉ IP và port. Đối tượng IPEndPoint được dùng để gắn kết các Socket với các địa chỉ cục bộ hoặc các địa chỉ ở xa.

Hai thuộc tính của IPEndPoint có thể được dùng để lấy được vùng các port trên hệ thống là MinPort và MaxPort.

II.4. Lập trình Socket hướng kết nối

Trong lập trình Socket hướng kết nối, giao thức TCP được dùng để thành lập phiên làm việc giữa hai endpoint. Khi sử dụng giao thức TCP để thành lập kết nối ta phải đàm phán kết nối trước nhưng khi kết nối đã được thành lập dữ liệu có thể truyền đi giữa các thiết bị một cách tin tưởng.

Để lập trình Socket hướng kết nối ta phải thực hiện một loạt các thao tác giữa clien và Server như trong mô hình bên dưới



Hình II.1: Mô hình lập trình Socket hướng kết nối

II.4.1. Lập trình phía Server

Đầu tiên Server sẽ tạo một Socket, Socket này sẽ được gắn vào một địa chỉ ip và một port cục bộ, hàm để thực hiện việc này là hàm Bind(). Hàm này cần một danh đối số là một IPEndPoint cục bộ:

```
IPEndPoint ipep = new IPEndPoint(IPAddress.Any, 5000);
Socket server = new Socket(AddressFamily.InterNetwork,
SocketType.Stream, ProtocolType.Tcp);
server.Bind(ipep);
```

Bởi vì Server thường chấp nhận kết nối trên chính địa chỉ IP và port riêng của nó nên ta dùng `IPAddress.Any` để chấp nhận kết nối trên bất kỳ card mạng nào

Địa chỉ IP ta dùng ở đây là địa chỉ IP version 4 và kiểu giao thức là TCP nên AddressFamily là InterNetwork và SocketType là Stream.

Sau khi Socket đã được gắn kết vào một địa chỉ và một port, Server phải sẵn sàng chấp nhận kết nối từ Client. Việc này được thực hiện nhờ vào hàm Listen().Hàm Listen() có một đối số, đó chính là số Client tối đa mà nó lắng nghe.

```
server.Listen(10);
```

Tiếp theo Server dùng hàm Accept() để chấp nhận kết nối từ Client:

```
Socket client = server.Accept();
```

Hàm Accept() này sẽ dừng Server lại và chờ cho đến khi nào có Client kết nối đến nó sẽ trả về một Socket khác, Socket này được dùng để trao đổi dữ liệu với Client. Khi đã chấp nhận kết nối với Client thì Server có thể gởi và nhận dữ liệu với Client thông qua phương thức Send() và Receive().

```
string welcome = "Hello Client";
buff = Encoding.ASCII.GetBytes(welcome);
client.Send(buff, buff.Length, SocketFlags.None);
```

Phương thức Send() của Socket dùng để gởi dữ liệu, phương thức này có một số đối số quan trọng sau:

- ✓ Buff : mảng các byte cần gởi
- ✓ Offset: vị trí đầu tiên trong mảng cần gởi
- ✓ Size: số byte cần gởi
- ✓ SocketFlags: chỉ ra cách gởi dữ liệu trên Socket

Việc gởi và nhận dữ liệu được thực hiện liên tục thông qua một vòng lặp vô hạn:

```
while (true)
{
      buff = new byte[1024];
      recv = client.Receive(buff);
      if (recv == 0)
           break;

      Console.WriteLine(Encoding.ASCII.GetString(buff, 0, recv));
      client.Send(buff, recv, SocketFlags.None);
}
```

Phương thức Receive() đặt dữ liệu vào buffer, kích thước buffer được thiết lập lại, do đó nếu buffer không được thiết lập lại, lần gọi phương thức Receive() kế tiếp sẽ chỉ có thể nhận được dữ liệu tối đa bằng lần nhận dữ liệu trước.

Phương thức này có một số đối số quan trọng sau:

- ✓ Buff : mảng các byte cần gởi
- ✓ Offset: vị trí đầu tiên trong mảng cần nhận
- ✓ Size: số byte cần gởi
- ✓ SocketFlags: chỉ ra cách nhận dữ liệu trên Socket

Phương thức Receive() trả về số byte dữ liệu nhận được từ Client. Nếu không có dữ liệu được nhận, phương thức Receive() sẽ bị dừng lại và chờ cho tới khi có dữ liệu. Khi Client gởi tín hiệu kết thúc phiên làm việc (bằng cách gởi cờ FIN trong gói TCP), phương thức Receive() sẽ trả về giá trị 0. Khi phương thức Receive() trả về giá trị 0, ta đóng Socket của Client lại bằng phương thức Close(). Socket chính (Server Socket) vẫn còn hoạt động để chấp nhận các kết nối khác. Nếu không muốn Client nào kết nối đến nữa thì ta đóng Server lại luôn:

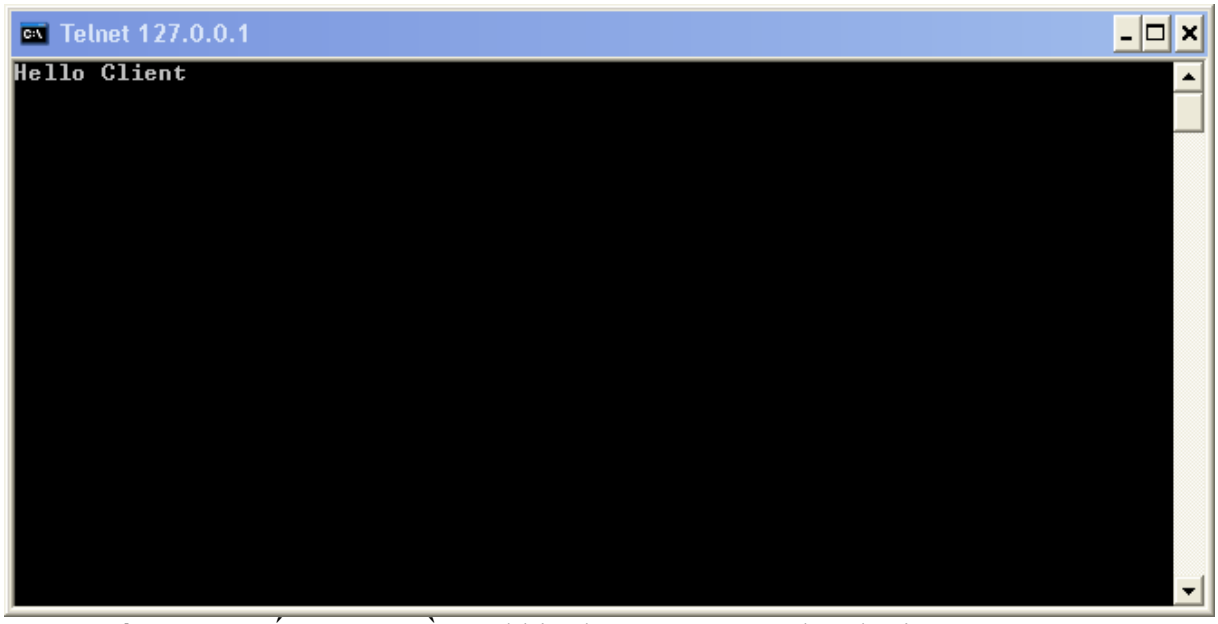
```
client.Close();
server.Close();
```

Chương trình TCP Server đơn giản:

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
class TcpServerDonGian
{
    public static void Main()
    {
        //Số byte thực sự nhận được dùng hàm Receive()
        int byteReceive;
        //buffer để nhận và gởi dữ liệu
        byte[] buff = new byte[1024];
        //EndPoint cục bộ
        IPEndPoint ipep = new IPEndPoint(IPAddress.Any, 5000);
        //Server Socket
        Socket server = new Socket(AddressFamily.InterNetwork,
SocketType.Stream, ProtocolType.Tcp);
        //Kết nối server với 1 EndPoint
        server.Bind(ipep);
        //Server lắng nghe tối đa 10 kết nối
        server.Listen(10);
        Console.WriteLine("Dang cho Client ket noi den...");
        //Hàm Accept() sẽ block server lại cho đến khi có Client kết nối đến
        Socket client = server.Accept();
        //Client EndPoint
        IPEndPoint clientep = (IPEndPoint)client.RemoteEndPoint;
        Console.WriteLine("Da ket noi voi Client {0} tai port {1}",
clientep.Address, clientep.Port);

        string welcome = "Hello Client";
```

```
            //Chuyển chuỗi thành mảng các byte
            buff = Encoding.ASCII.GetBytes(welcome);
            //Gởi câu chào cho Client
            client.Send(buff, buff.Length, SocketFlags.None);
            while (true)
            {
                //Reset lại buffer
                buff = new byte[1024];
                //Lấy số byte thực sự nhận được
                byteReceive = client.Receive(buff);
                //Nếu Client ngắt kết nối thì thoát khỏi vòng lặp
                if (byteReceive == 0)
                    break;

                Console.WriteLine(Encoding.ASCII.GetString(buff, 0, byteReceive));
                //Sau khi nhận dữ liệu xong, gởi lại cho Client
                client.Send(buff, byteReceive, SocketFlags.None);
            }
            Console.WriteLine("Da dong ket noi voi Client: {0}", clientep.Address);
            //Đóng kết nối
            client.Close();
            server.Close();
        }
}
```

Để kiểm tra thử chương trình ta có thể dùng chương trình Telnet của Windows để kiểm tra. Dùng lệnh telnet 127.0.0.1 5000



Hình II.2: Kết quả trả về sau khi telnet vào Server local tại port 5000

Sau khi dùng lệnh telnet, kết quả trả về như trên hình là đã kết nối thành công

II.4.2. Lập trình phía Client

Lập trình Socket hướng kết nối phía Client đơn giản hơn phía Server. Client cũng phải gắn kết một địa chỉ của một Socket đã được tạo ra nhưng sử dụng phương thức Connect() chứ không sử dụng phương thức Bind() giống như phía Server. Phương thức Connect() yêu cầu một IPEndPoint của Server mà Client cần kết nối đến.

```
IPEndPoint ipep = new IPEndPoint(IPAddress.Parse("127.0.0.1"),
5000);
Socket server = new Socket(AddressFamily.InterNetwork,
SocketType.Stream, ProtocolType.Tcp);

try
{
        server.Connect(ipep);
}
catch (SocketException e)
{
        Console.WriteLine("Không thể kết nối đến Server");
        Console.WriteLine(e.ToString());
        return;
}
```

Phương thức Connect() sẽ dừng lại cho đến khi Client kết nối được với Server. Nếu kết nối không thể được thực hiện thì nó sẽ phát sinh ra một biệt lệ, do đó hàm Connect() tra phải để trong khối try, catch để không bị lỗi chương trình.

Khi kết nối được thành lập, Client có thể dùng phương thức Send() và Receive() của lớp Socket để gởi và nhận dữ liệu tương tự như Server đã làm. Khi quá trình trao đổi dữ liệu đã hoàn tất, đối tượng Socket phải được đóng lại. Client Socket dùng phương thức Shutdown() để dừng Socket và dùng phương thức Close() để thực sự đóng phiên làm việc. Phương thức Shutdown() của Socket dùng một tham số để quyết định cách Socket sẽ dừng lại. Các phương thức đó là:

Giá trị	**Mô tả**

Giá trị	Mô tả
SocketShutdown.Both	Ngăn cản gởi và nhận dữ liệu trên Socket.
SocketShutdown.Receive	Ngăn cản nhận dữ liệu trên Socket. Cờ RST sẽ được gởi nếu có thêm dữ liệu được nhận.
SocketShutdown.Send	Ngăn cản gởi dữ liệu trên Socket. Cờ FIN sẽ được gởi sau khi tất cả dữ liệu còn lại trong buffer đã được gởi đi.

Chương trình TCP Client đơn giản:

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
class SimpleTcpClient
{
    public static void Main()
    {
        //Buffer để gởi và nhận dữ liệu
        byte[] buff = new byte[1024];
        //Chuỗi nhập vào và chuỗi nhận được
        string input, stringData;
        //IPEndPoint ở server
        IPEndPoint ipep = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 5000);
        //Server Socket
        Socket server = new Socket(AddressFamily.InterNetwork,
SocketType.Stream, ProtocolType.Tcp);
        //Hàm Connect() sẽ bị block lại và chờ khi kết nối được với server thì
mới hết block
        try
        {
            server.Connect(ipep);
        }
        //Quá trình kết nối có thể xảy ra lỗi nên phải dùng try, catch
        catch (SocketException e)
        {
            Console.WriteLine("Không thể kết nối đến Server");
            Console.WriteLine(e.ToString());
            return;
        }
        //Số byte thực sự nhận được
        int byteReceive = server.Receive(buff);
        //Chuỗi nhận được
        stringData = Encoding.ASCII.GetString(buff, 0, byteReceive);
```

```
            Console.WriteLine(stringData);
            while (true)
            {
                //Nhập dữ liệu từ bàn phím
                input = Console.ReadLine();
                //Nếu nhập exit thì thoát và đóng Socket
                if (input == "exit")
                    break;
                //Gởi dữ liệu cho server
                server.Send(Encoding.ASCII.GetBytes(input));
                //Reset lại buffer
                buff = new byte[1024];
                //Số byte thực sự nhận được
                byteReceive = server.Receive(buff);
                //Chuỗi nhận được
                stringData = Encoding.ASCII.GetString(buff, 0, byteReceive);
                Console.WriteLine(stringData);
            }
            Console.WriteLine("Dong ket noi voi server...");
            //Dừng kết nối, không cho phép nhận và gởi dữ liệu
            server.Shutdown(SocketShutdown.Both);
            //Đóng Socket
            server.Close();
        }
    }
```

II.4.3. Vấn đề với bộ đệm dữ liệu

Trong ví dụ Client, Server đơn giản trên thì một mảng các byte được dùng như là bộ đệm để gởi và nhận dữ liệu trên Socket. Bởi vì chương trình được chạy trong môi trường được điều khiển, tất cả các thông điệp đều thuộc dạng text và kích thước nhỏ nên loại buffer này không phải là một vấn đề.

Trong thế giới thực, chúng ta không biết kích thước và kiểu dữ liệu đến trong khi truyền thông giữa Client và Server. Vấn đề xảy ra khi khi dữ liệu đến lớn hơn kích thước bộ đệm dữ liệu.

Khi nhận dữ liệu thông qua TCP, dữ liệu được lưu trữ trong bộ đệm hệ thống. Mỗi khi gọi phương thức Receive(), nó sẽ đọc dữ liệu từ bộ đệm TCP và lấy dữ liệu ra khỏi bộ đệm. Số lượng dữ liệu được đọc bởi phương thức Receive() được điều khiển bởi hai yếu tố sau:

- Kích thước bộ đệm dữ liệu được chỉ ra trong phương thức Receive()
- Kích thước bộ đệm được chỉ ra trong tham số của phương thức Receive()

Trong ví dụ đơn giản trên, buffer được định nghĩa là một mảng byte kích thước 1024. Bởi vì kích thước dữ liệu không được chỉ ra trong phương thức Receive() nên kích thước bộ đệm tự động lấy kích thước mặc định của bộ đệm dữ liệu là 1024 byte. Phương thức Receive() sẽ đọc 1024 byte dữ liệu một lần và đặt dữ liệu đọc được vào biến buff

```
byteReceive = client.Receive(buff);
```

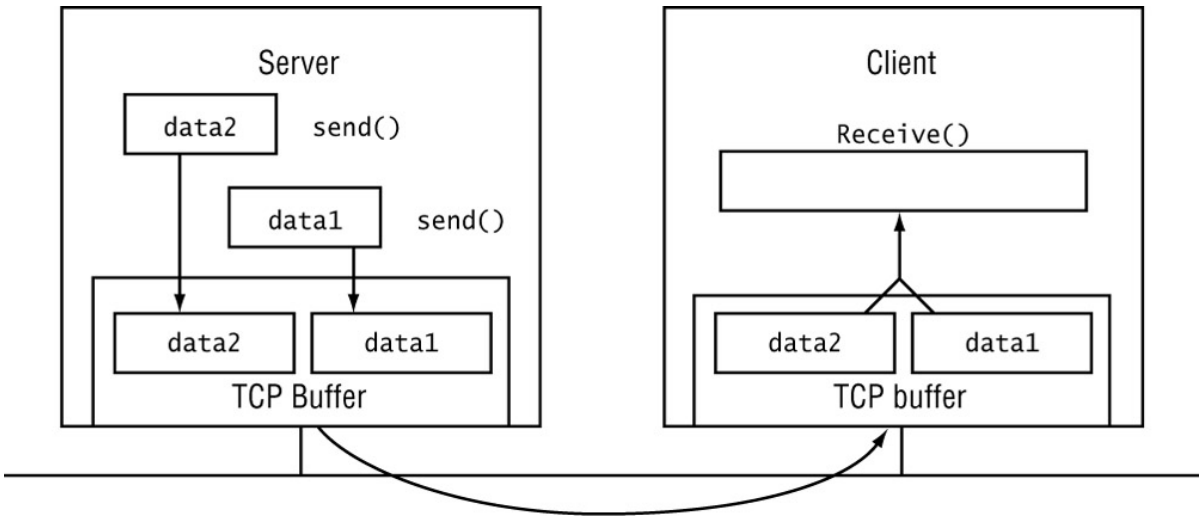
Vào lúc phương thức Receive() được gọi, nếu bộ đệm TCP chứa ít hơn 1024 byte, phương thức này sẽ trả về số lượng dữ liệu mà nó thực sự đọc được trong biến byte Receive. Để chuyển dữ liệu thành chuỗi, ta dùng phương thức GetString() như sau:

```
stringData = Encoding.ASCII.GetString(buff, 0, byteReceive);
```

Trong đối số của hàm GetString, ta phải truyền vào số byte thực sự đã đọc được nếu không ta sẽ nhận được một chuỗi với các byte thừa ở đằng sau.

II.4.4. Xử lý với các bộ đệm có kích thước nhỏ

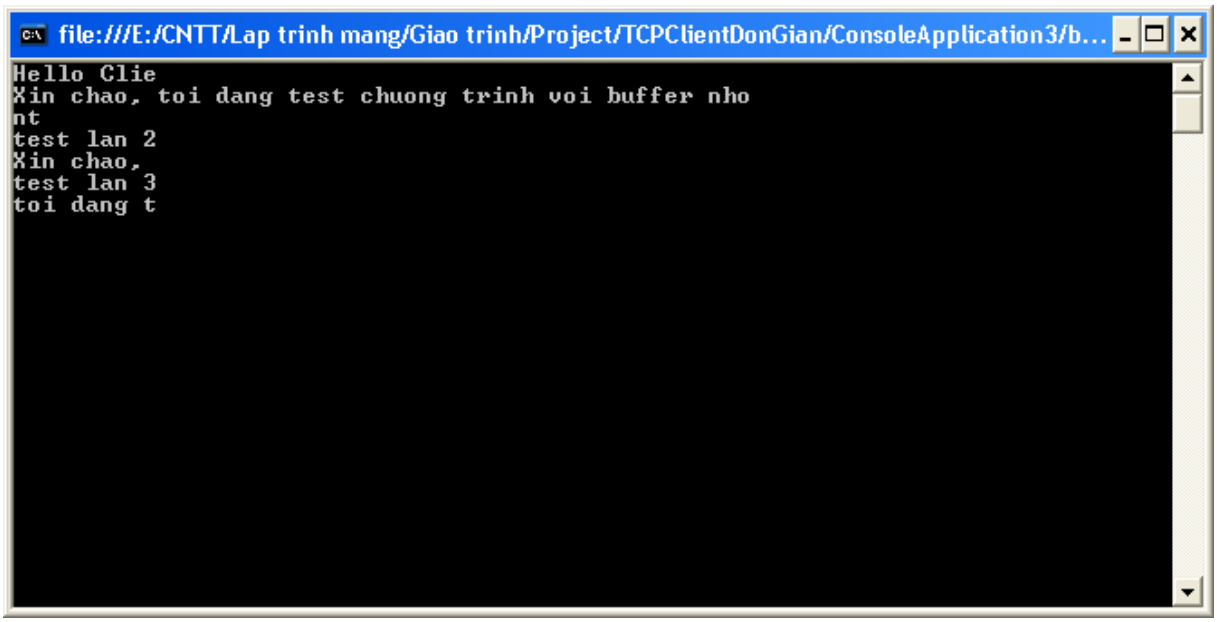
Hệ điều hành Window dùng bộ đệm TCP để gởi và nhận dữ liệu. Điều này là cần thiết để TCP có thể gởi lại dữ liệu bất cứ lúc nào cần thiết. Một khi dữ liệu đã được hồi báo nhận thành công thì nó mới được xóa khỏi bộ đệm.



Hình II.3: TCP Buffer

Dữ liệu đến cũng được hoạt động theo cách tương tự. Nó sẽ ở lại trong bộ đệm cho đến khi phương thức Receive() được dùng để đọc nó. Nếu phương thức Receive() không đọc toàn bộ dữ liệu ở trong bộ đệm, phần còn lại vẫn được nằm ở đó và chờ phương thức Receive() tiếp theo được đọc. Dữ liệu sẽ không bị mất nhưng chúng ta sẽ không lấy được các đoạn dữ liệu mình mong muốn.

Để thấy được vấn đề, ta tiến hành thay đổi kích thước bộ đệm từ 1024 byte xuống còn 10 byte. Và chạy lại chương trình Client, Server đơn giản trên



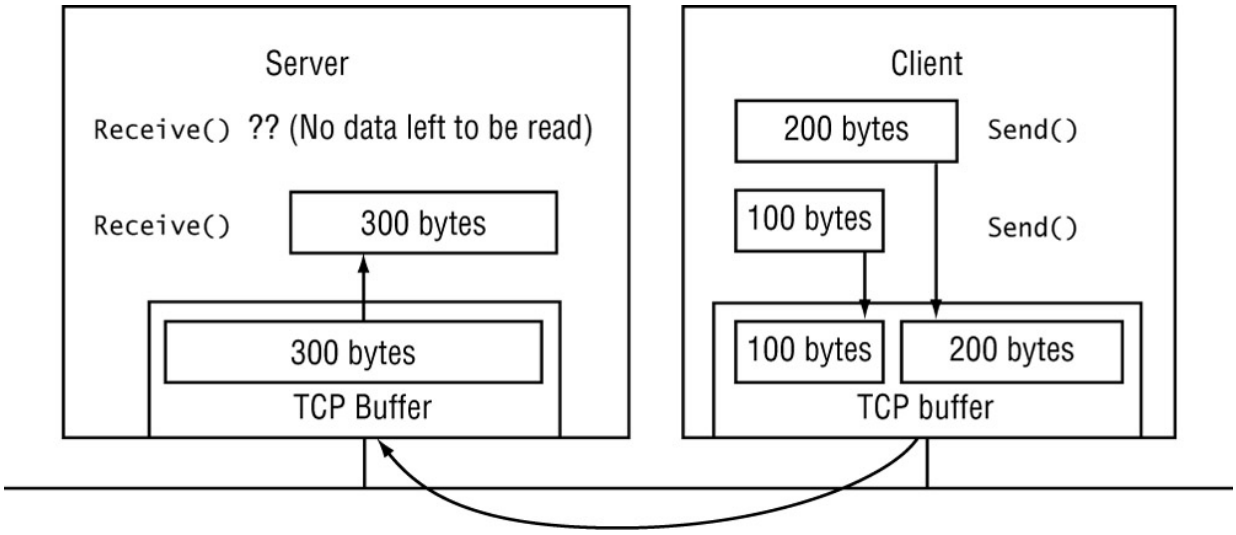
Hình II.4: Kết quả trả về khi chạy chương trình với buffer nhỏ

Bởi vì bộ đệm dữ liệu không đủ lớn để lấy hết dữ liệu ở bộ đệm TCP nên phương thức Receive() chỉ có thể lấy được một lượng dữ liệu có độ lớn đúng bằng độ lớn của bộ đệm dữ liệu, phần còn lại vẫn nằm ở bộ đệm TCP và nó được lấy khi gọi lại phương thức Receive(). Do đó câu chào Client của Server phải dùng tới hai lần gọi phương thức Receive() mới lấy được hết. Trong lần gởi và nhận dữ liệu kế tiếp, đoạn dữ liệu tiếp theo được đọc từ bộ đệm TCP do đó nếu ta gởi dữ liệu với kích thước lớn hơn 10 byte thì khi nhận ta chỉ nhận được 10 byte đầu tiên.

Bởi vì vậy nên trong khi lập trình mạng chúng ta phải quan tâm đến việc đọc dữ liệu từ bộ đệm TCP một cách chính xác. Bộ đệm quá nhỏ có thể dẫn đến tình trạng thông điệp nhận sẽ không khớp với thông điệp gởi, ngược lại bộ đệm quá lớn sẽ làm cho các thông điệp bị trộn lại, khó xử lý. Việc khó nhất là làm sao phân biệt được các thông điệp được đọc từ Socket.

II.4.5. Vấn đề với các thông điệp TCP

Một trong những khó khăn của những nhà lập trình mạng khi sử dụng giao thức TCP để chuyển dữ liệu là giao thức này không quan tâm đến biên dữ liệu.



Hình II.5: Client Send hai lần rồi Server mới Receive

Như trên hình vấn đề xảy ra khi truyền dữ liệu là không đảm bảo được mỗi phương thức Send() sẽ không được đọc bởi một phương thức Receive(). Tất cả dữ liệu được đọc từ phương thức Receive() không thực sự được đọc trực tiếp từ mạng mà nó được đọc từ bộ đệm TCP. Khi các gói tin TCP được nhận từ mạng sẽ được đặt theo thứ tự trong bộ đệm TCP. Mỗi khi phương thức Receive() được gọi, nó sẽ đọc dữ liệu trong bộ đệm TCP, không quan tâm đến biên dữ liệu.

Chúng ta hãy xem xét ví dụ sau, Chương Trình BadTCPServer:

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
class BadTcpServer
{
    public static void Main()
    {
        //Số byte thực sự nhận được dùng hàm Receive()
        int byteReceive;
        //buffer để nhận và gởi dữ liệu
        byte[] buff = new byte[1024];
        //EndPoint cục bộ
        IPEndPoint ipep = new IPEndPoint(IPAddress.Any, 5000);
        //Server Socket
```

```
        Socket server = new Socket(AddressFamily.InterNetwork,
SocketType.Stream, ProtocolType.Tcp);
        //Kết nối server với 1 EndPoint
        server.Bind(ipep);
        //Server lắng nghe tối đa 10 kết nối
        server.Listen(10);
        Console.WriteLine("Dang cho Client ket noi den...");
        //Hàm Accept() sẽ block server lại cho đến khi có Client kết nối đến
        Socket client = server.Accept();
        //Client EndPoint
        IPEndPoint clientep = (IPEndPoint)client.RemoteEndPoint;
        Console.WriteLine("Da ket noi voi Client {0} tai port {1}",
clientep.Address, clientep.Port);

        string welcome = "Hello Client";
        //Chuyển chuỗi thành mảng các byte
        buff = Encoding.ASCII.GetBytes(welcome);
        //Gởi câu chào cho Client
        client.Send(buff, buff.Length, SocketFlags.None);

        for (int i = 0; i < 5; i++)
        {
            byteReceive = client.Receive(buff);
            Console.WriteLine(Encoding.ASCII.GetString(buff, 0, byteReceive));
        }
        Console.WriteLine("Da dong ket noi voi Client: {0}", clientep.Address);
        //Đóng kết nối
        client.Close();
        server.Close();
        Console.Read();

    }
}
```

Chương trình Server thành lập Socket TCP bình thường để lắng nghe kết nối, khi kết nối được thành lập, Server gởi câu chào cho Client và cố gắng nhận năm thông điệp riêng biệt từ Client:

```
for (int i = 0; i < 5; i++)
{
    byteReceive = client.Receive(data);
    Console.WriteLine(Encoding.ASCII.GetString(data, 0,
byteReceive));
```

```
}
```

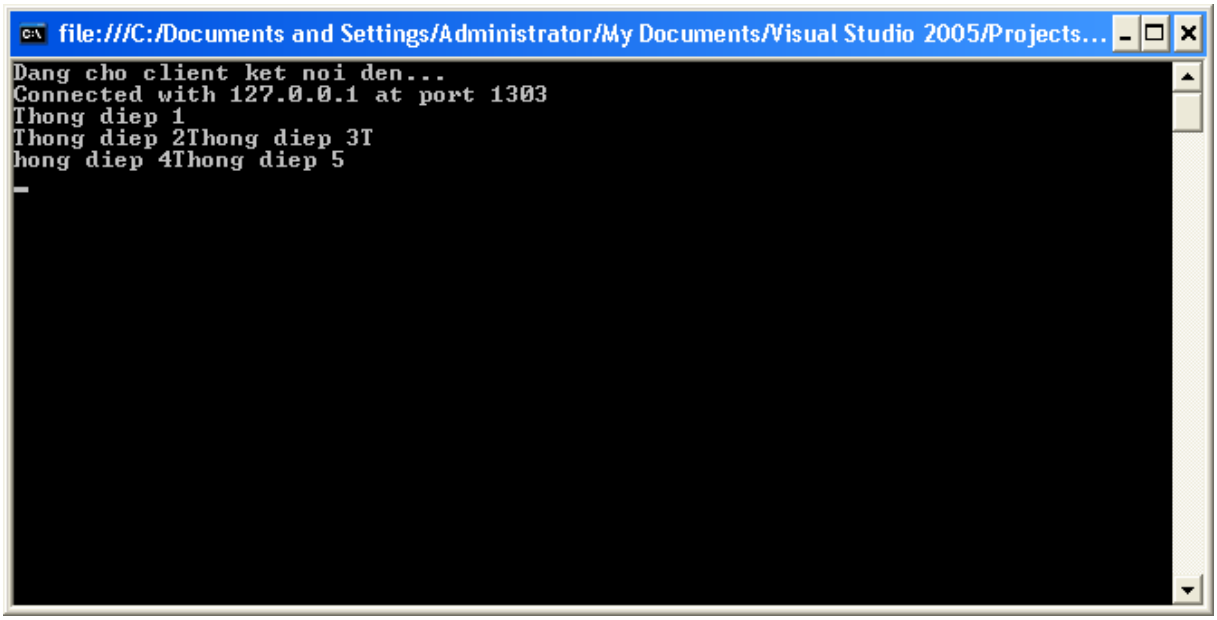
Mỗi khi được gọi, phương thức Receive() đọc toàn bộ dữ liệu trong bộ đệm TCP, sau khi nhận năm thông điệp, kết nối được đóng lại.

Chương trình BadTCPClient:

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
class BadTcpClient
{
    public static void Main()
    {
        //Buffer để gởi và nhận dữ liệu
        byte[] buff = new byte[10];
        //Chuỗi nhập vào và chuỗi nhận được
        string input, stringData;
        //IPEndPoint ở server
        IPEndPoint ipep = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 5000);
        //Server Socket
        Socket server = new Socket(AddressFamily.InterNetwork,
SocketType.Stream, ProtocolType.Tcp);
        //Hàm Connect() sẽ bị block lại và chờ khi kết nối được với server thì
mới hết block
        try
        {
            server.Connect(ipep);
        }
        //Quá trình kết nối có thể xảy ra lỗi nên phải dùng try, catch
        catch (SocketException e)
        {
            Console.WriteLine("Khon the ket noi den Server");
            Console.WriteLine(e.ToString());
            return;
        }
        //Số byte thực sự nhận được
        int byteReceive = server.Receive(buff);
        //Chuỗi nhận được
        stringData = Encoding.ASCII.GetString(buff, 0, byteReceive);
        Console.WriteLine(stringData);

        server.Send(Encoding.ASCII.GetBytes("Thong diep 1"));
        server.Send(Encoding.ASCII.GetBytes("Thong diep 2"));
```

```
            server.Send(Encoding.ASCII.GetBytes("Thong diep 3"));
            server.Send(Encoding.ASCII.GetBytes("Thong diep 4"));
            server.Send(Encoding.ASCII.GetBytes("Thong diep 5"));

            Console.WriteLine("Dong ket noi voi server...");
            //Dừng kết nối, không cho phép nhận và gởi dữ liệu
            server.Shutdown(SocketShutdown.Both);
            //Đóng Socket
            server.Close();
            Console.Read();

        }
}
```

Kết quả chương trình như hình bên dưới



Hình II.6: Kết quả trên Server

Trong lần gọi phương thức Receive() lần đầu tiên, phương thức này nhận toàn bộ dữ liệu từ phương thức Send() của Client gởi lên, trong lần gọi phương thức Receive() lần thứ hai, phương thức Receive() đọc dữ liệu từ hai phương thức Send() và một phương thức Send() khác gởi dữ liệu chưa xong. Trong lần gọi thứ ba thì phương thức Receive() sẽ đọc hết dữ liệu đang được gởi dở từ phương thức Send() và đọc dữ liệu được gởi từ phương thức Send() cuối cùng và sau khi Client thực hiện xong năm phương thức Send() nó sẽ đóng kết nối với Server và Server cũng sẽ thoát ra.

II.4.6. Giải quyết các vấn đề với thông điệp TCP

Để giải quyết vấn đề với biên dữ liệu không được bảo vệ, chúng ta phải tìm hiểu một số kỹ thuật để phân biệt các thông điệp. Ba kỹ thuật thông thường dùng để phân biệt các thông điệp được gởi thông qua TCP:

- ✓ Luôn luôn sử dụng các thông điệp với kích thước cố định
- ✓ Gởi kèm kích thước thông điệp cùng với mỗi thông điệp
- ✓ Sử dụng các hệ thống đánh dấu để phân biệt các thông điệp

II.4.6.1. Sử dụng các thông điệp với kích thước cố định

Cách dễ nhất nhưng cũng là cách tốn chi phí nhất để giải quyết vấn đề với các thông điệp TCP là tạo ra các giao thức luôn luôn truyền các thông điệp với kích thước cố định. Bằng cách thiết lập tất cả các thông điệp có cùng kích thước, chương trình TCP nhận có thể biết toàn bộ thông điệp được gởi từ Client.

Khi gởi dữ liệu với kích thước cố định, chúng ta phải đảm bảo toàn bộ thông điệp được gởi từ phương thức Send(). Phụ thuộc vào kích thước của bộ đệm TCP và bao nhiêu dữ liệu được truyền, phương thức Send() sẽ trả về số byte mà nó thực sự đã gởi đến bộ đệm TCP. Nếu phương thức Send() chưa gởi hết dữ liệu thì chúng ta phải gởi lại phần dữ liệu còn lại. Việc này thường được thực hiện bằng cách sử dụng vòng lặp while() và trong vòng lặp ta kiểm tra số byte thực sự đã gởi với kích thước cố định.

```
private static int SendData(Socket s, byte[] data)
{
    int total = 0;
    int size = data.Length;
    int dataleft = size;
    int sent;
    while (total < size)
    {
        sent = s.Send(data, total, dataleft, SocketFlags.None);
        total += sent;
        dataleft -= sent;
    }
    return total;
}
```

Cũng giống như việc gởi dữ liệu, chúng ta phải luôn luôn đảm bảo nhận tất cả dữ liệu trong phương thức Receive(). Bằng cách dùng vòng lặp gọi phương thức Receive() chúng ta có thể nhận được toàn bộ dữ liệu mong muốn.

```
private static byte[] ReceiveData(Socket s, int size)
{
    int total = 0;
    int dataleft = size;
    byte[] data = new byte[size];
    int recv;
    while (total < size)
    {
        recv = s.Receive(data, total, dataleft, 0);
        if (recv == 0)
        {
            data = Encoding.ASCII.GetBytes("exit");
            break;
        }
        total += recv;
        dataleft -= recv;
    }
    return data;
```

Phương thức ReceiveData() sẽ đọc dữ liệu với kích thước được đọc là đối số được truyền vào, nếu phương thức Receive() sẽ trả về số byte thực sụ đọc được, nếu số byte thực sự đọc được mà còn nhỏ hơn số byte truyền vào phương thức ReceiveData() thì vòng lặp sẽ tiếp tục cho đến khi số byte đọc được đúng bằng kích thước yêu cầu.

Chương trình Server gởi và nhận dữ liệu với kích thước cố định

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
class FixedTcpSrvr
{
    private static int SendData(Socket s, byte[] data)
    {
        int total = 0;
        int size = data.Length;
```

```
        int dataleft = size;
        int sent;
        while (total < size)
        {
            sent = s.Send(data, total, dataleft, SocketFlags.None);
            total += sent;
            dataleft -= sent;
        }
        return total;
    }
    private static byte[] ReceiveData(Socket s, int size)
    {
        int total = 0;
        int dataleft = size;
        byte[] data = new byte[size];
        int recv;
        while (total < size)
        {
            recv = s.Receive(data, total, dataleft, 0);
            if (recv == 0)
            {
                data = Encoding.ASCII.GetBytes("exit");
                break;
            }
            total += recv;
            dataleft -= recv;
        }
        return data;
    }
    public static void Main()
    {
        byte[] data = new byte[1024];
        IPEndPoint ipep = new IPEndPoint(IPAddress.Any, 5000);
        Socket newsock = new Socket(AddressFamily.InterNetwork,
                SocketType.Stream, ProtocolType.Tcp);
        newsock.Bind(ipep);
        newsock.Listen(10);
        Console.WriteLine("Dang cho Client ket noi den...");
        Socket client = newsock.Accept();
        IPEndPoint newclient = (IPEndPoint)client.RemoteEndPoint;
        Console.WriteLine("Da ket noi voi Client {0} tai port {1}",
                newclient.Address, newclient.Port);
        string welcome = "Hello Client";
```

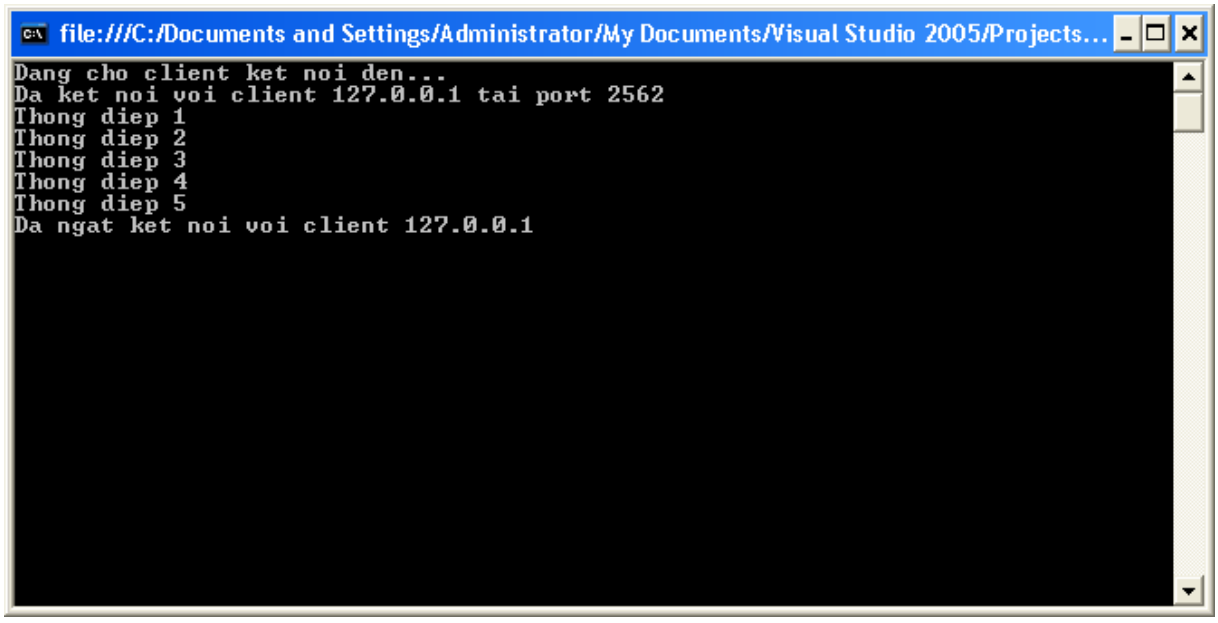
```
        data = Encoding.ASCII.GetBytes(welcome);
        int sent = SendData(client, data);
        for (int i = 0; i < 5; i++)
        {
            data = ReceiveData(client, 12);
            Console.WriteLine(Encoding.ASCII.GetString(data));
        }
        Console.WriteLine("Da ngat ket noi voi Client {0}", newclient.Address);
        client.Close();
        newsock.Close();
    }
}
```

Chương trình Client gởi và nhận dữ liệu với kích thước cố định

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
class FixedTcpClient
{
    private static int SendData(Socket s, byte[] data)
    {
        int total = 0;
        int size = data.Length;
        int dataleft = size;
        int sent;
        while (total < size)
        {
            sent = s.Send(data, total, dataleft, SocketFlags.None);
            total += sent;
            dataleft -= sent;
        }
        return total;
    }
    private static byte[] ReceiveData(Socket s, int size)
    {
        int total = 0;
        int dataleft = size;
        byte[] data = new byte[size];
        int recv;
        while (total < size)
        {
```

```
            recv = s.Receive(data, total, dataleft, 0);
            if (recv == 0)
            {
                data = Encoding.ASCII.GetBytes("exit ");
                break;
            }
            total += recv;
            dataleft -= recv;
        }
        return data;
    }
    public static void Main()
    {
        byte[] data = new byte[1024];
        int sent;
        IPEndPoint ipep = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 5000);
        Socket server = new Socket(AddressFamily.InterNetwork,
                SocketType.Stream, ProtocolType.Tcp);
        try
        {
            server.Connect(ipep);
        }
        catch (SocketException e)
        {
            Console.WriteLine("Khong the ket noi den server");
            Console.WriteLine(e.ToString());
            return;
        }
        int recv = server.Receive(data);
        string stringData = Encoding.ASCII.GetString(data, 0, recv);
        Console.WriteLine(stringData);
        sent = SendData(server, Encoding.ASCII.GetBytes("Thong diep 1"));
        sent = SendData(server, Encoding.ASCII.GetBytes("Thong diep 2"));
        sent = SendData(server, Encoding.ASCII.GetBytes("Thong diep 3"));
        sent = SendData(server, Encoding.ASCII.GetBytes("Thong diep 4"));
        sent = SendData(server, Encoding.ASCII.GetBytes("Thong diep 5"));
        Console.WriteLine("Dong ket noi voi server...");
        server.Shutdown(SocketShutdown.Both);
        server.Close();
    }
}
```

Kết quả trên Server



Hình II.7: Kết quả gởi và nhận dữ liệu với kích thước cố định

II.4.6.2. Gởi kèm kích thước thông điệp cùng với thông điệp

Cách giải quyết vấn đề biên thông điệp của TCP bằng cách sử dụng các thông điệp với kích thước cố định là một giải pháp lãng phí bởi vì tất cả các thông điệp đều phải cùng kích thước. Nếu các thông điệp nào chưa đủ kích thước thì phải thêm phần đệm vào, gây lãng phí băng thông mạng.

Một giải pháp cho vấn đề cho phép các thông điệp được gởi với các kích thước khác nhau là gởi kích thước thông điệp kèm với thông điệp. Bằng cách này thiết bị nhận sẽ biết được kích thước của mỗi thông điệp.

Để thực hiện việc này ta sửa đổi phương thức SendData() trong ví dụ trước

```
private static int SendVarData(Socket s, byte[] buff)
{
   int total = 0;
   int size = buff.Length;
   int dataleft = size;
   int sent;
   byte[] datasize = new byte[4];
   datasize = BitConverter.GetBytes(size);
   sent = s.Send(datasize);
   while (total < size)
   {
       sent = s.Send(buff, total, dataleft, SocketFlags.None);
```

```
            total += sent;
            dataleft -= sent;
        }
        return total;
    }
```

Trong phương thức SendVarData(), ta sẽ lấy kích thước của thông điệp và gắn nó vào đầu của thông điệp, kích thước này là một số interger 4 byte. Kích thước tối đa của mỗi thông điệp này là 65KB. Giá trị interger 4 byte này đầu tiên được chuyển thành mảng các byte, hàm GetBytes() của lớp BitConverter được dùng để thực hiện việc này. Mảng kích thước sau đó được gởi đến thiết bị ở xa, sau khi gởi kích thước thông điệp xong, phần chính của thông điệp được gởi đi, kỹ thuật gởi cũng giống như trong ví dụ trước, chúng ta sẽ lặp cho đến khi tất cả các byte đã được gởi.

Bước tiếp theo là tạo ra một phương thức có thể nhận 4 byte kích thước thông điệp và toàn bộ thông điệp. phương thức ReceiveData() trong ví dụ trước được sửa đổi để thực hiện việc này.

```
    private static byte[] ReceiveVarData(Socket s)
    {
      int total = 0;
      int recv;
      byte[] datasize = new byte[4];
      recv = s.Receive(datasize, 0, 4, 0);
      int size = BitConverter.ToInt32(datasize, 0);
      int dataleft = size;
      byte[] data = new byte[size];
      while (total < size)
      {
          recv = s.Receive(data, total, dataleft, 0);
          if (recv == 0)
          {
              data = Encoding.ASCII.GetBytes("exit ");
              break;
          }
          total += recv;
          dataleft -= recv;
      }
```

```
        return data;
    }
```

Hàm ReceiveVarData() nhận 4 byte đầu tiên của thông điệp và chuyển nó thành giá trị interger bằng phương thức GetInt32() của lớp BitConverter.

Chương trình Server gởi và nhận thông điệp cùng với kích thước

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
class VarTcpSrvr
{
    private static int SendVarData(Socket s, byte[] data)
    {
        int total = 0;
        int size = data.Length;
        int dataleft = size;
        int sent;
        byte[] datasize = new byte[4];
        datasize = BitConverter.GetBytes(size);
        sent = s.Send(datasize);
        while (total < size)
        {
            sent = s.Send(data, total, dataleft, SocketFlags.None);
            total += sent;
            dataleft -= sent;
        }
        return total;
    }
    private static byte[] ReceiveVarData(Socket s)
    {
        int total = 0;
        int recv;
        byte[] datasize = new byte[4];
        recv = s.Receive(datasize, 0, 4, 0);
        int size = BitConverter.ToInt32(datasize, 0);
        int dataleft = size;
        byte[] data = new byte[size];
        while (total < size)
        {
            recv = s.Receive(data, total, dataleft, 0);
            if (recv == 0)
            {
```

```
                data = Encoding.ASCII.GetBytes("exit ");
                break;
            }
            total += recv;
            dataleft -= recv;
        }
        return data;
    }
    public static void Main()
    {
        byte[] data = new byte[1024];
        IPEndPoint ipep = new IPEndPoint(IPAddress.Any, 5000);
        Socket newsock = new Socket(AddressFamily.InterNetwork,
                SocketType.Stream, ProtocolType.Tcp);
        newsock.Bind(ipep);
        newsock.Listen(10);
        Console.WriteLine("Dang cho Client ket noi den...");
        Socket client = newsock.Accept();
        IPEndPoint newclient = (IPEndPoint)client.RemoteEndPoint;
        Console.WriteLine("Da ket noi voi client {0} tai port {1}",
                newclient.Address, newclient.Port);
        string welcome = "Hello client";
        data = Encoding.ASCII.GetBytes(welcome);
        int sent = SendVarData(client, data);
        for (int i = 0; i < 5; i++)
        {
            data = ReceiveVarData(client);
            Console.WriteLine(Encoding.ASCII.GetString(data));
        }
        Console.WriteLine("Dong ket noi voi Client {0}", newclient.Address);
        client.Close();
        newsock.Close();
    }
}
```

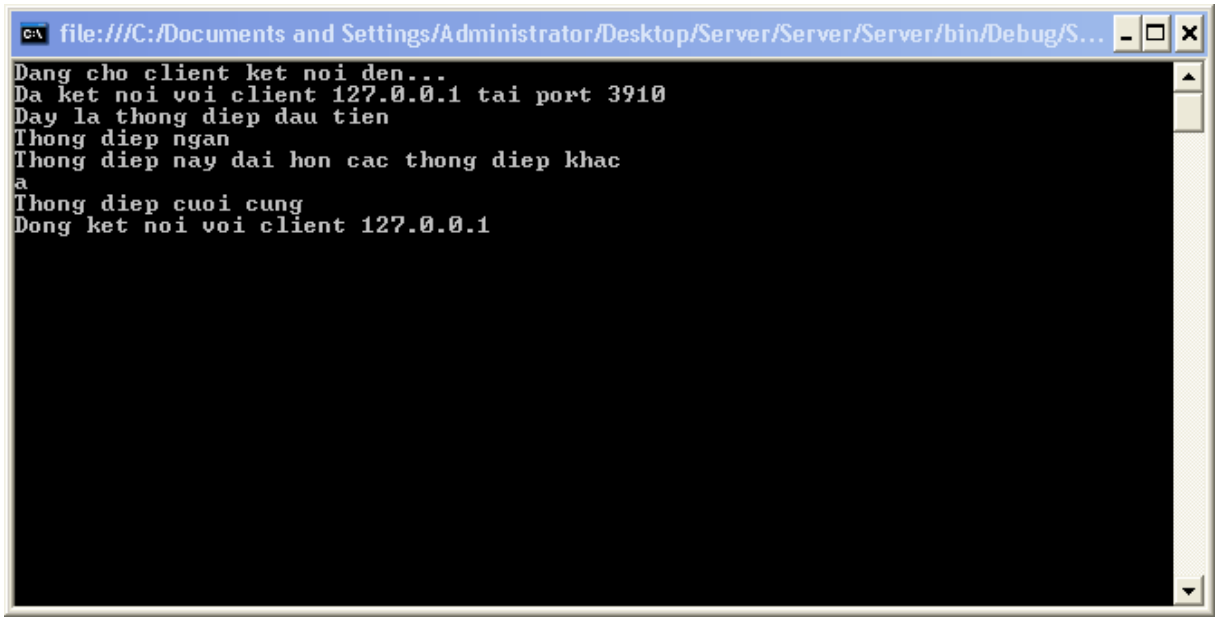
Chương trình Client gởi và nhận thông điệp cùng với kích thước

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
class VarTcpClient
{
```

```
private static int SendVarData(Socket s, byte[] data)
{
    int total = 0;
    int size = data.Length;
    int dataleft = size;
    int sent;
    byte[] datasize = new byte[4];
    datasize = BitConverter.GetBytes(size);
    sent = s.Send(datasize);
    while (total < size)
    {
        sent = s.Send(data, total, dataleft, SocketFlags.None);
        total += sent;
        dataleft -= sent;
    }
    return total;
}
private static byte[] ReceiveVarData(Socket s)
{
    int total = 0;
    int recv;
    byte[] datasize = new byte[4];
    recv = s.Receive(datasize, 0, 4, 0);
    int size = BitConverter.ToInt32(datasize, 0);
    int dataleft = size;
    byte[] data = new byte[size];
    while (total < size)
    {
        recv = s.Receive(data, total, dataleft, 0);
        if (recv == 0)
        {
            data = Encoding.ASCII.GetBytes("exit ");
            break;
        }
        total += recv;
        dataleft -= recv;
    }
    return data;
}
public static void Main()
{
    byte[] data = new byte[1024];
    int sent;
```

```
        IPEndPoint ipep = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 5000);
        Socket server = new Socket(AddressFamily.InterNetwork,
                SocketType.Stream, ProtocolType.Tcp);
        try
        {
            server.Connect(ipep);
        }
        catch (SocketException e)
        {
            Console.WriteLine("Khong the ket noi voi server");
            Console.WriteLine(e.ToString());
            return;
        }
        data = ReceiveVarData(server);
        string stringData = Encoding.ASCII.GetString(data);
        Console.WriteLine(stringData);
        string message1 = "Day la thong diep dau tien";
        string message2 = "Thong diep ngan";
        string message3 = "Thong diep nay dai hon cac thong diep khac";
        string message4 = "a";
        string message5 = "Thong diep cuoi cung";
        sent = SendVarData(server, Encoding.ASCII.GetBytes(message1));
        sent = SendVarData(server, Encoding.ASCII.GetBytes(message2));
        sent = SendVarData(server, Encoding.ASCII.GetBytes(message3));
        sent = SendVarData(server, Encoding.ASCII.GetBytes(message4));
        sent = SendVarData(server, Encoding.ASCII.GetBytes(message5));
        Console.WriteLine("Dang ngat ket noi voi server...");
        server.Shutdown(SocketShutdown.Both);
        server.Close();
    }
}
```

Kết quả



Hình II.8: Kết quả gởi và thông điệp cùng với kích thước

II.4.6.3. Sử dụng các hệ thống đánh dấu để phân biệt các thông điệp

Một cách khác để gởi các thông điệp với kích thước khác nhau là sử dụng các hệ thống đánh dấu. Hệ thống này sẽ chia các thông điệp bởi các ký tự phân cách để báo hiệu kết thúc thông điệp. Khi dữ liệu được nhận từ Socket, dữ liệu được kiểm tra từng ký tự một để phát hiện các ký tự phân cách, khi các ký tự phân cách được phát hiện thì dữ liệu trước ký tự phân cách chính là một thông điệp và dữ liệu sau ký tự phân cách sẽ bắt đầu một thông điệp mới.

Phương pháp này có một số hạn chế, nếu thông điệp lớn nó sẽ làm giảm tốc độ của hệ thống vì toàn bộ các ký tự của thông điệp đều phải được kiểm tra. Cũng có trường hợp một số ký tự trong thông điệp trùng với các ký tự phân cách và thông điệp này sẽ bị tách ra thành các thông điệp con, điều này làm cho chương trình chạy bị sai lệch.

II.4.7. Sử dụng C# Stream với TCP

Điều khiển thông điệp dùng giao thức TCP thường gây ra khó khăn cho các lập trình viên nên .NET Framework cung cấp một số lớp để giảm gánh nặng lập trình. Một trong những lớp đó là NetworkStream, và hai lớp dùng để gởi và nhận text sử dụng giao thức TCP là StreamWriter và StreamReader

II.4.7.1. Lớp NetworkStream

Lớp NetworkStream nằm trong namespace System.Net.Socket, lớp này có nhiều phương thức tạo lập để tạo một thể hiện của lớp NetworkStream nhưng phương thức tạo lập sau hay được dùng nhất:

```
Socket server = new Socket(AddressFamily.InterNetwork,
                SocketType.Stream, ProtocolType.Tcp);
NetworkStream ns = new NetworkStream(server);
```

Một số thuộc tính của lớp NetworkStream:

Thuộc Tính	**Mô Tả**
CanRead	true nếu NetworkStream hỗ trợ đọc
CanSeek	Luôn luôn false
CanWrite	true nếu NetworkStream hỗ trợ ghi
DataAvailable	true nếu có dữ liệu để đọc

Một số phương thức của lớp NetworkStream:

Phương Thức	**Mô Tả**
BeginRead()	Bắt đầu đọc NetworkStream bất đồng bộ
BeginWrite()	Bắt đầu ghi NetworkStream bất đồng bộ
Close()	Đóng đối tượng NetworkStream
CreateObjRef()	Tạo ra một đối tượng dùng như là proxy cho NetworkStream
EndRead()	Kết thúc đọc NetworkStream bất đồng bộ
EndWrite()	Kết thúc ghi NetworkStream bất đồng bộ
Equals()	So sánh hai đối tượng NetworkStreams
Flush()	Đẩy tất cả dữ liệu từ NetworkStream đi
GetHashCode()	Lấy hash code cho NetworkStream
GetLifetimeService()	Lấy đối tượng lifetime service cho NetworkStream
GetType()	Lấy kiểu NetworkStream
InitializeLifetimeService()	Lấy đối tượng lifetime service object để điều khiển chính sách lifetime choNetworkStream
Read()	Đọc dữ liệu từ NetworkStream
ReadByte()	Đọc một byte dữ liệu từ NetworkStream
ToString()	Trả về chuỗi mô tả NetworkStream
Write()	Ghi dữ liệu từ NetworkStream

Phương thức Read() được dùng để đọc các khối dữ liệu từ NetworkStream. Định dạng của phương thức này:

int Read(byte[] buffer, int offset, int size)

Trong đó:

- ✓ buffer: mảng các byte được đọc vào
- ✓ offset: vị trí bắt đầu để đọc vào trong bộ đệm
- ✓ size: số byte tối đa đọc được

Phương thức này trả về một giá trị interger mô tả số byte thực sự đọc được từ NetworkStream và dặt dữ liệu đọc được vào buffer.

Phương thức Write() dùng để gởi các khối dữ liệu đi cũng có định dạng tương tự:

void Write(byte[] buffer, int offset, int size)

Trong đó:

- ✓ buffer: mảng các byte để ghi
- ✓ offset: vị trí bắt đầu để ghi trong bộ đệm
- ✓ size: số byte tối đa được ghi bắt đầu tại vị trí offset

Chương trình TCP Client NetworkStream

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
class NetworkStreamTcpClient
{
    public static void Main()
    {
        byte[] data = new byte[1024];
        string input, stringData;
        int recv;
        IPEndPoint ipep = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 500);
        Socket server = new Socket(AddressFamily.InterNetwork,
                SocketType.Stream, ProtocolType.Tcp);
        try
        {
            server.Connect(ipep);
        }
        catch (SocketException e)
        {
```

```
            Console.WriteLine("Khong the ket noi den server");
            Console.WriteLine(e.ToString());
            return;
        }
        NetworkStream ns = new NetworkStream(server);
        if (ns.CanRead)
        {
            recv = ns.Read(data, 0, data.Length);
            stringData = Encoding.ASCII.GetString(data, 0, recv);
            Console.WriteLine(stringData);
        }
        else
        {
            Console.WriteLine("Error: Can't read from this Socket");
            ns.Close();
            server.Close();
            return;
        }
        while (true)
        {
            input = Console.ReadLine();
            if (input == "exit")
                break;
            if (ns.CanWrite)
            {
                ns.Write(Encoding.ASCII.GetBytes(input), 0, input.Length);
                ns.Flush();
            }
            recv = ns.Read(data, 0, data.Length);
            stringData = Encoding.ASCII.GetString(data, 0, recv);
            Console.WriteLine(stringData);
        }
        Console.WriteLine("Dang ngat ket noi voi server...");
        ns.Close();
        server.Shutdown(SocketShutdown.Both);
        server.Close();
    }
}
```

Chương trình này tạo ra một đối tượng NetworkStream từ đối tượng Socket:

```
NetworkStream ns = new NetworkStream(server);
```

Khi đối tượng NetworkStream được tạo ra, đối tượng Socket sẽ không được tham chiếu đến nữa cho đến khi nó bị đóng lại vào cuối chương trình, tất cả các thông tin liên lạc với Server ở xa được thực hiện thông đối tượng NetworkStream:

```
recv = ns.Read(data, 0, data.Length);
ns.Write(Encoding.ASCII.GetBytes(input), 0, input.Length);
ns.Flush();
```

Phương thức Flush() được dùng sau mỗi phương thức Write() để đảm bảo dữ liệu đặt vào NetworkStream sẽ lập tức được gởi đến hệ thống ở xa. Mặc dù đối tượng NetworkStream có thêm một số chức năng của Socket nhưng vẫn còn tồn tại vấn đề với biên thông điệp. Vấn đề này được giải quyết thông qua hai lớp hỗ trợ là StreamReader và StreamWriter.

Ta có thể kiểm tra chương trình này với chương trình TCP Server đơn giản ở trên.

II.4.7.2. Lớp StreamReader và StreamWriter

Namespcace System.IO chứa hai lớp StreamReader và StreamWriter điều khiển việc đọc và ghi các thông điệp text từ mạng. Cả hai lớp đều có thể được triển khai với một đối tượng NetworkStream để xác định các hệ thống đánh dấu cho các thông điệp TCP.

Lớp StreamReader có nhiều phương thức tạo lập, trong đó phương thức tạo lập đơn giản nhất của lớp StreamReader:

```
public StreamReader(Stream stream);
```

Biến stream có thể được tham chiếu đến bất kỳ kiểu đối tượng Stream nào kể cả đối tượng NetworkStream. Có nhiều phương thức và thuộc tính có thể được dùng với đối tượng StreamReader sau khi nó được tạo ra như trong bảng sau:

Phương Thức	**Mô Tả**
Close()	Đóng đối tượng StreamReader
CreateObjRef()	Tạo ra một đối tượng được dùng như là một proxy cho StreamReader
DiscardBufferedData()	Bỏ dữ liệu hiện tại ở StreamReader
Equals()	So sánh hai đối tượng StreamReader
GetHashCode()	Lấy hash code cho đối tượng StreamReader
GetLifetimeService()	Lấy đối tượng lifetime service object cho StreamReader

Phương Thức	**Mô Tả**
GetType()	Lấy kiểu của đối tượng StreamReader
InitializeLifetimeService()	Tạo ra một đối tượng lifetime service cho StreamReader
Peek()	Trả về byte dữ liệu hợp lệ tiếp theo từ mà không gỡ bỏ nó khỏi stream
Read()	Đọc một hoặc nhiều byte dữ liệu từ StreamReader
ReadBlock()	Đọc một nhóm các byte từ stream StreamReader và đặt nó vào một bộ đệm
ReadLine()	Đọc dữ liệu từ bắt đầu đối tượng StreamReader trở lên cho đến khi gặp ký tự xuống dòng đầu tiên
ReadToEnd()	Đọc dữ liệu cho đến khi hết stream
ToString()	Tạo ra một chuỗi mô tả đối tượng StreamReader

Tương tự đối tượng StreamReader, đối tượng StreamWriter có thể được tạo ra từ một đối tượng NetworkStream:

```
public StreamWriter(Stream stream);
```

StreamWriter cũng có nhiều phương thức và thuộc tính kết hợp với nó, một số phương thức và thuộc tính của lớp StreamReader cũng có trong đối tượng StreamWriter, ngoài ra nó còn có một số phương thức và thuộc tính riêng:

Phương Thức	**Mô Tả**
Flush()	Gởi tất cả dữ liệu trong bộ đệm StreamWriter ra stream
Write()	Gởi một hoặc nhiều byte dữ liệu ra stream
WriteLine()	Gởi dữ liệu cùng với ký tự xuống dòng ra stream

Phương thức ReadLine() là phương thức hay nhất của lớp StreamReader. Nó đọc các ký tự từ stream cho tới khi nó gặp ký tự xuống dòng. Tính năng này cho phép sử dụng ký tự xuống dòng như là một ký tự phân tách các thông điệp. Phương thức WriteLine() của lớp StreamWriter sẽ so khớp với phương thức ReadLine của lớp StreamReader do đó việc xử lý các thông điệp TCP trở nên dễ dàng hơn.

Chương trình Stream TCP Server

```
using System;
using System.IO;
using System.Net;
using System.Net.Sockets;
using System.Text;
```

```
class StreamTcpSrvr
{
    public static void Main()
    {
        string data;
        IPEndPoint ipep = new IPEndPoint(IPAddress.Any, 5000);
        Socket newsock = new Socket(AddressFamily.InterNetwork,
                SocketType.Stream, ProtocolType.Tcp);
        newsock.Bind(ipep);
        newsock.Listen(10);
        Console.WriteLine("Dang cho Client ket noi toi...");
        Socket client = newsock.Accept();
        IPEndPoint newclient = (IPEndPoint)client.RemoteEndPoint;
        Console.WriteLine("Da ket noi voi Client {0} tai port {1}",
                newclient.Address, newclient.Port);
        NetworkStream ns = new NetworkStream(client);
        StreamReader sr = new StreamReader(ns);
        StreamWriter sw = new StreamWriter(ns);
        string welcome = "Hello Client";
        sw.WriteLine(welcome);
        sw.Flush();
        while (true)
        {
            try
            {
                data = sr.ReadLine();
            }
            catch (IOException)
            {
                break;
            }

            Console.WriteLine(data);
            sw.WriteLine(data);
            sw.Flush();
        }
        Console.WriteLine("Da dong ket noi voi Client {0}", newclient.Address);
        sw.Close();
        sr.Close();
        ns.Close();
    }
}
```

Chương trình StreamTcpSrvr dùng phương thức WriteLine() của lớp StreamWriter để gởi các thông điệp text và kết thúc bằng ký tự xuống dòng. Đối với đối tượng NetworkStream, tốt hơn hết là ta phương thức Flush() sau khi gọi phương thức WriteLine() để đảm bảo rằng tất cả dữ liệu được gởi từ bộ đệm TCP.

Điểm khác biệt của chương trình này với chương trình TCP Server đơn giản ở trên là cách chương trình StreamTcpSrvr biết khi nào Client ngắt kết nối. Bởi vì phương thức ReadLine() hoạt động trên stream chứ không phải là Socket nên nó không thể trả về giá trị 0 khi Client ngắt kết nối. Thay vì vậy, phương thức ReadLine() sẽ phát sinh ra một biệt lệ nếu Client ngắt kết nối và ta phải dùng catch để bắt biệt lệ này và xử lý khi Client ngắt kết nối:

```
try
{
      data = sr.ReadLine();
}
catch (IOException)
{
      break;
}
```

Chương trình Stream TCP Client

```
using System;
using System.IO;
using System.Net;
using System.Net.Sockets;
using System.Text;
class StreamTcpClient
{
    public static void Main()
    {
        string data;
        string input;
        IPEndPoint ipep = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 5000);
        Socket server = new Socket(AddressFamily.InterNetwork,
                SocketType.Stream, ProtocolType.Tcp);
        try
        {
            server.Connect(ipep);
        }
        catch (SocketException e)
```

```
        {
            Console.WriteLine("Khong the ket noi den server");
            Console.WriteLine(e.ToString());
            return;
        }
        NetworkStream ns = new NetworkStream(server);
        StreamReader sr = new StreamReader(ns);
        StreamWriter sw = new StreamWriter(ns);
        data = sr.ReadLine();
        Console.WriteLine(data);
        while (true)
        {
            input = Console.ReadLine();
            if (input == "exit")
                break;
            sw.WriteLine(input);
            sw.Flush();
            data = sr.ReadLine();
            Console.WriteLine(data);
        }
        Console.WriteLine("Dang dong ket noi voi server...");
        sr.Close();
        sw.Close();
        ns.Close();
        server.Shutdown(SocketShutdown.Both);
        server.Close();
    }
}
```

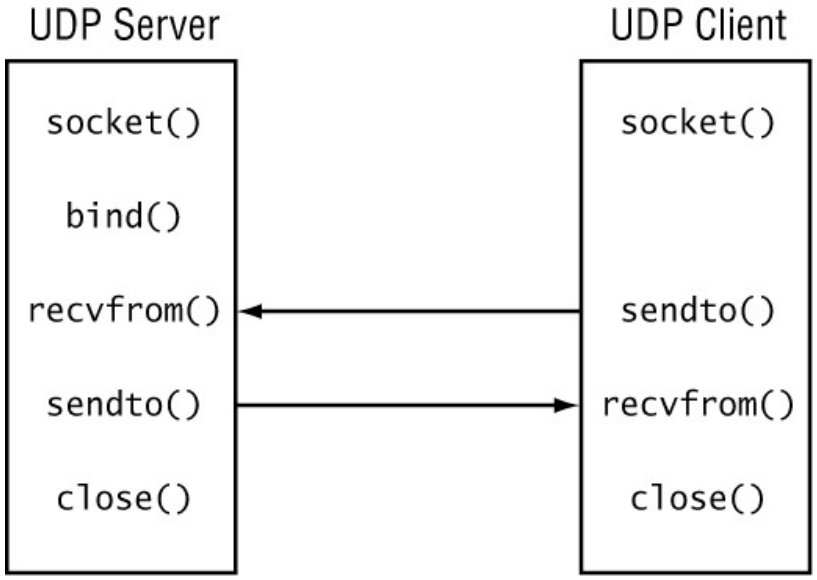
CHƯƠNG III: LẬP TRÌNH SOCKET PHI KẾT NỐI

III.1. Tổng quan

Các Socket phi kết nối cho phép gởi các thông điệp mà không cần phải thiết lập kết nối trước. Một phương thức đọc sẽ đọc toàn bộ thông điệp được gởi bởi một phương thức gởi, điều này làm tránh được các rắc rối, phức tạp với biên dữ liệu. Thật không may mắn là giao thức phi kết nối UDP không đảm bảo dữ liệu được truyền tới đích. Nhiều yếu tố như mạng bận, mạng bị đứt nữa chừng có thể ngăn cản các gói tin được truyền tới đích.

Nếu một thiết bị chờ dữ liệu từ một thiết bị ở xa, nó phải được gán một địa chỉ và port cục bộ, dùng hàm Bind() để gán. Một khi đã thực hiện xong, thiết bị có thể dùng Socket để gởi dữ liệu ra ngoài hay nhận dữ liệu từ Socket.

Bởi vì thiết bị Client không tạo ra kết nối đến một địa chỉ Server cụ thể do đó phương thức Connect() không cần dùng trong chương trình UDP Client. Mô hình bên dưới mô tả các bước lập trình Socket phi kết nối:



Hình V.1: Mô hình lập trình Socket phi kết nối

Khi kết nối không được thành lập thì phương thức Send() và Receive() không được dùng bởi vì trong hai phương thức trên đều không chỉ ra địa chỉ đích của dữ liệu.

Thay vào đó, Socket phi kết nối cung cấp hai phương thức để thực hiện việc này là SendTo() và ReceiveFrom()

III.2. Lập trình phía Server

UDP là một giao thức phi kết nối do đó các lập trình viên chỉ phải làm hai việc để tạo ra một ứng dụng Server gởi và nhận dữ liệu:

- ✓ Tạo ra Socket
- ✓ Kết nối Socket đến một IPEndPoint cục bộ

```
IPEndPoint ipep = new IPEndPoint(IPAddress.Any, 5000);
Socket newsock = new Socket(AddressFamily.InterNetwork,
                SocketType.Dgram, ProtocolType.Udp);
newsock.Bind(ipep);
```

Để thực hiện truyền thông phi kết nối, chúng ta phải chỉ ra SocketType là Dgram và ProtocolType là Udp.

Sau khi thực hiện xong hai bước trên, Socket có thể được dùng hoặc để chấp nhận các gói tin UDP đến trên IPEndPoint hoặc gởi các gói tin udp đến các thiết bị nhận khác trên mạng.

Phương thức SendTo() dùng để gởi dữ liệu, phương thức này chỉ ra dữ liệu để gởi và IPEndPoint của thiết bị nhận. Có nhiều quá tải hàm của phương thức SendTo() có thể được dùng tùy vào yêu cầu cụ thể.

```
SendTo(byte[] data, EndPoint Remote)
```

Phương thức trên gởi một mảng dữ liệu đến một EndPoint được chỉ ra bởi Remote. Một quá tải hàm khác phức tạp hơn của phương thức SendTo()

```
SendTo(byte[] data, SocketFlags Flags, EndPoint Remote)
```

Phương thức này cho phép thêm cờ SocketFlag, nó chỉ ra các tùy chọn UDP được sử dụng. Để chỉ ra số byte được gởi từ mảng byte ta sử dụng quá tải hàm sau của phương thức SendTo():

```
SendTo(byte[] data, int Offset, int Size, SocketFlags Flags,
EndPoint Remote)
```

Phương thức ReceiveFrom() có dùng định dạng với phương thức SendTo(), chỉ có một điểm khác biệt sau ở cách EndPoint được khai báo. Phương thức ReceiveFrom() đơn giản được định nghĩa như sau:

```
ReceiveFrom(byte[] data, ref EndPoint Remote)
```

Cũng như thông thường, tham số thứ nhất là một mảng byte được định nghĩa để nhận dữ liệu, tham số thứ hai ra phải truyền tham chiếu của đối tượng EndPoint. Tham chiếu này tham chiếu đến vị trí bộ nhớ nơi biến được lưu trữ. Phương thức ReceiveFrom() sẽ đặt thông tin EndPoint từ thiết bị ở xa vào vùng bộ nhớ của đối tượng EndPoint tham chiếu đến. Bằng việc sử dụng đối số thứ hai là tham chiếu ta sẽ lấy được địa chỉ IP và port của máy ở xa.

Chương trình UDP đơn giản

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
class SimpleUdpSrvr
{
    public static void Main()
    {
        int recv;
        byte[] data = new byte[1024];
        IPEndPoint ipep = new IPEndPoint(IPAddress.Any, 5000);
        Socket newsock = new Socket(AddressFamily.InterNetwork,
                SocketType.Dgram, ProtocolType.Udp);
        newsock.Bind(ipep);
        Console.WriteLine("Dang cho Client ket noi den...");
        IPEndPoint sender = new IPEndPoint(IPAddress.Any, 0);
        EndPoint Remote = (EndPoint)(sender);
        recv = newsock.ReceiveFrom(data, ref Remote);
        Console.WriteLine("Thong diep duoc nhan tu {0}:", Remote.ToString());
        Console.WriteLine(Encoding.ASCII.GetString(data, 0, recv));
        string welcome = "Hello Client";
        data = Encoding.ASCII.GetBytes(welcome);
        newsock.SendTo(data, data.Length, SocketFlags.None, Remote);
        while (true)
        {
            data = new byte[1024];
            recv = newsock.ReceiveFrom(data, ref Remote);

            Console.WriteLine(Encoding.ASCII.GetString(data, 0, recv));
            newsock.SendTo(data, recv, SocketFlags.None, Remote);
        }
    }
```

```
}
```

Để chương trình UDP chấp nhận các thông điệp UDP đến, nó phải được gắn với một port trên hệ thống. Việc này được thực hiện bằng cách tạo ra một đối tượng IPEndPoint sử dụng một địa chỉ IP cục bộ thích hợp, trong trường hợp này ta chỉ ra IPAddresss.Any để có thể dùng bất kỳ địa chỉ IP nào trên máy cục bộ để lắng nghe.

Sau khi gắn Socket vào một IPEndPoint, Server sẽ chờ Client kết nối đến, khi Client kết nối đến, Client sẽ gởi thông điệp đến Server. Server sau khi nhận được thông điệp từ Client nó sẽ gởi câu chào ngược lại cho Client:

```
recv = newsock.ReceiveFrom(data, ref Remote);
Console.WriteLine("Thong diep duoc nhan tu {0}:",Remote.ToString());
Console.WriteLine(Encoding.ASCII.GetString(data, 0, recv));
string welcome = "Hello client";
```

Khi gởi câu chào cho Client xong, Server sẽ bắt đầu nhận và gởi thông điệp

III.3. Lập trình phía Client

Bởi vì Client không cần chờ trên một port UDP định sắn nên nó cũng chẳng cần dùng phương thức Bind(), thay vì vậy nó sẽ lấy một port ngẫu nhien trên hệ thống khi dữ liệu được gởi và nó giữa port này để nhận dữ liệu trả về. Chương trình UDP Client cũng tương tự chương trình UDP Server:

Chương trình UDP Client đơn giản

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
class SimpleUdpClient
{
    public static void Main()
    {
        byte[] data = new byte[1024];
        string input, stringData;
        IPEndPoint ipep = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 5000);
        Socket server = new Socket(AddressFamily.InterNetwork,
                SocketType.Dgram, ProtocolType.Udp);
        string welcome = "Hello server";
        data = Encoding.ASCII.GetBytes(welcome);
        server.SendTo(data, data.Length, SocketFlags.None, ipep);
```

```
        IPEndPoint sender = new IPEndPoint(IPAddress.Any, 0);
        EndPoint Remote = (EndPoint)sender;
        data = new byte[1024];
        int recv = server.ReceiveFrom(data, ref Remote);
        Console.WriteLine("Thong diep duoc nhan tu {0}:", Remote.ToString());
        Console.WriteLine(Encoding.ASCII.GetString(data, 0, recv));
        while (true)
        {
            input = Console.ReadLine();
            if (input == "exit")
                break;
            server.SendTo(Encoding.ASCII.GetBytes(input), Remote);
            data = new byte[1024];
            recv = server.ReceiveFrom(data, ref Remote);
            stringData = Encoding.ASCII.GetString(data, 0, recv);
            Console.WriteLine(stringData);
        }
        Console.WriteLine("Dang dong client");
        server.Close();
    }
}
```

Chương trình UDP Client đầu tiên định nghĩa một IPEndPoint mà UDP Server sẽ gởi các gói tin:

```
IPEndPoint ipep = new IPEndPoint(IPAddress.Parse("127.0.0.1"),
5000);
```

Chương trình Client gởi thông điệp đến Server và chờ câu chào trả về từ Server. Bởi vì Client không cần chấp nhận các thông điệp UDP trên một port định trước nên Client không dùng phương thức Bind(). Nó sẽ nhận các thông điệp UDP trên cùng port mà nó đã gởi.

Chương trình SimpleUdpClient đọc dữ liệu nhập vào từ bàn phím rồi gởi đến và chờ dữ liệu từ Server gởi trả về. Khi Server gởi trả dữ liệu về, Client sẽ lấy thông điệp đó ra và hiển thị lên màn hình. Nếu người dùng nhận vào "exit" thì vòng lặp sẽ thoát và kết nối bị đóng lại.

Không giống như chương trình TCP Server, chương trình UDP Server sẽ không biết khi nào Client ngắt kết nối do đó khi Client ngắt kết nối thì nó phải gởi thông điệp ngắt kết nối cho Server biết.

III.3.1. Sử dụng phương thức Connect() trong chương trình UDP Client

Các phương thức UDP được thiết kế để cho phép các lập trình viên gởi các gói tin đến bất kỳ máy nào trên mạng bất cứ lúc nào. Bởi vì giao thức UDP không yêu cầu kết nối trước khi gởi dữ liệu nên phải chỉ ra địa chỉ của máy nhận trong phương thức SendTo() và phương thức ReceiveFrom(). Nếu chương trình của chúng ta chỉ cần gởi và nhận dữ liệu từ một máy, chúng ta có thể dùng phương thức Connect().

Sau khi UDP socket được tạo ra, chúng ta có thể dùng phương thức Connect() giống như trong chương trình TCP để chỉ ra udp Server ở xa. Sau khi dùng phương thức Connect() xong ta có thể dùng phương thức Send() và Receive() để truyền tải dữ liệu giữa các thiết bị với nhau. Kỹ thuật này được minh họa trong chương trình UDP Client sau:

Chương trình udp Client dùng phương thức Connect()

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
class OddUdpClient
{
    public static void Main()
    {
        byte[] data = new byte[1024];
        string input, stringData;
        IPEndPoint ipep = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 9050);
        Socket server = new Socket(AddressFamily.InterNetwork,
                SocketType.Dgram, ProtocolType.Udp);
        server.Connect(ipep);
        string welcome = "Xin chao server";
        data = Encoding.ASCII.GetBytes(welcome);
        server.Send(data);
        data = new byte[1024];
        int recv = server.Receive(data);
        Console.WriteLine("Nhan thong diep tu {0}:", ipep.ToString());
        Console.WriteLine(Encoding.ASCII.GetString(data, 0, recv));
        while (true)
        {
            input = Console.ReadLine();
            if (input == "exit")
                break;
```

```
            server.Send(Encoding.ASCII.GetBytes(input));
            data = new byte[1024];
            recv = server.Receive(data);
            stringData = Encoding.ASCII.GetString(data, 0, recv);
            Console.WriteLine(stringData);
        }
        Console.WriteLine("Dang dong client");
        server.Close();
    }
}
```

III.3.2. Phân biệt các thông điệp UDP

Một trong những tính năng quan trọng của UDP mà TCP không có được đó là khả năng xử lý thông điệp mà không cần quan tâm đến biên thông điệp. UDP bảo vệ biên thông điệp của tất cả các thông điệp được gởi. Mỗi lần gọi phương thức ReceiveFrom() nó chỉ đọc dữ liệu được gởi từ một phương thức SendTo().

Khi UDP Socket được tạo ra, nó có thể nhận thông điệp từ bất kỳ Client nào. Để udp Socket phân biệt được Client nào gởi dữ liệu nó bắt buộc mỗi thông điệp phải được chứa trong một gói tin riêng và được đánh dấu bởi thông tin IP của thiết bị gởi. Điều này cho phép thiết bị nhận phân biệt được các thông điệp và thiết bị gởi.

Chương trình Client và Server sau sẽ minh họa điều này.

Chương trình UDP Server

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
class TestUdpSrvr
{
    public static void Main()
    {
        int recv;
        byte[] data = new byte[1024];
        IPEndPoint ipep = new IPEndPoint(IPAddress.Any, 5000);
        Socket newsock = new Socket(AddressFamily.InterNetwork,
                SocketType.Dgram, ProtocolType.Udp);
        newsock.Bind(ipep);
        Console.WriteLine("Dang cho client ket noi den...");
        IPEndPoint sender = new IPEndPoint(IPAddress.Any, 0);
        EndPoint tmpRemote = (EndPoint)(sender);
```

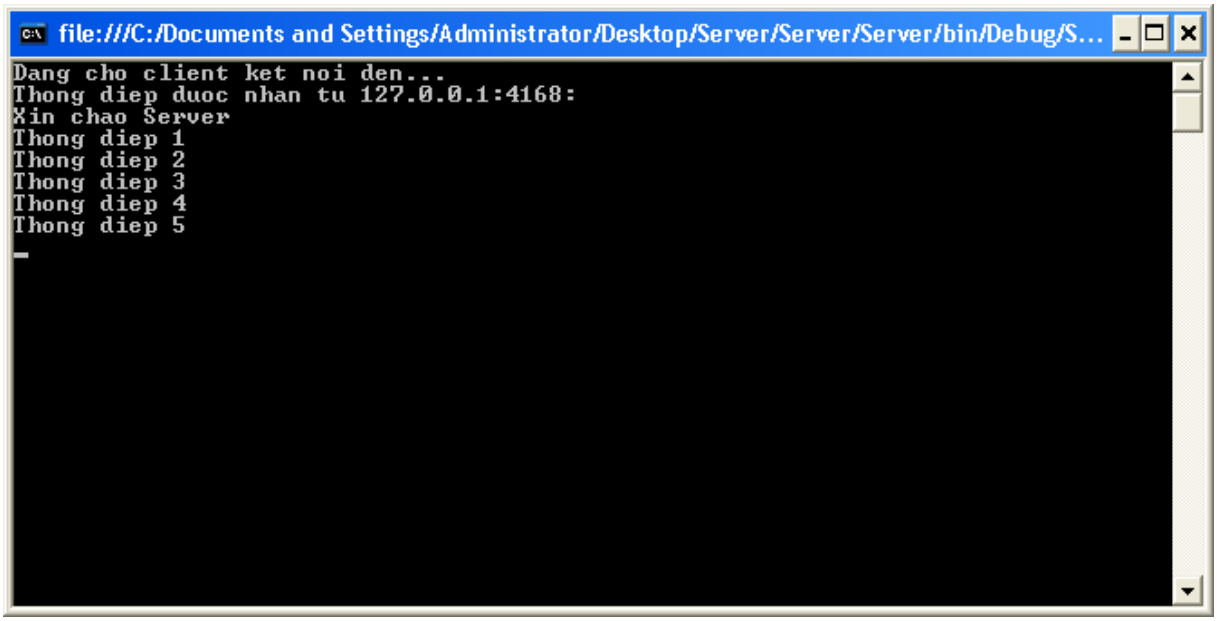
```
        recv = newsock.ReceiveFrom(data, ref tmpRemote);
        Console.WriteLine("Thong diep duoc nhan tu {0}:",
tmpRemote.ToString());
        Console.WriteLine(Encoding.ASCII.GetString(data, 0, recv));
        string welcome = "Xin chao client";
        data = Encoding.ASCII.GetBytes(welcome);
        newsock.SendTo(data, data.Length, SocketFlags.None, tmpRemote);
        for (int i = 0; i < 5; i++)
        {
            data = new byte[1024];
            recv = newsock.ReceiveFrom(data, ref tmpRemote);
            Console.WriteLine(Encoding.ASCII.GetString(data, 0, recv));
        }
        newsock.Close();
    }
}
```

Chương trình UDP Client

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
class TestUdpClient
{
    public static void Main()
    {
        byte[] data = new byte[1024];
        IPEndPoint ipep = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 5000);
        Socket server = new Socket(AddressFamily.InterNetwork,
                SocketType.Dgram, ProtocolType.Udp);
        string welcome = "Xin chao Server";
        data = Encoding.ASCII.GetBytes(welcome);
        server.SendTo(data, data.Length, SocketFlags.None, ipep);
        IPEndPoint sender = new IPEndPoint(IPAddress.Any, 0);
        EndPoint tmpRemote = (EndPoint)sender;
        data = new byte[1024];
        int recv = server.ReceiveFrom(data, ref tmpRemote);
        Console.WriteLine("Thong diep duoc nhan tu {0}:",
tmpRemote.ToString());
        Console.WriteLine(Encoding.ASCII.GetString(data, 0, recv));
        server.SendTo(Encoding.ASCII.GetBytes("Thong diep 1"), tmpRemote);
        server.SendTo(Encoding.ASCII.GetBytes("Thong diep 2"), tmpRemote);
```

```
            server.SendTo(Encoding.ASCII.GetBytes("Thong diep 3"), tmpRemote);
            server.SendTo(Encoding.ASCII.GetBytes("Thong diep 4"), tmpRemote);
            server.SendTo(Encoding.ASCII.GetBytes("Thong diep 5"), tmpRemote);
            Console.WriteLine("Dang dong client");
            server.Close();
        }
}
```

Kết quả ở Server



Hình V.2: UDP Server nhận biết được các thông điệp riêng rẽ

III.4. Ngăn cản mất dữ liệu

Một thuận lợi của việc truyền thông dùng giao thức TCP là giao thức TCP sử dụng bộ đệm TCP. Tất cả dữ liệu được gởi bởi TCP Socket được đặt vào bộ đệm TCP trước khi được gởi ra ngoài mạng. Cũng giống như vậy, tất cả dữ liệu nhận từ Socket được đặt vào bộ đệm TCP trước khi được đọc bởi phương thức Receive(). Khi phương thức Receive() cố gắng đọc dữ liệu từ bộ đệm, nếu nó không đọc hết dữ liệu thì phần còn lại vẫn nằm trong bộ đệm và chờ lần gọi phương thức Receive() kế tiếp.

Vì UDP không quan tâm đến việc gởi lại các gói tin nên nó không dùng bộ đệm. Tất cả dữ liệu được gởi từ Socket đều được lập tức gởi ra ngoài mạng và tất cả dữ liệu được nhận từ mạng lập tức được chuyển cho phương thức ReceiveFrom() trong lần gọi tiếp theo. Khi phương thức ReceiveFrom() được dùng trong chương trình, các lập trình viên phải đảm bảo rằng bộ đệm phải đủ lớn để chấp nhận hết dữ liệu từ UDP

Socket. Nếu bộ đệm quá nhỏ, dữ liệu sẽ bị mất. Để thấy được điều này, ta tiến hành thay đổi kích thước bộ đệm trong chương trình UDP đơn giản trên:

Chương trình BadUDPClient

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
class BadUdpClient
{
    public static void Main()
    {
        byte[] data = new byte[30];
        string input, stringData;
        IPEndPoint ipep = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 5000);
        Socket server = new Socket(AddressFamily.InterNetwork,
                SocketType.Dgram, ProtocolType.Udp);
        string welcome = "Xin chao serveer";
        data = Encoding.ASCII.GetBytes(welcome);
        server.SendTo(data, data.Length, SocketFlags.None, ipep);
        IPEndPoint sender = new IPEndPoint(IPAddress.Any, 0);
        EndPoint tmpRemote = (EndPoint)sender;
        data = new byte[30];
        int recv = server.ReceiveFrom(data, ref tmpRemote);
        Console.WriteLine("Thong diep duoc nhan tu {0}:",
tmpRemote.ToString());
        Console.WriteLine(Encoding.ASCII.GetString(data, 0, recv));
        while (true)
        {
            input = Console.ReadLine();
            if (input == "exit")
                break;
            server.SendTo(Encoding.ASCII.GetBytes(input), tmpRemote);
            data = new byte[30];
            recv = server.ReceiveFrom(data, ref tmpRemote);
            stringData = Encoding.ASCII.GetString(data, 0, recv);
            Console.WriteLine(stringData);
        }
        Console.WriteLine("Dang dong client");
        server.Close();
    }
}
```

Ta có thể test chương trình này với chương trình UDP Server đơn giản ở trên. Khi ta nhận dữ liệu ít hơn 10 byte thì chương trình vẫn chạy bình thường nhưng khi ta nhập dữ liệu lớn hơn 10 byte thì chương trình BadUdpClient sẽ phát sinh ra một biệt lệ. Mặc dầu ta không thể lấy lại dữ liệu đã bị mất nhưng ta có thể hạn chế mất dữ liệu bằng cách đặt phương thức ReceiveFrom() trong khối try-catch, khi dữ liệu bị mất bởi kích thước bộ đệm nhỏ, ta có thể tăng kích thước bộ đệm vào lần kế tiếp nhận dữ liệu. Chương trình BetterUdpClient sau minh họa việc này:

Chương trình BetterUdpClient

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
class BetterdUdpClient
{
    public static void Main()
    {
        byte[] data = new byte[30];
        string input, stringData;
        IPEndPoint ipep = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 5000);
        Socket server = new Socket(AddressFamily.InterNetwork,
                SocketType.Dgram, ProtocolType.Udp);
        string welcome = "Xin chao server";
        data = Encoding.ASCII.GetBytes(welcome);
        server.SendTo(data, data.Length, SocketFlags.None, ipep);
        IPEndPoint sender = new IPEndPoint(IPAddress.Any, 0);
        EndPoint tmpRemote = (EndPoint)sender;
        data = new byte[30];
        int recv = server.ReceiveFrom(data, ref tmpRemote);
        Console.WriteLine("Thong diep duoc nhan tu {0}:",
tmpRemote.ToString());
        Console.WriteLine(Encoding.ASCII.GetString(data, 0, recv));
        int i = 30;
        while (true)
        {
            input = Console.ReadLine();
            if (input == "exit")
                break;
            server.SendTo(Encoding.ASCII.GetBytes(input), tmpRemote);
            data = new byte[i];
            try
```

```
            {
                recv = server.ReceiveFrom(data, ref tmpRemote);
                stringData = Encoding.ASCII.GetString(data, 0, recv);
                Console.WriteLine(stringData);
            }
            catch (SocketException)
            {
                Console.WriteLine("Canh bao: du lieu bi mat, hay thu lai");
                i += 10;
            }
        }
        Console.WriteLine("Dang dong client");
        server.Close();
    }
}
```

Thay vì sử dụng mảng buffer với chiều dài cố định, chương trình BetterUdpClient dùng một biết có thể thiết lập giá trị khác nhau mỗi lần phương thức ReceiveFrom() được dùng.

```
data = new byte[i];
try
{
      recv = server.ReceiveFrom(data, ref tmpRemote);
      stringData = Encoding.ASCII.GetString(data, 0, recv);
      Console.WriteLine(stringData);
}
catch (SocketException)
{
      Console.WriteLine("Canh bao: du lieu bi mat, hay thu lai");
      i += 10;
}
```

III.5. Ngăn cản mất gói tin

Một khó khăn khác khi lập trình với giao thức udp là khả năng bị mất gói tin bởi vì udp là một giao thức phi kết nối nên không có cách nào mà thiết bị gởi biết được gói tin gởi có thực sự đến được đích hay không. Cách đơn giản nhất để ngăn chặn việc mất các gói tin là phải có cơ chế hồi báo giống như giao thức TCP. Các gói tin được gởi thành công đến thiết bị nhận thì thiết bị nhận phải sinh ra gói tin hồi báo

cho thiết bị gởi biết đã nhận thành công. Nếu gói tin hồi báo không được nhận trong một khoảng thời gian nào đó thì thiết bị nhận sẽ cho là gói tin đó đã bị mất và gởi lại gói tin đó.

Có hai kỹ thuật dùng để truyền lại các gói tin UDP:

- ✓ Sử dụng Socket bất đồng bộ và một đối tượng Timer. Kỹ thuật này yêu cầu sử dụng một Socket bất đồng bộ mà nó có thể lắng nghe các gói tin đến không bị block. Sau khi Socket được thiết lập đọc bất đồng bộ, một đối tượng Timer có thể được thiết lập, nếu đối tượng Timer tắt trước khi hành động đọc bất đồng bộ kết thúc thì việc gởi lại dữ liệu diễn ra.
- ✓ Sử dụng Socket đồng bộ và thiết lập giá trị Socket time-out. Để làm được việc này, ta dùng phương thức SetSocketOption().

III.5.1. Sử dụng Soket Time-out

Phương thức ReceiveFrom() là phương thức bị block. Nó sẽ block chương trình lại cho đến khi chương trình nhận dữ liệu. Nếu dữ liệu không bao giờ nhận, chương trình sẽ block mã mãi. Mặc định phương thức ReceiveFrom() sẽ bị block mãi mãi nếu không có dữ liệu được đọc. phương thức SetSocketOption() cung cấp nhiều tùy chọn cho các Socket đã được tạo, một trong những tùy chọn đó là ReceiveTimeout. Nó sẽ thiết lập khoảng thời gian Socket sẽ chờ dữ liệu đến trước khi phát ra tín hiệu time-out.

Định dạng của phương thức SetSocketOption() như sau:

```
SetSocketOption(SocketOptionLevel so, SocketOptionName sn, int
value)
```

SocketOptionLevel chỉ ra kiểu tùy chọn Socket để thực thi. SocketOptionName định nghĩa tùy chọn được thiết lập, và tham số cuối cùng thiết lập giá trị cho tùy chọn. Để chỉ ra giá trị TimeOut ta dùng như sau:

```
server.SetSocketOption(SocketOptionLevel.Socket,
    SocketOptionName.ReceiveTimeout, 3000);
```

Trong đó giá trị của tham số cuối cùng chỉ ra số miligiây tối đa hàm ReceiveFrom() sẽ chờ cho đến khi có dữ liệu để đọc. Chương trình sau sẽ minh họa cách dùng TimeOut:

Chương trình TimeOutUdpClient

```
using System;
using System.Net;
```

```
using System.Net.Sockets;
using System.Text;
class TimeoutUdpClient
{
    public static void Main()
    {
        byte[] data = new byte[1024];
        string input, stringData;
        int recv;
        IPEndPoint ipep = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 5000);
        Socket server = new Socket(AddressFamily.InterNetwork,
                SocketType.Dgram, ProtocolType.Udp);
        int sockopt = (int)server.GetSocketOption(SocketOptionLevel.Socket,
        SocketOptionName.ReceiveTimeout);
        Console.WriteLine("Gia tri timeout mac dinh: {0}", sockopt);
        server.SetSocketOption(SocketOptionLevel.Socket,
        SocketOptionName.ReceiveTimeout, 3000);
        sockopt = (int)server.GetSocketOption(SocketOptionLevel.Socket,
        SocketOptionName.ReceiveTimeout);
        Console.WriteLine("Gia tri timeout moi: {0}", sockopt);
        string welcome = "Xin chao server";
        data = Encoding.ASCII.GetBytes(welcome);
        server.SendTo(data, data.Length, SocketFlags.None, ipep);
        IPEndPoint sender = new IPEndPoint(IPAddress.Any, 0);
        EndPoint tmpRemote = (EndPoint)sender;
        data = new byte[1024];
        recv = server.ReceiveFrom(data, ref tmpRemote);
        Console.WriteLine("Thong diep duoc nhan tu {0}:", mpRemote.ToString());
        Console.WriteLine(Encoding.ASCII.GetString(data, 0, recv));
        while (true)
        {
            input = Console.ReadLine();
            if (input == "exit")
                break;
            server.SendTo(Encoding.ASCII.GetBytes(input), tmpRemote);
            data = new byte[1024];
            recv = server.ReceiveFrom(data, ref tmpRemote);
            stringData = Encoding.ASCII.GetString(data, 0, recv);
            Console.WriteLine(stringData);
        }
        Console.WriteLine("Dang dong client");
        server.Close();
    }
```

```
}
```

Chương trình TimeoutUdpClient đầu tiên lấy giá trị ReceiveTimeout ban đầu từ Socket và hiển thị nó, sau đó thiết lập giá trị này thành 3 giây:

```
int sockopt = (int)server.GetSocketOption(SocketOptionLevel.Socket,
SocketOptionName.ReceiveTimeout);
Console.WriteLine("Gia tri timeout mac dinh: {0}", sockopt);
server.SetSocketOption(SocketOptionLevel.Socket,
SocketOptionName.ReceiveTimeout, 3000);
```

Phương thức GetSocketOption() trả về một đối tượng Object, vì thế nó phải được ép kiểu thành kiểu interger. Sau khi biên dịch và chạy chương trình với chương trình SimpleUdpServer ở trên, kết quả xuất ra như sau:

```
C:\>TimeoutUdpClient
Gia tri timeout mac dinh: 0
Gia tri timeout moi: 3000
Unhandled Exception: System.Net.Sockets.SocketException: An existing
connection
was forcibly closed by the remote host
  at System.Net.Sockets.Socket.ReceiveFrom(Byte[] buffer, Int32 offset,
Int32
size, SocketFlags socketFlags, EndPoint& remoteEP)
  at System.Net.Sockets.Socket.ReceiveFrom(Byte[] buffer, EndPoint&
remoteEP)
  at TimeoutUdpClient.Main()
C:\>
```

Giá trị ban đầu của ReceiveTimeout được thiết lập là 0 cho biết nó sẽ chờ dữ liệu mãi mãi. Sau khi thêm phương thức SetSocketOpition() và được thiết lập giá trị 3000 mili giây thì hàm ReceiveFrom() sẽ đợi dữ liệu trong 3 giây, sau 3 giây nếu không có dữ liệu để đọc thì nó sẽ phát sinh ra biệt lệ do đó ta phải đặt hàm này trong khối try – catch để xử lý biệt lệ.

III.6. Điều khiển việc truyền lại các gói tin

Có nhiều lý do các gói tin UDP không thể đến được đích, có thể lần đầu tiên gởi không tới được đích nhưng khi gởi lại lần thứ hai, ba thì tới được đích. Hầu hết các ứng dụng udp đều cho phép gởi lại các gói tin một số lần trước khi loại bỏ nó. Khi gởi một gói tin mà không có thông điệp trả về, chúng ta có thể gởi lại thông điệp ban đầu nhiều lần, nếu sau khi gởi lại một số lần thông điệp đã đến được đích thì ta tiếp tục với phần còn lại của chương trình ngược lại ta sẽ phát sinh ra một thông báo lỗi.

Cách đơn giản nhất để thực hiện việc truyền lại là tạo ra một phương thức riêng trong lớp để điều khiển tất cả việc gởi và nhận các thông điệp. Các bước thực hiện như sau:

1) Gởi một thông điệp đến máy ở xa

2) Chờ câu trả lời từ máy ở xa

3) Nếu câu trả lời được nhận, chấp nhận nó và thoát khỏi phương thức với dữ liệu nhận và kích thước của dữ liệu.

4) Nếu không nhận được câu trả lời nào trong khoảng thời gian time-out, tăng biến đếm thử lên

5) Kiểm tra biến đếm thử, nếu nó nhỏ hơn số lần đã định nghĩa trước thì quay lại bước 1, nếu nó bằng số lần đã định nghĩa trước, không truyền lại nữa và thông báo lỗi.

Một khi phương thức để gởi và nhận các gói tin udp đã được tạo ra, nó có thể được dùng bất cứ đâu trong chương trình nơi có dữ liệu được gởi tới thiết bị ở xa và chờ câu trả lời. Phương thức này được cài đặt như sau:

```
private int SndRcvData(Socket s, byte[] message, EndPoint rmtdevice)
{
      int recv;
      int retry = 0;
      while (true)
{
Console.WriteLine("Truyen lai lan thu: #{0}", retry);
try
{
      s.SendTo(message, message.Length, SocketFlags.None, rmtdevice);
      data = new byte[1024];
      recv = s.ReceiveFrom(data, ref Remote);
}
catch (SocketException)
{
      recv = 0;
}
if (recv > 0)
{
```

```
        return recv;
}
else
{
        retry++;
        if (retry > 4)
        {
                return 0;
        }
}
```

Phương thức này yêu cầu ba tham số:

- ✓ Một đối tượng socket đã được thành lập
- ✓ Mảng dữ liệu chứa thông điệp để gởi tới thiết bị ở xa
- ✓ Một đối tượng EndPoint chứa địa chỉ IP và port của thiết bị ở xa

Đối tượng Socket được truyền vào phương thức trên phải được khởi tạo trước và giá trị ReceiveTimeout đã được thiết lập một khoảng thời gian chờ câu trả lời từ thiết bị ở xa.

Phương thức SndRcvData() đầu tiên gởi dữ liệu đến thiết bị ở xa dùng phương thức SendTo() truyền thống. Sau khi gởi thông điệp, phương thức SndRcvData() sẽ block ở phương thức ReceiveFrom() và chờ thông điệp trả về. Nếu thông điệp được nhận từ thiết bị ở xa trong khoảng giá trị ReceiveTimeout thì phương thức SndRcvData() sẽ đặt dữ liệu vào mảng byte đã được định nghĩa trong lớp và trả về số byte đọc được. Nếu không có thông điệp trả về vào lúc kết thúc giá trị ReceiveTimeout, một biệt lệ sẽ được phát ra và khối catch được xử lý. Trong khối catch, giá trị recv được thiết lập về 0. Sau khối try-catch, giá trị recv sẽ được kiểm tra. Nếu giá trị đó là số dương thì thông điệp đã được nhận thành công, nếu là số 0 thì không có thông điệp nào được nhận và giá trị này được tăng lên, sau đó kiểm tra nó đã đạt tới giá trị tối đa hay chưa, nếu chưa đạt tới giá trị tối đa toàn bộ quá trình sẽ được lặp lại và bắt đầu gởi lại thông điệp, nếu đã tới giá trị tối đa rồi thì phương thức SndRcvData() sẽ trả về 0.

Chương trình RetryUdpClient

```
using System;
using System.Net;
```

```
using System.Net.Sockets;
using System.Text;
class RetryUdpClient
{
    private byte[] data = new byte[1024];
    private static IPEndPoint sender = new IPEndPoint(IPAddress.Any, 0);
    private static EndPoint Remote = (EndPoint)sender;
    private int SndRcvData(Socket s, byte[] message, EndPoint rmtdevice)
    {
        int recv;
        int retry = 0;
        while (true)
        {
            Console.WriteLine("Truyen lai lan thu: #{0}", retry);
            try
            {
                s.SendTo(message, message.Length, SocketFlags.None, rmtdevice);
                data = new byte[1024];
                recv = s.ReceiveFrom(data, ref Remote);
            }
            catch (SocketException)
            {
                recv = 0;
            }
            if (recv > 0)
            {
                return recv;
            }
            else
            {
                retry++;
                if (retry > 4)
                {
                    return 0;
                }
            }
        }
    }
    public RetryUdpClient()
    {
        string input, stringData;
        int recv;
        IPEndPoint ipep = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 5000);
```

```
        Socket server = new Socket(AddressFamily.InterNetwork,
                SocketType.Dgram, ProtocolType.Udp);
        int sockopt = (int)server.GetSocketOption(SocketOptionLevel.Socket,
         SocketOptionName.ReceiveTimeout);
        Console.WriteLine("Gia tri timeout mac dinh: {0}", sockopt);
        server.SetSocketOption(SocketOptionLevel.Socket,
         SocketOptionName.ReceiveTimeout, 3000);
        sockopt = (int)server.GetSocketOption(SocketOptionLevel.Socket,
         SocketOptionName.ReceiveTimeout);
        Console.WriteLine("Gia tri timeout moi: {0}", sockopt);
        string welcome = "Xin chao Server";
        data = Encoding.ASCII.GetBytes(welcome);
        recv = SndRcvData(server, data, ipep);
        if (recv > 0)
        {
            stringData = Encoding.ASCII.GetString(data, 0, recv);
            Console.WriteLine(stringData);
        }
        else
        {
            Console.WriteLine("Khong the lien lac voi thiet bi o xa");
            return;
        }
        while (true)
        {
            input = Console.ReadLine();
            if (input == "exit")
                break;
            recv = SndRcvData(server, Encoding.ASCII.GetBytes(input), ipep);
            if (recv > 0)
            {
                stringData = Encoding.ASCII.GetString(data, 0, recv);
                Console.WriteLine(stringData);
            }
            else
                Console.WriteLine("Khong nhan duoc cau tra loi");
        }
        Console.WriteLine("Dang dong client");
        server.Close();
    }
    public static void Main()
    {
        RetryUdpClient ruc = new RetryUdpClient();
```

```
        }
}
```

CHƯƠNG IV: SỬ DỤNG CÁC LỚP HELPER CỦA C# SOCKET

IV.1. Lớp TCP Client

Lớp TcpClient nằm ở namespace System.Net.Sockets được thiết kế để hỗ trợ cho việc viết các ứng dụng TCP Client được dễ dàng.

Lớp TcpClient cho phép tạo ra một đối tượng Tcp Client sử dụng một trong ba phương thức tạo lập sau:

TcpClient(): là phương thức tạo lập đầu tiên, đối tượng được tạo ra bởi phương thức tạo lập này sẽ gắn kết với một địa chỉ cục bộ và một port TCP ngẫu nhiên. Sau khi đối tượng TcpClient được tạo ra, nó phải được kết nối đến thiết bị ở xa thông qua phương thức Connect() như ví dụ dưới đây:

```
TcpClient newcon = new TcpClient();
newcon.Connect("192.168.6.1", 8000);
```

TcpClient(IPEndPoint localEP): phương thức tạo lập này cho phép chúng ta chỉ ra địa chỉ IP cục bộ cùng với port được dùng. Đây là phương thức tạo lập thường được sử dụng khi thiết bị có nhiều hơn một card mạng và chúng ta muốn dữ liệu được gởi trên card mạng nào. Phương thức Connect() cũng được dùng để kết nối với thiết bị ở xa:

```
IPEndPoint iep = new IPEndPoint(IPAddress,Parse("192.168.6.1"),
8000);
TcpClient newcon = new TcpClient(iep);
newcon.Connect("192.168.6.2", 8000);
```

TcpClient(String host, int port): phương thức tạo lập thứ ba này thường được sử dụng nhất, nó cho phép chỉ ra thiết bị nhận trong phương thức tạo lập và không cần phải dùng phương thức Connect(). Địa chỉ của thiết bị ở xa có thể là một chuỗi hostname hoặc một chuỗi địa chỉ IP. Phương thức tạo lập của TcpClient sẽ tự động phân giải hostname thành địa chỉ IP. Ví dụ:

```
TcpClient newcon = new TcpClient("www.isp.net", 8000);
```

Mỗi khi đối tượng TcpClient được tạo ra, nhiều thuộc tính và phương thức có thể được dùng để xử lý việc truyền dữ liệu qua lại giữa các thiết bị.

Phương Thức	**Mô Tả**
Close()	Đóng kết nối TCP
Connect()	Thành lập kết nối TCP với thiết bị ở xa
Equals()	So sánh hai đối tượng TcpClient
GetHashCode()	Lấy mã hash code
GetStream()	Lấy đối tượng Stream nó có thể dùng để gởi và nhận dữ liệu
GetType()	Lấy kiểu của thể hiện hiện tại
ToString()	Chuyển thể hiện hiện tại sang kiểu chuỗi

Phương thức Connect() dùng để kết nối đối tượng TcpClient đến thiết bị ở xa. Mỗi khi kết nối được thành lập, phương thức GetStream() gán một đối tượng NetworkStream để gởi và nhận dữ liệu nhờ vào phương thức Read() và Write(). Lớp TcpClient còn có nhiều thuộc tính được mô tả trong bảng sau:

Thuộc Tính	**Mô Tả**
LingerState	Lấy hoặc thiết lập thời gian kết nối TCP vẫn còn sau khi gọi phương thức Close()
NoDelay	Lấy hoặc thiết lập thời gian trễ được dùng để gởi hoặc nhận ở bộ đệm TCP
ReceiveBufferSize	Lấy hoặc thiết lập kích thước bộ đệm TCP nhận
ReceiveTimeout	Lấy hoặc thiết lập thời gian timeout của Socket
SendBufferSize	Lấy hoặc thiết lập kích thước bộ đệm TCP gởi
SendTimeout	Lấy hoặc thiết lập giá trị timeout của Socket

Chương trình TCPClient đơn giản

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
class TcpClientSample
{
    public static void Main()
    {
        byte[] data = new byte[1024];
        string input, stringData;
        TcpClient server;
```

```
        try
        {
            server = new TcpClient("127.0.0.1", 5000);
        }
        catch (SocketException)
        {
            Console.WriteLine("Khong the ket noi den server");
            return;
        }
        NetworkStream ns = server.GetStream();
        int recv = ns.Read(data, 0, data.Length);
        stringData = Encoding.ASCII.GetString(data, 0, recv);
        Console.WriteLine(stringData);
        while (true)
        {
            input = Console.ReadLine();
            if (input == "exit")
                break;
            ns.Write(Encoding.ASCII.GetBytes(input), 0,
input.Length);
            ns.Flush();
            data = new byte[1024];
            recv = ns.Read(data, 0, data.Length);
            stringData = Encoding.ASCII.GetString(data, 0, recv);
            Console.WriteLine(stringData);
        }
        Console.WriteLine("Dang ngat ket noi voi server...");
        ns.Close();
        server.Close();
    }
}
```

Trong chương trình trên phương thức tạo lập sẽ tự động kết nối đến Server được chỉ ra ở xa, nó nên được đặt ở trong khối try-catch để phòng trường hợp Server không hợp lệ.

Sau khi đối tượng NetworkStream được tạo ra, ta có thể dùng phương thức Read() và Write() để nhận và gởi dữ liệu:

```
while (true)
{
      input = Console.ReadLine();
      if (input == "exit")
      break;
      ns.Write(Encoding.ASCII.GetBytes(input), 0, input.Length);
      ns.Flush();
      data = new byte[1024];
      recv = ns.Read(data, 0, data.Length);
      stringData = Encoding.ASCII.GetString(data, 0, recv);
      Console.WriteLine(stringData);
}
```

Phương thức Read() yêu cầu 3 tham số:

- ✓ Mảng các byte để đặt dữ liệu nhận vào
- ✓ Vị trí offset trong bộ đệm mà tại đó ta muốn đặt dữ liệu
- ✓ Chiều dài của bộ đệm dữ liệu

Cũng giống như phương thức Receive() của Socket, phương thức Read() sẽ đọc một lượng dữ liệu có độ lớn tối đa đúng bằng độ lớn bộ đệm. Nếu bộ đệm quá nhỏ, phần dữ liệu còn lại sẽ nằm ở trong stream và đợi lần gọi phương thức Read() tiếp theo.

Phương thức Write() cũng yêu cầu ba tham số:

- ✓ Mảng các byte để gởi dữ liệu
- ✓ Vị trí offset trong bộ đệm mà tại đó ta muốn gởi dữ liệu
- ✓ Chiều dài của dữ liệu được gởi

Cần chú ý rằng TCP không bảo vệ các biên thông điệp. Điều này cũng áp dụng cho lớp TcpClient, do đó ta cần phải xử lý vấn đề biên thông điệp giống như phương thức Receive() của lớp Socket bằng cách tạo ra vòng lặp để đảm bảo tất cả dữ liệu đều được đọc từ stream.

Ta có thể test chương trình này với chương trình TCP Server đơn giản ở phần trên.

IV.2. Lớp TCPListener

Cũng giống như lớp TcpClient, lớp TcpListener cũng cho phép chúng ta tạo ra các chương trình TCP Server một cách đơn giản

Lớp TcpListener có ba phương thức tạo lập:

- ✓ TcpListener(int port): gắn một đối tượng TcpListener vào một port được chỉ ra trên máy cục bộ.
- ✓ TcpListener(IPEndPoint ie): gắn một đối tượng TcpListener vào một đối tượng EndPoint cục bộ
- ✓ TcpListener(IPAddress addr, int port): gắn một đối tượng TcpListener vào một đối tượng IPAddress và một port

Không giống như lớp TcpClient, các phương thức tạo lập của lớp TcpListener yêu cầu ít nhất một tham số: số port mà Server lắng nghe kết nối. Nếu Server có nhiều card mạng và ta muốn lắng nghe trên một card mạng nào đó thì ta có thể dùng một đối tượng IPEndPoint để chỉ ra địa chỉ IP của card cùng với số port dùng để lắng nghe.

Lớp TcpListener có một số phương thức sau:

Phương Thức	**Mô Tả**
AcceptSocket()	Chấp nhận kết nối trên port và gán kết nối cho một đối tượng Socket
AcceptTcpClient()	Chấp nhận kết nối trên port và gán kết nối cho một đối tượng TCPClient
Equals()	So sánh hai đối tượng TcpListener
GetHashCode()	Lấy hash code
GetType()	Lấy kiểu của thể hiện hiện tại
Pending()	Kiểm tra xem có yêu cầu đang chờ kết nối hay không
Start()	Bắt đầu lắng nghe kết nối
Stop()	Ngừng lắng nghe kết nối
ToString()	Chuyển đối tượng TcpListener thành chuỗi

Phương thức Start() tương tự như phương thứcBind() và Listen() được dùng ở lớp socket. Phương thức Start() kết nối Socket đến EndPoint được định nghĩa ở phương thức tạo lập của lớp TcpListener và đặt TCP port vào chế độ lắng nghe, sẵng sàng chấp nhận kết nối. Phương thức AcceptTcpClient() có thể so sánh với phương thức Accept() của Socket, chấp nhận kết nối và gán nó cho một đối tượng TcpClient.

Sau khi đối tượng TcpClient được tạo ra, tất cả các truyền thông với thiết bị ở xa được thực hiện với đối tượng TcpClient mới chứ không phải với đối tượng

TcpListener ban đầu, do đó đối tượng TcpListener có thể được dùng để chấp nhận kết nối khác. Để đóng đối tượng TcpListener ta dùng phương thức Stop().

Chương trình TCPListener

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
class TcpListenerSample
{
    public static void Main()
    {
        int recv;
        byte[] data = new byte[1024];
        TcpListener newsock = new TcpListener(5000);
        newsock.Start();
        Console.WriteLine("Dan cho client ket noi den...");
        TcpClient client = newsock.AcceptTcpClient();
        NetworkStream ns = client.GetStream();
        string welcome = "Hello Client";
        data = Encoding.ASCII.GetBytes(welcome);
        ns.Write(data, 0, data.Length);
        while (true)
        {
            data = new byte[1024];
            recv = ns.Read(data, 0, data.Length);
            if (recv == 0)
                break;

            Console.WriteLine(Encoding.ASCII.GetString(data, 0, recv));
            ns.Write(data, 0, recv);
        }
        ns.Close();
        client.Close();
        newsock.Stop();
    }
}
```

Chương trình TcpListenerSample đầu tiên tạo ra một đối tượng TcpListener, sử dụng port 5000 và dùng phương thức Start() để đặt đối tượng mới tạo ra vào chế độ

lắng nghe. Sau đó phương thức AcceptTcpClient() chờ kết nối TCP đến và gán kết nối đến này vào một đối tượng TcpClient:

```
TcpListener newsock = new TcpListener(5000);
newsock.Start();
Console.WriteLine("Dan cho client ket noi den...");
TcpClient client = newsock.AcceptTcpClient();
NetworkStream ns = client.GetStream();
```

Khi đối tượng TcpClient được thành lập, một đối tượng NetworkStream được gán vào để truyền thông với máy ở xa. Tất cả các thông tin liên lạc đều được thực hiện bằng cách sử dụng phương thức Read() và Write()

IV.3. Lớp UdpClient

Lớp UdpClient được tạo ra để giúp cho việc lập trình mạng với giao thức UDP được đơn giản hơn. Lớp UdpClient có bốn phương thức tạo lập:

- ✓ UdpClient(): tạo ra một đối tượng UdpClient nhưng không gắn vào bất kỳ địa chỉ hay port nào.
- ✓ UdpClient(int port): gắn đối tượng UdpClient mới tạo vào một port/
- ✓ UdpClient(IPEndPoint iep): gắn đối tượng UdpClient mới tạo vào một địa chỉ Ip cục bộ và một port.
- ✓ UdpClient(string host, int port): gắn đối tượng UdpClient mới tạo vào một địa chỉ IP và một port bất kỳ và kết hợp nó với một địa chỉ IP và port ở xa.

Các phương thức tạo lập của lớp UdpClient tương tự như các phương thức tạo lập của lớp TcpClient, chúng ta có thể để cho hệ thống chọn port thích hợp cho ứng dụng hoặc ta có thể chỉ ra port được dùng trong ứng dụng. Nếu ứng dụng UDP phải chấp nhận dữ liệu trên một port nào đó, ta phải định nghĩa port đó trong phương thức tạo lập của lớp UdpClient.

Một số phương thức của lớp UdpClient:

Phương Thức	**Mô Tả**
Close()	Đóng Socket ở bên dưới
Connect()	Cho phép chỉ ra IP endpoint ở xa để gởi và nhận dữ liệu
DropMulticastGroup()	Gỡ bỏ Socket từ 1 nhóm UDP multicast
Equals()	So sánh hai đối tượng UdpClient

Phương Thức	**Mô Tả**
GetHashCode()	Lấy hash code
GetType()	Lấy kiểu của đối tượng hiện tại
JoinMulticastGroup()	Thêm Socket vào một nhóm UDP multicast
Receive()	Nhận dữ liệu từ Socket
Send()	Gởi dữ liệu đến thiết bị ở xa từ Socket
ToString()	Chuyển đối tượng UdpClient thành chuỗi

Có nhiều chỗ khác nhau giữa phương thức Send(), Receive() của lớp UdpClient và phương thức SendTo(), ReceiveFrom() của Socket.

Phương thức Receive()

Lớp UdpClient sử dụng phương thức Receive() để chấp nhận các gói tin trên một card mạng và một port. Chỉ có một cách sử dụng của phương thức Receive():

```
byte[] Receive(ref IPEndPoint iep)
```

Khi dữ liệu được nhận từ Socket, nó không đặt vào mảng byte trong phương thức như trong phương thức ReceiveFrom() mà nó sẽ trả về một mảng byte. Sự khác nhau thứ hai là phương thức ReceiveFrom() đặt thông tin của máy ở xa vào một đối tượng EndPoint còn phương thức Receive() đặt thông tin của máy ở xa vào một đối tượng IPEndPoint, việc này có thể làm cho lập trình viên cảm thấy dễ lập trình hơn.

Khi nhiều dữ liệu được nhận hơn kích thước bộ đệm, thay vì phát sinh ra một biệt lệ như trong phương thức ReceiveFrom() của Socket, UdpClient tả về một bộ đệm dữ liệu đủ lớn để chứa dữ liệu nhận đây là một tính năng rất hay của phương thức Receive().

Phương thức Send()

Phương thức Send() có ba quá tải hàm để gởi dữ liệu tới thiết bị ở xa:

- ✓ Send(byte[] data, int sz): gởi một mảng dữ liệu với kích thước là sz đến thiết bị ở xa mặc định. Để dùng quá tải hàm này, ta phải chỉ ra thiết bị ở xa mặc định bằng cách hoặc sử dụng phương thức tạo lập của lớp UdpClient hoặc dùng phương thức Connect().
- ✓ Send(byte[] data, int sz, IPEndPoint iep): cho phép gởi mảng dữ liệu có kích thước sz đến thiết bị ở xa được chỉ ra bởi iep.

- ✓ Send(byte[] data, int sz, string host, int port): gởi mảng dữ liệu kích thước sz đến máy ở xa và port được chỉ ra.

Chương trình UdpClient Server

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
class UdpSrvrSample
{
    public static void Main()
    {
        byte[] data = new byte[1024];
        IPEndPoint ipep = new IPEndPoint(IPAddress.Any, 5000);
        UdpClient newsock = new UdpClient(ipep);
        Console.WriteLine("Dang cho client ket noi den...");
        IPEndPoint sender = new IPEndPoint(IPAddress.Any, 0);
        data = newsock.Receive(ref sender);
        Console.WriteLine("Thong diep duoc nhan tu {0}:", sender.ToString());
        Console.WriteLine(Encoding.ASCII.GetString(data, 0, data.Length));
        string welcome = "Xin chao client";
        data = Encoding.ASCII.GetBytes(welcome);
        newsock.Send(data, data.Length, sender);
        while (true)
        {
            data = newsock.Receive(ref sender);

            Console.WriteLine(Encoding.ASCII.GetString(data, 0, data.Length));
            newsock.Send(data, data.Length, sender);
        }
    }
}
```

Chương trình UdpClient Client

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
class UdpClientSample
{
    public static void Main()
    {
        byte[] data = new byte[1024];
```

```
        string input, stringData;
        UdpClient server = new UdpClient("127.0.0.1", 5000);
        IPEndPoint sender = new IPEndPoint(IPAddress.Any, 0);
        string welcome = "Xin chao server";
        data = Encoding.ASCII.GetBytes(welcome);
        server.Send(data, data.Length);
        data = server.Receive(ref sender);
        Console.WriteLine("Thong diep duoc nhan tu {0}:", sender.ToString());
        stringData = Encoding.ASCII.GetString(data, 0, data.Length);
        Console.WriteLine(stringData);
        while (true)
        {
            input = Console.ReadLine();
            if (input == "exit")
                break;

            server.Send(Encoding.ASCII.GetBytes(input), input.Length);
            data = server.Receive(ref sender);
            stringData = Encoding.ASCII.GetString(data, 0, data.Length);
            Console.WriteLine(stringData);
        }
        Console.WriteLine("Dang dong client");
        server.Close();
    }
}
```

CHƯƠNG V: ĐA NHIỆM TIỂU TRÌNH

V.1. Khái niệm tiến trình và tiểu trình của Windows

Tiến trình là một thể hiện của chương trình đang hoạt động. Một tiến trình luôn sở hữu một không gian địa chỉ có kích thước 4GB chứa mã chương trình, các dữ liệu, sở hữu tài nguyên của hệ thống như tập tin, đối tượng đồng bộ hóa....

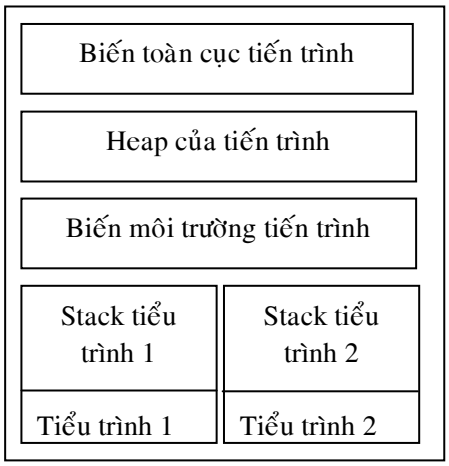
Mỗi tiến trình khi mới tạo lập đều chỉ có một tiểu trình chính nhưng sau đó có thể tạo lập nhiều tiến trình khác.

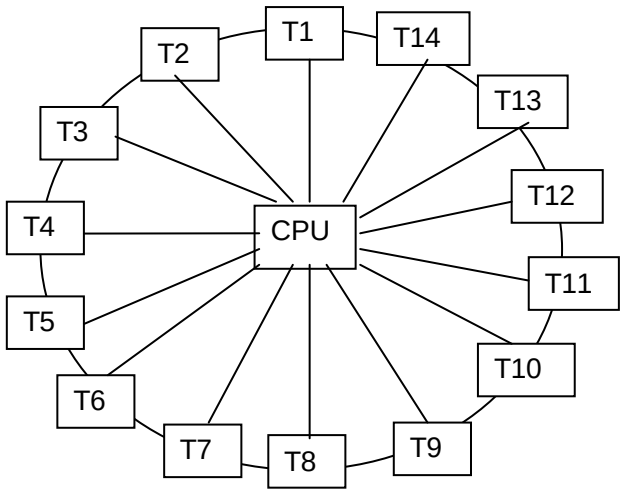
Tiểu trình là một thành phần đơn vị của tiến trình có thể thực hiện các chỉ thị ứng với một đoạn mã nào đó của chương trình.

Hệ điều hành Windows cho phép các tiểu trình hoạt động độc lập và tổ chức điều phối (lập lịch tiến trình) CPU để các tiểu trình hoạt động đồng thời. Hệ điều hành phân chia thời gian sử dụng CPU cho mỗi tiến trình rất mịn theo kiểu xoay vòng. Mỗi tiểu trình có thể có 1 trong 3 trạng thái : Running, Ready, Blocked.

Các tiểu trình trong một tiến trình có thể cùng truy xuất đến các biến toàn cục của tiến trình.

V.2. Mô hình





Các ứng dụng cài đặt theo mô hình đa tiến trình hay đa tiểu trình đều phải đối diện với các vấn đề sau :

- Hệ thống tiêu thụ thêm bộ nhớ để lưu trữ các cấu trúc mô tả tiến trình hay tiểu trình.

- Hệ thống tốn thêm thời gian để theo vết chương trình, quản lý các tiểu trình.
- Nhiều tiến trình tranh chấp tài nguyên dùng chung đòi hỏi thực hiện đồng bộ hóa.

V.3. Các kỹ thuật trong .NET tạo tiểu trình

Thư viện lớp .NET Framework cung cấp một số phương pháp tạo tiểu trình mới:

- ✓ Thực thi một phương phức bằng tiểu trình trong Thread-pool
- ✓ Thực thi phương thức một cách bất đồng bộ
- ✓ Thực thi một phương thức bằng tiểu trình theo chu kỳ hay ở một thời điểm xác định
- ✓ Thực thi phương thức bằng cách ra hiệu đối tượng WaitHandle

V.3.1. Tạo tiểu trình trong Thread-pool

Cách tạo:

- ✓ Khai báo một phương thức chứa mã lệnh cần thực thi
- ✓ Phương thức này phải trả về void và chỉ nhận một đối số.
- ✓ Tạo một thể hiện của ủy nhiệm System.Threading.WaitCallback tham chiếu đến phương thức này.
- ✓ Gọi phương thức tĩnh QueueUserWorkItem của lớp System.Threading.ThreadPool,
- ✓ Truyền thể hiện ủy nhiệm đã tạo làm đối số.
- ✓ Bộ thực thi sẽ xếp thể hiện ủy nhiệm này vào hàng đợi và thực thi nó khi một tiểu trình trong thread-pool sẵn sàng.

Ví dụ thực thi một phương thức có tên là DisplayMessage:

Truyền DisplayMessage đến thread-pool hai lần

Lần đầu không có đối số

Lần sau có đối số là đối tượng MessageInfo

Chương trình ThreadPoolExample

```
using System;
using System.Threading;
// Lớp dùng để truyền dữ liệu cho phương thức DisplayMessage
// khi nó được thực thi bằng thread-pool.
```

```
public class MessageInfo
{
    private int iterations;
    private string message;
    // Phương thức khởi dựng nhận các thiết lập cấu hình cho tiểu trình.
    public MessageInfo(int iterations, string message)
    {
        this.iterations = iterations;
        this.message = message;
    }
    // Các thuộc tính dùng để lấy các thiết lập cấu hình.
    public int Iterations { get { return iterations; } }
    public string Message { get { return message; } }
}
public class ThreadPoolExample
{
    // Hiển thị thông tin ra cửa sổ Console.
    public static void DisplayMessage(object state)
    {
        // Ép đối số state sang MessageInfo.
        MessageInfo config = state as MessageInfo;
        // Nếu đối số config là null, không có đối số nào truyền cho phương
thức
        //               ThreadPool.QueueUserWorkItem;
        // sử dụng các giá trị mặc định.
        if (config == null)
        {
            // Hiển thị một thông báo ra cửa sổ Console ba lần.
            for (int count = 0; count < 3; count++)
            {
                Console.WriteLine("A thread-pool example.");
                // Vào trạng thái chờ, dùng cho mục đích minh họa.
                // Tránh đưa các tiểu trình của thread-pool
                // vào trạng thái chờ trong các ứng dụng thực tế.
                Thread.Sleep(1000);
            }
        }
        else
        {
            // Hiển thị một thông báo được chỉ định trước
            // với số lần cũng được chỉ định trước.
            for (int count = 0; count < config.Iterations; count++)
            {
```

```
                Console.WriteLine(config.Message);
                // Vào trạng thái chờ, dùng cho mục đích minh họa.
                // Tránh đưa các tiểu trình của thread-pool
                // vào trạng thái chờ trong các ứng dụng thực tế.
                Thread.Sleep(1000);
            }
        }
    }
    public static void Main()
    {
        // Tạo một đối tượng ủy nhiệm, cho phép chúng ta
        // truyền phương thức DisplayMessage cho thread-pool.
        WaitCallback workMethod = new
            WaitCallback(ThreadPoolExample.DisplayMessage);
        // Thực thi DisplayMessage bằng thread-pool (không có đối số).
        ThreadPool.QueueUserWorkItem(workMethod);
        // Thực thi DisplayMessage bằng thread-pool (truyền một
        // đối tượng MessageInfo cho phương thức DisplayMessage).
        MessageInfo info = new MessageInfo(5, "A thread-pool example with
arguments.");
        ThreadPool.QueueUserWorkItem(workMethod, info);
        // Nhấn Enter để kết thúc.
        Console.WriteLine("Main method complete. Press Enter.");
        Console.ReadLine();
    }
}
```

Tình huống sử dụng:

Khi một tiểu trình trong thread-pool sẵn sàng, nó nhận công việc kế tiếp từ hàng đợi và thực thi công việc này. Khi đã hoàn tất công việc, thay vì kết thúc, tiểu trình này quay về thread-pool và nhận công việc kế tiếp từ hàng đợi.

Việc sử dụng thread-pool giúp đơn giản hóa việc lập trình hỗ-trợ-đa-tiểu-trình. Tuy nhiên, cần lưu ý khi quyết định sử dụng thread-pool, cần xem xét các điểm sau:

- Không nên sử dụng thread-pool để thực thi các tiến trình chạy trong một thời gian dài. Vì số tiểu trình trong thread-pool là có giới hạn. Đặc biệt, tránh đặt các tiểu trình trong thread-pool vào trạng thái đợi trong một thời gian quá dài.

- Không thể điều khiển lịch trình của các tiểu trình trong thread-pool, cũng như không thể thay đổi độ ưu tiên của các công việc. Thread-pool xử lý các công việc theo thứ tự thêm chúng vào hàng đợi.

V.3.2. Tạo tiểu trình bất đồng bộ

Khi cho gọi một phương thức,thường thực hiện một cách đồng bộ; nghĩa là mã lệnh thực hiện lời gọi phải đi vào trạng thái dừng (block) cho đến khi phương thức được thực hiện xong.

Trong một số trường hợp, cần thực thi phương thức một cách bất đồng bộ; nghĩa là cho thực thi phương thức này trong một tiểu trình riêng trong khi vẫn tiếp tục thực hiện các công việc khác. Sau khi phương thức đã hoàn tất, cần lấy trị trả về của nó .

Nguyên tắc hoạt động:

NET Framework hỗ trợ chế độ thực thi bất đồng bộ, cho phép thực thi bất kỳ phương thức nào một cách bất đồng bộ bằng một ủy nhiệm.

Khi khai báo và biên dịch một ủy nhiệm, trình biên dịch sẽ tự động sinh ra hai phương thức hỗ trợ chế độ thực thi bất đồng bộ: BeginInvoke và EndInvoke. Khi gọi phương thức BeginInvoke của một thể hiện ủy nhiệm, phương thức được tham chiếu bởi ủy nhiệm này được xếp vào hàng đợi để thực thi bất đồng bộ.

Quyền kiểm soát quá trình thực thi được trả về cho mã gọi BeginInvoke ngay sau đó, và phương thức được tham chiếu sẽ thực thi trong ngữ cảnh của tiểu trình sẵn sàng trước tiên trong thread-pool.

Các bước thực hiện:

Khai báo một ủy nhiệm có chữ ký giống như phương thức cần thực thi.

Tạo một thể hiện của ủy nhiệm tham chiếu đến phương thức này.

Gọi phương thức BeginInvoke của thể hiện ủy nhiệm để thực thi phương thức

Sử dụng phương thức EndInvoke để kiểm tra trạng thái của phương thức cũng như thu lấy trị trả về của nó nếu đã hoàn tất .

Các đối số của phương thức BeginInvoke gồm các đối số được chỉ định bởi ủy nhiệm, cộng với hai đối số dùng khi phương thức thực thi bất đồng bộ kết thúc:

- ✓ Một thể hiện của ủy nhiệm System.AsyncCallback tham chiếu đến phương thức mà bộ thực thi sẽ gọi khi phương thức thực thi bất đồng bộ

kết thúc. Phương thức này sẽ được thực thi trong ngữ cảnh của một tiểu trình trong thread-pool. Truyền giá trị null cho đối số này nghĩa là không có phương thức nào được gọi và phải sử dụng một cơ chế khác để xác định khi nào phương thức thực thi bất bộ kết thúc.

- ✓ Một tham chiếu đối tượng mà bộ thực thi sẽ liên kết với quá trình thực thi bất đồng bộ. Phương thức thực thi bất đồng bộ không thể sử dụng hay truy xuất đến đối tượng này, nhưng mã lệnh có thể sử dụng nó khi phương thức này kết thúc, cho phép liên kết thông tin trạng thái với quá trình thực thi bất đồng bộ.

Ví dụ, đối tượng này cho phép ánh xạ các kết quả với các thao tác bất đồng bộ đã được khởi tạo trong trường hợp khởi tạo nhiều thao tác bất đồng bộ nhưng sử dụng chung một phương thức callback để xử lý việc kết thúc.

Phương thức EndInvoke cho phép lấy trị trả về của phương thức thực thi bất đồng bộ, nhưng trước hết phải xác định khi nào nó kết thúc.

Có bốn kỹ thuật dùng để xác định một phương thức thực thi bất đồng bộ đã kết thúc hay chưa:

Blocking: dừng quá trình thực thi của tiểu trình hiện hành cho đến khi phương thức thực thi bất đồng bộ kết thúc. Điều này rất giống với sự thực thi đồng bộ. Tuy nhiên, nếu linh hoạt chọn thời điểm chính xác để đưa mã lệnh vào trạng thái dừng (block) thì vẫn còn cơ hội thực hiện thêm một số việc trước khi mã lệnh đi vào trạng thái này.

Polling: lặp đi lặp lại việc kiểm tra trạng thái của phương thức thực thi bất đồng bộ để xác định nó kết thúc hay chưa. Đây là một kỹ thuật rất đơn giản, nhưng nếu xét về mặt xử lý thì không được hiệu quả. Nên tránh các vòng lặp chặt làm lãng phí thời gian của bộ xử lý; tốt nhất là nên đặt tiểu trình thực hiện polling vào trạng thái nghỉ (sleep) trong một khoảng thời gian bằng cách sử dụng Thread.Sleep giữa các lần kiểm tra trạng thái. Bởi kỹ thuật polling đòi hỏi phải duy trì một vòng lặp nên hoạt động của tiểu trình chờ sẽ bị giới hạn, tuy nhiên có thể dễ dàng cập nhật tiến độ công việc

Waiting: sử dụng một đối tượng dẫn xuất từ lớp System.Threading.WaitHandle để báo hiệu khi phương thức thực thi bất đồng bộ kết thúc. Waiting là một cải tiến của kỹ thuật polling, nó cho phépchờ nhiều phương thức thực thi bất đồng bộ kết thúc.

Có thể chỉ định các giá trị time-out cho phép tiểu trình thực hiện waiting dừng lại nếu phương thức thực thi bất đồng bộ đã diễn ra quá lâu, hoặc muốn cập nhật định kỳ bộ chỉ trạng thái.

Callback: Callback là một phương thức mà bộ thực thi sẽ gọi khi phương thức thực thi bất đồng bộ kết thúc. Mã lệnh thực hiện lời gọi không cần thực hiện bất kỳ thao tác kiểm tra nào, nhưng vẫn có thể tiếp tục thực hiện các công việc khác.
Callback rất linh hoạt nhưng cũng rất phức tạp, đặc biệt khi có nhiều phương thức thực thi bất đồng bộ chạy đồng thời nhưng sử dụng cùng một callback. Trong những trường hợp như thế, phải sử dụng các đối tượng trạng thái thích hợp để so trùng các phương thức đã hoàn tất với các phương thức đã khởi tạo.

Ví dụ:

Lớp AsyncExecutionExample trong mô tả cơ chế thực thi bất đồng bộ. Nó sử dụng một ủy nhiệm có tên là AsyncExampleDelegate để thực thi bất đồng bộ một phương thức có tên là LongRunningMethod.

Phương thức LongRunningMethod sử dụng Thread.Sleep để mô phỏng một phương thức có thời gian thực thi dài:

```
    // Ủy nhiệm cho phép bạn thực hiện việc thực thi bất đồng bộ
    //của AsyncExecutionExample.LongRunningMethod.
    public delegate DateTime AsyncExampleDelegate(int delay, string
name);
    // Phương thức có thời gian thực thi dài.
    public static DateTime LongRunningMethod(int delay, string name)
    {
        Console.WriteLine("{0} : {1} example - thread starting.",
                    DateTime.Now.ToString("HH:mm:ss.ffff"), name);
        // Mô phỏng việc xử lý tốn nhiều thời gian.
        Thread.Sleep(delay);
        Console.WriteLine("{0}:{1}example-thread finishing.",
                    DateTime.Now.ToString("HH:mm:ss.ffff"), name);
        // Trả về thời gian hoàn tất phương thức.
        return DateTime.Now;
    }
```

AsyncExecutionExample chứa 5 phương thức diễn đạt các cách tiếp cận khác nhau về việc kết thúc phương thức thực thi bất đồng bộ.

V.3.2.1. Phương thức BlockingExample

Phương thức BlockingExample thực thi bất đồng bộ phương thức LongRunningMethod và tiếp tục thực hiện công việc của nó trong một khoảng thời gian. Khi xử lý xong công việc này, BlockingExample chuyển sang trang thái dừng (block) cho đến khi phương thức LongRunningMethod kết thúc.

Để vào trạng thái dừng, BlockingExample thực thi phương thức EndInvoke của thể hiện ủy nhiệm AnsyncExampleDelegate. Nếu phương thức LongRunningMethod kết thúc, EndInvoke trả về ngay lập tức, nếu không, BlockingExample chuyển sang trạng thái dừng cho đến khi phương thức LongRunningMethod kết thúc.

```
public static void BlockingExample()
    {
        Console.WriteLine(Environment.NewLine + "*** Running
Blocking Example ***");
        // Gọi LongRunningMethod một cách bất đồng bộ. Truyền null
cho
        // cả ủy nhiệm callback và đối tượng trạng thái bất đồng bộ.
        AsyncExampleDelegate longRunningMethod = new
AsyncExampleDelegate(LongRunningMethod);
        IAsyncResult asyncResult =
longRunningMethod.BeginInvoke(2000, "Blocking", null, null);
        // Thực hiện công việc khác cho đến khi
        // sẵn sàng đi vào trạng thái dừng.
        for (int count = 0; count < 3; count++)
        {
            Console.WriteLine("{0} : Continue processing until " +
"ready to block...",
                    DateTime.Now.ToString("HH:mm:ss.ffff"));
            Thread.Sleep(200);
        }
        // Đi vào trạng thái dừng cho đến khi phương thức
        // thực thi bất đồng bộ kết thúc và thu lấy kết quả.
        Console.WriteLine("{0} : Blocking until method is
complete...", DateTime.Now.ToString("HH:mm:ss.ffff"));
```

```
        DateTime completion =
                                longRunningMethod.EndInvoke(asyncResult);
        // Hiển thị thông tin kết thúc.
        Console.WriteLine("{0} : Blocking example complete.",
completion.ToString("HH:mm:ss.ffff"));
    }
```

V.3.2.2. Phương thức PollingExample

Phương thức PollingExample thực thi bất đồng bộ phương thức LongRunningMethod và sau đó thực hiện vòng lặp polling cho đến khi LongRunningMethod kết thúc.

PollingExample kiểm tra thuộc tính IsComplete của thể hiện IAsyncResult (được trả về bởi BeginInvoke) để xác định phương thức LongRunningMethod đã kết thúc hay chưa, nếu chưa, PollingExample sẽ gọi Thread.Sleep.

```
public static void PollingExample()
    {
        Console.WriteLine(Environment.NewLine + " Running Polling
Example");
        // Gọi LongRunningMethod một cách bất đồng bộ. Truyền null
cho
        // cả ủy nhiệm callback và đối tượng trạng thái bất đồng bộ.
AsyncExampleDelegate longRunningMethod =  new
     AsyncExampleDelegate(LongRunningMethod);

        IAsyncResult asyncResult =
longRunningMethod.BeginInvoke(2000, "Polling", null, null);
        // Thực hiện polling để kiểm tra phương thức thực thi
        // bất đồng bộ kết thúc hay chưa. Nếu chưa kết thúc thì đi
vào
        // trạng thái chờ trong 300 mini-giây trước khi thực hiện
polling lần nữa.
        Console.WriteLine("{0} : Poll repeatedly until method is
complete...", DateTime.Now.ToString("HH:mm:ss.ffff")); while
(!asyncResult.IsCompleted)
        {
            Console.WriteLine("{0} : Polling...",
```

```
                DateTime.Now.ToString("HH:mm:ss.ffff"));
            Thread.Sleep(300);
        }
        // Thu lấy kết quả của phương thức thực thi bất đồng bộ.
        DateTime completion =
longRunningMethod.EndInvoke(asyncResult);
        // Hiển thị thông tin kết thúc.
        Console.WriteLine("{0} : Polling example complete.",
completion.ToString("HH:mm:ss.ffff"));
    }
```

V.3.2.3. Phương thức WaitingExample

Phương thức WaitingExample thực thi bất đồng bộ phương thức LongRunningExample và sau đó chờ cho đến khi LongRunningMethod kết thúc. WaitingExample sử dụng thuộc tính AsyncWaitHandle của thể hiện IAsyncResult (được trả về bởi BeginInvoke) để có được một WaitHandle.

Gọi phương thức WaitOne của WaitHandle. Việc sử dụng giá trị time-out cho phép WaitingExample dừng quá trình đợi để thực hiện công việc khác hoặc dừng hoàn toàn nếu phương thức thực thi bất đồng bộ diễn ra quá lâu.

```
  public static void WaitingExample()
    {
        Console.WriteLine(Environment.NewLine + "*** Running Waiting
Example ***");
        // Gọi LongRunningMethod một cách bất đồng bộ. Truyền null
cho
        // cả ủy nhiệm callback và đối tượng trạng thái bất đồng bộ.
        AsyncExampleDelegate longRunningMethod = new
                AsyncExampleDelegate(LongRunningMethod);
        IAsyncResult asyncResult =
                    longRunningMethod.BeginInvoke(2000, "Waiting",
null, null);
        // Đợi phương thức thực thi bất đồng bộ kết thúc.
        // Time-out sau 300 mili-giây và hiển thị trạng thái ra
        // cửa sổ Console trước khi tiếp tục đợi.
        Console.WriteLine("{0} : Waiting until method is
complete...", DateTime.Now.ToString("HH:mm:ss.ffff"));
```

```
        while (!asyncResult.AsyncWaitHandle.WaitOne(300, false))
        {
            Console.WriteLine("{0} : Wait timeout...",
DateTime.Now.ToString("HH:mm:ss.ffff"));
        }
        // Thu lấy kết quả của phương thức thực thi bất đồng bộ.
        DateTime completion =
longRunningMethod.EndInvoke(asyncResult);
        // Hiển thị thông tin kết thúc.
        Console.WriteLine("{0} : Waiting example complete.",
completion.ToString("HH:mm:ss.ffff"));
    }
```

V.3.2.4. Phương thức WaitAllExample

Phương thức WaitAllExample thực thi bất đồng bộ phương thức LongRunningMethod nhiều lần và sau đó sử dụng mảng các đối tượng WaitHandle để đợi cho đến khi tất cả các phương thức kết thúc.

```
public static void WaitAllExample()
    {
        Console.WriteLine(Environment.NewLine + "*** Running WaitAll
Example ***");
        // Một ArrayList chứa các thể hiện IAsyncResult
        // cho các phương thức thực thi bất đồng bộ.
        ArrayList asyncResults = new ArrayList(3);
        // Gọi ba lần LongRunningMethod một cách bất đồng bộ.
        // Truyền null cho cả ủy nhiệm callback và đối tượng
        // trạng thái bất đồng bộ. Thêm thể hiện IAsyncResult
        // cho mỗi phương thức vào ArrayList.
        AsyncExampleDelegate longRunningMethod = new
AsyncExampleDelegate(LongRunningMethod);
        asyncResults.Add(longRunningMethod.BeginInvoke(3000,
"WaitAll 1", null, null));
        asyncResults.Add(longRunningMethod.BeginInvoke(2500,
"WaitAll 2", null, null));
        asyncResults.Add(longRunningMethod.BeginInvoke(1500,
"WaitAll 3", null, null));
        // Tạo một mảng các đối tượng WaitHandle,
```

```
        // sẽ được sử dụng để đợi tất cả các phương thức
        // thực thi bất đồng bộ kết thúc.
        WaitHandle[] waitHandles = new WaitHandle[3];
        for (int count = 0; count < 3; count++)
        {
            waitHandles[count] =
            ((IAsyncResult)asyncResults[count]).AsyncWaitHandle;
        }
        // Đợi cả ba phương thức thực thi bất đồng bộ kết thúc.
        // Time-out sau 300 mili-giây và hiển thị trạng thái ra
        // cửa sổ Console trước khi tiếp tục đợi.
        Console.WriteLine("{0} : Waiting until all 3 methods are
complete...", DateTime.Now.ToString("HH:mm:ss.ffff")); while
(!WaitHandle.WaitAll(waitHandles, 300, false)) {
Console.WriteLine("{0} : WaitAll timeout...",
DateTime.Now.ToString("HH:mm:ss.ffff")); }
        // Kiểm tra kết quả của mỗi phương thức và xác định
        // thời gian phương thức cuối cùng kết thúc.
        DateTime completion = DateTime.MinValue;
        foreach (IAsyncResult result in asyncResults)
        {
            DateTime time = longRunningMethod.EndInvoke(result);
            if (time > completion) completion = time;
        }
        // Hiển thị thông tin kết thúc.
        Console.WriteLine("{0} : WaitAll example complete.",
completion.ToString("HH:mm:ss.ffff"));
    }
```

V.3.2.5. Phương thức CallbackExample

Phương thức CallbackExample thực thi bất đồng bộ phương thức LongRunningMethod và truyền một thể hiện ủy nhiệm AsyncCallback (tham chiếu đến phương thức CallbackHandler) cho phương thức BeginInvoke.

Phương thức CallbackHandler sẽ được gọi một cách tự động khi phương thức LongRunningMethod kết thúc. Kết quả là phương thức CallbackExample vẫn tiếp tục thực hiện công việc.

```
public static void CallbackExample()
```

```
    {
        Console.WriteLine(Environment.NewLine +
                "*** Running Callback Example ***");
        // Gọi LongRunningMethod một cách bất đồng bộ. Truyền một
        // thể hiện ủy nhiệm AsyncCallback tham chiếu đến
        // phương thức CallbackHandler. CallbackHandler sẽ
        // tự động được gọi khi phương thức thực thi bất đồng bộ
        // kết thúc. Truyền một tham chiếu đến thể hiện ủy nhiệm
        // AsyncExampleDelegate như một trạng thái bất đồng bộ;
        // nếu không, phương thức callback không thể truy xuất
        // thể hiện ủy nhiệm để gọi EndInvoke.
        AsyncExampleDelegate longRunningMethod = new
                    AsyncExampleDelegate(LongRunningMethod);
        IAsyncResult asyncResult =
longRunningMethod.BeginInvoke(2000, "Callback", new
AsyncCallback(CallbackHandler), longRunningMethod);
        // Tiếp tục với công việc khác.
        for (int count = 0; count < 15; count++)
        {
            Console.WriteLine("{0} : Continue processing...",
                DateTime.Now.ToString("HH:mm:ss.ffff"));
            Thread.Sleep(200);
        }
    }
    // Phương thức xử lý việc kết thúc bất đồng bộ bằng
callbacks.public
    static void CallbackHandler(IAsyncResult result)
    {
        // Trích tham chiếu đến thể hiện AsyncExampleDelegate
        // từ thể hiện IAsyncResult.
        AsyncExampleDelegate longRunningMethod =
(AsyncExampleDelegate)result.AsyncState;
        // Thu lấy kết quả của phương thức thực thi bất đồng bộ.
        DateTime completion = longRunningMethod.EndInvoke(result);
        // Hiển thị thông tin kết thúc.
        Console.WriteLine("{0} : Callback example complete.",
completion.ToString("HH:mm:ss.ffff"));
```

```
}
```

V.3.3. Thực thi phương thức bằng Timer

Kỹ thuật này giúp thực thi một phương thức trong một tiểu trình riêng theo chu kỳ hay ở một thời điểm xác định. Thông thường, rất hữu ích khi thực thi một phương thức ở một thời điểm xác định hay ở những thời khoảng xác định. Ví dụ, cần sao lưu dữ liệu lúc 1:00 AM mỗi ngày hay xóa vùng đệm dữ liệu mỗi 20 phút.

Lớp Timer giúp việc định thời thực thi một phương thức trở nên dễ dàng, cho phép thực thi một phương thức được tham chiếu bởi ủy nhiệm TimerCallback ở những thời khoảng nhất định. Phương thức được tham chiếu sẽ thực thi trong ngữ cảnh của một tiểu trình trong thread-pool.

Cách thực hiện

Khai báo một phương thức trả về void và chỉ nhận một đối tượng làm đối số. Sau đó, tạo một thể hiện ủy nhiệm System.Threading.TimerCallback tham chiếu đến phương thức này.

Tiếp theo, tạo một đối tượng System.Threading.Timer và truyền nó cho thể hiện ủy nhiệm TimerCallback cùng với một đối tượng trạng thái mà Timer sẽ truyền cho phương thức khi Timer hết hiệu lực.

Bộ thực thi sẽ chờ cho đến khi Timer hết hiệu lực và sau đó gọi phương thức bằng một tiểu trình trong thread-pool.

Khi tạo một đối tượng Timer, cần chỉ định hai thời khoảng (thời khoảng có thể được chỉ định là các giá trị kiểu int, long, uint, hay System.TimeSpan):

- ✓ Giá trị đầu tiên là thời gian trễ (tính bằng mili-giây) để phương thức được thực thi lần đầu tiên. Chỉ định giá trị 0 để thực thi phương thức ngay lập tức, và chỉ định System.Threading.Timeout.Infinite để tạo Timer ở trạng thái chưa bắt đầu (unstarted).
- ✓ Giá trị thứ hai là khoảng thời gian mà Timer sẽ lặp lại việc gọi phương thức sau lần thực thi đầu tiên. Nếu chỉ định giá trị 0 hay Timeout.Infinite thì Timer chỉ thực thi phương thức một lần duy nhất (với điều kiện thời gian trễ ban đầu không phải là Timeout.Infinite). Đối số thứ hai có thể cung cấp bằng các trị kiểu int, long, uint, hay System.TimeSpan.

Sau khi tạo đối tượng Timer, cũng có thể thay đổi các thời khoảng được sử dụng bởi Timer bằng phương thức Change, nhưng không thể thay đổi phương thức sẽ được gọi.

Khi đã dùng xong Timer, nên gọi phương thức Timer.Depose để giải phóng tài nguyên hệ thống bị chiếm giữ bởi Timer. Việc hủy Timer cũng hủy luôn phương thức đã được định thời thực thi.

Ví dụ

Lớp TimerExample dưới đây trình bày cách sử dụng Timer để gọi một phương thức có tên là TimerHandler. Ban đầu, Timer được cấu hình để gọi TimerHandler sau hai giây và lặp lại sau một giây. Ví dụ này cũng trình bày cách sử dụng phương thức Timer.Change để thay đổi các thời khoảng.

Chương trình TimerExample

```
using System;
using System.Threading;
public class TimerExample
{
    // Phương thức sẽ được thực khi Timer hết hiệu lực.
    // Hiển thị một thông báo ra cửa sổ Console.
    private static void TimerHandler(object state)
    {
        Console.WriteLine("{0} : {1}", DateTime.Now.ToString("HH:mm:ss.ffff"),
state);
    }
    public static void Main()
    {
        // Tạo một thể hiện ủy nhiệm TimerCallback mới
        // tham chiếu đến phương thức tĩnh TimerHandler.
        // TimerHandler sẽ được gọi khi Timer hết hiệu lực.
        TimerCallback handler = new TimerCallback(TimerHandler);
        // Tạo một đối tượng trạng thái, đối tượng này sẽ được
        // truyền cho phương thức TimerHandler.
        // Trong trường hợp này, một thông báo sẽ được hiển thị.
        string state = "Timer expired.";
        Console.WriteLine("{0} : Creating Timer.",
DateTime.Now.ToString("HH:mm:ss.ffff"));
        // Tạo một Timer, phát sinh lần đầu tiên sau hai giây
        // và sau đó là mỗi giây.
        using (Timer timer = new Timer(handler, state, 2000, 1000))
```

```
        {
            int period;
            // Đọc thời khoảng mới từ Console cho đến khi
            // người dùng nhập 0. Các giá trị không hợp lệ
            // sẽ sử dụng giá trị mặc định là 0 (dừng ví dụ).
            do
            {
                try
                {
                    period = Int32.Parse(Console.ReadLine());
                }
                catch { period = 0; }
                // Thay đổi Timer với thời khoảng mới.
                if (period > 0) timer.Change(0, period);
            } while (period > 0);
        }
        // Nhấn Enter để kết thúc.
        Console.WriteLine("Main method complete. Press Enter.");
        Console.ReadLine();
    }
}
```

V.3.4. Thực thi phương thức bằng tiểu trình mới

Thực thi mã lệnh trong một tiểu trình riêng, và muốn kiểm soát hoàn toàn quá trình thực thi và trạng thái của tiểu trình đó.

Để tăng độ linh hoạt và mức độ kiểm soát khi hiện thực các ứng dụng hỗ-trợ-đa-tiểu-trình, bạn phải trực tiếp tạo và quản lý các tiểu trình. Lớp Thread cung cấp một cơ chế mà qua đó bạn có thể tạo và kiểm soát các tiểu trình.

Các bước thực hiện:

Tạo một đối tượng ủy nhiệm ThreadStart tham chiếu đến phương thức chứa mã lệnh mà muốn dùng một tiểu trình mới để chạy nó. Giống như các ủy nhiệm khác, ThreadStart có thể tham chiếu đến một phương thức tĩnh hay phương thức của một đối tượng. Phương thức được tham chiếu phải trả về void và không có đối số.

Tạo một đối tượng Thread, và truyền thể hiện ủy nhiệm ThreadStart cho phương thức khởi dựng của nó. Tiểu trình mới có trạng thái ban đầu là Unstarted (một thành viên thuộc kiểu liệt kê System.Threading.ThreadState).

Gọi thực thi phương thức Start của đối tượng Thread để chuyển trạng thái của nó sang ThreadState.Running và bắt đầu thực thi phương thức được tham chiếu bởi thể hiện ủy nhiệm ThreadStart.

Vì ủy nhiệm ThreadStart khai báo không có đối số, nên không thể truyền dữ liệu trực tiếp cho phương thức được tham chiếu. Để truyền dữ liệu cho tiểu trình mới, cần phải cấu hình dữ liệu là khả truy xuất đối với mã lệnh đang chạy trong tiểu trình mới.

Cách tiếp cận thông thường là tạo một lớp đóng gói cả dữ liệu cần cho tiểu trình và phương thức được thực thi bởi tiểu trình. Khi muốn chạy một tiểu trình mới, thì tạo một đối tượng của lớp này, cấu hình trạng thái cho nó, và rồi chạy tiểu trình.

Ví dụ

```
using System;
using System.Threading;
public class ThreadExample
{
    // Các biến giữ thông tin trạng thái.
    private int iterations;
    private string message;
    private int delay;
    public ThreadExample(int iterations, string message, int delay)
    {
        this.iterations = iterations;
        this.message = message;
        this.delay = delay;
    }
    public void Start()
    {
        // Tạo một thể hiện ủy nhiệm ThreadStart
        // tham chiếu đến DisplayMessage.
        ThreadStart method = new ThreadStart(this.DisplayMessage);
        // Tạo một đối tượng Thread và truyền thể hiện ủy nhiệm
        // ThreadStart cho phương thức khởi dựng của nó.
        Thread thread = new Thread(method);
        Console.WriteLine("{0} : Starting new thread.",
                DateTime.Now.ToString("HH:mm:ss.ffff"));
        // Khởi chạy tiểu trình mới.
        thread.Start();
    }
    private void DisplayMessage()
```

```
    {
        // Hiển thị thông báo ra cửa sổ Console với số lần
        // được chỉ định (iterations), nghỉ giữa mỗi thông báo
        // một khoảng thời gian được chỉ định (delay).
        for (int count = 0; count < iterations; count++)
        {
            Console.WriteLine("{0} : {1}",
DateTime.Now.ToString("HH:mm:ss.ffff"), message);
            Thread.Sleep(delay);
        }
    }
    public static void Main()
    {
        // Tạo một đối tượng ThreadExample.
        ThreadExample example = new
            ThreadExample(5, "A thread example.", 500);
        // Khởi chạy đối tượng ThreadExample.
        example.Start();
        // Tiếp tục thực hiện công việc khác.
        for (int count = 0; count < 13; count++)
        {
            Console.WriteLine("{0} : Continue processing...",
DateTime.Now.ToString("HH:mm:ss.ffff"));
            Thread.Sleep(200);
        }
        // Nhấn Enter để kết thúc.
        Console.WriteLine("Main method complete. Press Enter.");
        Console.ReadLine();
    }
}
```

V.3.5. Điều khiển quá trình thực thi của một tiểu trình

Cần nắm quyền điều khiển khi một tiểu trình chạy và dừng, có thể tạm dừng quá trình thực thi của một tiểu trình

Các phương thức của lớp Thread: Abort, Interrupt, Resume, Start, và Suspend cung cấp một cơ chế điều khiển mức cao lên quá trình thực thi của một tiểu trình. Mỗi phương thức này trở về tiểu trình đang gọi ngay lập tức. Kết quả là phải viết mã để bắt và thụ lý các ngoại lệ có thể bị ném khi ta cố điều khiển quá trình thực thi của một Thread.

Phương Thức	**Mô Tả**
Abort()	- Kết thúc một tiểu trình bằng cách ném ngoại lệ System.Threading. ThreadAbortException trong mã lệnh đang được chạy. - Mã lệnh của tiểu trình bị hủy có thể bắt ngoại lệ ThreadAbortException để thực hiện việc dọn dẹp, nhưng bộ thực thi sẽ tự động ném ngoại lệ này lần nữa để bảo đảm tiểu trình kết thúc, trừ khi ResetAbort được gọi.
Interrupt()	- Ném ngoại lệ - System.Threading.ThreadInterruptedException (trong mã lệnh đang được chạy) lúc tiểu trình đang ở trạng thái WaitSleepJoin. Điều này nghĩa là tiểu trình này đã gọi Sleep, Join hoặc đang đợi WaitHandle - Nếu tiểu trình này không ở trạng thái WaitSleepJoin, ThreadInterruptedException sẽ bị ném sau khi tiểu trình đi vào trạng thái WaitSleepJoin.
Resume	- Phục hồi quá trình thực thi của một tiểu trình đã bị tạm ngưng . - Việc gọi Resume trên một tiểu trình chưa bị tạm hoãn sẽ sinh ra ngoại lệ System.Threading.ThreadStateException trong tiểu trình đang gọi.
Start	Khởi chạy tiểu trình mới;
Suspend	- Tạm hoãn quá trình thực thi của một tiểu trình cho đến khi phương thức Resume được gọi. - Việc tạm hoãn một tiểu trình đã bị tạm hoãn sẽ không có hiệu lực - Việc gọi Suspend trên một tiểu trình chưa khởi chạy hoặc đã kết thúc sẽ sinh ra ngoại lệ ThreadStateException trong tiểu trình đang gọi.

Ví dụ:

Lớp ThreadControlExample:

- ✓ Ví dụ này khởi chạy một tiểu trình thứ hai, hiển thị định kỳ một thông báo ra cửa sổ Console và rồi đi vào trạng thái nghỉ (sleep).
- ✓ Bằng cách nhập các lệnh tại dấu nhắc lệnh, ta có thể gián đoạn, tạm hoãn, phục hồi, và hủy bỏ tiểu trình thứ hai.

Chương trình ThreadControlExample

```
using System;
using System.Threading;
public class ThreadControlExample
{
    private static void DisplayMessage()
    {
        // Lặp đi lặp lại việc hiển thị một thông báo ra cửa sổ Console.
        while (true)
        {
            try
            {
                Console.WriteLine("{0} : Second thread running. Enter
(S)uspend, (R)esume, (I)nterrupt, or (E)xit.",
DateTime.Now.ToString("HH:mm:ss.ffff"));
                // Nghỉ 2 giây.
                Thread.Sleep(2000);
            }
            catch (ThreadInterruptedException)
            {
                // Tiểu trình đã bị gián đoạn. Việc bắt ngoại lệ
                // ThreadInterruptedException cho phép ví dụ này
                // thực hiện hành động phù hợp và tiếp tục thực thi.

                Console.WriteLine("{0} : Second thread interrupted.",
DateTime.Now.ToString("HH:mm:ss.ffff"));
            }
            catch (ThreadAbortException abortEx)
            {
                // Đối tượng trong thuộc tính
                // ThreadAbortException.ExceptionState được cung cấp
                // bởi tiểu trình đã gọi Thread.Abort.
                // Trong trường hợp này, nó chứa một chuỗi
                // mô tả lý do của việc hủy bỏ.
                Console.WriteLine("{0} : Second thread aborted ({1})",
DateTime.Now.ToString("HH:mm:ss.ffff"), abortEx.ExceptionState);
```

```
                // Mặc dù ThreadAbortException đã được thụ lý,
                // bộ thực thi sẽ ném nó lần nữa để bảo đảmtiểu trình kết thúc
            }
        }
    }
    public static void Main()
    {
        // Tạo một đối tượng Thread và truyền cho nó một thể hiện
        // ủy nhiệm ThreadStart tham chiếu đến DisplayMessage.
        Thread thread = new Thread(new ThreadStart(DisplayMessage));
        Console.WriteLine("{0} : Starting second thread.",
DateTime.Now.ToString("HH:mm:ss.ffff"));
        // Khởi chạy tiểu trình thứ hai.
        thread.Start();
        // Lặp và xử lý lệnh do người dùng nhập.
        char command = ' ';
        do
        {
            string input = Console.ReadLine();
            if (input.Length > 0)
                command = input.ToUpper()[0];
            else command = ' ';
            switch (command)
            {
                case 'S':
                    // Tạm hoãn tiểu trình thứ hai.

                    Console.WriteLine("{0} : Suspending second thread.",
DateTime.Now.ToString("HH:mm:ss.ffff"));
                    thread.Suspend();
                    break;
                case 'R':
                    // Phục hồi tiểu trình thứ hai.
                    try
                    {
                        Console.WriteLine("{0} : Resuming second thread.",
DateTime.Now.ToString("HH:mm:ss.ffff"));
                        thread.Resume();
                    }
                    catch (ThreadStateException)
                    {
                        Console.WriteLine("{0} : Thread wasn't suspended.",
DateTime.Now.ToString("HH:mm:ss.ffff"));
```

```
                    } break;
                case 'I':
                    // Gián đoạn tiểu trình thứ hai.
                    Console.WriteLine("{0} : Interrupting second thread.",
DateTime.Now.ToString("HH:mm:ss.ffff"));
                    thread.Interrupt();
                    break;
                case 'E':
                    // Hủy bỏ tiểu trình thứ hai và truyền một đối tượng
                    // trạng thái cho tiểu trình đang bị hủy,
                    // trong trường hợp này là một thông báo.
                    Console.WriteLine("{0} : Aborting second thread.",
DateTime.Now.ToString("HH:mm:ss.ffff"));
                    thread.Abort("Terminating example.");
                    // Đợi tiểu trình thứ hai kết thúc.
                    thread.Join();
                    break;
            }
        }
        while (command != 'E');
        // Nhấn Enter để kết thúc.

        Console.WriteLine("Main method complete. Press Enter.");
        Console.ReadLine();
    }
}
```

V.3.6. Nhận biết khi nào một tiểu trình kết thúc

Để kiểm tra một tiểu trình đã kết thúc hay chưa là kiểm tra thuộc tính Thread.IsAlive. Thuộc tính này trả về true nếu tiểu trình đã được khởi chạy nhưng chưa kết thúc hay bị hủy.

Thông thường,cần một tiểu trình để đợi một tiểu trình khác hoàn tất việc xử lý của nó. Thay vì kiểm tra thuộc tính IsAlive trong một vòng lặp, có thể sử dụng phương thức Thread.Join. Phương thức này khiến tiểu trình đang gọi dừng lại (block) cho đến khi tiểu trình được tham chiếu kết thúc.

Có thể tùy chọn chỉ định một khoảng thời gian (giá trị int hay TimeSpan) mà sau khoảng thời gian này, Join sẽ hết hiệu lực và quá trình thực thi của tiểu trình đang

gọi sẽ phục hồi lại. Nếu chỉ định một giá trị time-out, Join trả về true nếu tiểu trình đã kết thúc, và false nếu Join đã hết hiệu lực.

Ví dụ

Ví dụ thực thi một tiểu trình thứ hai và rồi gọi Join để đợi tiểu trình thứ hai kết thúc. Vì tiểu trình thứ hai mất 5 giây để thực thi, nhưng phương thức Join chỉ định giá trị time-out là 3 giây, nên Join sẽ luôn hết hiệu lực và ví dụ này sẽ hiển thị một thông báo ra cửa sổ Console.

Chương trình ThreadFinishExample

```
using System;
using System.Threading;
public class ThreadFinishExample
{
    private static void DisplayMessage()
    {
        // Hiển thị một thông báo ra cửa sổ Console 5 lần.
        for (int count = 0; count < 5; count++)
        {
            Console.WriteLine("{0} : Second thread",
            DateTime.Now.ToString("HH:mm:ss.ffff"));
            // Nghỉ 1 giây.
            Thread.Sleep(1000);
        }
    }
    public static void Main()
    {
        // Tạo một thể hiện ủy nhiệm ThreadStart
        // tham chiếu đến DisplayMessage.
        ThreadStart method = new ThreadStart(DisplayMessage);
        // Tạo một đối tượng Thread và truyền thể hiện ủy nhiệm
        // ThreadStart cho phương thức khởi dựng của nó.
        Thread thread = new Thread(method);
        Console.WriteLine("{0} : Starting second thread.",
DateTime.Now.ToString("HH:mm:ss.ffff"));
        // Khởi chạy tiểu trình thứ hai.
        thread.Start();
        // Dừng cho đến khi tiểu trình thứ hai kết thúc,
        // hoặc Join hết hiệu lực sau 3 giây.
        if (!thread.Join(3000))
        {
```

```
            Console.WriteLine("{0} : Join timed out !!",
DateTime.Now.ToString("HH:mm:ss.ffff"));
        }
        // Nhấn Enter để kết thúc.
        Console.WriteLine("Main method complete. Press Enter.");
        Console.ReadLine();
    }
}
```

V.3.7. Khởi chạy một tiến trình mới

Lớp Process cung cấp một dạng biểu diễn được quản lý cho một tiến trình của hệ điều hành. Lớp Process hiện thực bốn quá tải hàm phương thức Start. Hai trong số này là các phương thức tĩnh, cho phép chỉ định tên và các đối số cho tiến trình mới. Ví dụ, hai lệnh dưới đây đều thực thi Notepad trong một tiến trình mới:

```
// Thực thi notepad.exe, không có đối số.
        Process.Start("notepad.exe");
// Thực thi notepad.exe, tên file cần mở là đối số.
        Process.Start("notepad.exe", "SomeFile.txt");
```

Hai dạng khác của phương thức Start yêu cầu tạo đối tượng ProcessStartInfo được cấu hình với các chi tiết của tiến trình cần chạy.Việc sử dụng đối tượng ProcessStartInfo cung cấp một cơ chế điều khiển tốt hơn trên các hành vi và cấu hình của tiến trình mới.

Tóm tắt một vài thuộc tính thông dụng của lớp ProcessStartInfo:

- ✓ Arguments: Các đối số dùng để truyền cho tiến trình mới
- ✓ ErrorDialog :Nếu Process.Start không thể khởi chạy tiến trình đã được chỉ định, nó sẽ ném ngoại lệ System.ComponentModel.Win32Exception. Nếu ErrorDialog là true, Start sẽ hiển thị một thông báo lỗi trước khi ném ngoại lệ
- ✓ FileName : Tên của ứng dụng.
- ✓ WindowStyle :Điều khiển cách thức hiển thị của cửa sổ. Các giá trị hợp lệ bao gồm: Hidden, Maximized, Minimized, và Normal
- ✓ WorkingDirectory : Tên đầy đủ của thư mục làm việc

Khi đã hoàn tất với một đối tượng Process, bạn nên hủy nó để giải phóng các tài nguyên hệ thống, gọi phương thức Close(), Dispose().

Ví dụ sau sử dụng Process để thực thi Notepad trong một cửa sổ ở trạng thái phóng to và mở một file có tên là C:\Temp\file.txt. Sau khi tạo, ví dụ này sẽ gọi phương thức Process.WaitForExit để dừng tiểu trình đang chạy cho đến khi tiến trình kết thúc hoặc giá trị time-out hết hiệu lực.

Chương trình StartProcessExample

```
using System;
using System.Diagnostics;
public class StartProcessExample
{
    public static void Main()
    {
        // Tạo một đối tượng ProcessStartInfo và cấu hình cho nó
        // với các thông tin cần thiết để chạy tiến trình mới.
        ProcessStartInfo startInfo = new ProcessStartInfo();
        startInfo.FileName = "notepad.exe";
        startInfo.Arguments = "file.txt";
        startInfo.WorkingDirectory = "C:\\Temp";
        startInfo.WindowStyle = ProcessWindowStyle.Maximized;
        startInfo.ErrorDialog = true;
        // Tạo một đối tượng Process mới.
        using (Process process = new Process())
        {
            // Gán ProcessStartInfo vào Process.
            process.StartInfo = startInfo;
            try
            {
                // Khởi chạy tiến trình mới.
                process.Start();
                // Đợi tiến trình mới kết thúc trước khi thoát.
                Console.WriteLine("Waiting 30 seconds for process to finish.");
                process.WaitForExit(30000);
            }
            catch (Exception ex)
            {
                Console.WriteLine("Could not start process.");
                Console.WriteLine(ex);
            }
        }
        // Nhấn Enter để kết thúc.
        Console.WriteLine("Main method complete. Press Enter.");
        Console.ReadLine();
```

```
    }
}
```

V.3.8. Kết thúc một tiến trình

Nếu khởi chạy một tiến trình mới từ mã lệnh được quản lý bằng lớp Process , có thể kết thúc tiến trình mới bằng đối tượng Process mô tả tiến trình này. Một khi đã có đối tượng Process mô tả tiến trình cần kết thúc, cần gọi phương thức CloseMainWindow hay phương thức Kill().

Phương thức CloseMainWindow gửi một thông điệp đến cửa sổ chính của ứng dụng. CloseMainWindow sẽ không kết thúc các ứng dụng không có cửa sổ chính hoặc các ứng dụng có cửa sổ chính bị vô hiệu. Với những tình huống như thế, CloseMainWindow sẽ trả về false. CloseMainWindow trả về true nếu thông điệp được gửi thành công, nhưng không bảo đảm tiến trình thật sự kết thúc.

Phương thức Kill() kết thúc một tiến trình ngay lập tức; người dùng không có cơ hội dừng việc kết thúc, và tất cả các dữ liệu chưa được lưu sẽ bị mất.

Ví dụ sau khởi chạy một thể hiện mới của Notepad, đợi 5 giây, sau đó kết thúc tiến trình Notepad.

Trước tiên, ví dụ này kết thúc tiến trình bằng CloseMainWindow. Nếu CloseMainWindow trả về false, hoặc tiến trình Notepad vẫn cứ chạy sau khi CloseMainWindow được gọi, ví dụ này sẽ gọi Kill() và buộc tiến trình Notepad kết thúc. Có thể buộc CloseMainWindow trả về false bằng cách bỏ mặc hộp thoại File Open mở.

Chương trình TerminateProcessExample

```
using System;
using System.Threading;
using System.Diagnostics;
public class TerminateProcessExample
{
    public static void Main()
    {
        // Tạo một Process mới và chạy notepad.exe.
        using (Process process = Process.Start("notepad.exe"))
        {
            // Đợi 5 giây và kết thúc tiến trình Notepad.
            Thread.Sleep(5000);
```

```
            // Kết thúc tiến trình Notepad.
            // Gửi một thông điệp đến cửa sổ chính.
            if (!process.CloseMainWindow())
            {
                // Không gửi được thông điệp. Kết thúc Notepad bằng Kill.

                process.Kill();
            }
            else
            {
                // Thông điệp được gửi thành công; đợi 2 giây
                // để chứng thực việc kết thúc trước khi viện đến Kill.
                if (!process.WaitForExit(2000))
                {
                    process.Kill();

                }

                // Nhấn Enter để kết thúc.
                Console.WriteLine("Main method complete. Press Enter.");
                Console.ReadLine();
            }


        }
    }
}
```

V.4. Thực thi phương thức bằng cách ra hiệu đối tượng WaitHandle

Sử dụng các lớp dẫn xuất từ WaitHandle để gọi thực thi một phương thức. Bằng phương thức RegisterWaitForSingleObject của lớp ThreadPool, có thể đăng ký thể hiện ủy nhiệm WaitOrTimerCallback với thread-pool khi một đối tượng dẫn xuất từ WaitHandle đi vào trạng thái signaled.

Có thể cấu hình thread-pool để thực thi phương thức chỉ một lần hay tự động đăng ký lại phương thức mỗi khi WaitHandle đi vào trạng thái signaled. Nếu WaitHandle đã ở trạng thái signaled khi gọi RegisterWaitForSingleObject, phương thức sẽ thực thi ngay lập tức. Phương thức Unregister của đối tượng System.Threading.RegisteredWaitHandle (được trả về bởi phương thức RegisterWaitForSingleObject) được sử dụng để hủy bỏ việc đăng ký.

Lớp thường được dùng làm bộ kích hoạt là AutoResetEvent, nó sẽ tự động chuyển sang trạng thái unsignaled sau khi ở trạng thái signaled. Tuy nhiên, cũng có thể thay đổi trạng thái signaled theo ý muốn bằng lớp ManualResetEvent hay Mutex.

CHƯƠNG VI: ĐỒNG BỘ HÓA

VI.1. Lý do đồng bộ hóa

Trong các hệ điều hành đa nhiệm cho phép nhiều công việc được thực hiện đồng thời. Việc tồn tại cùng lúc nhiều tiểu trình trong môi trường có thể dẫn đến sự tranh chấp, ngăn cản hoạt động lẫn nhau giữa các tiểu trình.

Ví dụ : Với một tiến trình mới đầu tạo lập 4 tiểu trình cùng có nội dung xử lý đồng nhất.: Mỗi tiểu trình sẽ tăng giá trị của biến toàncục Count lên 250.000 lần . Do vậy Count sẽ tăng lên 1.000.000 lần trong tòan bộ tiến trình.

Để hạn chế các trình trạng tranh chấp tài nguyên cần có cơ chế điều khiển các tiểu trình truy xuất tài nguyên một cách tuần tự . đó chính là thực hiện đồng bộ hóa các tiểu trình.

Các trường hợp cần thực hiện đồng bộ hóa.

Khi một tiểu trình truy xuất đến một tài nguyên dùng chung , cần chú ý thực hiện đồng bộ hóa việc truy xuất tài nguyên để tránh xảy ra tranh chấp. Các cơ chế đồng bộ hóa đều dựa trên ý tưởng chỉ cho phép một tiểu trình được truy xuất tài nguyên khi thỏa điều kiện không có tranh chấp trên đó. Những tiểu trình không hội đủ điều kiện để sử dụng tài nguyên thì được hệ điều hành đặt vào trạng thái chờ (Không chiếm CPU), và hệ điều hành sẽ luôn kiểm sóat tình trạng truy xuất tài nguyên của tiểu trình khác để có thể giải phóng kịp thời các tiểu trình đang chờ vào thời điểm thích hợp.

VI.2. Các phương pháp đồng bộ hóa

Để thực hiện được cơ chế đồng bộ hóa , hệ điều hành sử dụng các đối tượng đồng bộ hóa gắn liền với các tài nguyên để phản ánh tình trạng truy xuất trên các tài nguyên.

Các đối tượng đồng bộ hóa có thể nhận trạng thái TRUE hoặc FALSE. Khi đối tượng đồng bộ hóa ở trạng thái TRUE tiểu trình được phép truy cập tài nguyên, ngược lại thì không.

VI.3. Phương pháp Semaphore

Khái niệm: Semaphore là một đối tượng đồng bộ hóa lưu trữ một biến đếm có giá trị từ 0 đến Max, semaphore nhận trạng thái TRUE khi giá trị của biến đếm > 0 và nhận trạng thái FALSE nếu có giá trị biến đếm =0

Tình huống sử dụng: Sử dụng semaphore để kiểm soát việc cho phép một số hữu hạn tiểu trình cùng lúc truy xuất một tài nguyên dùng chung.

Biến đếm của đối tượng semaphore sẽ cho biết số tiểu trình đang truy xuất tài nguyên mà semaphore bảo vệ, biến đếm sẽ giảm 1 nếu có thêm một tiểu trình truy xuất tài nguyên, ngược lại sẽ tăng giá trị lên 1 nếu có một tiểu trình chấm dứt truy xuất tài nguyên. Khi biến đếm đạt giá trị 0 tài nguyên được bảo vệ, không tiểu trình nào ngòai Max tiểu trình đã đã đăng ký được truy xuất. Có thể dùng semaphore để đồng bộ hóa các tiểu trình trong cùng hoặc khác tiến trình.

Cách tạo đối tượng Semaphore

Class Semaphore

Semaphore(int InitCount, int MaxCount)

Đăng ký truy cập tài nguyên chung

Phương thức WaitOne( )

Kết thúc truy cập tài nguyên chung

Phương thức Release( )

Ví dụ: Tạo ra 10 tiểu trình nhưng tại một thời điểm chỉ có 3 tiểu trình truy cập tài nguyên chung:

```
class SemaphoreTest
{
    static Semaphore s = new Semaphore(3, 3);
    // Available=3; Capacity=3
    static void Main()
    {
        for (int i = 0; i < 10; i++)
        {
            Thead thread = new Thread(new ThreadStart(Go));
            thread.Start();
        }
    }
```

```
    static void Go()
    {
        while (true)
        {
            s.WaitOne();
            Thread.Sleep(100);
            // Only 3 threads can get here at once
            s.Release();
        }
    }
}
```

VI.4. Phương pháp dùng lớp Monitor

Một cơ chế đồng bộ hóa khác là lớp Monitor. Lớp này cho phép một tiểu trình đơn thu lấy khóa (lock) trên một đối tượng bằng cách gọi phương thức tĩnh Monitor.Enter. Bằng cách thu lấy khóa trước khi truy xuất một tài nguyên hay dữ liệu dùng chung, ta chắc chắn rằng chỉ có một tiểu trình có thể truy xuất tài nguyên đó cùng lúc. Một khi đã hoàn tất với tài nguyên, tiểu trình này sẽ giải phóng khóa để tiểu trình khác có thể truy xuất nó bằng phương thức tĩnh Monitor.Exit.

Khối mã được gói trong lệnh lock tương đương với gọi Monitor.Enter khi đi vào khối mã này, và gọi Monitor.Exit khi đi ra khối mã này. Tiểu trình chủ có thể gọi Monitor.Wait để giải phóng lock và đặt tiểu trình này vào hàng chờ (wait queue).

Các tiểu trình trong hàng chờ cũng có trạng thái là WaitSleepJoin và sẽ tiếp tục block cho đến khi tiểu trình chủ gọi phương thức Pulse hay PulseAll của lớp Monitor. Phương thức Pulse di chuyển một trong các tiểu trình từ hàng chờ vào hàng sẵn sàng, còn phương thức PulseAll thì di chuyển tất cả các tiểu trình. Khi một tiểu trình đã được di chuyển từ hàng chờ vào hàng sẵn sàng, nó có thể thu lấy lock trong lần giải phóng kế tiếp.

Lớp ThreadSyncExample trình bày cách sử dụng lớp Monitor và lệnh lock.

Ví dụ này khởi chạy ba tiểu trình, mỗi tiểu trình (lần lượt) thu lấy lock của một đối tượng có tên là consoleGate. Kế đó, mỗi tiểu trình gọi phương thức Monitor.Wait. Khi người dùng nhấn Enter lần đầu tiên, Monitor.Pulse sẽ được gọi để giải phóng một tiểu trình đang chờ. Lần thứ hai người dùng nhấn Enter, Monitor.PulseAll sẽ được gọi để giải phóng tất cả các tiểu trình đang chờ còn lại.

Chương trình ThreadSyncExample

```
using System;
using System.Threading;
public class ThreadSyncExample
{
    private static object consoleGate = new Object();
    private static void DisplayMessage()
    {
        Console.WriteLine("{0} : Thread started, acquiring lock...",
DateTime.Now.ToString("HH:mm:ss.ffff"));
        // Thu lấy chốt trên đối tượng consoleGate.
        try
        {
            Monitor.Enter(consoleGate);
            Console.WriteLine("{0} : {1}",
            DateTime.Now.ToString("HH:mm:ss.ffff"), "Acquired consoleGate lock,
waiting...");
            // Đợi cho đến khi Pulse được gọi trên đối tượng consoleGate.
            Monitor.Wait(consoleGate);
            Console.WriteLine("{0} : Thread pulsed, terminating.",
DateTime.Now.ToString("HH:mm:ss.ffff"));
        }
        finally
        {
            Monitor.Exit(consoleGate);
        }
    }
    public static void Main()
    {
        // Thu lấy chốt trên đối tượng consoleGate.
        lock (consoleGate)
        {
            // Tạo và khởi chạy ba tiểu trình mới
            // (chạy phương thức DisplayMesssage).
            for (int count = 0; count < 3; count++)
            {
                (new Thread(new ThreadStart(DisplayMessage))).Start();
            }
        }
        Thread.Sleep(1000);
        // Đánh thức một tiểu trình đang chờ.
        Console.WriteLine("{0} : {1}", DateTime.Now.ToString("HH:mm:ss.ffff"),
"Press Enter to pulse one waiting thread.");
```

```
            Console.ReadLine();
            // Thu lấy chốt trên đối tượng consoleGate.
            lock (consoleGate)
            {
                // Pulse một tiểu trình đang chờ.
                Monitor.Pulse(consoleGate);
            }
            // Đánh thức tất cả các tiểu trình đang chờ.
            Console.WriteLine("{0} : {1}", DateTime.Now.ToString("HH:mm:ss.ffff"),
"Press Enter to pulse all waiting threads.");
            Console.ReadLine();
            // Thu lấy chốt trên đối tượng consoleGate.
            lock (consoleGate)
            {
                // Pulse tất cả các tiểu trình đang chờ.
                Monitor.PulseAll(consoleGate);
            }
            // Nhấn Enter để kết thúc.
            Console.WriteLine("Main method complete. Press Enter.");
            Console.ReadLine();
        }
}
```

VI.5. System.Threading.WaitHandle, bao gồm AutoResetEvent, ManualResetEvent

Các lớp thông dụng khác dùng để đồng bộ hóa tiểu trình là các lớp con của lớp System.Threading.WaitHandle, bao gồm AutoResetEvent, ManualResetEvent. Thể hiện của các lớp này có thể ở trạng thái signaled hay unsignaled.

Các tiểu trình có thể sử dụng các phương thức của các lớp được liệt kê để đi vào trạng thái WaitSleepJoin và đợi trạng thái của một hay nhiều đối tượng dẫn xuất từ WaitHandle biến thành signaled.

Phương Thức	**Mô Tả**
WaitAny()	Tiểu trình gọi phương thức tĩnh này sẽ đi vào trạng thái WaitSleepJoin và đợi bất kỳ một trong các đối tượng WaitHandle thuộc một mảng WaitHandle biến thành *signaled*. Cũng có thể chỉ định giá trị time-out.
WaitAll()	Tiểu trình gọi phương thức tĩnh này sẽ đi vào trạng thái

Phương Thức	**Mô Tả**
	WaitSleepJoin và đợi tất cả các đối tượng WaitHandle trong một mảng WaitHandle biến thành *signaled*. Bạn cũng có thể chỉ định giá trị time-out.
WaitOne()	Tiểu trình gọi phương thức này sẽ đi vào trạng thái WaitSleepJoin và đợi một đối tượng WaitHandle cụ thể biến thành *signaled*.

Điểm khác biệt chính giữa các lớp AutoResetEvent, ManualResetEvent, là cách thức chúng chuyển trạng thái từ signaled thành unsignaled.

Lớp AutoResetEvent và ManualResetEvent là cục bộ đối với một tiến trình. Để ra hiệu một AutoResetEvent, bạn hãy gọi phương thức Set của nó, phương thức này chỉ giải phóng một tiểu trình đang đợi sự kiện. AutoResetEvent sẽ tự động trở về trạng thái unsignaled.

Lớp ManualResetEvent phải được chuyển đổi qua lại giữa signaled và unsignaled bằng phương thức Set và Reset của nó.
Gọi Set trên một ManualResetEvent sẽ đặt trạng thái của nó là signaled, giải phóng tất cả các tiểu trình đang đợi sự kiện. Chỉ khi gọi Reset mới làm cho ManualResetEvent trở thành unsignaled.

Sử dụng các lớp dẫn xuất từ WaitHandle để gọi thực thi một phương thức. Bằng phương thức RegisterWaitForSingleObject của lớp ThreadPool, có thể đăng ký thể hiện ủy nhiệm WaitOrTimerCallback với thread-pool khi một đối tượng dẫn xuất từ WaitHandle đi vào trạng thái signaled.

Có thể cấu hình thread-pool để thực thi phương thức chỉ một lần hay tự động đăng ký lại phương thức mỗi khi WaitHandle đi vào trạng thái signaled.
Nếu WaitHandle đã ở trạng thái signaled khi gọi RegisterWaitForSingleObject, phương thức sẽ thực thi ngay lập tức.

Phương thức Unregister của đối tượng System.Threading.RegisteredWaitHandle (được trả về bởi phương thức RegisterWaitForSingleObject) được sử dụng để hủy bỏ việc đăng ký.

Lớp thường được dùng làm bộ kích hoạt là AutoResetEvent, nó sẽ tự động chuyển sang trạng thái unsignaled sau khi ở trạng thái signaled. Tuy nhiên, chúng ta cũng có thể thay đổi trạng thái signaled theo ý muốn bằng lớp ManualResetEvent hay Mutex.

Ví dụ dưới đây trình bày cách sử dụng một AutoResetEvent để kích hoạt thực thi một phương thức có tên là EventHandler:

Chương trình EventExecutionExample

```
using System.Threading;
public class EventExecutionExample
{
    // Phương thức sẽ được thực thi khi AutoResetEvent đi vào trạng
    // thái signaled hoặc quá trình đợi hết thời gian (time-out).
    private static void EventHandler(object state, bool timedout)
    {
        // Hiển thị thông báo thích hợp ra cửa sổ Console
        // tùy vào quá trình đợi đã hết thời gian hay
        // AutoResetEvent đã ở trạng thái signaled.
        if (timedout)
        {
            Console.WriteLine("{0} : Wait timed out.",
                DateTime.Now.ToString("HH:mm:ss.ffff"));
        }
        else
        {
            Console.WriteLine("{0} : {1}",
            DateTime.Now.ToString("HH:mm:ss.ffff"), state);
        }
    }
    public static void Main()
    {
        // Tạo một AutoResetEvent ở trạng thái unsignaled.
        AutoResetEvent[] autoEvent;
        autoEvent[0] = new AutoResetEvent(false);

        // Tạo một thể hiện ủy nhiệm WaitOrTimerCallback
        // tham chiếu đến phương thức tĩnh EventHandler.
        // EventHandler sẽ được gọi khi AutoResetEvent đi vào
        // trạng thái signaled hay quá trình đợi hết thời gian.
        WaitOrTimerCallback handler = new WaitOrTimerCallback(EventHandler);
        // Tạo đối tượng trạng thái (được truyền cho phương thức
```

```
        // thụ lý sự kiện khi nó được kích hoạt). Trong trường hợp
        // này, một thông báo sẽ được hiển thị.
        string state = "AutoResetEvent signaled.";
        // Đăng ký thể hiện ủy nhiệm để đợi AutoResetEvent đi vào
        // trạng thái signaled. Thiết lập giá trị time-out là 3 giây.
        RegisteredWaitHandle handle =
ThreadPool.RegisterWaitForSingleObject(autoEvent, handler, state, 3000, false);
        Console.WriteLine("Press ENTER to signal the AutoResetEvent or enter
\"Cancel\" to unregister the wait operation.");
        while (Console.ReadLine().ToUpper() != "CANCEL")
        {
            // Nếu "Cancel" không được nhập vào Console,
            // AutoResetEvent sẽ đi vào trạng thái signal,
            // và phương thức EventHandler được thực thi.
            // AutoResetEvent sẽ tự động trở về trạng thái unsignaled.
            autoEvent.Set();
        }
        // Hủy bỏ việc đăng ký quá trình đợi.
        Console.WriteLine("Unregistering wait operation.");
        handle.Unregister(null);
        // Nhấn Enter để kết thúc.
        Console.WriteLine("Main method complete. Press Enter.");
        Console.ReadLine();
    }
}
```

VI.6. Phương pháp Mutex

Mutex cung cấp một cơ chế để đồng bộ hóa quá trình thực thi của các tiểu trình vượt qua biên tiến trình. Một Mutex là signaled khi nó không thuộc sở hữu của bất kỳ tiểu trình nào. Một tiểu trình giành quyền sở hữu Mutex lúc khởi dựng hoặc sử dụng một trong các phương thức được liệt kê ở trên.

Quyền sở hữu Mutex được giải phóng bằng cách gọi phương thức Mutex.ReleaseMutex (ra hiệu Mutex và cho phép một tiểu trình khác thu lấy quyền sở hữu này).

Ví dụ dưới đây sử dụng một Mutex có tên là MutexExample để bảo đảm chỉ một thể hiện của ví dụ có thể thực thi.

Chương trình MutexExample

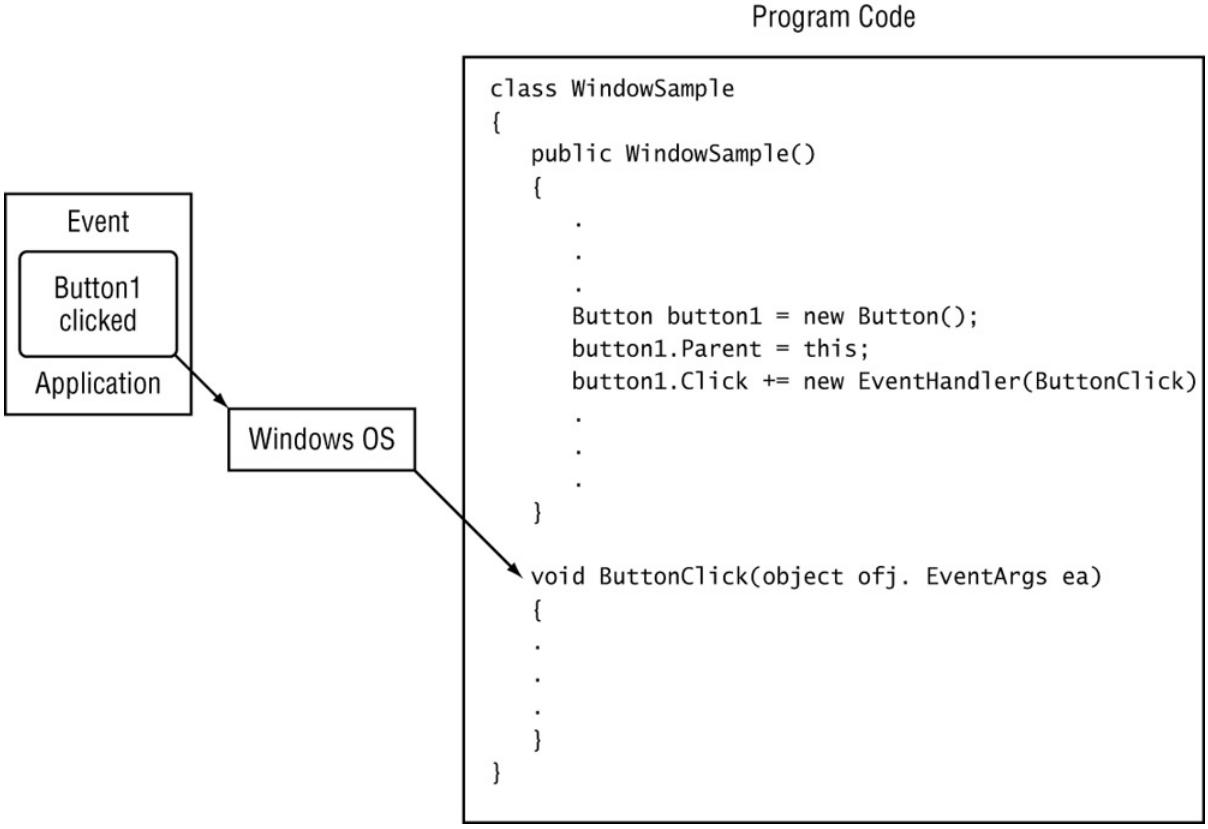
```
using System;
using System.Threading;
public class MutexExample
```

```
{
    public static void Main()
    {
        // Giá trị luận lý cho biết ứng dụng này
        // có quyền sở hữu Mutex hay không.
        bool ownsMutex;
        // Tạo và lấy quyền sở hữu một Mutex có tên là MutexExample.
        using (Mutex mutex = new Mutex(true, "MutexExample", out ownsMutex))
        {
            // Nếu ứng dụng sở hữu Mutex, nó có thể tiếp tục thực thi;
            // nếu không, ứng dụng sẽ thoát.
            if (ownsMutex)
            {
                Console.WriteLine("This application currently owns the mutex
named MutexExample. Additional instances of this application will not run until
you release the mutex by pressing Enter.");
                Console.ReadLine();
                // Giải phóng
                Mutex.mutex.ReleaseMutex();
            }
            else
            {
                Console.WriteLine("Another instance of this" + " application
already owns the mutex named" + " MutexExample. This instance of the" + "
application will terminate.");
            }
        }
        // Nhấn Enter để kết thúc.
        Console.WriteLine("Main method complete. Press Enter.");
        Console.ReadLine();
    }
}
```

CHƯƠNG VII: LẬP TRÌNH SOCKET BẤT ĐỒNG BỘ

VII.1. Lập trình sự kiện trong Windows

Trong lập trình sự kiện trong Windows, mỗi khi một sự kiện xảy ra, một phương thức được gọi để thực thi dựa trên sự kiện đó như trong hình dưới đây:



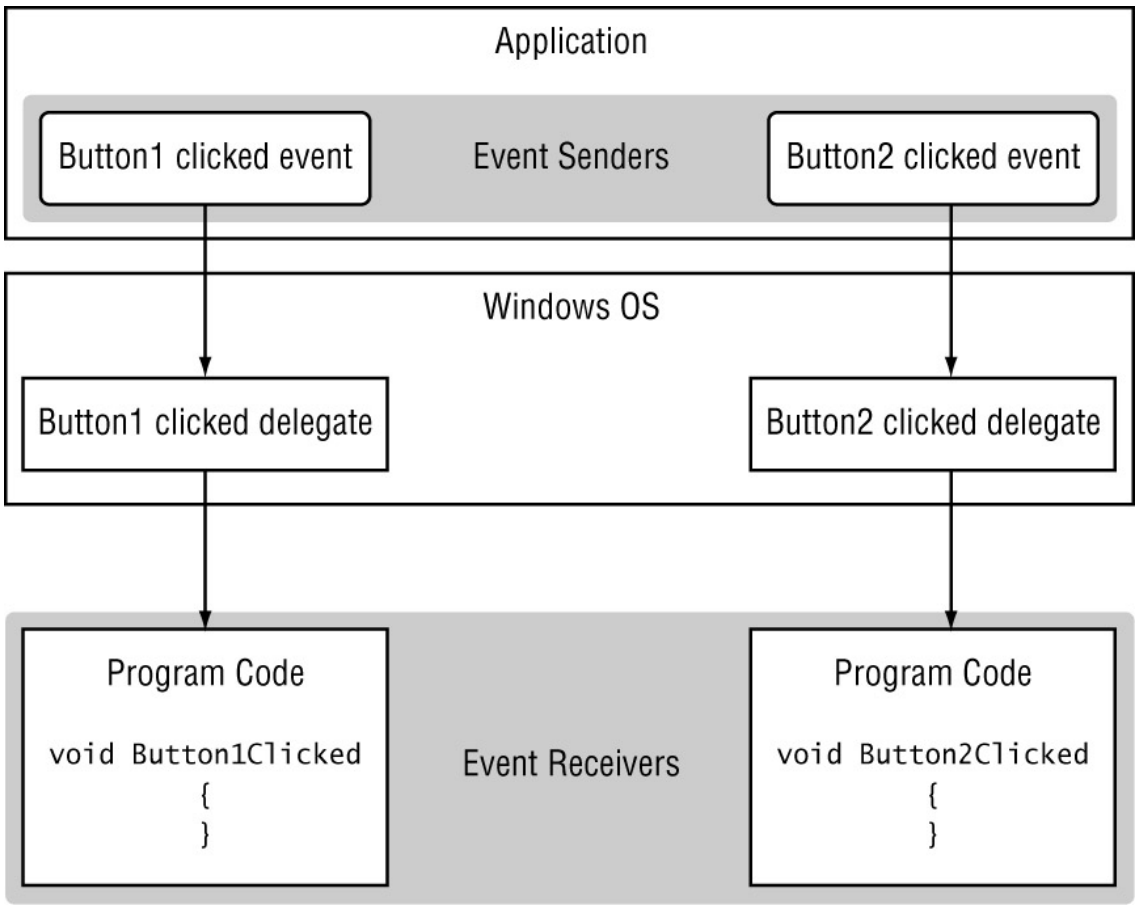
Hình VI.1: Lập trình sự kiện trên Windows

Trong các chương trước, chúng ta đã lập trình Socket trong chế độ blocking. Socket ở chế độ blocking sẽ chờ mãi mãi cho đến khi hoàn thành nhiệm vụ của nó. Trong khi nó bị blocking thì các chức năng khác của chương trình không thực hiện được.

Khi lập trình Windows thì lúc gọi một phương thức bị blocking thì toàn bộ chương trình sẽ đứng lại và không thực hiện các chức năng khác được. Do đó việc lập trình bất đồng bộ là cần thiết để cho chương trình khỏi bị đứnng.

VII.1.1. Sử dụng Event và Delegate

Event là một thông điệp được gởi bởi một đối tượng mô tả một hoạt động mà nó diễn ra. Thông điệp này xác định hoạt động và truyền các dữ liệu cần thiết cho hoạt động. Event có thể là mô tả hoạt động nào đó, chẳng hạn như hoạt động click một Button, hoạt động nhận và gởi dữ liệu trên Socket. Event sender không cần thiết phải biết đối tượng nào sẽ điều khiển thông điệp sự kiện mỗi khi nó được gởi thông qua hệ thống Windows. Nó để cho bộ nhận sự kiện đăng ký với hệ thống Windows và thông báo kiểu sự kiện mà bộ nhận sự kiện muốn nhận như hình minh họa sau:



Hình VI.2: Gởi và nhận sự kiện trong Windows

Bộ nhận sự kiện được xác định trong hệ thống Windows bởi một con trỏ lớp được gọi là delegate. Một delegate là một lớp nó giữ tham chiếu đến một phương thức mà phương thức này điều khiển sự kiện được nhận. Khi hệ thống Windows nhận sự kiện. nó kiểm tra coi thử có delegate nào đăng ký để xử lý nó không. Nếu có delegate đăng ký để xử lý sự kiện, thông điệp sự kiện được truyền vào phương thức được định

nghĩa bởi delegate. Sau khi phương thức hoàn tất, hệ thống Windows sẽ xử lý sự kiện tiếp theo xảy ra cho tới khi sự kiện kết thúc chương trình được phát ra.

Ví dụ đơn giản sau mô tả cách lập trình sự kiện trên Windows Forrm

Chương trình WindowSample

```
using System;
using System.Drawing;
using System.Windows.Forms;
class WindowSample : Form
{
    private TextBox data;
    private ListBox results;
    public WindowSample()
    {
        Text = "Sample Window Program";
        Size = new Size(400, 380);
        Label label1 = new Label();
        label1.Parent = this;
        label1.Text = "Enter text string:";
        label1.AutoSize = true;
        label1.Location = new Point(10, 10);
        data = new TextBox();
        data.Parent = this;
        data.Size = new Size(200, 2 * Font.Height);
        data.Location = new Point(10, 35);
        results = new ListBox();
        results.Parent = this;
        results.Location = new Point(10, 65);
        results.Size = new Size(350, 20 * Font.Height);
        Button checkit = new Button();
        checkit.Parent = this;
        checkit.Text = "test";
        checkit.Location = new Point(235, 32);
        checkit.Size = new Size(7 * Font.Height, 2 * Font.Height);
        checkit.Click += new EventHandler(checkit_OnClick);
    }
    void checkit_OnClick(object obj, EventArgs ea)
    {
        results.Items.Add(data.Text);
        data.Clear();
    }
    public static void Main()
    {
```

```
        Application.Run(new WindowSample());
    }
}
```

Điểm chính trong chương trình này là EvenHandler đăng ký phương thức ButtonOnClick() cho đối tượng Button với sự kiện click:

```
checkit.Click += new EventHandler(checkit_OnClick);
```

Khi người dùng click button, điều khiển chương trình sẽ được chyển đến phương thức ButtonOnClick()

```
void checkit_OnClick(object obj, EventArgs ea)
    {
        results.Items.Add(data.Text);
        data.Clear();
    }
```

VII.1.2. Lớp AsyncCallback trong lập trình Windows

Khi sự kiện kích hoạt delegate, .NET cung cấp một cơ chế để kích hoạt delegate. Lớp AsyncCallback cung cấp các phương thức để bắt đầu một chức năng bất đồng bộ và cung cấp một phương thức delegate để gọi khi chức năng bất đồng bộ kết thúc.

Tiến trình này khác với cách lập trình sự kiện cơ bản, sự kiện này không phát sinh ra từ đối tượng Windows mà nó xuất phát từ một phương thức khác trong chương trình. Phương thức này chính nó đăng ký một delegate AsyncCallback để gọi khi phương thức hoàn tất chức năng của nó.

Socket sử dụng phương thức được định nghĩa trong lớp AsyncCallback để cho phép các chức năng mạng hoạt động một cách bất đồng bộ. Nó phát tín hiệu cho hệ điều hành khi chức năng mạng hoàn tất. Trong môi trường lập trình Windows, những phương thức này giúp cho ứng dụng khỏi bị treo trong khi chờ các chức năng mạng hoàn tất.

VII.2. Sử dụng Socket bất đồng bộ

Đối tượng Socket có nhiều phương thức sử dụng lớp AsyncCallback để gọi các phương thức khác khi các chức năng mạng hoàn tất. Nó cho phép dùng tiếp tục xử lý các sự kiện khác trong khi chờ cho các chức năng mạng hòa thành công việc của nó.

Các phương thức bất đồng bộ của Socket chia các chức năng mạng làm hai phần:

- ✓ Một phương thức Begin bắt đầu các chức năng mạng và đăng ký phương thức AsyncCallback.
- ✓ Một phương thức End hoàn thành chức năng mạng khi phương thức AsyncCallback được gọi.

Bảng sau cho biết các phương thức bất đồng bộ có thể được giữa với Socket. Mỗi phương thức Begin được kết hợp với một phương thức End để hoàn tất chức năng của chúng:

Phương thức bắt đầu	**Mô tả**	**Phương thức kết thúc**
BeginAccept()	Chấp nhận kết nối	EndAccept()
BeginConnect()	Kết nối đến thiết bị ở xa	EndConnect()
BeginReceive()	Nhận dữ liệu từ Socket	EndReceive()
BeginReceiveFrom()	Nhận dữ liệu từ thiết bị ở xa	EndReceiveFrom()
BeginSend()	Gởi dữ liệu từ Socket	EndSend()
BeginSendTo()	Gởi dữ liệu đến thiết bị ở xa	EndSendTo

VII.2.1. Thành lập kết nối

Phương thức được dùng để thành lập kết nối với thiết bị ở xa phụ thuộc vào chương trình đóng vai trò là Server hay Client. Nếu là Server thì phương thức BeginAccept() được dùng, nếu là Client thì phương thức BeginConnect() được dùng.

VII.2.1.1. Phương thức BeginAccept() và EndAccept()

Để chấp nhận kết nối từ thiết bị ở xa, phương thức BeginAccetp() được dùng. Định dạng của nó như sau:

```
IAsyncResult BeginAccept(AsyncCallback callback, object state)
```

Phương thức BeginAccept() nhận hai đối số: tên của phương thức AsyncCallback dùng để kết thúc phương thức và một đối tượng state có thể được dùng để truyền thông tin giữa các phương thức bất đồng bộ. Phương thức này được dùng như sau:

```
Socket sock = new Socket(AddressFamily.InterNetwork,
SocketType.Stream,
 ProtocolType.Tcp);
IPEndPoint iep = new IPEndPoint(IPAddress.Any, 9050);
sock.Bind(iep);
```

```
sock.Listen(5);
sock.BeginAccept(new AsyncCallback(CallAccept), sock);
```

Đoạn code ở trên tạo ra một đối tượng Socket và gán nó đến một địa chỉ IP cục bộ và một port TCP cục bộ để chấp nhận kết nối. Phương thức BeginAccept() định nghĩa delegate được dùng khi có kết nối trên Socket. Tham số cuối cùng được truyền vào phương thức BeginAccept() là Socket ban đầu được tạo ra.

Sau khi phương thức BeginAccept() kết thúc, phương thức AsyncCallback định nghĩa sẽ được gọi khi kết nối xảy ra. Phương thức AsyncCallback phải bao gồm phương thức EndAccept() để kết thúc việc chấp nhận Socket. Sau đây là định dạng của phương thức EndAccept():

```
Socket EndAccept(IAsyncResult iar);
```

Đối tượng IasyncResult truyền giá trị bất đồng bộ từ phương thức BeginAccept() đến phương thức EndAccept(). Cũng giống như phương thức đồng bộ Accept(), phương thức EndAccept() trả về một đối tượng Socket được dùng để kết nối với Client. Tất cả các thông tin liên lạc vói thiết bị ở xa đều được thực hiện thông qua đối tượng Socket này.

```
private static void CallAccept(IAsyncResult iar)
{
  Socket server = (Socket)iar.AsyncState;
  Socket client = server.EndAccept(iar);
  .
  .
  .
}
```

Tên của phương thức AsyncCallback phải giống với tên của phương thức được dùng làm tham số của phương thức BeginAccept(). Bước đầu tiên trong phương thức là nhận Socket ban đầu của Server. Việc này được thực hiện bằng cách dùng property AsyncState của lớp IasyncResult. Property này có kiểu là object do đó nó phải được ép kiểu sang một đối tượng Socket.

Sau khi Socket ban đầu được lấy về, phương thức EndAccept() có thể lấy được đối tượng Socket mới để truyền thông với thiết bị ở xa. Đối số truyền vào của của phương thức EndAccept() phải giống với đối số truyền vào phương thức AsyncCallback.

Đối tượng Client Socket sau khi được tạo ra, nó có thể được dùng giống như bất kỳ đối tượng Socket nào khác, nó có thể được sử dụng các phương thức đồng bộ hoặc bất đồng bộ để gởi và nhận dữ liệu.

VII.2.1.2. Phương thức BeginConnect() và EndConnect()

Để ứng dụng Client kết nối đến Server ở xa bằng phương thức bất đồng bộ ta phải dùng phương thức BeginConnect(). Định dạng của phương thức này nhu sau:

```
IAsyncResult BeginConnect(EndPoint ep, AsyncCallback callback,
Object state)
```

Tham số truyền vào phương thức BeginConnect() là một EndPoint của thiết bị ở xa cần kết nối đến. Giống như phương thức BeginAccept(), phương thức BeginConnect() cũng chỉ ra tên phương thức do delegate AsyncCallback tham chiếu đến và phương thức này được gọi khi kết nối hoàn tất. Tham số cuối cùng là một đối tượng state, nó có thể được truyền vào phương thức EndConnect() để truyền tải các dữ liệu cần thiết.

Đây là đoạn code ví dụ của phương thức BeginConnect():

```
Socket newsock = new Socket(AddressFamily.InterNetwork,
SocketType.Stream,
ProtocolType.Tcp);
IPEndPoint iep = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 9050);
newsock.BeginConnect(iep, new AsyncCallback(Connected), newsock);
```

Đoạn code này tạo ra một đối tượng Socket newsock và một đối tượng IPEndPoint iep cho thiết bị ở xa. Phương thức BeginConnect() tham chiếu đến phương thức AsyncCallback (Connect) và truyền đối tượng Socket ban đầu newsock đến phương thức AsyncCallback.

Khi kết nối được thành lập phương thức do delegate AsyncCallback tham chiếu tới được gọi. Phương thức này dùng phương thức EndConnect() để hoàn thành việc kết nối. Định dạng của phương thức EndConnect() như sau:

```
EndConnect(IAsyncResult iar)
```

Đối tượng IasyncResult truyền giá trị từ phương thức BeginConnect(). Sau đây là ví dụ cách dùng phương thức này:

```
public static void Connected(IAsyncResult iar)
{
```

```
  Socket sock = (Socket)iar.AsyncState;
  try
  {
   sock.EndConnect(iar);
  } catch (SocketException)
  {
   Console.WriteLine("Unable to connect to host");
  }
}
```

Đầu tiên ta lấy Socket ban đầu được sử dụng bởi phương thức BeginConnect(), Socket này lấy được là nhờ vào thuộc tính object của đối tượng IAsyncResult được truyền vào trong phương thức tham chiếu bởi delegate AsyncCallback.

Sau khi Socket ban đầu được tạo ra, phương thức EndConnect() được gọi, đối số truyền vào phương thức EndConnect() là một đối tượng IAsyncResult chỏ ngược trở lại phương thức BeginConnect() ban đầu. Bởi vì thiết bị ở xa có thể kết nối được và cũng có thể không nên tốt nhất là đặt nó vào trong khối try – catch.

VII.2.2. Gởi dữ liệu

VII.2.2.1. Phương thức BeginSend() và phương thức EndSend()

Phương thức BeginSend() cho phép gởi dữ liệu đến một Socket đã được kết nối. Định dạng của phương thức này như sau:

```
IAsyncResult BeginSend(byte[] buffer, int offset, int size,
 SocketFlags sockflag, AsyncCallback callback, object state)
```

Hầu hết các đối số của phương thức này giống như các đối số của phương thức đồng bộ Send() chỉ có thêm hai đối số là AsyncCallback và object.

- ✓ Đối số AsyncCallback: xác định phương thức được gọi khi phương thức BeginSend() thực hiện thành công.
- ✓ Đối số object: gởi thông tin tình trạng đến phương thức EndSend()

Sau đây là một ví dụ của phương thức BeginSend():

```
sock.BeginSend(data, 0, data.Length, SocketFlags.None,
 new AsyncCallback(SendData), sock);
```

Trong ví dụ này toàn bộ dữ liệu trong bộ đệm data được gởi đi và phương thức SendData được gọi khi Socket sẵng sàng gởi dữ liệu. Đối tượng Socket sock sẽ được truyền đến phương thức do delegate AsyncCallback tham chiếu tới.

Phương thức EndSend() sẽ hoàn tất việc gởi dữ liệu. Định dạng của phương thức này như sau:

```
int EndSend(IAsyncResult iar)
```

Phương thức EndSend() trả về số byte được gởi thành công thừ Socket. Sau đây là một ví dụ của phương thức EndSend():

```
private static void SendData(IAsyncResult iar)
{
  Socket server = (Socket)iar.AsyncState;
  int sent = server.EndSend(iar);
}
```

Socket ban đầu được tạo lại bằng cách dùng thuộc tính AsyncState của đối tượng IAsyncResult

VII.2.2.2. Phương thức BeginSendTo() và EndSendTo()

Phương thức BeginSendTo() được dùng với Socket phi kết nối để bắt đầu truyền tải dữ liệu bất đồng bộ tói thiết bị ở xa. Định dạng của phương thức BeginSendTo() như sau:

```
IAsyncResult BeginSendTo(byte[] buffer, int offset, int size,
 SocketFlags sockflag, EndPoint ep, AsyncCallback callback, object
state)
```

Các đối số của phương thức BeginSendTo() cũng giống như các đối số của phương thức SendTo().

Sau đây là một ví dụ của phương thức BeginSendTo():

```
     IPEndPoint iep = new IPEndPoint(IPAddress.Parse("192.168.1.6"),
9050);
sock.BeginSendTo(data, 0, data.Length, SocketFlags.None, iep,
new AsynCallback(SendDataTo), sock);
```

Phương thức SendDataTo() do delegate AsyncCallback tham chiếu đến được truyền vào làm đối số của phương thức BeginSendTo(). Phương thức này sẽ được thực thi khi dữ liệu bắt đầu được gởi đi từ Socket. Phương thức EndSendTo() sẽ được thực thi khi quá trình gởi dữ liệu kết thúc. Định dạng của phương thức này như sau:

```
int EndSendTo(IAsyncResult iar)
```

Đối số truyền vào của phương thức EndSendTo() là một đối tượng IAsyncResult, đối tượng này mang giá trị được truyền từ phương thức BeginSendTo(). Khi quá trình gởi dữ liệu bất đồng bộ kết thúc thì phương thức EndSendTo() sẽ trả về số byte mà nó thực sự gởi được.

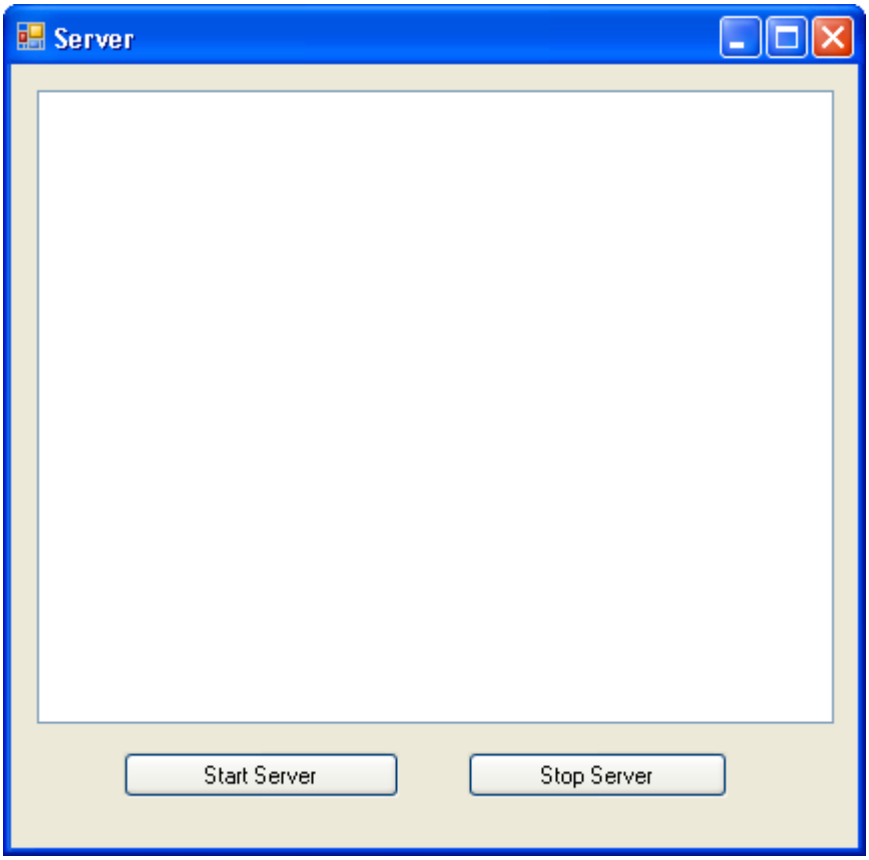
VII.2.3. Nhận dữ liệu

VII.2.3.1. Phương thức BeginReceive(), EndReceive, BeginReceiveFrom(), EndReceiveFrom()

Cách sử dụng của các phương thức này cũng tương tự như cách sử dụng của các phương thức: BeginSend(), EndSend(), BeginSendTo() và EndSendTo()

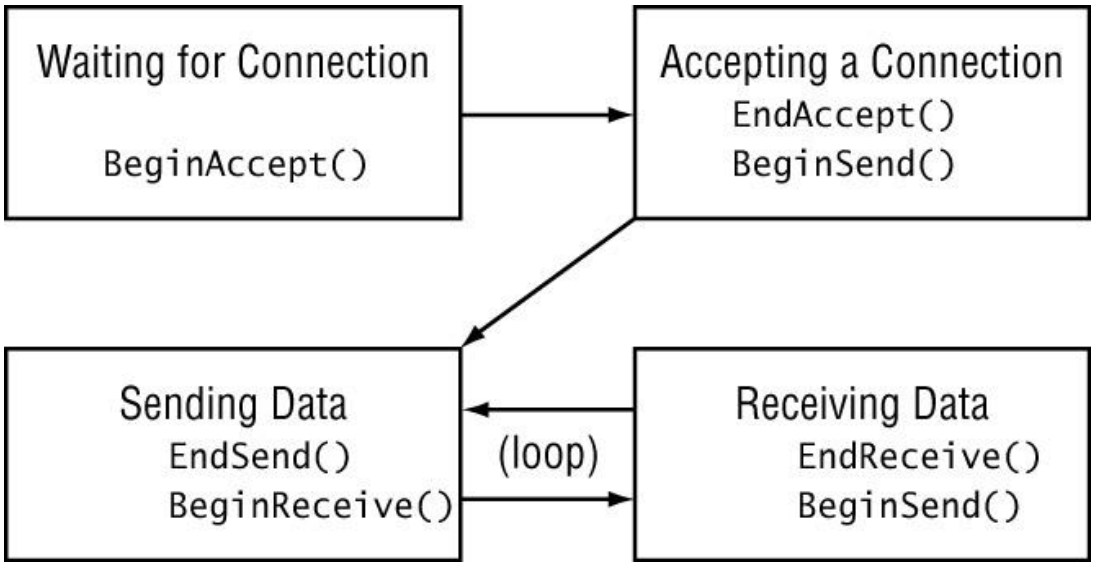
VII.2.4. Chương trình WinForm gởi và nhận dữ liệu giữa Client và Server

VII.2.4.1. Chương trình Server



Hình VI.3: Giao diện Server

VII.2.4.2. Mô hình chương trình Server



Hình VI.4: Mô hình chương trình Server

VII.2.4.3. Lớp ServerProgram

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Net;
using System.Net.Sockets;
namespace Server
{
    class ServerProgram
    {
        private IPAddress serverIP;
        public IPAddress ServerIP
        {
            get
            {
                return serverIP;
            }
            set
            {
                this.serverIP = value;
            }
        }
        private int port;
        public int Port
        {
            get
```

```
            {
                return this.port;
            }
            set
            {
                this.port = value;
            }
        }
        //delegate để set dữ liệu cho các Control
        //Tại thời điểm này ta chưa biết dữ liệu sẽ được hiển thị vào đâu nên
ta phải dùng delegate
        public delegate void SetDataControl(string Data);
        public SetDataControl SetDataFunction = null;
        Socket serverSocket = null ;
        IPEndPoint serverEP = null;
        Socket clientSocket = null;
        //buffer để nhận và gởi dữ liệu
        byte[] buff = new byte[1024];
        //Số byte thực sự nhận được
        int byteReceive = 0;
        string stringReceive = "";

        public ServerProgram(IPAddress ServerIP, int Port)
        {
            this.ServerIP = ServerIP;
            this.Port = Port;
        }
        //Lắng nghe kết nối
        public void Listen()
        {
            serverSocket = new Socket(AddressFamily.InterNetwork,
SocketType.Stream, ProtocolType.Tcp);
            serverEP = new IPEndPoint(ServerIP, Port);
            //Kết hợp Server Socket với Local Endpoint
            serverSocket.Bind(serverEP);
            //Lắng nghe kết nối trên Server Socket
            //-1: không giới hạn số lượng client kết nối đến
            serverSocket.Listen(-1);
            SetDataFunction("Dang cho ket noi");
            //Bắt đầu chấp nhận Client kết nối đến
            serverSocket.BeginAccept(new AsyncCallback(AcceptScoket),
serverSocket);
        }
```

```
        //Hàm callback chấp nhận Client kết nối
        private void AcceptScoket(IAsyncResult ia)
        {
            Socket s = (Socket)ia.AsyncState;
            //Hàm Accept sẽ block server lại và chờ Client kết nối đến
            //Sau khi Client kết nối đến sẽ trả về socket chứa thông tin của
Client
            clientSocket = s.EndAccept(ia);
            string hello = "Hello Client";
            buff = Encoding.ASCII.GetBytes(hello);
            SetDataFunction("Client " + clientSocket.RemoteEndPoint.ToString()
+ "da ket noi den");
            clientSocket.BeginSend(buff, 0, buff.Length, SocketFlags.None, new
AsyncCallback(SendData), clientSocket);
        }
        private void SendData(IAsyncResult ia)
        {
            Socket s = (Socket)ia.AsyncState;
            s.EndSend(ia);
            //khởi tạo lại buffer để nhận dữ liệu
            buff = new byte[1024];
            //Bắt đầu nhận dữ liệu
            s.BeginReceive(buff, 0, buff.Length, SocketFlags.None, new
AsyncCallback(ReceiveData), s);
        }
        public void Close()
        {
            clientSocket.Close();
            serverSocket.Close();
        }

        private void ReceiveData(IAsyncResult ia)
        {
            Socket s = (Socket)ia.AsyncState;
            try
            {
                //Hàm EmdReceive sẽ bị block cho đến khi có dữ liệu trong TCP
buffer
                byteReceive = s.EndReceive(ia);
            }
            catch
            {
                //Trường hợp lỗi xảy ra khi Client ngắt kết nối
```

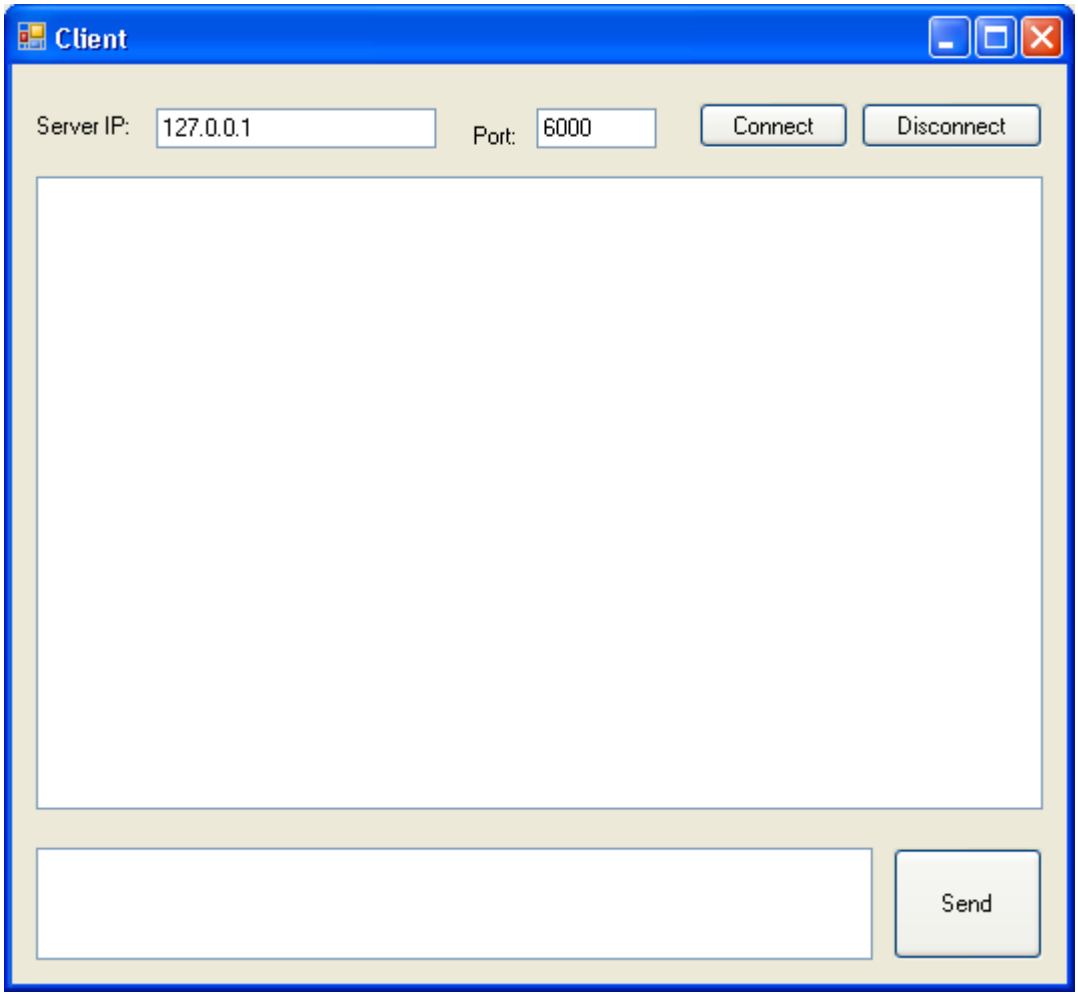
```
                this.Close();
                SetDataFunction("Client ngat ket noi");
                this.Listen();
                return;
            }

            //Nếu Client shutdown thì hàm EndReceive sẽ trả về 0
            if (byteReceive == 0)
            {
                Close();
                SetDataFunction("Clien dong ket noi");
            }
            else
            {
                stringReceive = Encoding.ASCII.GetString(buff, 0, byteReceive);
                SetDataFunction(stringReceive);

                //Sau khi Server nhận dữ liệu xong thì bắt đầu gởi dữ liệu
xuống cho Client
                s.BeginSend(buff, 0, buff.Length, SocketFlags.None, new
AsyncCallback(SendData), s);
            }
        }
    }
}
```

VII.2.4.4. Lớp ServerForm

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Net;
using System.Net.Sockets;

namespace Server
{
    public partial class ServerForm : Form
    {
        ServerProgram server = new ServerProgram(IPAddress.Any, 6000);
```

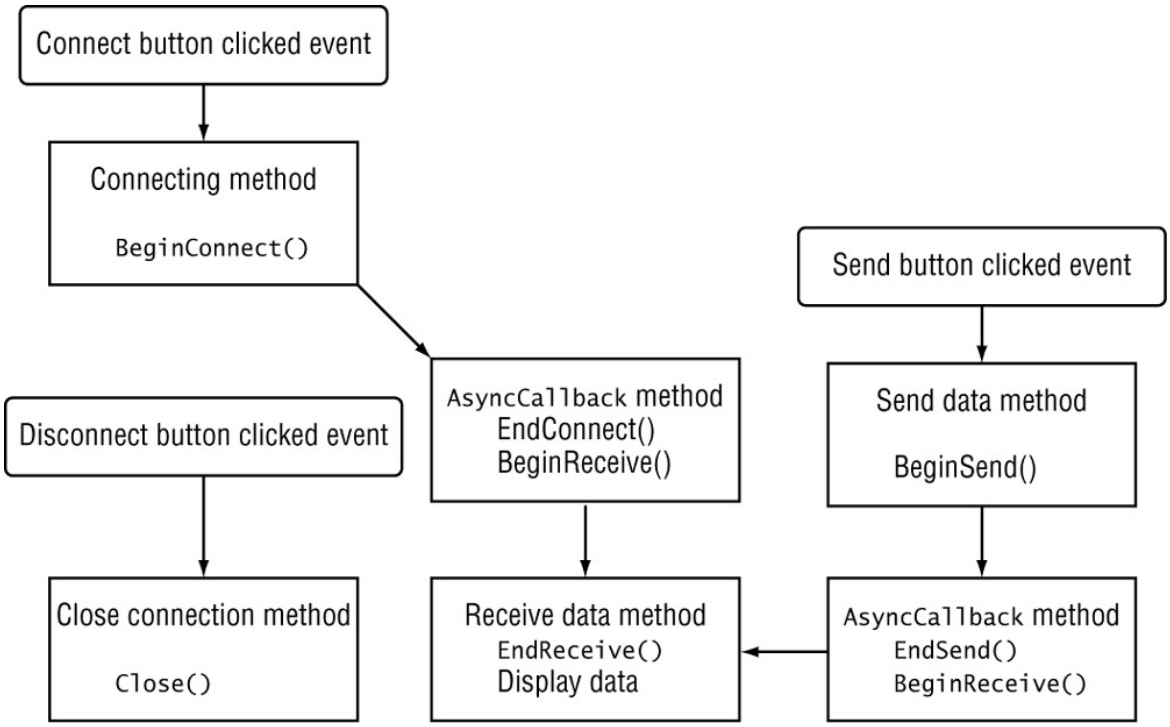
```
        public ServerForm()
        {
            InitializeComponent();
            CheckForIllegalCrossThreadCalls = false;
            server.SetDataFunction = new ServerProgram.SetDataControl(SetData);
        }

        private void SetData(string Data)
        {
            this.listBox1.Items.Add(Data);
        }
        private void cmdStart_Click(object sender, EventArgs e)
        {
            server.Listen();
        }
        private void cmdStop_Click(object sender, EventArgs e)
        {
            this.server.Close();
            SetData("Server dong ket noi");
        }
        private void ServerForm_Load(object sender, EventArgs e)
        {

        }
    }
}
```

VII.2.5. Chương trình Client



Hình VI.5: Giao diện Client

VII.2.5.1. Mô hình chương trình Client



Hình VI.6: Mô hình chương trình Client

VII.2.5.2. Lớp ClientProgram

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Net;
using System.Net.Sockets;
namespace Client
{
    class ClientProgram
    {
        //delegate để set dữ liệu cho các Control
        //Tại thời điểm này ta chưa biết dữ liệu sẽ được hiển thị vào đâu nên
ta phải dùng delegate
        public delegate void SetDataControl(string Data);
        public SetDataControl SetDataFunction = null;

        //buffer để nhận và gởi dữ liệu
        byte[] buff = new byte[1024];
        //Số byte thực sự nhận được
        int byteReceive = 0;
        //Chuỗi nhận được
        string stringReceive = "";
```

```
        Socket serverSocket = new Socket(AddressFamily.InterNetwork,
SocketType.Stream, ProtocolType.Tcp);
        IPEndPoint serverEP = null;
        //Lắng nghe kết nối
        public void Connect(IPAddress serverIP, int Port)
        {
            serverEP = new IPEndPoint(serverIP, Port);
            //Việc kết nối có thể mất nhiều thời gian nên phải thực hiện bất
đồng bộ
            serverSocket.BeginConnect( serverEP, new
AsyncCallback(ConnectCallback), serverSocket);
        }
        //Hàm callback chấp nhận Client kết nối
        private void ConnectCallback(IAsyncResult ia)
        {
            //Lấy Socket đang thực hiện việc kết nối bất đồng bộ
            Socket s = (Socket)ia.AsyncState;
            try
            {
                //Set dữ liệu cho Control
                SetDataFunction("Đang chờ kết nối");
                //Hàm EndConnect sẽ bị block cho đến khi kết nối thành công
                s.EndConnect(ia);
                SetDataFunction("Kết nối thành công");
            }
            catch
            {
                SetDataFunction("Kết nối thất bại");
                return;
            }
            //Ngay sau khi kết nối xong bắt đầu nhận câu chào từ Server gởi
xuống
            s.BeginReceive(buff, 0, buff.Length, SocketFlags.None, new
AsyncCallback(ReceiveData), s);
        }

        private void ReceiveData(IAsyncResult ia)
        {
            Socket s = (Socket)ia.AsyncState;
            byteReceive = s.EndReceive(ia);
            stringReceive = Encoding.ASCII.GetString(buff, 0, byteReceive);
            SetDataFunction(stringReceive);
        }
```

```
        private void SendData(IAsyncResult ia)
        {
            Socket s = (Socket)ia.AsyncState;
            s.EndSend(ia);
            //khởi tạo lại buffer để nhận dữ liệu
            buff = new byte[1024];
            //Bắt đầu nhận dữ liệu
            s.BeginReceive(buff, 0, buff.Length, SocketFlags.None, new
AsyncCallback(ReceiveData), s);
        }

        //Hàm ngắt kết nối
        public bool Disconnect()
        {
            try
            {
                //Shutdown Soket đang kết nối đến Server
                serverSocket.Shutdown(SocketShutdown.Both);
                serverSocket.Close();
                return true;
            }
            catch
            {
                return false;
            }
        }
        //Hàm gởi dữ liệu
        public void SendData(string Data)
        {
            buff = Encoding.ASCII.GetBytes(Data);
            serverSocket.BeginSend(buff, 0, buff.Length, SocketFlags.None, new
AsyncCallback(SendToServer), serverSocket);
        }
       //Hàm CallBack gởi dữ liệu
        private void SendToServer(IAsyncResult ia)
        {
            Socket s = (Socket)ia.AsyncState;
            s.EndSend(ia);
            buff = new byte[1024];
            s.BeginReceive(buff, 0, buff.Length, SocketFlags.None, new
AsyncCallback(ReceiveData), s);
        }
    }
```

```
}
```

VII.2.5.3. Lớp ClientForm

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Net;
using System.Net.Sockets;

namespace Client
{
    public partial class ClientForm : Form
    {
        ClientProgram client = new ClientProgram();
        public ClientForm()
        {
            InitializeComponent();
            CheckForIllegalCrossThreadCalls = false;
            client.SetDataFunction = new ClientProgram.SetDataControl(SetData);
        }
        private void SetData(string Data)
        {
            this.listBox1.Items.Add(Data);
        }

        private void cmdConnect_Click(object sender, EventArgs e)
        {
            client.Connect(IPAddress.Parse(this.txtServerIP.Text),
int.Parse(this.txtPort.Text));
        }
        private void cmdDisconnect_Click(object sender, EventArgs e)
        {
            client.Disconnect();
        }
        private void cmdSend_Click_1(object sender, EventArgs e)
        {
            client.SendData(this.txtInput.Text);
            this.txtInput.Text = "";
        }
    }
```

```
}
```

VII.3. Lập trình Socket bất đồng bộ sử dụng tiểu trình

VII.3.1. Lập trình sử dụng hàng đợi gởi và hàng đợi nhận thông điệp

Trong cách lập trình này, chúng ta sẽ dùng hai hàng đợi, một hàng đợi gởi và một hàng đợi nhận để thực hiện việc gởi và nhận dữ liệu. Lớp Queue nằm trong namespace System.Collections giúp ta thực hiện việc này.

```
Queue inQueue = new Queue();
Queue outQueue = new Queue();
```

Hai phương thức quan trọng của lớp Queue được dùng trong lập trình mạng là EnQueue() và DeQueue(). Phương thức EnQueue() sẽ đưa một đối tượng vào hàng đợi và phương thức DeQueue() sẽ lấy đối tượng đầu tiên từ hàng đợi ra.

Ý tưởng của phương pháp lập trình này là ta sẽ dùng hai vòng lặp vô hạn để kiểm tra hàng đợi gởi và hàng đợi nhận, khi hàng đợi gởi có dữ liệu thì dữ liệu đó được gởi ra ngoài mạng thông qua Socket, tương tự khi hàng đợi nhận có dữ liệu thì nó sẽ ngay lập tức lấy dữ liệu đó ra và xử lý.

```
private void Send()
{
	while (true)
	{
		if (OutQ.Count > 0)
		{
			streamWriter.WriteLine(OutQ.Dequeue().ToString());
			streamWriter.Flush();
		}
	}
}

private void Receive()
{
	string s;
	while (true)
	{
		s = streamReader.ReadLine();
		InQ.Enqueue(s);
	}
}
```

Để chạy song song hai vòng lặp vô hạn này ta phải tạo ra hai tiểu trình riêng, mỗi vòng lặp vô hạn sẽ chạy trong một tiểu trình riêng biệt do đó trong khi hai vòng lặp vô hạn này chạy thì tiến trình chính vẫn có thể làm được các công việc khác.

```
Thread tSend = new Thread(new ThreadStart(Send));
tSend.Start();

Thread tReceive = new Thread(new ThreadStart(Receive));
tReceive.Start();
```

Ngoài ra, ta còn sử dụng một tiểu trình khác thực hiện việc kết nối nhằm tránh gây treo tiến trình chính.

```
Thread tConnect = new Thread(new ThreadStart(WaitingConnect));
tConnect.Start();
```

Việc điều khiển kết nối được thực hiện trong một tiểu trình khác và được xử lý bằng phương thức WaitingConnect():

```
private void WaitingConnect()
{
      tcpListener = new TcpListener(1001);
      tcpListener.Start();

      socketForClient = tcpListener.AcceptSocket();

      if (socketForClient.Connected)
      {
            MessageBox.Show("Client Connected");
            netWorkStream = new NetworkStream(socketForClient);
            streamReader = new StreamReader(netWorkStream);
            streamWriter = new StreamWriter(netWorkStream);
            tSend = new Thread(new ThreadStart(Send));
            tSend.Start();
            tReceive = new Thread(new ThreadStart(Receive));
            tReceive.Start();
      }
      else
      {
            MessageBox.Show("Ket noi khong thanh cong");
      }
}
```

Việc nhập dữ liệu được thực hiện bằng phương thức InputData()

```
public void InputData(string s)
{
      InQ.Enqueue(s);
      OutQ.Enqueue(s);
}
```

Phương thức này đơn giản chỉ đưa dữ liệu nhập vào hàng đợi nhận để hiển thị dữ liệu lên màn hình và đưa dữ liệu nhập này vào hàng đợi gởi để gởi ra ngoài mạng.

Lớp ServerObject

```
using System;
using System.IO;
using System.Windows.Forms;
using System.Threading;
using System.Collections;
using System.Net.Sockets;
using System.Collections.Generic;
using System.Text;

namespace Server
{
    class ServerObject
    {
        Thread tSend, tReceive, tConnect;
        public Queue InQ = new Queue();
        public Queue OutQ = new Queue();

        private TcpListener tcpListener;
        Socket socketForClient;
        private NetworkStream netWorkStream;
        private StreamWriter streamWriter;
        private StreamReader streamReader;

        public void CreateConnection()
        {
            tConnect = new Thread(new ThreadStart(WaitingConnect));
            tConnect.Start();
        }

        private void WaitingConnect()
        {
            tcpListener = new TcpListener(1001);
```

```
        tcpListener.Start();

        socketForClient = tcpListener.AcceptSocket();

        if (socketForClient.Connected)
        {
            MessageBox.Show("Client Connected");
            netWorkStream = new NetworkStream(socketForClient);
            streamReader = new StreamReader(netWorkStream);
            streamWriter = new StreamWriter(netWorkStream);

            tSend = new Thread(new ThreadStart(Send));
            tSend.Start();

            tReceive = new Thread(new ThreadStart(Receive));
            tReceive.Start();
        }
        else
        {
            MessageBox.Show("Ket noi khong thanh cong");
        }
        //socketForClient.Close();
    }
    private void Send()
    {
        while (true)
        {
            if (OutQ.Count > 0)
            {
                streamWriter.WriteLine(OutQ.Dequeue().ToString());
                streamWriter.Flush();
            }
        }
    }
    private void Receive()
    {
        string s;
        while (true)
        {
            s = streamReader.ReadLine();
            InQ.Enqueue(s);
        }
    }
```

```
        public void InputData(string s)
        {
            InQ.Enqueue(s);
            OutQ.Enqueue(s);
        }

    }
}
```

Lớp ClientObject

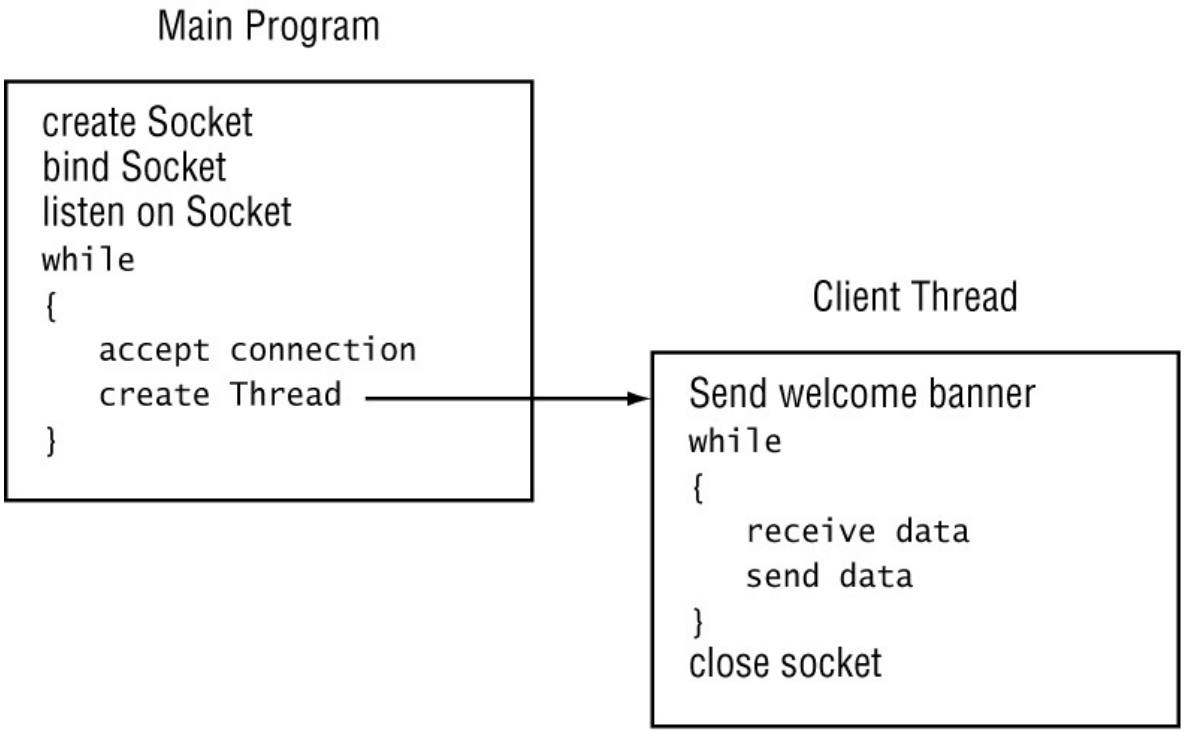
```
using System;
using System.Windows.Forms;
using System.Collections;
using System.Net.Sockets;
using System.Threading;
using System.IO;
using System.Collections.Generic;
using System.Text;

namespace Client
{
    class ClientObject
    {
        Thread tSend, tReceive, tConnect;

        public Queue InQ = new Queue();
        public Queue OutQ = new Queue();

        private TcpClient socketForServer;
        private NetworkStream networkStream;
        private StreamWriter streamWriter;
        private StreamReader streamReader;

        public void Connect()
        {
            tConnect = new Thread(new ThreadStart(WaitConnect));
            tConnect.Start();
        }
        public void WaitConnect()
        {
            try
            {
```

```
                socketForServer = new TcpClient("localhost", 1001);
            }
            catch {
                MessageBox.Show("Ket noi that bai");
            }
            networkStream = socketForServer.GetStream();
            streamReader = new StreamReader(networkStream);
            streamWriter = new StreamWriter(networkStream);

            tSend = new Thread(new ThreadStart(Send));
            tSend.Start();

            tReceive = new Thread(new ThreadStart(Receive));
            tReceive.Start();
        }
        private void Send()
        {
            while (true)
            {
                if (OutQ.Count > 0)
                {
                    streamWriter.WriteLine(OutQ.Dequeue().ToString());
                    streamWriter.Flush();
                }
            }
        }
        private void Receive()
        {
            string s;
            while (true)
            {
                s = streamReader.ReadLine();
                InQ.Enqueue(s);
            }
        }
        public void InputData(string s)
        {
            InQ.Enqueue(s);
            OutQ.Enqueue(s);
        }
    }
}
```

VII.3.2. Lập trình ứng dụng nhiều Client

Một trong những khó khăn lớn nhất của các nhà lập trình mạng đó là khả năng xử lý nhiều Client kết nối đến cùng một lúc. Để ứng dụng server xử lý được với nhiều Client đồng thời thì mỗi Client kết nối tới phải được xử lý trong một tiểu trình riêng biệt.

Mô hình xử lý như sau:



Hình VI.7: Mô hình xử lý một Server nhiều Client

Chương trình Server sẽ tạo ra đối tượng Socket chính trong chương trình chính, mỗi khi Client kết nối đến, chương trình chính sẽ tạo ra một tiểu trình riêng biệt để điều khiển kết nối. Bởi vì phương thức Accept() tạo ra một đối tượng Socket mới cho mỗi kết nối nên đối tượng mới này được sử dùng để thông tin liên lạc với Client trong tiểu trình mới. Socket ban đầu được tự do và có thể chấp nhận kết nối khác.

Chương trình ThreadedTcpSrvr

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
using System.Threading;
using System.Net;
using System.Net.Sockets;
```

```
class ThreadedTcpSrvr
{
    private TcpListener client;
    public ThreadedTcpSrvr()
    {
        client = new TcpListener(5000);
        client.Start();
        Console.WriteLine("Dang cho client...");
        while (true)
        {
            while (!client.Pending())
            {
                Thread.Sleep(1000);
            }
            ConnectionThread newconnection = new ConnectionThread();
            newconnection.threadListener = this.client;
            Thread newthread = new Thread(new
ThreadStart(newconnection.HandleConnection));
            newthread.Start();
        }
    }
    public static void Main()
    {
        ThreadedTcpSrvr server = new ThreadedTcpSrvr();
    }
}
class ConnectionThread
{
    public TcpListener threadListener;
    private static int connections = 0;
    public void HandleConnection()
    {
        int recv;
        byte[] data = new byte[1024];
        TcpClient client = threadListener.AcceptTcpClient();
        NetworkStream ns = client.GetStream();
        connections++;
        Console.WriteLine("Client moi duoc chap nhan, Hien tai co {0} ket noi",
connections);
        string welcome = "Xin chao client";
        data = Encoding.ASCII.GetBytes(welcome);
        ns.Write(data, 0, data.Length);
```

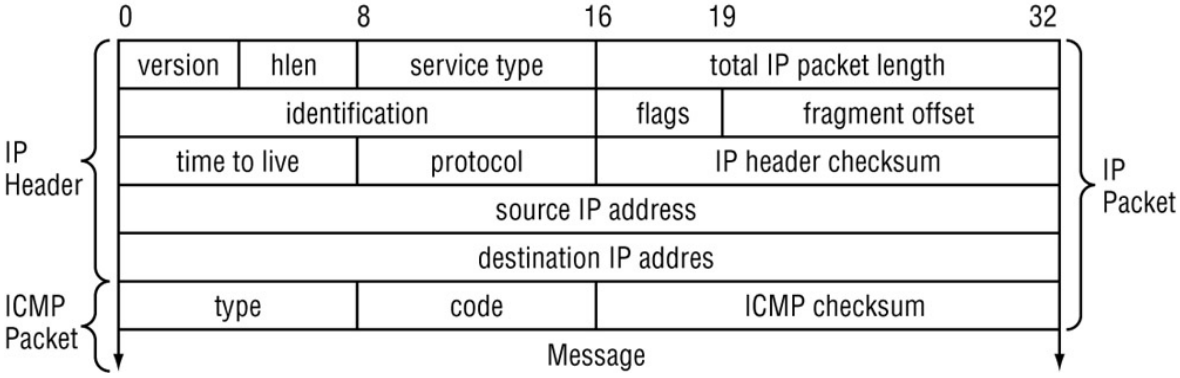
```
        while (true)
        {
            data = new byte[1024];
            recv = ns.Read(data, 0, data.Length);
            if (recv == 0)
                break;

            ns.Write(data, 0, recv);
        }
        ns.Close();
        client.Close();
        connection--;
        Console.WriteLine("Client disconnected: {0} active connections",
connections);
    }
}
```

CHƯƠNG VIII: LẬP TRÌNH VỚI CÁC GIAO THỨC

VIII.1. LẬP TRÌNH VỚI GIAO THỨC ICMP

VIII.1.1. Giao thức ICMP

ICMP được định nghĩa trong RFC 792, giao thức này giúp xác định lỗi của cac thiết bị mạng. ICMP sử dụng IP để truyền thông trên mạng. Mặc dù nó dùng IP nhưng ICMP hoàn toàn là một giao thức độc lập với TCP, UDP. Giao thức tầng kế tiếp của các gói tin IP được xác định trong phần dữ liệu sử dụng trường Type. Các gói tin ICMP được xác định bởi trường Type bằng 1 của gói tin IP, toàn bộ gói tin ICMP được kết hợp với phần dữ liệu của gói tin IP.



Định dạng của gói tin ICMP và IP

VIII.1.1.1. Định dạng của gói tin ICMP

Tương tự như TCP và UDP, ICMP dùng một đặc tả định dạng gói tin đặc biệt để xác định thông tin. Các gói tin ICMP gồm những trường sau:

Trường Type: kích thước 1 byte, xác định loại thông điệp ICMP trong gói tin. Nhiều loại gói tin ICMP được dùng để gởi thông điệp điều khiển đến các thiết bị ở xa. Mỗi loại thông điệp có định dạng riêng và các dữ liệu cần thiết.

Trường Code: kích thước 1 byte, các kiểu thông điệp ICMP khác nhau yêu cầu các tùy chọn dữ liệu và điều khiển riêng, những tùy chọn này được định nghĩa ở trường Code.

Trường Checksum: kích thước 2 byte, trường này đảm bảo gói tin ICMP đến đích mà không bị hư hại.

Trường Message: có nhiều byte, những byte này chứa các thành phần dữ liệu cần thiết cho mỗi kiểu thông điệp ICMP. Trường Message thường chứa đựng những thông tin được gởi và nhận từ thiết bị ở xa. Nhiều kiểu thông điệp ICMP định nghĩa 2 trường đầu tiên trong Message là Identifier và số Sequense. Các trường này dùng để định danh duy nhất gói tin ICMP đến các thiết bị.

VIII.1.1.2. Các tường Type của gói tin ICMP

Có nhiều kiểu gói tin ICMP, mỗi kiểu của gói tin ICMP được định nghĩa bởi 1 byte trong trường Type. Bảng sau liệt kê danh sách các kiểu ICMP ban đầu được định nghĩa trong RFC 792.

Type Code	**Mô Tả**
0	Echo reply
3	Destination unreachable
4	Source quench
5	Redirect
8	Echo request
11	Time exceeded
12	Parameter problem
13	Timestamp request
14	Timestamp reply
15	Information request
16	Information reply

Các trường Type của gói tin ICMP

Từ khi phát hành RFC 792 vào tháng 9 năm 1981, nhiều trường ICMP được tạo ra. Các gói tin ICMP được dùng cho việc thông báo lỗi mạng. Những mô tả sau hay dùng các gói tin ICMP:

VIII.1.1.3. Echo Request and Echo Reply Packets

Hai gói tin ICMP thường được sử dụng nhất là Echo Request và Echo Reply. Những gói tin này cho phép một thiết bị yêu cầu một trả lời ICMP từ một thiết bị ở xa

trên mạng. Đây chính là nhân của lệnh ping hay dùng để kiểm tra tình trạng của các thiết bị mạng.

Gói tin Echo Request dùng Type 8 của ICMP với giá trị code bằng 0. Phần dữ liệu của Message gồm các thành phần sau:

- 1 byte Indentifier: xác định duy nhất gói tin Echo Request
- 1 byte số Sequence: cung cấp thêm định danh cho gói tin gói tin ICMP trong một dòng các byte chứa dữ liệu được trả về gởi thiết bị nhận.

Khi một thiết bị nhận nhận một gói tin Echo Request, nó phải trả lời với một gói tin Echo Reply, trường Type ICMP bằng 0. Gói tin Echo Reply phải chứa cùng Identifier và số Sequence như gói tin Echo Request tương ứng. Phần giá trị dữ liệu cũng phải giống như của gói tin Echo Request.

VIII.1.1.4. Gói tin Destination Unreachable

Gói tin Destination Unreachable với trường Type bằng 3 thường trả về bởi thiết bị Router sau khi nó nhận được một gói tin IP mà nó không thể chuyển tới đích. Phần dữ liệu của gói tin Destination Unreachable chứa IP Header cộng với 64 bit giản đồ. Trong gói tin, trường Code xác định lý do gói tin không thể được chuyển đi bởi router. Bảng sau cho biết một số giá trị Code có thể gặp phải:

Code	**Mô Tả**
0	Network unreachable
1	Host unreachable
2	Protocol unreachable
3	Port unreachable
4	Fragmentation needed and DF flag set
5	Source route failed
6	Destination network unknown
7	Destination host unknown
8	Source host isolated
9	Communication with destination network prohibited
10	Communication with destination host prohibited
11	Network unreachable for type of service
12	Host unreachable for type of service

Các giá trị Code Destination Unreachable

VIII.1.1.5. Gói tin Time Exceeded

Gói tin Time Exceeded với trường Type của ICMP bằng 11 là một công cụ quan trọng để khắc phục các vấn đề mạng. Nó cho biết một gói tin IP đã vượt quá giá trị thời gian sống (TTL) được định nghĩa trong IP Header.

Mỗi khi một gói tin IP đến 1 router, giá trị TTL giảm đi một. Nếu giá trị TTL về 0 trước khi gói tin IP đến đích, router cuối cùng nhận được gói tin với giá trị TTL bằng 0 sẽ phải gởi một gói tin Time Exceeded cho thiết bị gởi. Lệnh tracert hay dùng gói tin này.

VIII.1.2. Sử dụng Raw Socket

Bởi vì các gói tin ICMP không dùng cả TCP và UDP, nên ta không dùng được các lớp hellper như TcpClient, UdpClient. Thay vì vậy, ta phải sử dụng Raw Socket, đây là một tính năng của lớp Socket. Raw Socket cho phép định nghĩa riêng các gói tin mạng phía trên tầng IP. Tất nhiên là tả phải làm tất cả mọi việc bằng tay như là tạo tất cả các trường của gói tin ICMP chứ không dùng các thư viện có sẵn của .NET như đã làm với TCP và UDP.

VIII.1.2.1. Định dạng của Raw Socket

Để tạo ra một Raw Socket, ta phải dùng SocketType.Raw khi Socket được tạo ra. Có nhiều giá trị ProtocolType ta có thể dùng với Raw Socket được liệt kê trong bảng sau:

Giá Trị	**Mô Tả**
Ggp	Gateway-to-Gateway Protocol
Icmp	Internet Control Message Protocol
Idp	IDP Protocol
Igmp	Internet Group Management Protocol
IP	A raw IP packet
Ipx	Novell IPX Protocol
ND	Net Disk Protocol
Pup	Xerox PARC Universal Protocol (PUP)
Raw	A raw IP packet
Spx	Novell SPX Protocol

Giá Trị	Mô Tả
SpxII	Novell SPX Version 2 Protocol
Unknown	An unknown protocol
Unspecified	An unspecified protocol

Các giá trị của Raw Socket ProtocolType

Những giao thức được liệt kê cho Raw Socket ở trên cho phép thư viện .NET tạo ra các gói tin IP bên dưới. Bằng cách sử dụng giá trị ProtocolType.Icmp, gói tin IP được tạo ra bởi Socket xác định giao thức tầng tiếp theo là ICMP (Trường Type bằng 1). Điều này cho phép thiết bị ở xa nhận ra ngay gói tin là 1 gói tin ICMP và xử lý nó một cách tương ứng. Sau đây là lệnh tạo ra một Socket cho các gói tin ICMP:

```
Socket sock = new Socket(AddressFamily.InterNetwork, SocketType.Raw, ProtocolType.Icmp);
```

VIII.1.2.2. Gởi các gói tin Raw

Bởi vì ICMP là một giao thức phi kết nối, nên ta không thể kết nối socket đến một port cục bộ để gởi một gói tin hoặc sử dụng phương thức Connect() để kết nối nó đến một thiết bị ở xa. Ta phải dùng phương thức SendTo() để chỉ ra đối tượng IPEndPoint của địa chỉ đích. ICMP không dùng các port vì thế giá trị port của đối tượng IPEndPoint không quan trọng. Ví dụ sau tạo một đối tượng IPEndPoint đích không có port và gởi một gói tin đến nó:

```
IPEndPoint iep = new IPEndPoint(IPAddress.Parse("192.168.1.2"), 0);
sock.SendTo(packet, iep);
```

VIII.1.2.3. Nhận các gói tin Raw

Nhận dữ liệu từ một Raw Socket khó hơn gởi dữ liệu từ một Raw Socket. Để nhận dữ liệu từ Raw Socket, ta phải dụng phương thức ReceiveFrom(). Bởi vì Raw Socket không định nghĩa giao thức tầng cao hơn, dữ liệu trả về từ một lần gọi phương thức ReceiveFrom() chứa toàn bộ nội dung của gói tin IP. Dữ liệu của gói tin IP bắt đầu từ byte thứ 20 do đó để lấy ra được dữ liệu và header của gói tin ICMP, ta phải bắt đầu đọc từ byte thứ 20 của dữ liệu nhận được.

VIII.1.3. Tạo ra một lớp ICMP

Như đã đề cập ở trước, Raw Socket không tự động định dạng gói tin ICMP vì vậy ta phải tự định dạng. Lớp ICMP phải định nghĩa mỗi biến cho mỗi thành phần trong gói tin ICMP. Các biến được định nghĩa mô tả một gói tin ICMP.

Các thành phần của một lớp ICMP điển hình

Biến Dữ Liệu	**Kích Thước**	**Kiểu**
Type	1 byte	Byte
Code	1 byte	Byte
Checksum	2 bytes	Unsigned 16-bit integer
Message	multibyte	Byte array

```
class ICMP
    {
        public byte Type;
        public byte Code;
        public UInt16 Checksum;
        public int MessageSize;
        public byte[] Message = new byte[1024];
        public ICMP()
        {
        }
    }
```

Sau khi gởi một gói tin ICMP thì hầu như sẽ nhận được một gói tin ICMP trả về từ thiết bị ở xa. Để dễ dàng giải mã nội dung của gói tin, chúng ta nên tạo một phương thức tạo lập khác của lớp ICMP nó có thể nhận một mảng byte Raw ICMP và đặt các giá trị vào phần dữ liệu thích hợp trong lớp:

```
public ICMP(byte[] data, int size)
  {
      Type = data[20];
      Code = data[21];
      Checksum = BitConverter.ToUInt16(data, 22);
      MessageSize = size - 24;
      Buffer.BlockCopy(data, 24, Message, 0, MessageSize);
  }
```

Raw Socket trả về toàn bộ gói tin IP do đó ta phải bỏ qua thông tin IP header trước khi lấy thông tin của gói tin ICMP vì thế phần Type ở tại vị trí 20 của mảng byte. Phần dữ liệu trong gói tin ICMP được lấy ra từng byte một và được đặt vào thành phần thích hợp của ICMP

Sua khi tạo ra một đối tượng ICMP mới với dữ liệu nhận được, ta có thể lấy các thành phần dữ liệu ra:

```
int recv = ReceiveFrom(data, ref ep);
ICMP response = new ICMP(data, recv);
Console.WriteLine("Received ICMP packet:");
Console.WriteLine(" Type {0}", response.Type);
Console.WriteLine(" Code: {0}", response.Code);
Int16 Identifier = BitConverter.ToInt16(response.Message, 0);
Int16 Sequence = BitConverter.ToInt16(response.Message, 2);
Console.WriteLine(" Identifier: {0}", Identifier);
Console.WriteLine(" Sequence: {0}", Sequence);
stringData = Encoding.ASCII.GetString(response.Message, 4, response.MessageSize
- 4);
Console.WriteLine(" data: {0}", stringData);
```

VIII.1.4. Tạo gói tin ICMP

Sau khi một đối tượng ICMP đã được tạo ra và phần dữ liệu của gói tin đã được định nghĩa, thì ta vẫn chưa thể gởi trực tiếp đối tượng ICMP bằng phương thức SendTo() được, nó phải chuyển thành 1 mảng byte, việc này được thực hiện nhờ vào phương thức Buffer.BlockCopy():

```
public byte[] getBytes()
{
      byte[] data = new byte[MessageSize + 9];
      Buffer.BlockCopy(BitConverter.GetBytes(Type), 0, data, 0, 1);
      Buffer.BlockCopy(BitConverter.GetBytes(Code), 0, data, 1, 1);
      Buffer.BlockCopy(BitConverter.GetBytes(Checksum), 0, data, 2, 2);
      Buffer.BlockCopy(Message, 0, data, 4, MessageSize);
      return data;
}
```

VIII.1.5. Tạo phương thức Checksum

Có lẽ phần khó khăn nhất của việc tạo một gói tin ICMP là tính toán giá trị checksum của gói tin. Cách dễ nhất để thực hiện điều này là tạo ra mọt phương thức để tính checksum rồi đặt đặt nó vào trong lớp ICMP để nó được sử dụng bởi chương trình ứng dụng ICMP.

```
public UInt16 getChecksum()
{
      UInt32 chcksm = 0;
      byte[] data = getBytes();
```

```
        int packetsize = MessageSize + 8;
        int index = 0;
        while (index < packetsize)
        {
                chcksm += Convert.ToUInt32(BitConverter.ToUInt16(data, index));
                index += 2;
        }
        chcksm = (chcksm >> 16) + (chcksm & 0xffff);
        chcksm += (chcksm >> 16);
        return (UInt16)(~chcksm);
}
```

Bởi vì giá trị checksum sử dụng mộ số 16 bit, thuật toán này đọc từng khúc 2 byte của gói tin ICMP một lúc (sử dụng phương thức ToUInt16() của lớp BitConverter) và thực hiện các phép toán số học cần thiết trên các byte. Giá trị trả về là một số nguyên dương 16 bit.

Để sử dụng giá trị checksum trong một chương trình ứng dụng ICMP, đầu tiên điền tất cả các thành phần dữ liệu, thiết lập thành phần Checksum thành 0, tiếp theo gọi phương thức getChecksum() để tính toán checksum của gói tin ICMP và sau đó đặt kết quả vào thành phần Checksum của gói tin:

```
packet.Checksum = 0;
packet.Checksum = packet.getChecksum();
```

Sau khi thành phần Checksum được tính toán, gói tin đã sẵn sàng được gởi ra ngoài thiết bị đích dùng phương thức SendTo().

Khi một gói tin ICMP nhận được từ một thiết bị ở xa, chúng ta phải lấy phần Checksum ra và so sánh với giá trị checksum tính được, nếu 2 giá trị đó không khớp nhau thì đã có lỗi xảy ra trong quá trình truyền và gói tin phải được truyền lại.

VIII.1.6. Lớp ICMP hoàn chỉnh

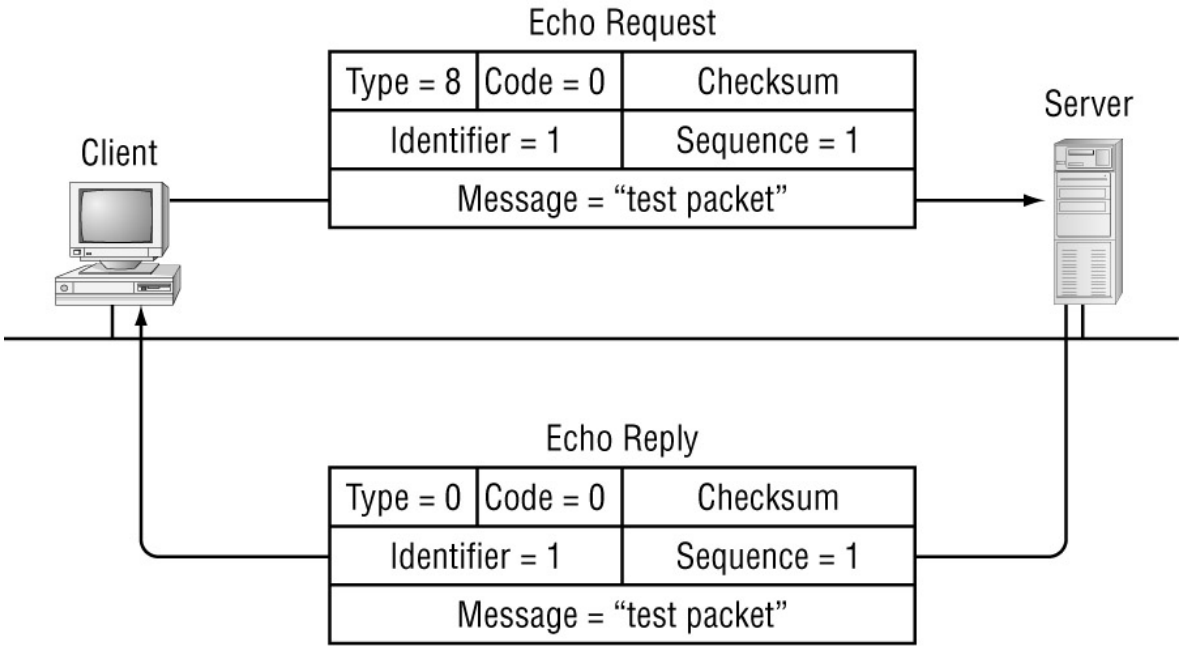
```
using System;
using System.Net;
using System.Text;
class ICMP
{
    public byte Type;
    public byte Code;
    public UInt16 Checksum;
```

```
    public int MessageSize;
    public byte[] Message = new byte[1024];
    public ICMP()
    {
    }
    public ICMP(byte[] data, int size)
    {
        Type = data[20];
        Code = data[21];
        Checksum = BitConverter.ToUInt16(data, 22);
        MessageSize = size - 24;
        Buffer.BlockCopy(data, 24, Message, 0, MessageSize);
    }
    public byte[] getBytes()
    {
        byte[] data = new byte[MessageSize + 9];
        Buffer.BlockCopy(BitConverter.GetBytes(Type), 0, data, 0, 1);
        Buffer.BlockCopy(BitConverter.GetBytes(Code), 0, data, 1, 1);
        Buffer.BlockCopy(BitConverter.GetBytes(Checksum), 0, data, 2, 2);
        Buffer.BlockCopy(Message, 0, data, 4, MessageSize);
        return data;
    }
    public UInt16 getChecksum()
    {
        UInt32 chcksm = 0;
        byte[] data = getBytes();
        int packetsize = MessageSize + 8;
        int index = 0;
        while (index < packetsize)
        {
            chcksm += Convert.ToUInt32(BitConverter.ToUInt16(data, index));
            index += 2;
        }
        chcksm = (chcksm >> 16) + (chcksm & 0xffff);
        chcksm += (chcksm >> 16);
        return (UInt16)(~chcksm);
    }
}
```

VIII.1.7. Chương trình ping đơn giản

Chương trình ping là một chương trình quan tronjng, nó là công cụ ban đầu để chuẩn đoán khả năng kết nối của một thiết bị mạng. Chương trình ping dùng các gói

tin ICMP Echo Request (Type 8) để gởi một thông điệp đơn giản đến thiết bị ở xa. Khi thiết bị ở xa nhận thông điệp, nó trả lời lại với một gói tin ICMP Echo Reply (Type 0) chứa thông điệp ban đầu



Thông điệp ICMP được dùng trong lệnh ping

Chương trình ping đơn giản:

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
class SimplePing
{
    public static void Main(string[] argv)
    {
        byte[] data = new byte[1024];
        int recv;
        Socket host = new Socket(AddressFamily.InterNetwork, SocketType.Raw,
ProtocolType.Icmp);
        IPEndPoint iep = new IPEndPoint(IPAddress.Parse(argv[0]), 0);
        EndPoint ep = (EndPoint)iep;
        ICMP packet = new ICMP();
        packet.Type = 0x08;
        packet.Code = 0x00;
        packet.Checksum = 0;
```

```
        Buffer.BlockCopy(BitConverter.GetBytes((short)1), 0, packet.Message, 0,
2);
        Buffer.BlockCopy(BitConverter.GetBytes((short)1), 0, packet.Message, 2,
2);
        data = Encoding.ASCII.GetBytes("goi tin test");
        Buffer.BlockCopy(data, 0, packet.Message, 4, data.Length);
        packet.MessageSize = data.Length + 4;
        int packetsize = packet.MessageSize + 4;
        UInt16 chcksum = packet.getChecksum();
        packet.Checksum = chcksum;
        host.SetSocketOption(SocketOptionLevel.Socket,
SocketOptionName.ReceiveTimeout, 3000);
        host.SendTo(packet.getBytes(), packetsize, SocketFlags.None, iep);
        try
        {
            data = new byte[1024];
            recv = host.ReceiveFrom(data, ref ep);
        }
        catch (SocketException)
        {
            Console.WriteLine("Khong co tra loi tu thiet bi o xa");
            return;
        }
        ICMP response = new ICMP(data, recv);
        Console.WriteLine("tra loi tu: {0}", ep.ToString());
        Console.WriteLine(" Type {0}", response.Type);
        Console.WriteLine(" Code: {0}", response.Code);
        int Identifier = BitConverter.ToInt16(response.Message, 0);
        int Sequence = BitConverter.ToInt16(response.Message, 2);
        Console.WriteLine(" Identifier: {0}", Identifier);
        Console.WriteLine(" Sequence: {0}", Sequence);
        string stringData = Encoding.ASCII.GetString(response.Message, 4,
response.MessageSize - 4);
        Console.WriteLine(" data: {0}", stringData);

        host.Close();
    }
}
```

Chương trình ping đơn giản này chỉ yêu cầu một địa chỉ IP được dùng ở command line. Trong phần đầu tiên của chương trình này, một gói tin ICMP được tạo ra, trường Type có giá trị là 8 và Code có giá trị 0. Nó tạo ra một gói tin Echo Request

sử dụng trường Identifier và Sequence để theo dõi các gói tin riêng biệt và cho phép nhập text vào phần dữ liệu.

Cũng giống như các chương trình hướng phi kết nối như UDP, một giá trị time-out được thiết lập cho Socket dùng phương thức SetSocketOption(). Nếu không có gói tin ICMP trả về trong 3 giây, một biệt lệ sẽ được ném ra và chương trình sẽ thoát.

Gói tin ICMP trả về tạo ra một đối tượng ICMP mới, đối tượng này có thể dùng để quyết định nếu gói tin nhận được khớp với gói tin ICMP gởi. Trường Identifier, Sequence và phần dữ liệu của gói tin nhận phải khớp giá trị của gói tin gởi.

VIII.1.8. Chương trình TraceRoute đơn giản

Chương trình traceroute gởi các ICMP Echo Request đến máy ở xa để biết được các router mà nó sẽ đi qua cho đến đích. Bằng cách thiết lập trường TTL (time to live) của gói tin IP tăng lên 1 giá trị khi đi qua mỗi router, chương trình traceroute có thể bắt buộc gói tin ICMP bị loại bỏ tại các router khác nhau trên đường nó tới đích. Mỗi lần trường TTL hết hạn, router cuối cùng sẽ gởi trả về một gói tin (Type 11) cho thiết bị gởi.

Bằng cách đặt giá trị khởi đầu cho trường TTL bằng 1, khi gói tin IP đi qua mỗi router thì giá trị TTL sẽ giảm đi một. Sau mỗi lần thiết bị gởi nhận được gói tin ICMP Time Exceeded trường TTL sẽ được tăng lên 1. Bằng cách hiển thị các địa chỉ gởi các gói tin ICMP Time Exceeded, ta có thể xác định được chi tiết đường đi của một gói tin tới đích.

Chương trình traceroute sử dụng phương thức SetSocketOption() và tùy chọn IPTimeToLive của Socket để thay đổi giá trị TTL của gói tin IP. Bởi vì chỉ có mỗi trường TTL của gói tin IP bị thay đổi, gói tin ICMP có thể được tạo ra một lần và được sử dụng trong các lần tiếp theo mỗi khi sử dụng gói tin IP này.

Chương trình traceroute đơn giản:

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
class TraceRoute
{
    public static void Main(string[] argv)
    {
```

```
        byte[] data = new byte[1024];
        int recv, timestart, timestop;
        Socket host = new Socket(AddressFamily.InterNetwork, SocketType.Raw,
ProtocolType.Icmp);
        IPHostEntry iphe = Dns.Resolve(argv[0]);
        IPEndPoint iep = new IPEndPoint(iphe.AddressList[0], 0);
        EndPoint ep = (EndPoint)iep;
        ICMP packet = new ICMP();
        packet.Type = 0x08;
        packet.Code = 0x00;
        packet.Checksum = 0;
        Buffer.BlockCopy(BitConverter.GetBytes(1), 0, packet.Message, 0, 2);
        Buffer.BlockCopy(BitConverter.GetBytes(1), 0, packet.Message, 2, 2);
        data = Encoding.ASCII.GetBytes("goi tin test");
        Buffer.BlockCopy(data, 0, packet.Message, 4, data.Length);
        packet.MessageSize = data.Length + 4;
        int packetsize = packet.MessageSize + 4;
        UInt16 chcksum = packet.getCchecksum();
        packet.Checksum = chcksum;
        host.SetSocketOption(SocketOptionLevel.Socket,
                   SocketOptionName.ReceiveTimeout, 3000);
        int badcount = 0;
        for (int i = 1; i < 50; i++)
        {
            host.SetSocketOption(SocketOptionLevel.IP,
                  SocketOptionName.IpTimeToLive, i);
            timestart = Environment.TickCount;
            host.SendTo(packet.getBytes(), packetsize, SocketFlags.None, iep);
            try
            {
                data = new byte[1024];
                recv = host.ReceiveFrom(data, ref ep);
                timestop = Environment.TickCount;
                ICMP response = new ICMP(data, recv);
                if (response.Type == 11)
                    Console.WriteLine("Chang {0}: tra loi tu {1}, {2}ms", i,
ep.ToString(), timestop - timestart);
                if (response.Type == 0)
                {
                    Console.WriteLine("{0} den {1} chang, {2}ms.",
ep.ToString(), i, timestop - timestart);
                    break;
                }
                badcount = 0;
            }
            catch (SocketException)
            {
                Console.WriteLine("chang {0}: khong co tra loi tu thiet bi o
xa", i);
                badcount++;
                if (badcount == 5)
                {
                    Console.WriteLine("Khong the lien lac voi thiet bi o xa");
                    break;
                }
            }
        }

        host.Close();
    }
}
```

VIII.2. LẬP TRÌNH VỚI GIAO THỨC SMTP

VIII.2.1. Cơ bản về Email

Hầu hết mọi gói tin email trên Internet đều dùng mô hình email của Unix. Mô hình này trở nên phổ biến và được sử dụng rộng rãi để phân phá thư đến cả người dùng cục bộ và người dùng ở xa.

Mô hình email Unix chia các chức năng email ra làm 3 phần:

- The Message Transfer Agent (MTA)
- The Message Delivery Agent (MDA)
- The Message User Agent (MUA)

Mỗi một phần thực hiện một chức năng riêng biệt trong quá trình gởi, nhận và hiển thị thư

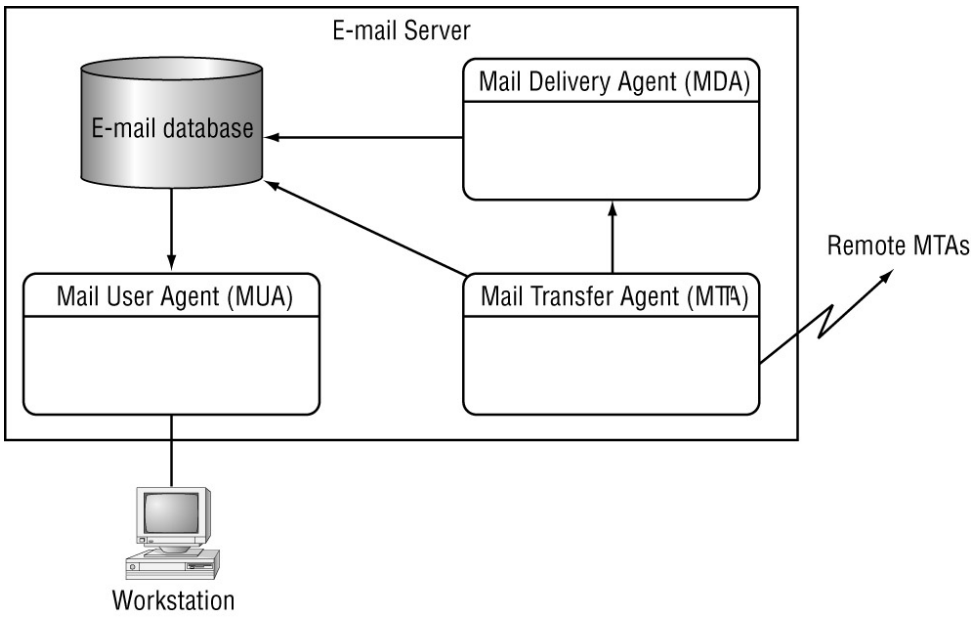
VIII.2.1.1. Hoạt động của MTA

MTA điều khiển cả các mail gởi đến và mail gởi đi, trước đây nó được chia ra làm 2 chức năng riêng biệt:

Quyết định nơi và cách gởi mail ra ngoài

Quyết định noi chuyển mail đến

Mỗi một chức năng trên yêu cầu một chức năng xử lý khác nhau trong MTA



Mô hình email Unix

VIII.2.1.2. Gởi mail ra ngoài

Với mỗi mail được gởi ra ngoài, MTA phải quyết định đích của địa chỉ nhận. Nếu đích là hệ thống cục bộ, MTA có thể hoặc là chuyển mail trực tiếp hệ thống mailbox cục bộ hoặc chuyển mail đến MDA để phát mail.

Tuy nhiên, nếu đích là một domain ở xa, MTA phải thực hiện 2 chức năng sau:

- Quyết định mail server của domain đó có sử dụng mục MX ở DNS hay không
- Thành lập một liên kết thông tin với mail server ở xa và chuyển mail.

Liên kết thông tin với mail server ở xa luôn luôn dùng SMTP (Simple Mail Transfer Protocol). Giao thức chuẩn này cung cấp một ký thuật giao tiếp chung để chuyển mail giữa 2 hệ thống mail khác nhau trên Internet

VIII.2.1.3. Nhận mail

MTA chịu trách nhiệm nhận các thư gởi đến cả từ hệ thống cục bộ và từ người dùng ở xa. Địa chỉ đích của thư phải được xem xét kỹ lưỡng và một quyết định phải được thực hiện cho biết thư có thể thực sự được gởi từ hệ thống cục bộ hay không.

Có 3 danh mục địa chỉ đích có thể được dùng trong thư: các tài khoản cục bộ của hệ thống, các tài khoản bí danh cục bộ, các tài khoản người dùng ở xa.

Các tài khoản cục bộ của hệ thống: mỗi hệ thống mail, dù là Window hay Unix hay ngay cả Macintosh cũng đều có một tập hợp các tài khoản cục bộ có thể truy cập vào hệ thống. MTA phải nhận ra các thư các mail có đích là tài khỏa người dùng và chuyển nó hoặc trực tiếp đến hộp thư của người dùng hoặc đến một MDA riêng biệt để chuyển đi.

Các tài khoản bí danh cục bộ: Nhiều MTA cho phép các tên bí danh được tạo ra. Tên bí danh chính nó không thể lưu giữ thư, thay vì vậy nó sử dụng con trỏ đến một hoặc nhiều người dùng của hệ thống thực sự nơi mà thư được lưu trữ. Mỗi khi MTA quyết định tên bí danh hợp lệ, nó chuyển đổi địa chỉ đích thành hệ thống tên tài khoản người dùng thực sự,

Các tài khoản người dùng ở xa: nhiều MTA cũng chấp nhận các thư gởi đến mà đích là các tài khoản người dùng không nằm trên hệ thống cục bộ. Đây là công việc

đòi hỏi phải có nhiều kỹ thuật, nhiều khó khăn để MTA điều khiển việc gởi mail đến một hệ thống các tài khoản ở xa.

Kỹ thuật này được gọi là relying. Một mail server chấp nhận một thư gởi đến mà đích là một tài khoản của máy ở xa và tự động chuyển thư đó đến máy ở xa. Nhiều ISP, tính năng này là cần thiết bởi khác hàng không có khả năng gởi thư trực tiếp đến các máy ở xa. Thay vì vậy, họ sẽ gởi thư đến mail server của ISP, và mail server của ISP sẽ tự động chuyển thư này đến hệ thống đích.

Thật không may mắn, việc chuyển tiếp mail có thể bị khai thác, người dùng có thể sử dụng các mail server chuyển tiếp để ẩn địa chỉ nguồn khi chuyển hàng loạt thư rác đến các người dùng trên hệ thống email ở xa, những email này phải được chặn lại bởi người quản trị mail.

VIII.2.1.4. Hoạt động của MDA

Chức năng chính của MDA là chuyển các thư có đích là các tài khoản người dùng trên hệ thống cục bộ. Để làm được điều này, MDA phải biết kiểu và vị trí của các mailbox của các cá nhân. Hầu hết các hệ thống email sử dùng một số kiểu hệ thống cơ sở dữ liệu để theo dõi các thư được lưu giữ bởi người dùng cục bộ. MDA phải truy cập đến mỗi mailbox của người dùng để chèn các thư được gởi đến.

Nhiều MDA cũng thực hiện các kỹ thuật nâng cao chỉ để chuyển tiếp thư:

Tự động lọc thư: đây là chức năng phổ biến nhất của các MDA để lọc các thu đến. Đối với các người dùng phải nhận hàng đống email mỗi ngày thì đây là một tính năng tuyệt vời. Các thư có thể được tự động sắp xếp theo các thư mục riêng biệt dựa vào các giá trị header, hay ngay cả một vài từ trong header. Hầu hết các MDA cũng cho phép nhà quản trị mail cấu hình lọc trên toàn bộ hệ thống để có thể chặn các thư rác hoặc các thư có virus.

Tự động trả lời thư: nhiều chương trình MDA cho phép người dùng cấu hình tự chuyển thư, một số thư tự động trả lời có thể được cấu hình để trả lời tất cả các thư được nhận bởi một người dùng, một số khác được cấu hình tự động trả lời dựa vào một số từ đặc biệt trong header.

Tự động chạy chương trình: đây là khả năng khởi động chương trình hệ thống dựa vào một số mail đến, đây là một công cụ quản trị của nhà quản trị. Nhà quản trị hệ thống mail có thể khởi động hệ thống mail từ xa hay có thể cấu hình mail server từ một máy khác mail server.

VIII.2.1.5. Hoạt động của MUA

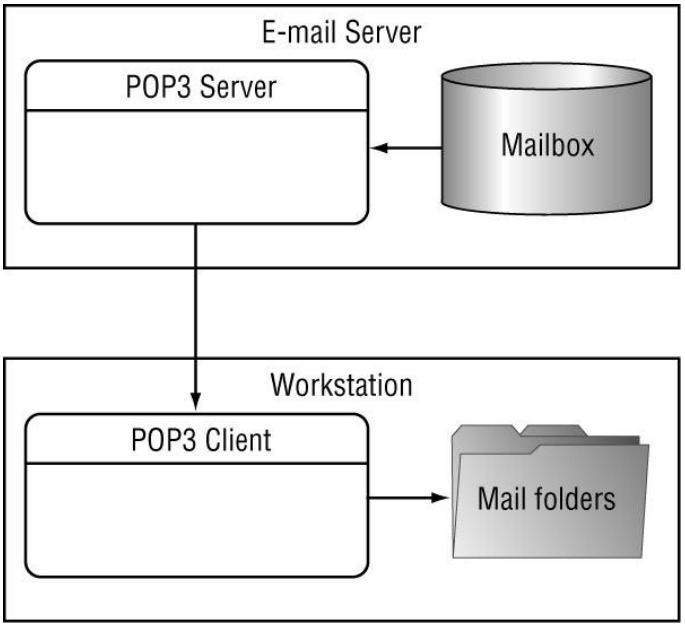
MUA cho phép người dùng không ngồi trước mail server vẫn có thể truy cập hộp thư của họ. MUA chỉ chịu trách nhiệm chính đọc các thư trong hộp thư, nó không thể nhận hoặc gởi thư.

Có 2 vấn đề phức tạp với MUA là việc đọc và lưu trữ các thư đến:

Lưu giữ các thư ở Client

Nhiều ISP thích người dùng tải các thư của họ về. Mỗi khi thư được tải về, nó sẽ bị gở bỏ khỏi mail server. Điều này giúp cho người quản trị mail bảo toàn được không gian lưu trữ trên server.

Giao thức được dùng cho loại truy cập này là POP (Post Office Protocol, thường được gọi là POP3 cho version 3). Chương trình MUA POP3 cho phép người dùng kết nối đến server và tải tất cả các thư ở hộp thư về. Quá trình này được mô tả trong hình sau:

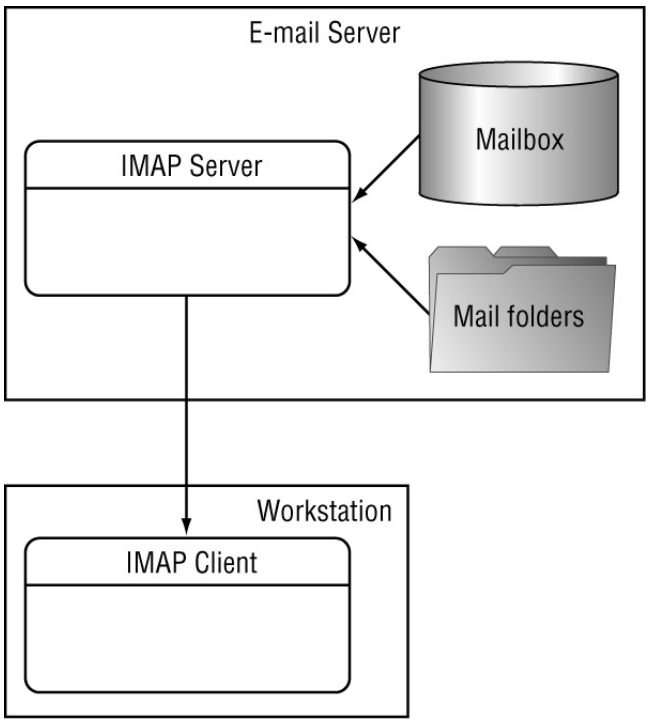


Quá trình tải thư sử dụng phần mềm POP3

Điều trở ngại khi sử dụng POP3 là các thư được lưu giữ ở trạm làm việc nơi mà người dùng kết nối tới mail server. Nếu người dùng kết nối sử dụng nhiều trạm làm việc thì các thư sẽ bị chia ra giữa các trạm làm việc làm cho người dùng khó truy cập các thư sau khi đã tải các thư xong. Hiện nay một số ISP cho phép tải thư sử dụng POP3 mà không xóa khỏi hộp thư.

Lưu giữ thư ở Server

Một phương thức truy cập thư thay cho POP3 là IMAP (Interactive Mail Access Protocol hay IMAPPrev4 cho version 4). IMAP cho phép người dùng xây dựng các thư mục ở mail server và lưu giữ các thư trong các thư mục thay vì phải tải xuống trạm làm việc. Bởi vì các thư được lưu giữ ở server, người dùng có thể kết nối từ bất kỳ trạm làm việc nào để đọc thư. Quá trình này được mô tả như trong hình sau:



Lưu giữ thư trên server ở các thư mục sử dụng IMAP

Điều trở ngại đối với hệ thống này là việc tiêu tốn dung lượng đĩa trên server.

The .NET mail library uses the Microsoft CDOSYS message component to send messages to remote hosts using SMTP. One significant area of confusion is when and how Windows OSes provide support for CDOSYS and how the CDOSYS relates to .NET. This section will shed some light on this topic, in addition to answering questions about CDOSYS.

VIII.2.2. SMTP và Windows

Thư viện mail của .NET sử dụng Microsoft CDOSYS để gởi thư đến máy ở xa dùng SMTP.

VIII.2.2.1. Collaboration Data Objects (CDO)

Hệ thống mail Microsoft Exchange 5 sử dụng thư viện Active Message (OLEMSG32.DLL) cho phép các lập trình viên viết code dùng hệ thống Exchange chuyển thư đến các hộp thư của người dùng cũng như đến hệ thống Exchange khác trên mạng Exchange.

Với sự phát hành của Exchange 5.5, một hệ thống thư mới được giới thiệu CDO (Collaboration Data Object). Không giống như Active Messaging, CDO sử dụng MAPI (Message Application Program Interface) để cung cấp cho các lập trình viên một cách dễ hơn để gởi thư. CDO cũng tồn tại qua vài năm với nhiều cải tiến bao gồm phiên bản 1.1, 1.2 và 1.2.1. Nền tảng Microsoft NT Server sau này sử dụng thư viện CDO NT Server (CDONTS). Thư viện này không dùng chuẩn MAPI được sử dụng trong CDO mà bắt đầu sử dụng các chuẩn của Internet như SMTP để chuyển thư đến các máy ở xa.

Phiên bản hiện tại của CDO (CDO 2) được phát hành chung với Windows 2000 và nó cũng được dùng trên Windows XP. Nó sử dụng CDONTS để cung cấp các thư viện mail và news cho lập trình viên. Các thành phần mới của CDO 2 bao gồm khả năng xử lý nhiều file đính kèm thư và nhiều sự kiện giao thức mail khác nhau. Những tính năng này giúp cho các lập trình viên dễ dàng hơn trong việc lập trình.

Trong Windows 2000, XP, thư viện CDO 2 là CDOSYS.DLL, tất cả các chức năng trong thư viện mail .NET phải cần tập tin này. Bởi vì CDO 2 hoàn toàn khác với các phiên bản CDO 1.x nên hệ thống cần cả 2 thư viện này. CDO 2 không tương thích lùi với các nền tảng Windows cũ hơn.

VIII.2.2.2. Dịch vụ mail SMTP

Cả Windows 2000 và Windows XP đều cung cấp một server SMTP cơ bản hỗ trợ cả gởi và nhận thư sử dụng giao thức SMTP. Chức năng này là một phần của IIS

(Internet Information Services). Các lớp mail của .NET có thể sử dụng IIS SMTP Server để gởi thư trực tiếp ra ngoài đến các mail server ở xa.

Trước khi sử dụng SMTP, chúng ta phải cấu hình cho nó, việc cấu hình được thực hiện từ cửa sổ Computer Management.

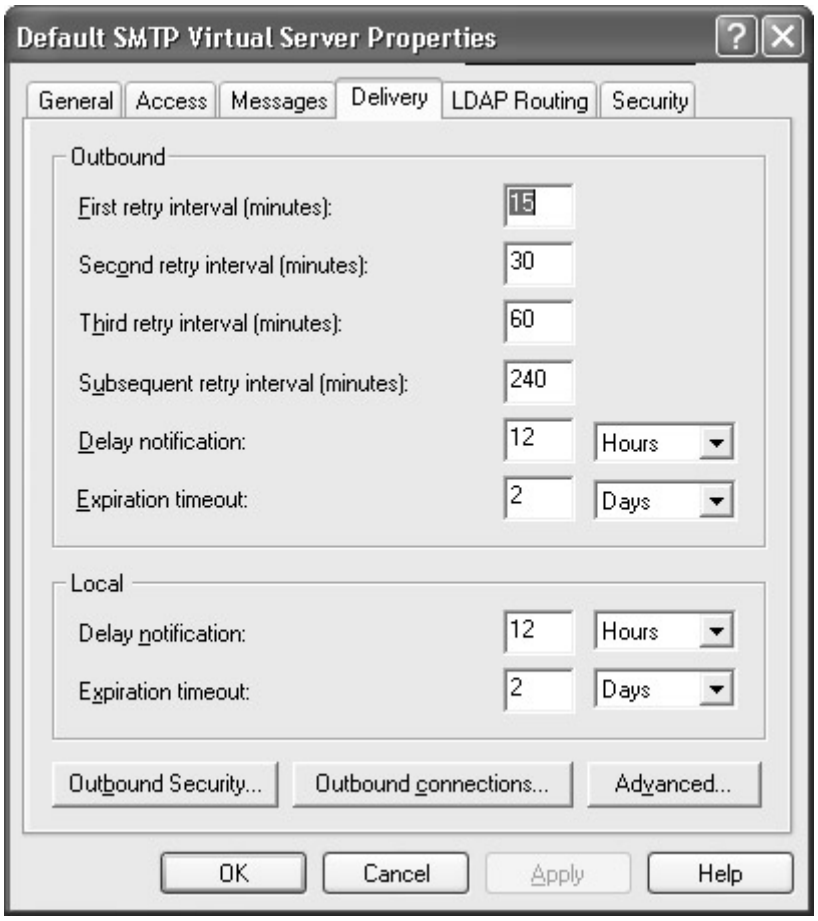
Các bước cấu hình như sau:

1. Nhấn chuột phải vào My Computer, chọn Manage
2. Khi cửa sổ Computer Management xuất hiện, mở Services and Applications và sau đó mở Internet Information Services
3. Nhấn chuột phải vào Default SMTP Virtual Server và chọn Properties

Cửa sổ Properties là nơi để chúng ta cấu hình các thiết lập cho dịch vụ SMTP

Trong thẻ General, chọn card mạng cho phép các kết nối SMTP tới và số lượng các kết nối SMTP đồng thời được phép. Chúng ta cũng có thể bật/tắt việc ghi log cho dịch vụ SMTP.

Trong thẻ Delivery, cấu hình số lần cố gắng gởi thư. Nút Advanced là các thiết lập thông minh cho mail server. Cấu hình này quan trọng nếu mail server của ISP là một server chuyển tếp đến mail server ở xa. Để làm điều này, chúng ta phải, nhập địa chỉ mail server của ISP và dịch vụ SMTP sẽ chuyển tiếp tất cả các thư ra ngoài thông qua server này.



Default SMTP Virtual Properties

VIII.2.3. Lớp SmtpMail

Lớp SmtpMail nằm trong namespace System.Web.Mail cho phép gởi thư theo giao thức SMTP trong chương trình C#.

VIII.2.4. Các phương thức và thuộc tính của lớp SmtpMail

Lớp SmtpMail cung cấp giao tiếp .NET cho thư viện mail CDOSYS trên hệ thống Windows 200 và Windows XP. Nó không dùng phương thức tạo lập để tạo 1 thể hiện của lớp. Thay vì vậy, ta phải dùng các phương thức và thuộc tính static để truyền thông tin cho thư viện CDOSYS.

Phương thức Send() của lớp SmtpMail là hay được dùng nhất, phương thức Send() được quá tải hàm và nó có 2 định dạng như sau:

```
Send(MailMessage message)
Send(string from, string to, string subject, string body)
```

Định dạng đầu tiên cho phép gởi một đối tượng MailMessage. Lớp MailMessage chứa một thông điệp email, thông điệp này được tạo ra bằng cách gán giá trị cho các thuộc tính của lớp với thông tin liên quan đến thông điệp và các địa chỉ đích.

Định dạng thứ 2 sẽ cho phép chúng ta gởi một thông điệp dạng raw, chúng ta phải chỉ ra các trường header của thông điệp email:

From: người gởi thông điệp

To: các địa chỉ nhận thông điệp, ngăn cách bởi dấu phẩy

Subject: tiêu đề thư

Đối số cuối cùng của phương thức Send() là phần thân của thư. Phần này có thể là text hoặc HTML.

Thuộc tính duy nhất của lớp SmtpMail là SmtpServer. Nó là một thuộc tính static và nó chỉ ra địa chỉ của server chuyển tiếp thư được dùng để chuyển tiếp các thông điệp mail ra ngoài. Mặc định, thuộc tính SmtpSerrver được thiết lập cho dịch vụ SMTP của IIS nếu IIS được cài. Nếu IIS không được cài, thuộc tính SmtpServer được thiết lập giá trị null và sẽ phát sinh ra lỗi khi gởi thông điệp đi.

Nếu chúng ta sử dụng một mail server chuyển tiếp, chúng ta phải thiết lập giá trị cho thuộc tính SmtpServer trước khi gởi thông điệp.

```
SmtpMail.SmtpServer = "mailsrvr.myisp.net";
```

Khi tất cả các giá trị đã được thiết lập, tất cả các thư gởi đi sẽ được chuyển tiếp qua mail server này. Tất nhiên, chúng ta phải đảm bảo rằng server này cho phép chúng ta chuyển tiếp mail qua nó.

VIII.2.4.1. Sử dụng lớp SmtpMail

Chương trình MailTest sau đây sẽ minh họa cách tạo một email đơn giản và gởi nó thông qua một mail rely đến người nhận:

TÀI LIỆU THAM KHẢO

[1] **C Sharp Network Programming,** Sybex
[2] **Microsoft Network Programming For The Microsoft Dot Net Framework,** Microsoft Press
[3] **Network programming in dot NET C Sharp and Visual Basic dot NET,** Digital Press

TRƯỜNG ĐẠI HỌC ĐÀ LẠT
KHOA CÔNG NGHỆ THÔNG TIN

BÀI GIẢNG TÓM TẮT

LẬP TRÌNH MẠNG

Dành cho sinh viên ngành Công Nghệ Thông Tin

(Lưu hành nội bộ)

Đà Lạt 2009

LỜI NÓI ĐẦU

Giáo trình " Lập trình mạng" được biên soạn theo chương trình đào tạo hệ thống tín chỉ của trường Đại Học Đà Lạt. Mục đích biên soạn giáo trình nhằm cung cấp cho sinh viên ngành Công Nghệ Thông Tin những kiến thức về lập trình mạng.

Tuy có rất nhiều cố gắng trong công tác biên soạn nhưng chắc chắn rằng giáo trình này còn nhiều thiếu sót. Chúng tôi xin trân trọng tiếp thu tất cả những ý kiến đóng góp của các đồng nghiệp cũng như các bạn sinh viên, trong lĩnh vực này để hoàn thiện giáo trình, phục vụ tốt hơn cho việc dạy và học tin học đang ngày càng phát triển ở nước ta.

Khoa Công Nghệ Thông Tin
Trường Đại Học Đà Lạt

MỤC LỤC

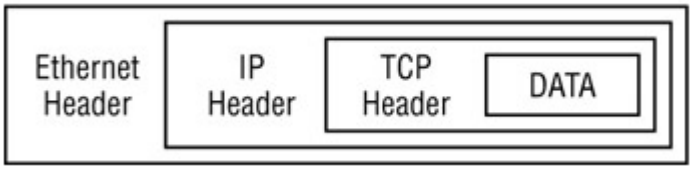
CHƯƠNG I: NHỮNG KIẾN THỨC CƠ BẢN VỀ LẬP TRÌNH MẠNG

I.1. Tổng quan

Internet Protocol (IP) là nền tảng của lập trình mạng. IP là phương tiện truyền tải dữ liệu giữa các hệ thống bất kể đó là hệ thống mạng cục bộ (LAN) hay hệ thống mạng diện rộng (WAN). Mặc dù lập trình viên mạng có thể chọn các giao thức khác để lập trình nhưng IP cung cấp các kỹ thuật mạnh nhất để gởi dữ liệu giữa các thiết bị, đặc biệt là thông qua mạng Internet.

Để hiểu rõ các khái niệm bên dưới lập trình mạng, chúng ta phải hiểu rõ giao thức IP, hiểu cách nó chuyển dữ liệu giữa các thiết bị mạng. Lập trình mạng dùng giao thức IP thường rất phức tạp. Có nhiều yếu tố cần quan tâm liên quan đến cách dữ liệu được gởi qua mạng: số lượng Client và Server, kiểu mạng, tắc nghẽn mạng, lỗi mạng,… Bởi vì các yếu tố này ảnh hưởng đến việc truyền dữ liệu từ thiết bị này đến thiết bị khác trên mạng do đó việc hiểu rõ chúng là vấn đề rất quan trọng để lập trình mạng được thành công.

Một gói dữ liệu mạng gồm nhiều tầng thông tin. Mỗi tầng thông tin chứa một dãy các byte được sắp đặt theo một trật tự đã được định sẵn. Hầu hết các gói dữ liệu dùng trong lập trình mạng đều chứa ba tầng thông tin cùng với dữ liệu được dùng để truyền tải giữa các thiết bị mạng. Hình sau mô tả hệ thống thứ bậc của một gói IP:

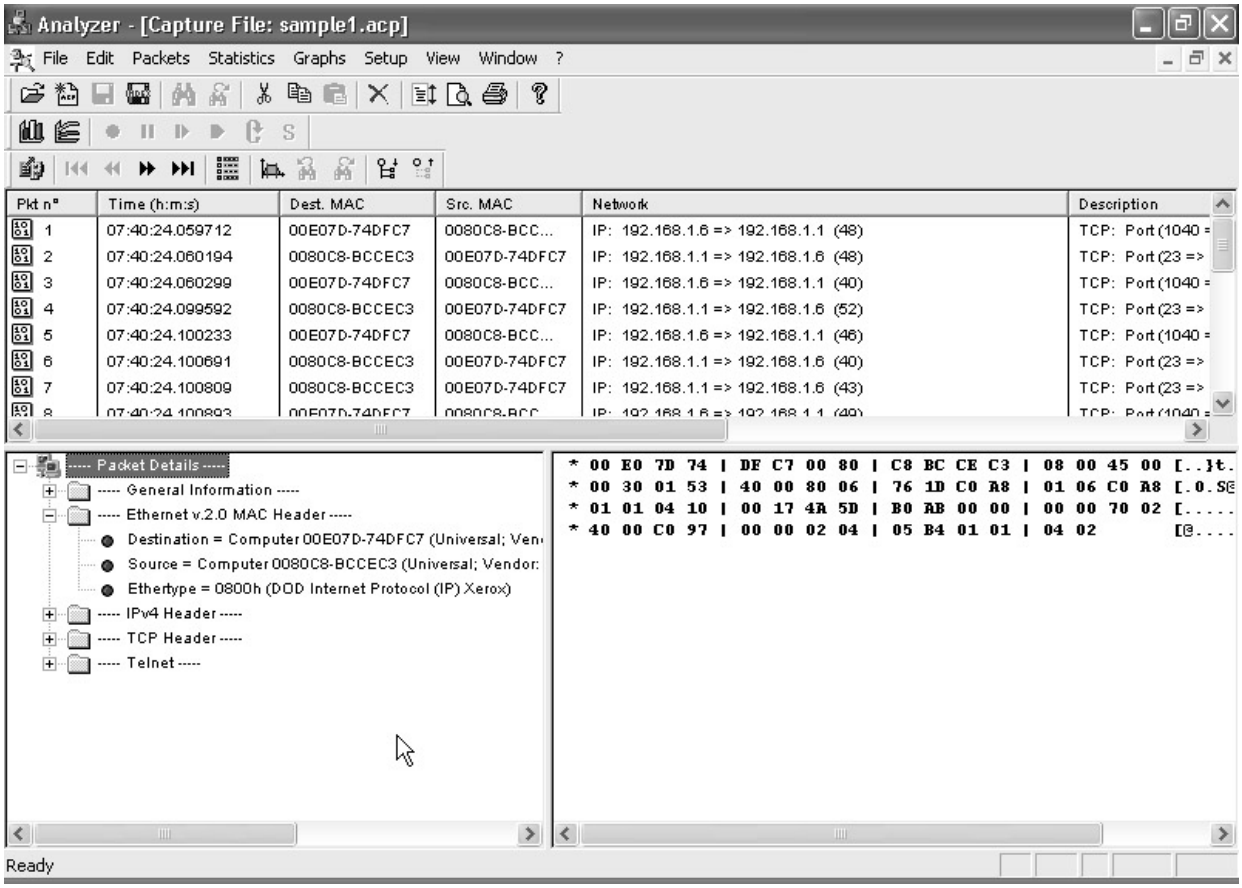


Hình I.1: Các tầng giao thức mạng trong các gói dữ liệu

I.1.1. Tầng Ethernet

Tầng đầu tiên của gói dữ liệu mạng được gọi là Ethernet Header, trong tầng này có ba gói giao thức Ethernet: Ethernet 802.2, Ethernet 802.3, và Ethernet phiên bản 2. Các giao thức Ethernet 802.2 và Ethernet 802.3 là các giao thức chuẩn của IEEE. Ethernet phiên bản 2 tuy không phải là giao thức chuẩn nhưng nó được sử dụng rộng

rãi trong mạng Ethernet. Hầu hết các thiết bị mạng kể cả hệ điều hành Windows mặc định dùng giao thức Ethernet phiên bản 2 để truyền tải các gói IP.



Hình I.2: Ethernet Header

Phần đầu của Ethernet phiên bản 2 là địa chỉ MAC (Media Access Card) dùng để xác định các thiết bị trên mạng cùng với số giao thức Ethernet xác định giao thức tầng tiếp theo chứa trong gói Ethernet. Mỗi gói Ethernet bao gồm:

- 6 byte địa chỉ MAC đích
- 6 byte địa chỉ MAC nguồn
- 2 byte xác định giao thức tầng kế tiếp
- Data payload từ 46 đến 1500 byte
- 4-byte checksum

I.1.2. Địa chỉ Ethernet

Địa chỉ Ethernet (địa chỉ MAC) là địa chỉ của các thiết bị, địa chỉ này được gán bởi các nhà sản xuất thiết bị mạng và nó không thay đổi được. Mỗi thiết bị trên mạng Ethernet phải có 1 địa chỉ MAC duy nhất. Địa chỉ MAC gồm 2 phần:

- 3 byte xác định nhà sản xuất
- 3 byte xác định số serial duy nhất của nhà sản xuất

Giản đồ địa chỉ Ethernet cho phép các địa chỉ broadcast và multicast. Đối với địa chỉ broadcast thì tất cả các bit của địa chỉ đích được gán bằng 1 (FFFFFFFFFFFF). Mỗi thiết bị mạng sẽ chấp nhận các gói có địa chỉ broadcast. Địa chỉ này hữu ích cho các giao thức phải gởi các gói truy vấn đến tất cả các thiết bị mạng. Địa chỉ multicast cũng là một loại địa chỉ đặc biệt của địa chỉ Ethernet, các địa chỉ multicast chỉ cho phép một số các thiết bị chấp nhận gói tin. Một số địa chỉ Ethernet multicast:

Địa Chỉ	**Mô Tả**
01-80-C2-00-00-00	Spanning tree (for bridges)
09-00-09-00-00-01	HP Probe
09-00-09-00-00-01	HP Probe
09-00-09-00-00-04	HP DTC
09-00-2B-00-00-00	DEC MUMPS
09-00-2B-00-00-01	DEC DSM/DTP
09-00-2B-00-00-02	DEC VAXELN
09-00-2B-00-00-03	DEC Lanbridge Traffic Monitor (LTM)
09-00-2B-00-00-04	DEC MAP End System Hello
09-00-2B-00-00-05	DEC MAP Intermediate System Hello
09-00-2B-00-00-06	DEC CSMA/CD Encryption
09-00-2B-00-00-07	DEC NetBios Emulator
09-00-2B-00-00-0F	DEC Local Area Transport (LAT)
09-00-2B-00-00-1x	DEC Experimental
09-00-2B-01-00-00	DEC LanBridge Copy packets (all bridges)
09-00-2B-02-00-00	DEC DNA Lev. 2 Routing Layer Routers
09-00-2B-02-01-00	DEC DNA Naming Service Advertisement
09-00-2B-02-01-01	DEC DNA Naming Service Solicitation
09-00-2B-02-01-02	DEC DNA Time Service
09-00-2B-03-xx-xx	DEC default filtering by bridges

Địa Chỉ	**Mô Tả**
09-00-2B-04-00-00	DEC Local Area System Transport (LAST)
09-00-2B-23-00-00	DEC Argonaut Console
09-00-4E-00-00-02	Novell IPX
09-00-77-00-00-01	Retix spanning tree bridges
09-00-7C-02-00-05	Vitalink diagnostics
09-00-7C-05-00-01	Vitalink gateway
0D-1E-15-BA-DD-06	HP
CF-00-00-00-00-00	Ethernet Configuration Test protocol (Loopback)

I.1.3. Ethernet Protocol Type

Một phần khác rất quan trọng của Ethernet Header là trường Protocol Type, trường này có kích thước hai byte. Sự khác nhau giữa gói tin Ethernet phiên bản 2 và Ethernet 802.2 và 802.3 xảy ra ở trường này. Các gói tin Ethernet 802.2 và 802.3 sử dụng trường này để cho biết kích thước của một gói tin Ethernet. Ethernet phiên bản 2 dùng trường này để định nghĩa giao thức tầng kế tiếp trong gói tin Ethernet. Một số giá trị của trường này:

Giá Trị	**Giao Thức**
0800	IP
0806	ARP
0BAD	Banyan VINES
8005	HP Probe
8035	Reverse ARP
809B	AppleTalk
80D5	IBM SNA
8137	Novell
8138	Novell
814C	Raw SNMP
86DD	IPv6
876B	TCP/IP compression

I.1.4. Data payload

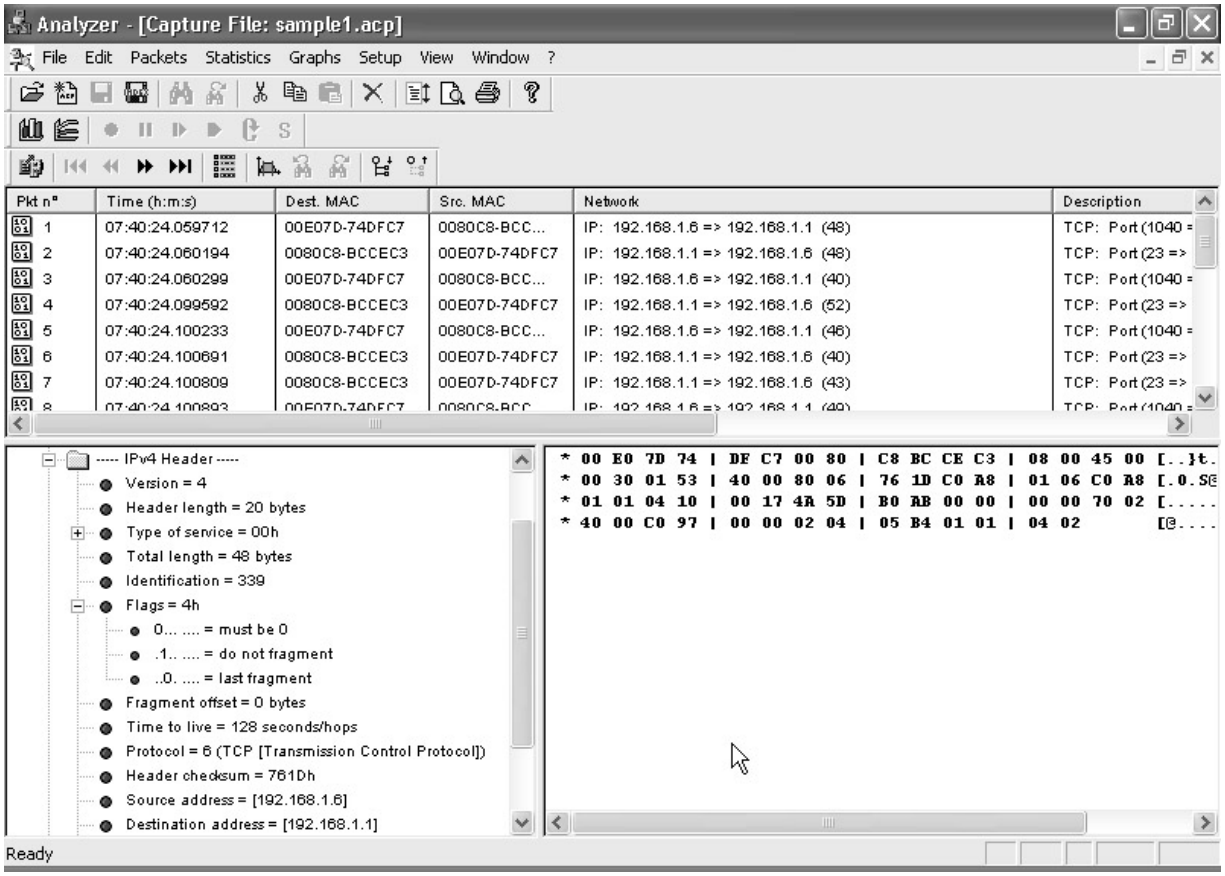
Data payload phải chứa tối thiểu 46 byte để đảm bảo gói Ethernet có chiều dài tối thiểu 64 byte. Nếu phần data chưa đủ 46 byte thì các ký tự đệm được thêm vào cho đủ. Kích thước của trường này từ 46 đến 1500 byte.

I.1.5. Checksum

Giá trị checksum cung cấp cơ chế kiểm tra lỗi cho dữ liệu, kích thước của trường này là 4 byte . Nếu gói tin bị hỏng trong lúc truyền, giá trị checksum sẽ bị tính toán sai và gói tin đó được đánh dấu là gói tin xấu.

I.2. Tầng IP

Tầng IP định nghĩa thêm nhiều trường thông tin của của giao thức Ethernet



Hình I.3: Thông tin tầng IP

Các trường trong tầng IP:

Trường	**Bit**	**Mô Tả**
Version	4	Phiên bản IP header (phiên bản hiện tại là 4)
Header Length	4	Chiều dài phần header của gói IP

Trường	**Bit**	**Mô Tả**
Type of Service	8	Kiểu chất lượng dịch vụ QoS (Quality of Service)
Total Length	16	Chiều dài của gói IP
Identification	16	Giá trị ID duy nhất xác định các gói IP
Flags	3	Cho biết gói IP có bị phân đoạn hay không hay còn các phân đoạn khác
Fragment offset	13	Vị trí của phân đoạn trong gói IP
Time to Live (TTL)	8	Thời gian tối đa gói tin được phép ở lại trên mạng (được tính bằng giây)
Protocol	8	Kiểu giao thức của tầng dữ liệu kế tiếp
Header Checksum	16	Checksum của dữ liệu gói IP header
Source Address	32	Địa chỉ IP của thiết bị gởi
Destination Address	32	Địa chỉ IP của thiết bị nhận
Options		Định nghĩa các đặc điểm của gói IP trong tươnglai

I.2.1. Trường địa chỉ

Địa chỉ Ethernet dùng để xác định các thiết bị trên mạng LAN nhưng nó không thể dùng để xác định địa chỉ của các thiết bị trên mạng ở xa. Để xác định các thiết bị trên các mạng khác nhau, địa chỉ IP được dùng. Một địa chỉ IP là một số 32 bit và địa chỉ IP được chia thành 4 lớp sau:

Lớp A	0.x.x.x–127.x.x.x
Lớp B	128.x.x.x–191.x.x.x
Lớp C	192.x.x.x–223.x.x.x
Lớp D	224.x.x.x–254.x.x.x

I.2.2. Các cờ phân đoạn

Một trong những phức tạp, rắc rối của gói IP là kích thước của chúng. Kích thước tối đa của gói IP có thể lên đến 65,536 byte. Đây là một lượng rất lớn dữ liệu cho một gói tin. Thực tế hầu hết các truyền tải dữ liệu ở cấp thấp như Ethernet không thể hỗ trợ một gói IP lớn (phần dữ liệu của Ethernet chỉ có thể tối đa 1500 byte). Để giải quyết vấn đề này, các gói IP dùng fragmentation (phân đoạn) để chia các gói IP thành các phần nhỏ hơn để truyền tải tới đích. Khi các mảnh được truyền tải tới đích, phần mềm của thiết bị nhận phải có cách để nhận ra các phân đoạn của gói tin và ráp chúng lại thành thành 1 gói IP.

Sự phân đoạn được thành lập nhờ vào việc sử dụng 3 trường của gói IP: fragmentation flags, fragment offset, và trường identification. Cờ phân đoạn bao gồm ba cờ một bit sau:

- Cờ reserved: giá trị zero
- Cờ Don't Fragment: cho biết gói IP không bị phân đoạn
- Cờ More Fragment: cho biết gói tin bị phân đoạn và còn các phân đoạn khác nữa

Trường IP Indentification xác định duy nhất định danh mỗi gói IP. Tất cả các phân đoạn của bất kỳ gói IP nào cũng đều có cùng số indentification. Số identification giúp cho phần mềm máy nhận biết được các phân đoạn nào thuộc gói IP nào và ráp lại cho đúng.

Trường fragment offset cho biết vị trí của phân đoạn trong gói tin ban đầu.

I.2.3. Trường Type of Service

Trường Type of Service xác định kiểu chất lượng dịch vụ QoS (Quality of Service) cho gói IP. Trường này được dùng để đánh dấu một gói IP có một độ ưu tiên nào đó chẳng hạn như được dùng để tăng độ ưu tiên của các dữ liệu cần thời gian thực như Video, Audio.

Trong hầu hết các truyền tải mạng, trường này được được thiết lập giá trị zero, cho biết đây là dữ liệu bình thường, tuy nhiên với các ứng dụng cần thời gian thực như Video hay Audio thì trường này sẽ được sử dụng để tăng độ ưu tiên cho gói dữ liệu. Trường này gồm tám bit và ý nghĩa các bit như sau:

- 3 bit được dùng làm trường ưu tiên
- 1 bit cho biết thời gian trễ là bình thường hay thấp
- 1 bit cho biết thông lượng bình thường hay cao
- 1 bit cho biết độ tin cậy bình thường hay cao
- 2 bit được dùng trong tương lai

I.2.4. Trường Protocol

Được dùng để xác định giao thức tầng tiếp theo trong gói IP, IANA định nghĩa 135 giá trị cho trường này có thể dùng trong gói IP nhưng chỉ có một số giá trị hay được dùng trong bảng sau:

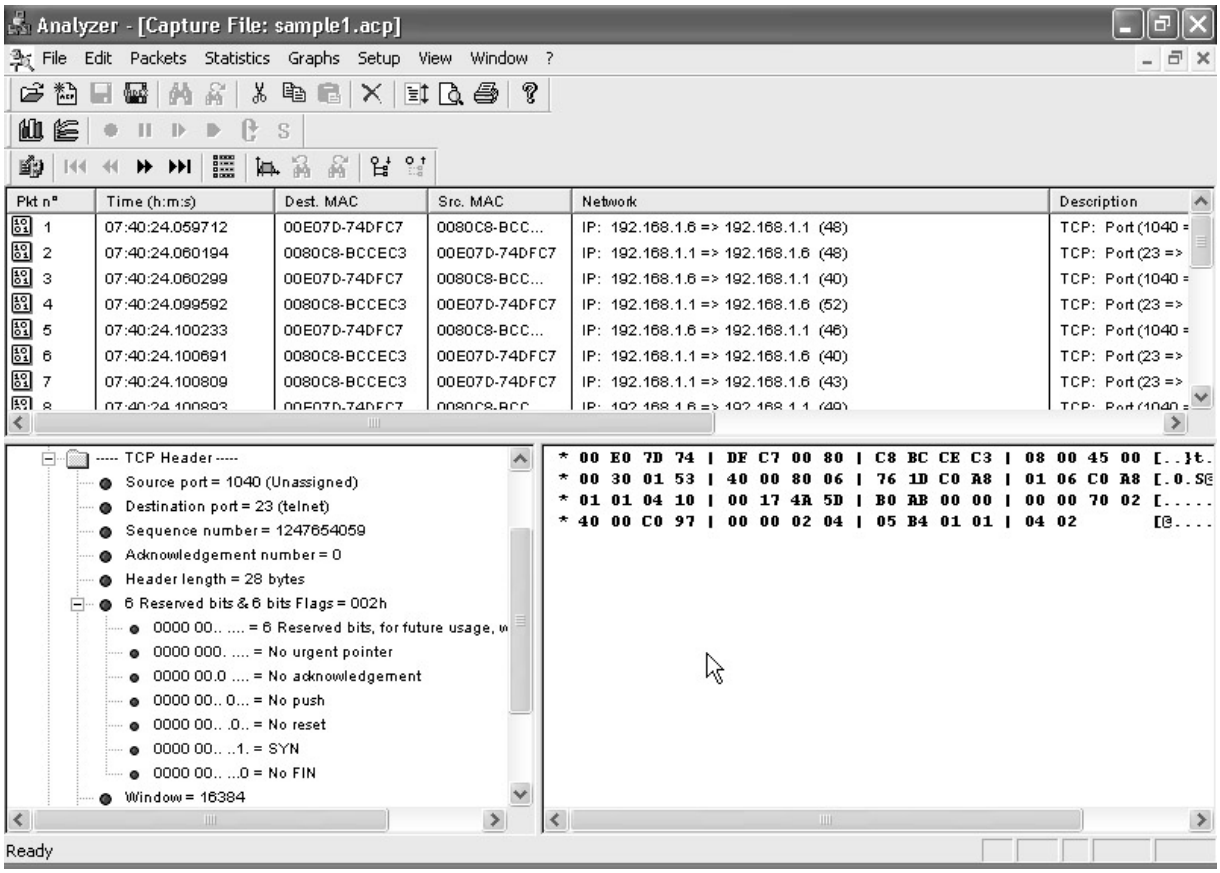
Giá Trị	**Giao Thức**
1	Internet Control Message (ICMP)
2	Internet Group Message (IGP)
6	Transmission Control (TCP)
8	Exterior Gateway (EGP)
9	Interior Gateway (Cisco IGP)
17	User Datagram (UDP)
88	Cisco EIGRP

Hai giao thức được dùng nhiều nhất trong lập trình mạng là TCP và UDP

I.3. Tầng TCP

Giao thức TCP (Transmission Control Protocol) là giao thức hướng kết nối, nó cho phép tạo ra kết nối điểm tới điểm giữa hai thiết bị mạng, thiết lập một đường nhất quán để truyền tải dữ liệu. TCP đảm bảo dữ liệu sẽ được chuyển tới thiết bị đích, nếu dữ liệu không tới được thiết bị đích thì thiết bị gởi sẽ nhận được thông báo lỗi.

Các nhà lập trình mạng phải hiểu cách hoạt động cơ bản của TCP và đặc biệt là phải hiểu cách TCP truyền tải dữ liệu giữ các thiết bị mạng. Hình sau cho thấy những trường của TCP Header. Những trường này chứa các thông tin cần thiết cho việc thực thi kết nối và truyền tải dữ liệu một cách tin tưởng.



Hình I.4: Các trường của TCP Header

Mỗi trường của TCP Header kết hợp với một chức năng đặc biệt của một phiên làm việc TCP. Có một số chức năng quan trọng sau:

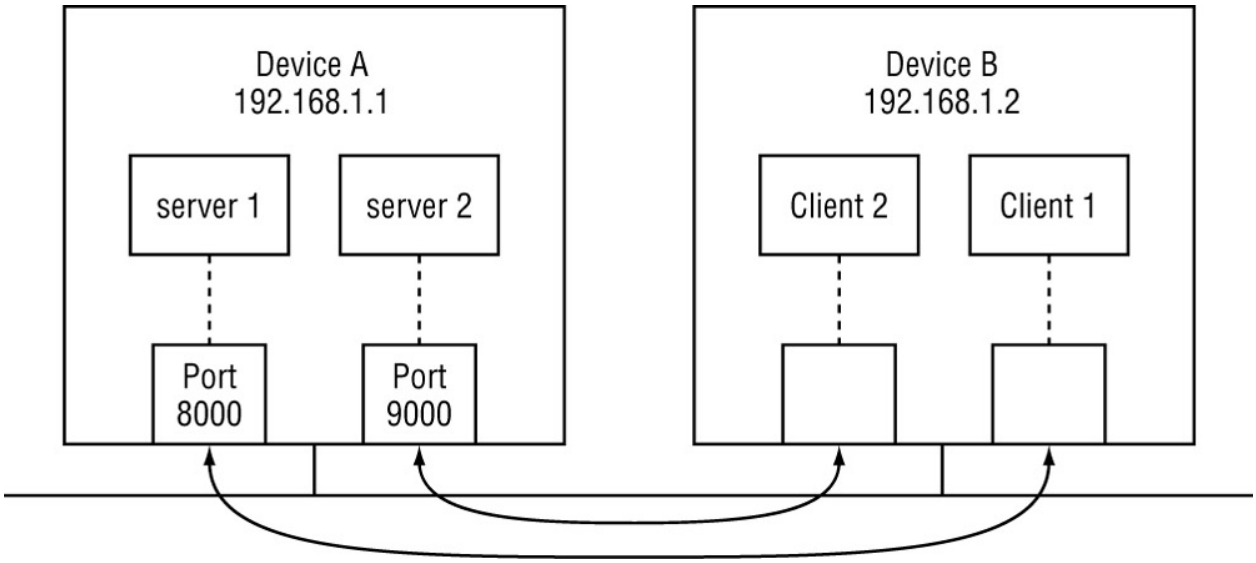
- Source port và Destination port: theo dõi các kết nối giữa các thiết bị
- Sequence và Acknowledgement number: theo dõi thứ tự các gói tin và truyền tải lại các gói tin bị mất
- Flag: mở và đóng kết nối giữa các thiết bị để truyền tải dữ liệu

I.3.1. TCP port

TCP sử dụng các port để xác định các kết nối TCP trên một thiết bị mạng. Để liên lạc với một ứng dụng chạy trên một thiết bị mạng ở xa ta cần phải biết hai thông tin :

- Địa chỉ IP của thiết bị ở xa
- TCP port được gán cho thiết bị ở xa

Để kết nối TCP được thành lập, thiết bị ở xa phải chấp nhận các gói tin truyền đến port đã được gán. Bởi vì có nhiều ứng dụng chạy trên một thiết bị sử dụng TCP do đó thiết bị phải cấp phát các cổng khác nhau cho các ứng dụng khác nhau.



Hình I.5: Kết nối TCP đơn giản

Trong hình trên thì thiết bị A đang chạy hai ứng dụng Server, hai ứng dụng này đang chờ các gói tin từ Client. Một ứng dụng được gán port 8000 và một ứng dụng được gán port 9000. Thiết bị mạng B muốn kết nối đến thiết bị mạng A thì nó phải được gán một TCP port còn trống từ hệ điều hành và port này sẽ được mở trong suốt phiên làm việc. Các port ở Client thường không quan trọng và có thể gán bất kỳ một port nào hợp lệ trên thiết bị.

Tổ hợp của một địa chỉ IP và một port là một IP endpoint. Một phiên làm việc TCP được định nghĩa là một sự kết hợp của một IP endpoint cục bộ và một IP endpoint ở xa. Một ứng dụng mạng có thể sử dụng cùng một IP endpoint cục bộ nhưng mỗi thiết bị ở xa phải sử dụng một địa chỉ IP hay port riêng.

IANA định nghĩa một danh sách các port TCP tiêu chuẩn được gán cho các ứng dụng đặc biệt:

Port	**Mô Tả**
7	Echo
13	Daytime
17	Quote of the day
20	FTP (data channel)
21	FTP (control channel)

Port	**Mô Tả**
22	SSH
23	Telnet
25	SMTP
37	Time
80	HTTP
110	POP3
119	NNTP
123	Network Time Protocol (NTP)
137	NETBIOS name service
138	NETBIOS datagram service
143	Internet Message Access Protocol (IMAP)
389	Lightweight Directory Access Protocol (LDAP)
443	Secure HTTP (HTTPS)
993	Secure IMAP
995	Secure POP3

Các port từ 0->1023 được gán cho các ứng dụng thông dụng do đó với các ứng dụng mà các lập trình viên tạo ra thì các port được gán phải từ 1024->65535.

I.3.2. Cơ chế đảm bảo độ tin cậy truyền tải các gói tin

Trường tiếp theo trong TCP Header sau port là số sequence và acknowledgement. Những giá trị này cho phép TCP theo dõi các gói tin và đảm bảo nó được nhận theo đúng thứ tự. Nếu bất kỳ gói tin nào bị lỗi, TCP sẽ yêu cầu truyền tải lại các gói tin bị lỗi và ráp chúng lại trước khi gởi gói tin cho ứng dụng.

Mỗi gói tin có một số duy nhất sequence cho một phiên làm việc TCP. Một số ngẫu nhiên được chọn cho gói tin đầu tiên được gởi đi trong phiên làm việc. Mỗi gói tin tiếp theo được gởi sẽ tăng số sequence bằng số byte dữ liệu TCP trong gói tin trước đó. Điều này đảm bảo mỗi gói tin được xác định duy nhất trong luồng dữ liệu TCP.

Thiết bị nhận sử dụng trường acknowledgement để hồi báo số sequence cuối cùng được nhận từ thiết bị gởi. Thiết bị nhận có thể nhận nhiều gói tin trước khi gởi lại một hồi báo. Số acknowledgement được trả về là số sequence cao nhất liền sau của dữ liệu được nhận. Kỹ thuật này được gọi là cửa sổ trượt. Các gói tin được nhận ngoài thứ

tự có thể được giữ trong bộ đệm và được đặt vào đúng thứ tự khi các gói tin khác đã được nhận thành công. Nếu một gói tin bị mất, thiết bị nhận sẽ thấy được số sequence bị lỗi và gởi một số acknowledgement thấp hơn để yêu cầu các gói tin bị lỗi. Nếu không có cửa sổ trượt mỗi gói tin sẽ phải hồi báo lại, làm tăng băng thông và độ trễ mạng.

I.3.3. Quá trình thành lập một phiên làm việc TCP

Quá trình làm thành lập một phiên làm việc TCP được thực hiện nhờ vào việc sử dụng các cờ (Flag):

Flag	**Mô Tả**
6 bit dành riêng	Dành riêng để sử dụng trong tương lai, giá trị luôn luôn là zero
1-bit URG flag	Đánh dấu gói tin là dữ liệu khẩn cấp
1-bit ACK flag	Hồi báo nhận một gói tin
1-bit PUSH flag	Cho biết dữ liệu được đẩy vào ứng dụng ngay lập tức
1-bit RESET flag	Thiết lập lại tình trạng khởi đầu kết nối TCP
1-bit SYN flag	Bắt đầu một phiên làm việc
1-bit FIN flag	Kết thúc một phiên làm việc

TCP sử dụng các tình trạng kết nối để quyết định tình trạng kết nối giữa các thiết bị. Một giao thức bắt tay đặc biệt được dùng để thành lập những kết nối này và theo dõi tình trạng kết nối trong suốt phiên làm việc. Một phiên làm việc TCP gồm ba pha sau:

- Mở bắt tay
- Duy trì phiên làm việc
- Đóng bắt tay

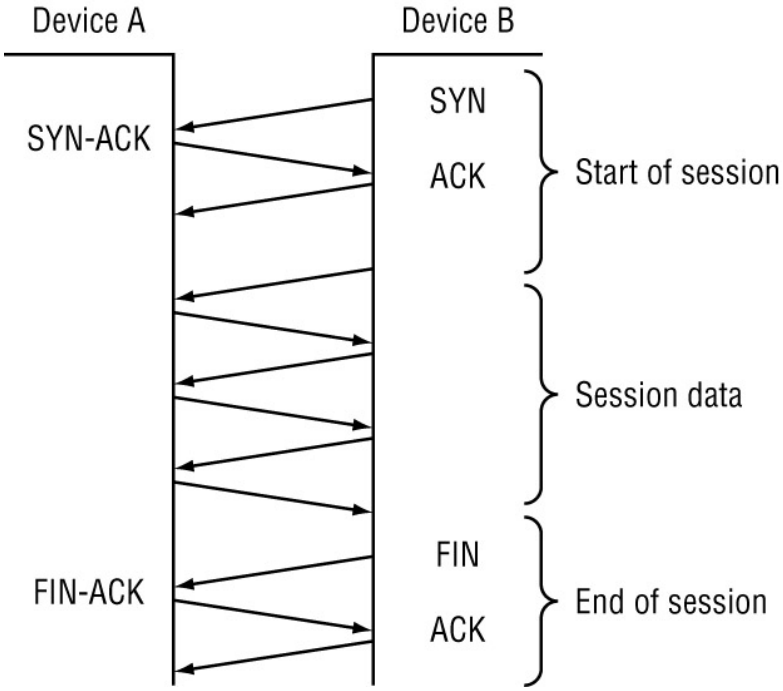
Mỗi pha yêu cầu các bit cờ được thiết lập trong một thứ tự nào đó. Quá trình mở bắt tay thường được gọi là ba cái bắt tay và nó yêu cầu ba bước để thành lập kết nối.

- Thiết bị gởi gởi cờ SYN cho biết bắt đầu phiên làm việc
- Thiết bị nhận gởi cả cờ SYN và cờ ACK trong cùng một gói tin cho biết nó chấp nhận bắt đầu phiên làm việc

- Thiết bị gởi gởi cờ ACK cho biết phiên làm việc đã mở và đã sẵng sàng cho việc gởi và nhận các gói tin.

Sau khi phiên làm việc được thành lập, cờ ACK sẽ được thiết lập trong các gói tin. Để đóng phiên làm việc, một quá trình bắt tay khác được thực hiện dùng cờ FIN:

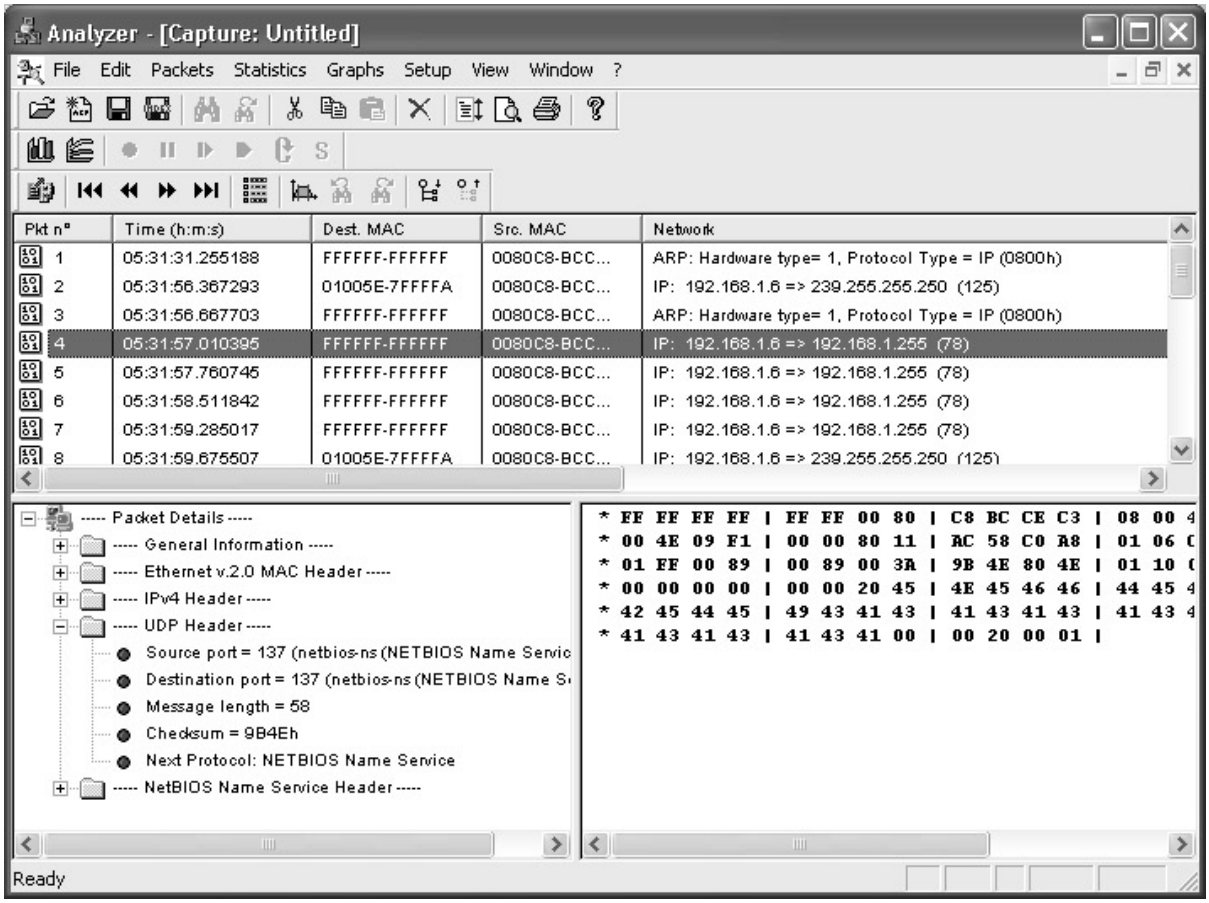
- Thiết bị khởi đầu đóng kết nối gởi cờ FIN
- Thiết bị bên kia gởi cờ FIN và ACK trong cùng một gói tin cho biết nó chấp nhận đóng kết nối
- Thiết bị khởi đầu đóng kết nối gởi cờ ACK để đóng kết nối



Hình I.6: Các bước bắt tay của giao thức TCP

I.4. Tầng UDP

User Datagram Protocol (UDP) là một giao thức phổ biến khác được dùng trong việc truyền tải dữ liệu của các gói IP. Không giống như TCP, UDP là giao thức phi nối kết. Mỗi phiên làm việc UDP không gì khác hơn là truyền tải một gói tin theo một hướng. Hình sau sẽ mô tả cấu trúc của một gói tin UDP



Hình I.7: UDP Header

UDP header gồm những trường sau:

- Source Port
- Destination Port
- Message Length
- Checksum
- Next Level Protocol

Cũng giống như TCP, UDP theo dõi các kết nối bằng cách sử dụng các port từ 1024->65536, các port UDP từ 0->1023 là các port dành riêng cho các ứng dụng phổ biến, một số dùng phổ biến như:

Port	**Mô Tả**
53	Domain Name System
69	Trivial File Transfer Protocol
111	Remote Procedure Call
137	NetBIOS name service
138	NetBIOS datagram

Port	**Mô Tả**
161	Simple Network Management Protocol

CHƯƠNG II: LẬP TRÌNH SOCKET HƯỚNG KẾT NỐI

II.1. Socket

Trong lập trình mạng dùng Socket, chúng ta không trực tiếp truy cập vào các thiết bị mạng để gởi và nhận dữ liệu. Thay vì vậy, một file mô tả trung gian được tạo ra để điều khiển việc lập trình. Các file mô tả dùng để tham chiếu đến các kết nối mạng được gọi là các Socket. Socket định nghĩa những đặc trưng sau:

- Một kết nối mạng hay một đường ống dẫn để truyền tải dữ liệu
- Một kiểu truyền thông như stream hay datagram
- Một giao thức như TCP hay UDP

Sau khi một Socket được tạo ra nó phải được gắn vào một địa chỉ mạng và một port trên hệ thống cục bộ hay ở xa. Một khi Socket đã được gắn vào các địa chỉ mạng và port, nó có thể được dùng để gởi và nhận dữ liệu trong mạng.

Trong .Net Framework lớp Socket hỗ trợ cho việc lập trình Socket. Phương thức tạo lập như sau:

```
Socket (AddressFamily, SocketType, ProtocolType)
```

Phương thức tạo lập của lớp Socket cần các đối số truyền vào sau:

+AddressFamily: họ địa chỉ được dùng, tham số này có thể có các giá trị sau:

AppleTalk	Địa chỉ AppleTalk
Atm	Native ATM services address.
Banyan	Địa chỉ Banyan
Ccitt	Địa chỉ cho giao thức CCITT, như là X25
Chaos	Địa chỉ cho giao thức MIT CHAOS
Cluster	Địa chỉ cho các sản phẩm cluster của Microsoft
DataKit	Địa chỉ cho giao thức Datakit
DataLink	Địa chỉ của giao thức tầng data-link
DecNet	Địa chỉ DECnet

Ecma	Địa chỉ ECMA (European Computer Manufacturers Association)
FireFox	Địa chỉ FireFox
HyperChannel	Địa chỉ NSC Hyperchannel
Ieee12844	Địa chỉ workgroup IEEE 1284.4
ImpLink	Địa chỉ ARPANET IMP
InterNetwork	Địa chỉ IP version 4
InterNetworkV6	Địa chỉ IP version 6
Ipx	Địa chỉ IPX hoặc SPX
Irda	Địa chỉ IrDA
Iso	Địa chỉ cho giao thức ISO
Lat	Địa chỉ LAT
Max	Địa chỉ MAX
NetBios	Địa chỉ NetBios
NetworkDesigners	Địa chỉ Network Designers
NS	Địa chỉ Xerox NS
Osi	Địa chỉ cho giao thức ISO
Pup	Địa chỉ cho giao thức PUP
Sna	Địa chỉ IBM SNA
Unix	Địa chỉ Unix
Unknown	Chưa biết họ địa chỉ
Unspecified	Chưa chỉ ra họ địa chỉ
VoiceView	Địa chỉ VoiceView

+SocketType: kiểu Socket, tham số này có thể có các giao thức sau:

Kiểu	Mô tả
Dgram	Được sử dụng trong các giao thức phi kết nối, không tin tưởng. Thông điệp có thể bị mất, bị trùng lặp hoặc có thể đến sai thứ tự.

	Dgram sử dụng giao thức **UDP** và họ địa chỉ InterNetwork.
Raw	Được sử trong các giao thức cấp thấp như Internet Control Message Protocol (Icmp) và Internet Group Management Protocol (Igmp). Ứng dụng phải cung cấp IP header khi gởi. Khi nhận sẽ nhận được IP header và các tùy chọn tương ứng.
Rdm	Được sử dụng trong các giao thức phi kết nối, hướng thông điệp, truyền thông điệp tin cậy, và biên của thông điệp được bảo vệ. Rdm (Reliably Delivered Messages) thông điệp đến không bị trùng lặp và đúng thứ tự. Hơn nữa, thiết bị nhận được thiết bị nếu thông điệp bị mất. Nếu khởi tạo **Socket** dùng Rdm, ta không cần yêu cầu kết nối tới host ở xa trước khi gởi và nhận dữ liệu.
Seqpacket	Cung cấp hướng kết nối và truyền 2 chiều các dòng byte một cách tin cậy . Seqpacket không trùng lập dữ liệu và bảo vệ biên dữ liệu. **Socket** kiểu **Seqpacket** truyền thông với 1 máy đơn và yêu cầu kết nối trước khi truyền dữ liệu.
Stream	Được sử dụng trong các giao thức hướng kết nối, không bị trùng lặp dữ liệu, không bảo vệ biên dữ liệu. **Socket** kiểu Stream chỉ truyền thông với một máy đơn và yêu cầu kết nối trước khi truyền dữ liệu. **Stream** dùng giao thức Transmission Control Protocol (**Tcp**) và họ địa chỉ **InterNetwork**
Unknown	Chưa biết kiểu **Socket**

+ProtocolType: kiểu giao thức, tham số này có thể có các giá trị sau:

ProtocolType	Mô tả
Ggp	Gateway To Gateway Protocol.
Icmp	Internet Control Message Protocol.
IcmpV6	Internet Control Message Protocol IPv6.
Idp	Internet Datagram Protocol.
Igmp	Internet Group Management Protocol.
IP	Internet Protocol.
IPSecAuthenticationHeader	IPv6 Authentication.
IPSecEncapsulatingSecurityPayload	IPv6 Encapsulating Security Payload header.

IPv4	Internet Protocol version 4.
IPv6	Internet Protocol version 6 (IPv6).
Ipx	Internet Packet Exchange Protocol.
ND	Net Disk Protocol (unofficial).
Pup	PARC Universal Packet Protocol.
Raw	Raw IP packet protocol.
Spx	Sequenced Packet Exchange protocol.
SpxII	Sequenced Packet Exchange version 2 protocol.
Tcp	Transmission Control Protocol.
Udp	User Datagram Protocol.
Unknown	Giao thức chưa biết
Unspecified	Giao thức chưa được chỉ ra

Ví dụ phương thức tạo lập của lớp Socket:

```
Socket sk = Socket(AddressFamily.InterNetwork, SocketType.Stream,
ProtocolType.Tcp);
```

II.2. IPAddress

IPAddress là một đối tượng dùng để mô tả một địa chỉ IP, đối tượng này có thể được sử dụng trong nhiều phương thức của Socket. Một số phương thức của lớp IPAddress

Phương Thức	**Mô Tả**
Equals	So sánh 2 địa chỉ IP
GetHashCode	Lấy giá trị has cho 1 đối tượng IPAddress
GetType	Trả về kiểu của một thể hiện địa chỉ IP
HostToNetworkOrder	Chuyển 1 địa chỉ IP từ host byte order thành network byte order
IsLoopBack	Cho biết địa chỉ IP có phải là địa chỉ LoopBack hay không
NetworkToHostOrder	Chuyển 1 địa chỉ IP từ network byte order thành host byte order

Phương Thức	**Mô Tả**
Parse	Chuyển 1 chuỗi thành 1 thể hiện IPAddress
ToString	Chuyển 1 đối tượng IPAddress thành một chuỗi

Phương thức Parse() thường được dùng để tạo ra 1 thể hiện của IPAddress:

```
IPAddress localIpAddress = IPAddress.Parse("127.0.0.1");
```

Lớp IPAddress cũng cung cấp 4 thuộc tính để mô tả các địa chỉ IP đặc biệt:

- Any: dùng để mô tả một địa chỉ IP bất kỳ của hệ thống.
- Broadcast: dùng để mô tả địa chỉ IP Broadcast cho mạng cục bộ
- Loopback: dùng để mô tả địa chỉ loopback của hệ thống
- None: không dùng địa chỉ IP

II.3. IPEndPoint

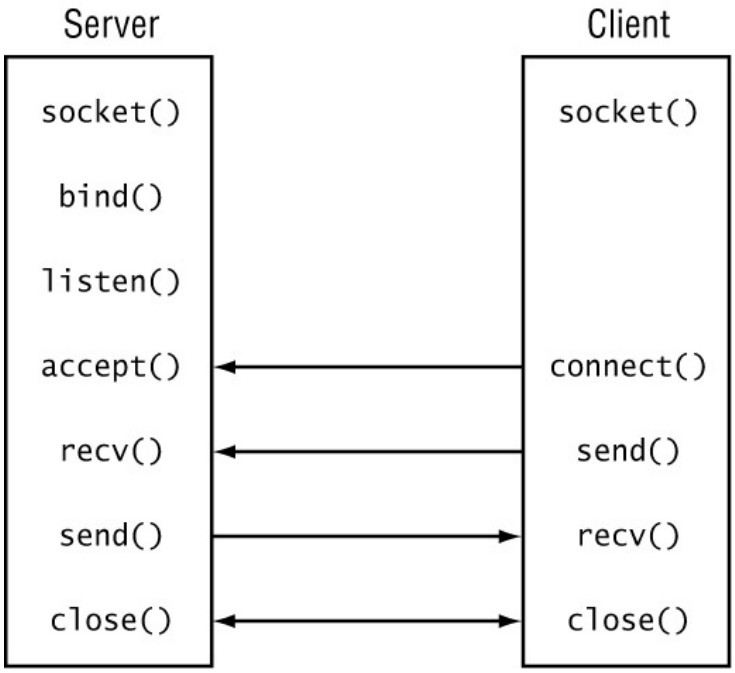
IPEndPoint là một đối tượng mô tả sự kết hợp của một địa chỉ IP và port. Đối tượng IPEndPoint được dùng để gắn kết các Socket với các địa chỉ cục bộ hoặc các địa chỉ ở xa.

Hai thuộc tính của IPEndPoint có thể được dùng để lấy được vùng các port trên hệ thống là MinPort và MaxPort.

II.4. Lập trình Socket hướng kết nối

Trong lập trình Socket hướng kết nối, giao thức TCP được dùng để thành lập phiên làm việc giữa hai endpoint. Khi sử dụng giao thức TCP để thành lập kết nối ta phải đàm phán kết nối trước nhưng khi kết nối đã được thành lập dữ liệu có thể truyền đi giữa các thiết bị một cách tin tưởng.

Để lập trình Socket hướng kết nối ta phải thực hiện một loạt các thao tác giữa clien và Server như trong mô hình bên dưới



Hình II.1: Mô hình lập trình Socket hướng kết nối

II.4.1. Lập trình phía Server

Đầu tiên Server sẽ tạo một Socket, Socket này sẽ được gắn vào một địa chỉ ip và một port cục bộ, hàm để thực hiện việc này là hàm Bind(). Hàm này cần một danh đối số là một IPEndPoint cục bộ:

```
IPEndPoint ipep = new IPEndPoint(IPAddress.Any, 5000);
Socket server = new Socket(AddressFamily.InterNetwork,
SocketType.Stream, ProtocolType.Tcp);
server.Bind(ipep);
```

Bởi vì Server thường chấp nhận kết nối trên chính địa chỉ IP và port riêng của nó nên ta dùng `IPAddress.Any` để chấp nhận kết nối trên bất kỳ card mạng nào

Địa chỉ IP ta dùng ở đây là địa chỉ IP version 4 và kiểu giao thức là TCP nên AddressFamily là InterNetwork và SocketType là Stream.

Sau khi Socket đã được gắn kết vào một địa chỉ và một port, Server phải sẵn sàng chấp nhận kết nối từ Client. Việc này được thực hiện nhờ vào hàm Listen().Hàm Listen() có một đối số, đó chính là số Client tối đa mà nó lắng nghe.

```
server.Listen(10);
```

Tiếp theo Server dùng hàm Accept() để chấp nhận kết nối từ Client:

```
Socket client = server.Accept();
```

Hàm Accept() này sẽ dừng Server lại và chờ cho đến khi nào có Client kết nối đến nó sẽ trả về một Socket khác, Socket này được dùng để trao đổi dữ liệu với Client. Khi đã chấp nhận kết nối với Client thì Server có thể gởi và nhận dữ liệu với Client thông qua phương thức Send() và Receive().

```
string welcome = "Hello Client";
buff = Encoding.ASCII.GetBytes(welcome);
client.Send(buff, buff.Length, SocketFlags.None);
```

Phương thức Send() của Socket dùng để gởi dữ liệu, phương thức này có một số đối số quan trọng sau:

- ✓ Buff : mảng các byte cần gởi
- ✓ Offset: vị trí đầu tiên trong mảng cần gởi
- ✓ Size: số byte cần gởi
- ✓ SocketFlags: chỉ ra cách gởi dữ liệu trên Socket

Việc gởi và nhận dữ liệu được thực hiện liên tục thông qua một vòng lặp vô hạn:

```
while (true)
{
      buff = new byte[1024];
      recv = client.Receive(buff);
      if (recv == 0)
           break;

      Console.WriteLine(Encoding.ASCII.GetString(buff, 0, recv));
      client.Send(buff, recv, SocketFlags.None);
}
```

Phương thức Receive() đặt dữ liệu vào buffer, kích thước buffer được thiết lập lại, do đó nếu buffer không được thiết lập lại, lần gọi phương thức Receive() kế tiếp sẽ chỉ có thể nhận được dữ liệu tối đa bằng lần nhận dữ liệu trước.

Phương thức này có một số đối số quan trọng sau:

- ✓ Buff : mảng các byte cần gởi
- ✓ Offset: vị trí đầu tiên trong mảng cần nhận
- ✓ Size: số byte cần gởi
- ✓ SocketFlags: chỉ ra cách nhận dữ liệu trên Socket

Phương thức Receive() trả về số byte dữ liệu nhận được từ Client. Nếu không có dữ liệu được nhận, phương thức Receive() sẽ bị dừng lại và chờ cho tới khi có dữ liệu. Khi Client gởi tín hiệu kết thúc phiên làm việc (bằng cách gởi cờ FIN trong gói TCP), phương thức Receive() sẽ trả về giá trị 0. Khi phương thức Receive() trả về giá trị 0, ta đóng Socket của Client lại bằng phương thức Close(). Socket chính (Server Socket) vẫn còn hoạt động để chấp nhận các kết nối khác. Nếu không muốn Client nào kết nối đến nữa thì ta đóng Server lại luôn:

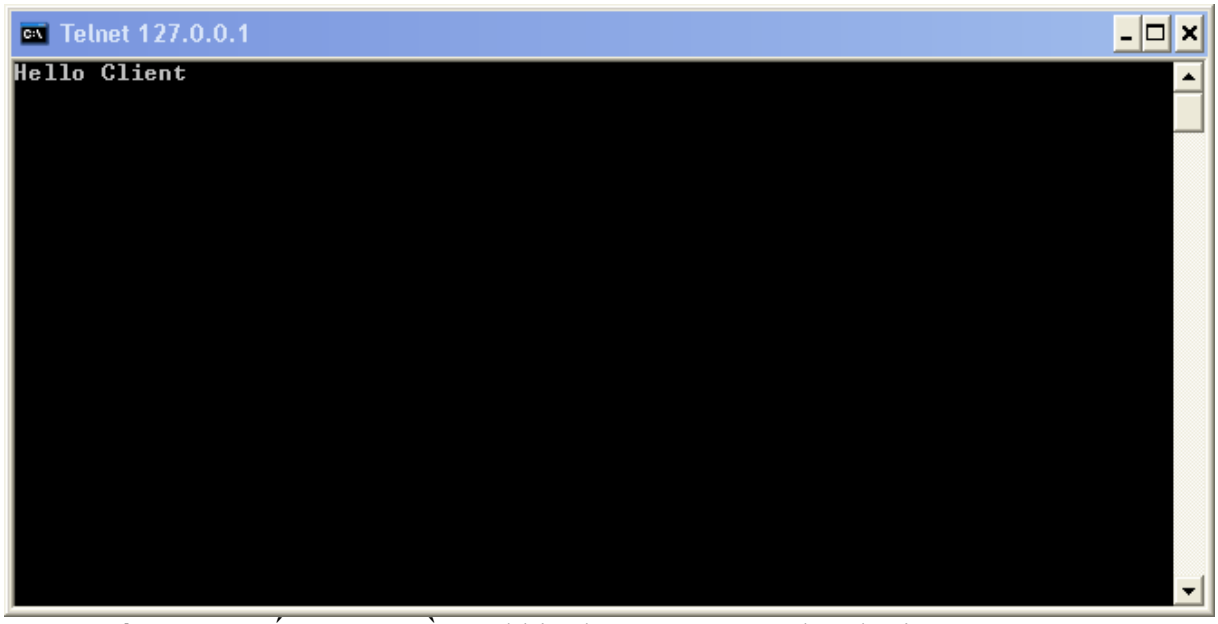
```
client.Close();
server.Close();
```

Chương trình TCP Server đơn giản:

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
class TcpServerDonGian
{
    public static void Main()
    {
        //Số byte thực sự nhận được dùng hàm Receive()
        int byteReceive;
        //buffer để nhận và gởi dữ liệu
        byte[] buff = new byte[1024];
        //EndPoint cục bộ
        IPEndPoint ipep = new IPEndPoint(IPAddress.Any, 5000);
        //Server Socket
        Socket server = new Socket(AddressFamily.InterNetwork,
SocketType.Stream, ProtocolType.Tcp);
        //Kết nối server với 1 EndPoint
        server.Bind(ipep);
        //Server lắng nghe tối đa 10 kết nối
        server.Listen(10);
        Console.WriteLine("Dang cho Client ket noi den...");
        //Hàm Accept() sẽ block server lại cho đến khi có Client kết nối đến
        Socket client = server.Accept();
        //Client EndPoint
        IPEndPoint clientep = (IPEndPoint)client.RemoteEndPoint;
        Console.WriteLine("Da ket noi voi Client {0} tai port {1}",
clientep.Address, clientep.Port);

        string welcome = "Hello Client";
```

```
            //Chuyển chuỗi thành mảng các byte
            buff = Encoding.ASCII.GetBytes(welcome);
            //Gởi câu chào cho Client
            client.Send(buff, buff.Length, SocketFlags.None);
            while (true)
            {
                //Reset lại buffer
                buff = new byte[1024];
                //Lấy số byte thực sự nhận được
                byteReceive = client.Receive(buff);
                //Nếu Client ngắt kết nối thì thoát khỏi vòng lặp
                if (byteReceive == 0)
                    break;

                Console.WriteLine(Encoding.ASCII.GetString(buff, 0, byteReceive));
                //Sau khi nhận dữ liệu xong, gởi lại cho Client
                client.Send(buff, byteReceive, SocketFlags.None);
            }
            Console.WriteLine("Da dong ket noi voi Client: {0}", clientep.Address);
            //Đóng kết nối
            client.Close();
            server.Close();
        }
}
```

Để kiểm tra thử chương trình ta có thể dùng chương trình Telnet của Windows để kiểm tra. Dùng lệnh telnet 127.0.0.1 5000



Hình II.2: Kết quả trả về sau khi telnet vào Server local tại port 5000

Sau khi dùng lệnh telnet, kết quả trả về như trên hình là đã kết nối thành công

II.4.2. Lập trình phía Client

Lập trình Socket hướng kết nối phía Client đơn giản hơn phía Server. Client cũng phải gắn kết một địa chỉ của một Socket đã được tạo ra nhưng sử dụng phương thức Connect() chứ không sử dụng phương thức Bind() giống như phía Server. Phương thức Connect() yêu cầu một IPEndPoint của Server mà Client cần kết nối đến.

```
IPEndPoint ipep = new IPEndPoint(IPAddress.Parse("127.0.0.1"),
5000);
Socket server = new Socket(AddressFamily.InterNetwork,
SocketType.Stream, ProtocolType.Tcp);

try
{
        server.Connect(ipep);
}
catch (SocketException e)
{
        Console.WriteLine("Không thể kết nối đến Server");
        Console.WriteLine(e.ToString());
        return;
}
```

Phương thức Connect() sẽ dừng lại cho đến khi Client kết nối được với Server. Nếu kết nối không thể được thực hiện thì nó sẽ phát sinh ra một biệt lệ, do đó hàm Connect() tra phải để trong khối try, catch để không bị lỗi chương trình.

Khi kết nối được thành lập, Client có thể dùng phương thức Send() và Receive() của lớp Socket để gởi và nhận dữ liệu tương tự như Server đã làm. Khi quá trình trao đổi dữ liệu đã hoàn tất, đối tượng Socket phải được đóng lại. Client Socket dùng phương thức Shutdown() để dừng Socket và dùng phương thức Close() để thực sự đóng phiên làm việc. Phương thức Shutdown() của Socket dùng một tham số để quyết định cách Socket sẽ dừng lại. Các phương thức đó là:

Giá trị	**Mô tả**

Giá trị	Mô tả
SocketShutdown.Both	Ngăn cản gởi và nhận dữ liệu trên Socket.
SocketShutdown.Receive	Ngăn cản nhận dữ liệu trên Socket. Cờ RST sẽ được gởi nếu có thêm dữ liệu được nhận.
SocketShutdown.Send	Ngăn cản gởi dữ liệu trên Socket. Cờ FIN sẽ được gởi sau khi tất cả dữ liệu còn lại trong buffer đã được gởi đi.

Chương trình TCP Client đơn giản:

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
class SimpleTcpClient
{
    public static void Main()
    {
        //Buffer để gởi và nhận dữ liệu
        byte[] buff = new byte[1024];
        //Chuỗi nhập vào và chuỗi nhận được
        string input, stringData;
        //IPEndPoint ở server
        IPEndPoint ipep = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 5000);
        //Server Socket
        Socket server = new Socket(AddressFamily.InterNetwork,
SocketType.Stream, ProtocolType.Tcp);
        //Hàm Connect() sẽ bị block lại và chờ khi kết nối được với server thì
mới hết block
        try
        {
            server.Connect(ipep);
        }
        //Quá trình kết nối có thể xảy ra lỗi nên phải dùng try, catch
        catch (SocketException e)
        {
            Console.WriteLine("Không thể kết nối đến Server");
            Console.WriteLine(e.ToString());
            return;
        }
        //Số byte thực sự nhận được
        int byteReceive = server.Receive(buff);
        //Chuỗi nhận được
        stringData = Encoding.ASCII.GetString(buff, 0, byteReceive);
```

```
            Console.WriteLine(stringData);
            while (true)
            {
                //Nhập dữ liệu từ bàn phím
                input = Console.ReadLine();
                //Nếu nhập exit thì thoát và đóng Socket
                if (input == "exit")
                    break;
                //Gởi dữ liệu cho server
                server.Send(Encoding.ASCII.GetBytes(input));
                //Reset lại buffer
                buff = new byte[1024];
                //Số byte thực sự nhận được
                byteReceive = server.Receive(buff);
                //Chuỗi nhận được
                stringData = Encoding.ASCII.GetString(buff, 0, byteReceive);
                Console.WriteLine(stringData);
            }
            Console.WriteLine("Dong ket noi voi server...");
            //Dừng kết nối, không cho phép nhận và gởi dữ liệu
            server.Shutdown(SocketShutdown.Both);
            //Đóng Socket
            server.Close();
        }
    }
```

II.4.3. Vấn đề với bộ đệm dữ liệu

Trong ví dụ Client, Server đơn giản trên thì một mảng các byte được dùng như là bộ đệm để gởi và nhận dữ liệu trên Socket. Bởi vì chương trình được chạy trong môi trường được điều khiển, tất cả các thông điệp đều thuộc dạng text và kích thước nhỏ nên loại buffer này không phải là một vấn đề.

Trong thế giới thực, chúng ta không biết kích thước và kiểu dữ liệu đến trong khi truyền thông giữa Client và Server. Vấn đề xảy ra khi khi dữ liệu đến lớn hơn kích thước bộ đệm dữ liệu.

Khi nhận dữ liệu thông qua TCP, dữ liệu được lưu trữ trong bộ đệm hệ thống. Mỗi khi gọi phương thức Receive(), nó sẽ đọc dữ liệu từ bộ đệm TCP và lấy dữ liệu ra khỏi bộ đệm. Số lượng dữ liệu được đọc bởi phương thức Receive() được điều khiển bởi hai yếu tố sau:

- Kích thước bộ đệm dữ liệu được chỉ ra trong phương thức Receive()
- Kích thước bộ đệm được chỉ ra trong tham số của phương thức Receive()

Trong ví dụ đơn giản trên, buffer được định nghĩa là một mảng byte kích thước 1024. Bởi vì kích thước dữ liệu không được chỉ ra trong phương thức Receive() nên kích thước bộ đệm tự động lấy kích thước mặc định của bộ đệm dữ liệu là 1024 byte. Phương thức Receive() sẽ đọc 1024 byte dữ liệu một lần và đặt dữ liệu đọc được vào biến buff

```
byteReceive = client.Receive(buff);
```

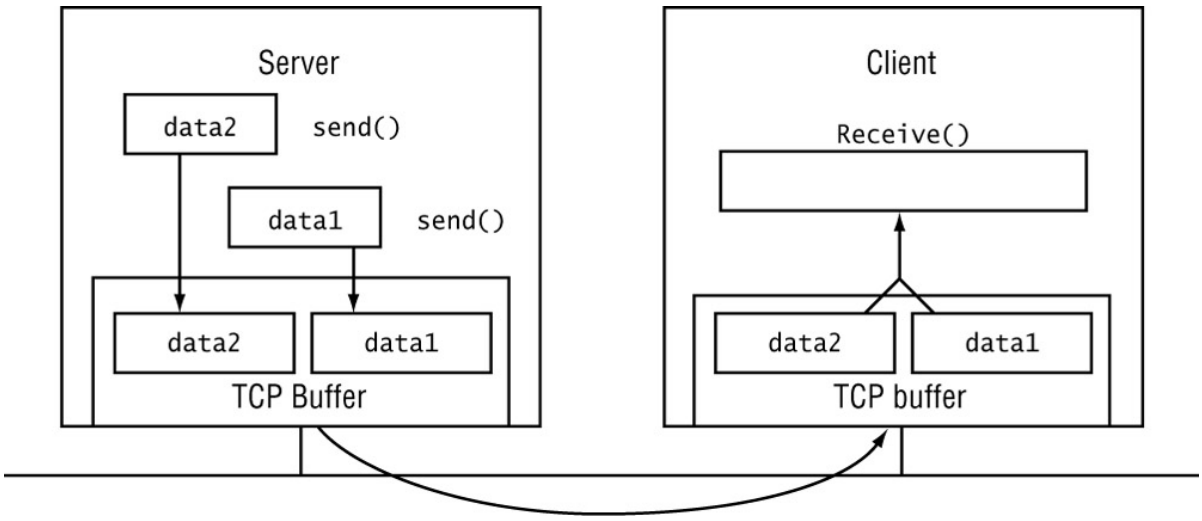
Vào lúc phương thức Receive() được gọi, nếu bộ đệm TCP chứa ít hơn 1024 byte, phương thức này sẽ trả về số lượng dữ liệu mà nó thực sự đọc được trong biến byte Receive. Để chuyển dữ liệu thành chuỗi, ta dùng phương thức GetString() như sau:

```
stringData = Encoding.ASCII.GetString(buff, 0, byteReceive);
```

Trong đối số của hàm GetString, ta phải truyền vào số byte thực sự đã đọc được nếu không ta sẽ nhận được một chuỗi với các byte thừa ở đằng sau.

II.4.4. Xử lý với các bộ đệm có kích thước nhỏ

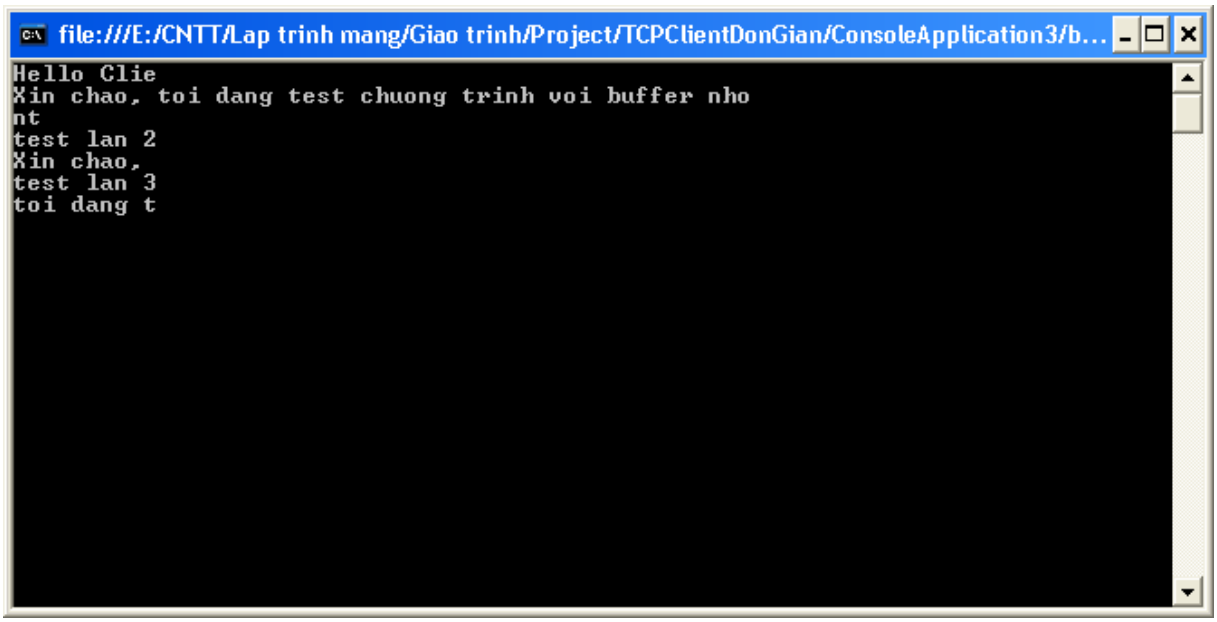
Hệ điều hành Window dùng bộ đệm TCP để gởi và nhận dữ liệu. Điều này là cần thiết để TCP có thể gởi lại dữ liệu bất cứ lúc nào cần thiết. Một khi dữ liệu đã được hồi báo nhận thành công thì nó mới được xóa khỏi bộ đệm.



Hình II.3: TCP Buffer

Dữ liệu đến cũng được hoạt động theo cách tương tự. Nó sẽ ở lại trong bộ đệm cho đến khi phương thức Receive() được dùng để đọc nó. Nếu phương thức Receive() không đọc toàn bộ dữ liệu ở trong bộ đệm, phần còn lại vẫn được nằm ở đó và chờ phương thức Receive() tiếp theo được đọc. Dữ liệu sẽ không bị mất nhưng chúng ta sẽ không lấy được các đoạn dữ liệu mình mong muốn.

Để thấy được vấn đề, ta tiến hành thay đổi kích thước bộ đệm từ 1024 byte xuống còn 10 byte. Và chạy lại chương trình Client, Server đơn giản trên



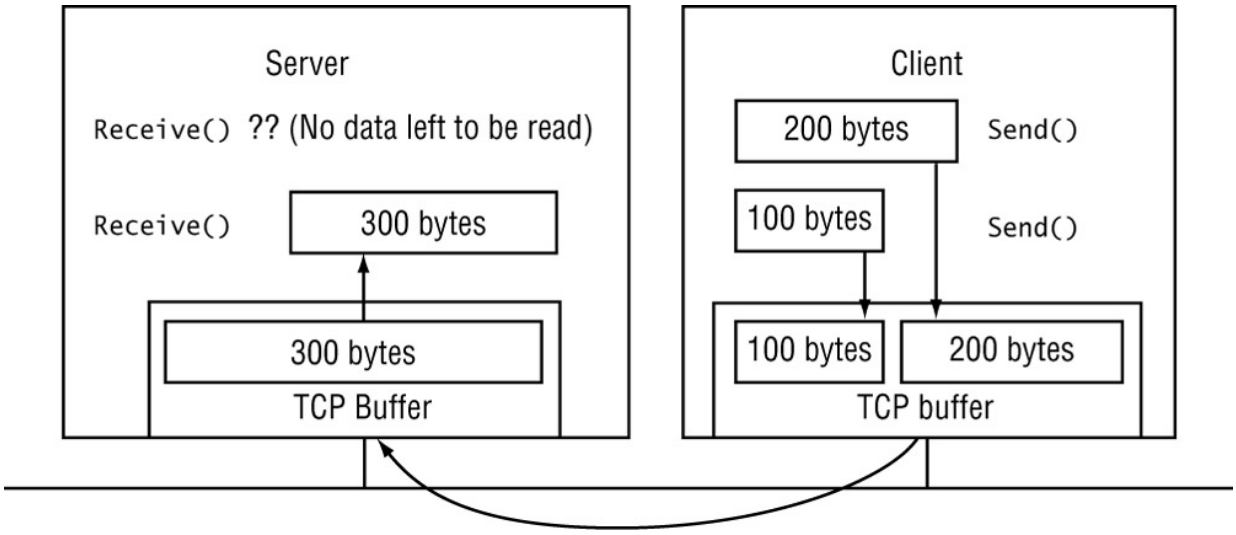
Hình II.4: Kết quả trả về khi chạy chương trình với buffer nhỏ

Bởi vì bộ đệm dữ liệu không đủ lớn để lấy hết dữ liệu ở bộ đệm TCP nên phương thức Receive() chỉ có thể lấy được một lượng dữ liệu có độ lớn đúng bằng độ lớn của bộ đệm dữ liệu, phần còn lại vẫn nằm ở bộ đệm TCP và nó được lấy khi gọi lại phương thức Receive(). Do đó câu chào Client của Server phải dùng tới hai lần gọi phương thức Receive() mới lấy được hết. Trong lần gởi và nhận dữ liệu kế tiếp, đoạn dữ liệu tiếp theo được đọc từ bộ đệm TCP do đó nếu ta gởi dữ liệu với kích thước lớn hơn 10 byte thì khi nhận ta chỉ nhận được 10 byte đầu tiên.

Bởi vì vậy nên trong khi lập trình mạng chúng ta phải quan tâm đến việc đọc dữ liệu từ bộ đệm TCP một cách chính xác. Bộ đệm quá nhỏ có thể dẫn đến tình trạng thông điệp nhận sẽ không khớp với thông điệp gởi, ngược lại bộ đệm quá lớn sẽ làm cho các thông điệp bị trộn lại, khó xử lý. Việc khó nhất là làm sao phân biệt được các thông điệp được đọc từ Socket.

II.4.5. Vấn đề với các thông điệp TCP

Một trong những khó khăn của những nhà lập trình mạng khi sử dụng giao thức TCP để chuyển dữ liệu là giao thức này không quan tâm đến biên dữ liệu.



Hình II.5: Client Send hai lần rồi Server mới Receive

Như trên hình vấn đề xảy ra khi truyền dữ liệu là không đảm bảo được mỗi phương thức Send() sẽ không được đọc bởi một phương thức Receive(). Tất cả dữ liệu được đọc từ phương thức Receive() không thực sự được đọc trực tiếp từ mạng mà nó được đọc từ bộ đệm TCP. Khi các gói tin TCP được nhận từ mạng sẽ được đặt theo thứ tự trong bộ đệm TCP. Mỗi khi phương thức Receive() được gọi, nó sẽ đọc dữ liệu trong bộ đệm TCP, không quan tâm đến biên dữ liệu.

Chúng ta hãy xem xét ví dụ sau, Chương Trình BadTCPServer:

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
class BadTcpServer
{
    public static void Main()
    {
        //Số byte thực sự nhận được dùng hàm Receive()
        int byteReceive;
        //buffer để nhận và gởi dữ liệu
        byte[] buff = new byte[1024];
        //EndPoint cục bộ
        IPEndPoint ipep = new IPEndPoint(IPAddress.Any, 5000);
        //Server Socket
```

```
        Socket server = new Socket(AddressFamily.InterNetwork,
SocketType.Stream, ProtocolType.Tcp);
        //Kết nối server với 1 EndPoint
        server.Bind(ipep);
        //Server lắng nghe tối đa 10 kết nối
        server.Listen(10);
        Console.WriteLine("Dang cho Client ket noi den...");
        //Hàm Accept() sẽ block server lại cho đến khi có Client kết nối đến
        Socket client = server.Accept();
        //Client EndPoint
        IPEndPoint clientep = (IPEndPoint)client.RemoteEndPoint;
        Console.WriteLine("Da ket noi voi Client {0} tai port {1}",
clientep.Address, clientep.Port);

        string welcome = "Hello Client";
        //Chuyển chuỗi thành mảng các byte
        buff = Encoding.ASCII.GetBytes(welcome);
        //Gởi câu chào cho Client
        client.Send(buff, buff.Length, SocketFlags.None);

        for (int i = 0; i < 5; i++)
        {
            byteReceive = client.Receive(buff);
            Console.WriteLine(Encoding.ASCII.GetString(buff, 0, byteReceive));
        }
        Console.WriteLine("Da dong ket noi voi Client: {0}", clientep.Address);
        //Đóng kết nối
        client.Close();
        server.Close();
        Console.Read();

    }
}
```

Chương trình Server thành lập Socket TCP bình thường để lắng nghe kết nối, khi kết nối được thành lập, Server gởi câu chào cho Client và cố gắng nhận năm thông điệp riêng biệt từ Client:

```
for (int i = 0; i < 5; i++)
{
    byteReceive = client.Receive(data);
    Console.WriteLine(Encoding.ASCII.GetString(data, 0,
byteReceive));
```

```
}
```

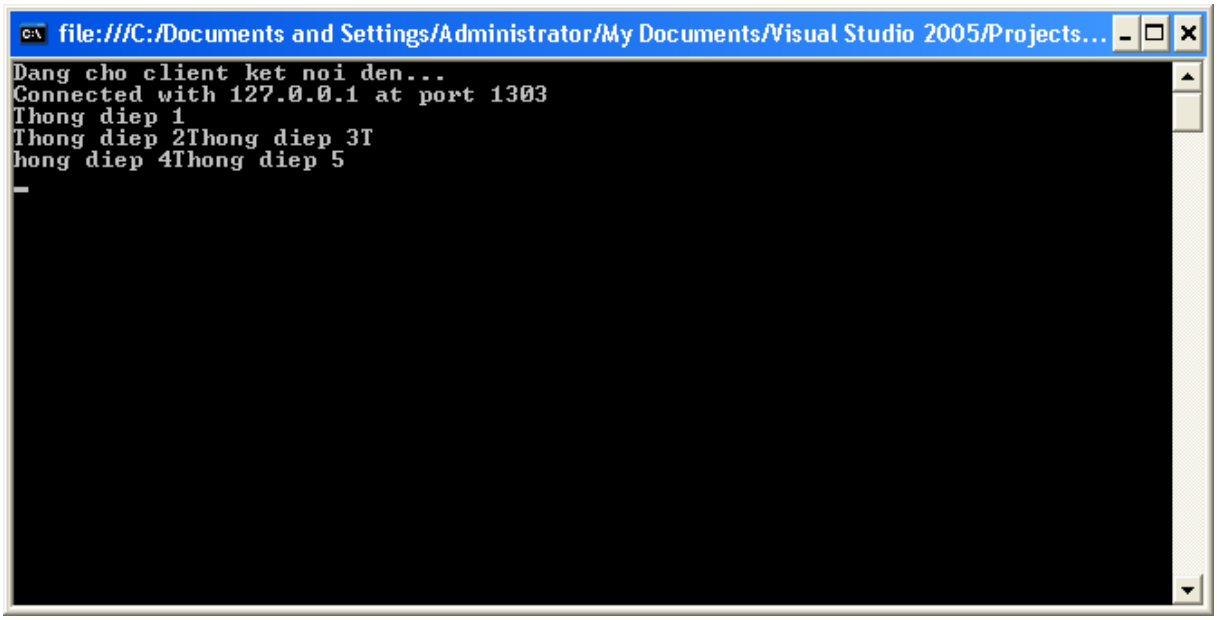
Mỗi khi được gọi, phương thức Receive() đọc toàn bộ dữ liệu trong bộ đệm TCP, sau khi nhận năm thông điệp, kết nối được đóng lại.

Chương trình BadTCPClient:

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
class BadTcpClient
{
    public static void Main()
    {
        //Buffer để gởi và nhận dữ liệu
        byte[] buff = new byte[10];
        //Chuỗi nhập vào và chuỗi nhận được
        string input, stringData;
        //IPEndPoint ở server
        IPEndPoint ipep = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 5000);
        //Server Socket
        Socket server = new Socket(AddressFamily.InterNetwork,
SocketType.Stream, ProtocolType.Tcp);
        //Hàm Connect() sẽ bị block lại và chờ khi kết nối được với server thì
mới hết block
        try
        {
            server.Connect(ipep);
        }
        //Quá trình kết nối có thể xảy ra lỗi nên phải dùng try, catch
        catch (SocketException e)
        {
            Console.WriteLine("Khon the ket noi den Server");
            Console.WriteLine(e.ToString());
            return;
        }
        //Số byte thực sự nhận được
        int byteReceive = server.Receive(buff);
        //Chuỗi nhận được
        stringData = Encoding.ASCII.GetString(buff, 0, byteReceive);
        Console.WriteLine(stringData);

        server.Send(Encoding.ASCII.GetBytes("Thong diep 1"));
        server.Send(Encoding.ASCII.GetBytes("Thong diep 2"));
```

```
            server.Send(Encoding.ASCII.GetBytes("Thong diep 3"));
            server.Send(Encoding.ASCII.GetBytes("Thong diep 4"));
            server.Send(Encoding.ASCII.GetBytes("Thong diep 5"));

            Console.WriteLine("Dong ket noi voi server...");
            //Dừng kết nối, không cho phép nhận và gởi dữ liệu
            server.Shutdown(SocketShutdown.Both);
            //Đóng Socket
            server.Close();
            Console.Read();

        }
}
```

Kết quả chương trình như hình bên dưới



Hình II.6: Kết quả trên Server

Trong lần gọi phương thức Receive() lần đầu tiên, phương thức này nhận toàn bộ dữ liệu từ phương thức Send() của Client gởi lên, trong lần gọi phương thức Receive() lần thứ hai, phương thức Receive() đọc dữ liệu từ hai phương thức Send() và một phương thức Send() khác gởi dữ liệu chưa xong. Trong lần gọi thứ ba thì phương thức Receive() sẽ đọc hết dữ liệu đang được gởi dở từ phương thức Send() và đọc dữ liệu được gởi từ phương thức Send() cuối cùng và sau khi Client thực hiện xong năm phương thức Send() nó sẽ đóng kết nối với Server và Server cũng sẽ thoát ra.

II.4.6. Giải quyết các vấn đề với thông điệp TCP

Để giải quyết vấn đề với biên dữ liệu không được bảo vệ, chúng ta phải tìm hiểu một số kỹ thuật để phân biệt các thông điệp. Ba kỹ thuật thông thường dùng để phân biệt các thông điệp được gởi thông qua TCP:

- ✓ Luôn luôn sử dụng các thông điệp với kích thước cố định
- ✓ Gởi kèm kích thước thông điệp cùng với mỗi thông điệp
- ✓ Sử dụng các hệ thống đánh dấu để phân biệt các thông điệp

II.4.6.1. Sử dụng các thông điệp với kích thước cố định

Cách dễ nhất nhưng cũng là cách tốn chi phí nhất để giải quyết vấn đề với các thông điệp TCP là tạo ra các giao thức luôn luôn truyền các thông điệp với kích thước cố định. Bằng cách thiết lập tất cả các thông điệp có cùng kích thước, chương trình TCP nhận có thể biết toàn bộ thông điệp được gởi từ Client.

Khi gởi dữ liệu với kích thước cố định, chúng ta phải đảm bảo toàn bộ thông điệp được gởi từ phương thức Send(). Phụ thuộc vào kích thước của bộ đệm TCP và bao nhiêu dữ liệu được truyền, phương thức Send() sẽ trả về số byte mà nó thực sự đã gởi đến bộ đệm TCP. Nếu phương thức Send() chưa gởi hết dữ liệu thì chúng ta phải gởi lại phần dữ liệu còn lại. Việc này thường được thực hiện bằng cách sử dụng vòng lặp while() và trong vòng lặp ta kiểm tra số byte thực sự đã gởi với kích thước cố định.

```
private static int SendData(Socket s, byte[] data)
{
    int total = 0;
    int size = data.Length;
    int dataleft = size;
    int sent;
    while (total < size)
    {
        sent = s.Send(data, total, dataleft, SocketFlags.None);
        total += sent;
        dataleft -= sent;
    }
    return total;
}
```

Cũng giống như việc gởi dữ liệu, chúng ta phải luôn luôn đảm bảo nhận tất cả dữ liệu trong phương thức Receive(). Bằng cách dùng vòng lặp gọi phương thức Receive() chúng ta có thể nhận được toàn bộ dữ liệu mong muốn.

```
private static byte[] ReceiveData(Socket s, int size)
{
    int total = 0;
    int dataleft = size;
    byte[] data = new byte[size];
    int recv;
    while (total < size)
    {
        recv = s.Receive(data, total, dataleft, 0);
        if (recv == 0)
        {
            data = Encoding.ASCII.GetBytes("exit");
            break;
        }
        total += recv;
        dataleft -= recv;
    }
    return data;
```

Phương thức ReceiveData() sẽ đọc dữ liệu với kích thước được đọc là đối số được truyền vào, nếu phương thức Receive() sẽ trả về số byte thực sụ đọc được, nếu số byte thực sự đọc được mà còn nhỏ hơn số byte truyền vào phương thức ReceiveData() thì vòng lặp sẽ tiếp tục cho đến khi số byte đọc được đúng bằng kích thước yêu cầu.

Chương trình Server gởi và nhận dữ liệu với kích thước cố định

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
class FixedTcpSrvr
{
    private static int SendData(Socket s, byte[] data)
    {
        int total = 0;
        int size = data.Length;
```

```
        int dataleft = size;
        int sent;
        while (total < size)
        {
            sent = s.Send(data, total, dataleft, SocketFlags.None);
            total += sent;
            dataleft -= sent;
        }
        return total;
    }
    private static byte[] ReceiveData(Socket s, int size)
    {
        int total = 0;
        int dataleft = size;
        byte[] data = new byte[size];
        int recv;
        while (total < size)
        {
            recv = s.Receive(data, total, dataleft, 0);
            if (recv == 0)
            {
                data = Encoding.ASCII.GetBytes("exit");
                break;
            }
            total += recv;
            dataleft -= recv;
        }
        return data;
    }
    public static void Main()
    {
        byte[] data = new byte[1024];
        IPEndPoint ipep = new IPEndPoint(IPAddress.Any, 5000);
        Socket newsock = new Socket(AddressFamily.InterNetwork,
                SocketType.Stream, ProtocolType.Tcp);
        newsock.Bind(ipep);
        newsock.Listen(10);
        Console.WriteLine("Dang cho Client ket noi den...");
        Socket client = newsock.Accept();
        IPEndPoint newclient = (IPEndPoint)client.RemoteEndPoint;
        Console.WriteLine("Da ket noi voi Client {0} tai port {1}",
                newclient.Address, newclient.Port);
        string welcome = "Hello Client";
```

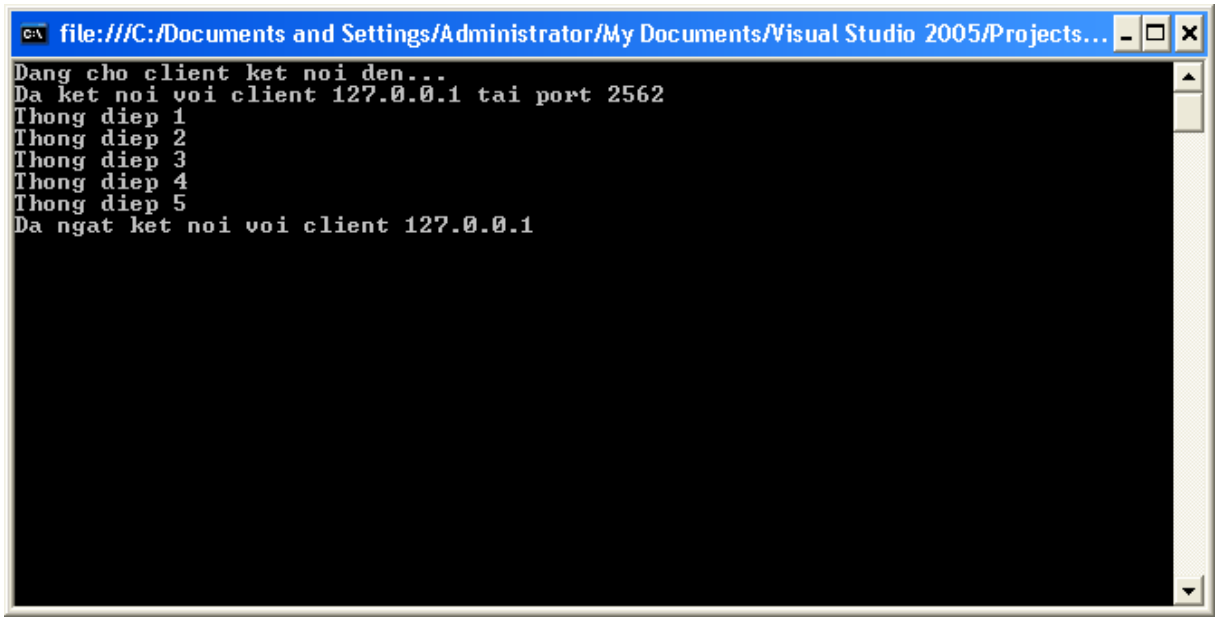
```
            data = Encoding.ASCII.GetBytes(welcome);
            int sent = SendData(client, data);
            for (int i = 0; i < 5; i++)
            {
                data = ReceiveData(client, 12);
                Console.WriteLine(Encoding.ASCII.GetString(data));
            }
            Console.WriteLine("Da ngat ket noi voi Client {0}", newclient.Address);
            client.Close();
            newsock.Close();
        }
}
```

Chương trình Client gởi và nhận dữ liệu với kích thước cố định

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
class FixedTcpClient
{
    private static int SendData(Socket s, byte[] data)
    {
        int total = 0;
        int size = data.Length;
        int dataleft = size;
        int sent;
        while (total < size)
        {
            sent = s.Send(data, total, dataleft, SocketFlags.None);
            total += sent;
            dataleft -= sent;
        }
        return total;
    }
    private static byte[] ReceiveData(Socket s, int size)
    {
        int total = 0;
        int dataleft = size;
        byte[] data = new byte[size];
        int recv;
        while (total < size)
        {
```

```
            recv = s.Receive(data, total, dataleft, 0);
            if (recv == 0)
            {
                data = Encoding.ASCII.GetBytes("exit ");
                break;
            }
            total += recv;
            dataleft -= recv;
        }
        return data;
    }
    public static void Main()
    {
        byte[] data = new byte[1024];
        int sent;
        IPEndPoint ipep = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 5000);
        Socket server = new Socket(AddressFamily.InterNetwork,
                SocketType.Stream, ProtocolType.Tcp);
        try
        {
            server.Connect(ipep);
        }
        catch (SocketException e)
        {
            Console.WriteLine("Khong the ket noi den server");
            Console.WriteLine(e.ToString());
            return;
        }
        int recv = server.Receive(data);
        string stringData = Encoding.ASCII.GetString(data, 0, recv);
        Console.WriteLine(stringData);
        sent = SendData(server, Encoding.ASCII.GetBytes("Thong diep 1"));
        sent = SendData(server, Encoding.ASCII.GetBytes("Thong diep 2"));
        sent = SendData(server, Encoding.ASCII.GetBytes("Thong diep 3"));
        sent = SendData(server, Encoding.ASCII.GetBytes("Thong diep 4"));
        sent = SendData(server, Encoding.ASCII.GetBytes("Thong diep 5"));
        Console.WriteLine("Dong ket noi voi server...");
        server.Shutdown(SocketShutdown.Both);
        server.Close();
    }
}
```

Kết quả trên Server



Hình II.7: Kết quả gởi và nhận dữ liệu với kích thước cố định

II.4.6.2. Gởi kèm kích thước thông điệp cùng với thông điệp

Cách giải quyết vấn đề biên thông điệp của TCP bằng cách sử dụng các thông điệp với kích thước cố định là một giải pháp lãng phí bởi vì tất cả các thông điệp đều phải cùng kích thước. Nếu các thông điệp nào chưa đủ kích thước thì phải thêm phần đệm vào, gây lãng phí băng thông mạng.

Một giải pháp cho vấn đề cho phép các thông điệp được gởi với các kích thước khác nhau là gởi kích thước thông điệp kèm với thông điệp. Bằng cách này thiết bị nhận sẽ biết được kích thước của mỗi thông điệp.

Để thực hiện việc này ta sửa đổi phương thức SendData() trong ví dụ trước

```
private static int SendVarData(Socket s, byte[] buff)
{
    int total = 0;
    int size = buff.Length;
    int dataleft = size;
    int sent;
    byte[] datasize = new byte[4];
    datasize = BitConverter.GetBytes(size);
    sent = s.Send(datasize);
    while (total < size)
    {
        sent = s.Send(buff, total, dataleft, SocketFlags.None);
```

```
            total += sent;
            dataleft -= sent;
        }
        return total;
    }
```

Trong phương thức SendVarData(), ta sẽ lấy kích thước của thông điệp và gắn nó vào đầu của thông điệp, kích thước này là một số interger 4 byte. Kích thước tối đa của mỗi thông điệp này là 65KB. Giá trị interger 4 byte này đầu tiên được chuyển thành mảng các byte, hàm GetBytes() của lớp BitConverter được dùng để thực hiện việc này. Mảng kích thước sau đó được gởi đến thiết bị ở xa, sau khi gởi kích thước thông điệp xong, phần chính của thông điệp được gởi đi, kỹ thuật gởi cũng giống như trong ví dụ trước, chúng ta sẽ lặp cho đến khi tất cả các byte đã được gởi.

Bước tiếp theo là tạo ra một phương thức có thể nhận 4 byte kích thước thông điệp và toàn bộ thông điệp. phương thức ReceiveData() trong ví dụ trước được sửa đổi để thực hiện việc này.

```
    private static byte[] ReceiveVarData(Socket s)
    {
      int total = 0;
      int recv;
      byte[] datasize = new byte[4];
      recv = s.Receive(datasize, 0, 4, 0);
      int size = BitConverter.ToInt32(datasize, 0);
      int dataleft = size;
      byte[] data = new byte[size];
      while (total < size)
      {
          recv = s.Receive(data, total, dataleft, 0);
          if (recv == 0)
          {
              data = Encoding.ASCII.GetBytes("exit ");
              break;
          }
          total += recv;
          dataleft -= recv;
      }
```

```
        return data;
    }
```

Hàm ReceiveVarData() nhận 4 byte đầu tiên của thông điệp và chuyển nó thành giá trị interger bằng phương thức GetInt32() của lớp BitConverter.

Chương trình Server gởi và nhận thông điệp cùng với kích thước

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
class VarTcpSrvr
{
    private static int SendVarData(Socket s, byte[] data)
    {
        int total = 0;
        int size = data.Length;
        int dataleft = size;
        int sent;
        byte[] datasize = new byte[4];
        datasize = BitConverter.GetBytes(size);
        sent = s.Send(datasize);
        while (total < size)
        {
            sent = s.Send(data, total, dataleft, SocketFlags.None);
            total += sent;
            dataleft -= sent;
        }
        return total;
    }
    private static byte[] ReceiveVarData(Socket s)
    {
        int total = 0;
        int recv;
        byte[] datasize = new byte[4];
        recv = s.Receive(datasize, 0, 4, 0);
        int size = BitConverter.ToInt32(datasize, 0);
        int dataleft = size;
        byte[] data = new byte[size];
        while (total < size)
        {
            recv = s.Receive(data, total, dataleft, 0);
            if (recv == 0)
            {
```

```
                data = Encoding.ASCII.GetBytes("exit ");
                break;
            }
            total += recv;
            dataleft -= recv;
        }
        return data;
    }
    public static void Main()
    {
        byte[] data = new byte[1024];
        IPEndPoint ipep = new IPEndPoint(IPAddress.Any, 5000);
        Socket newsock = new Socket(AddressFamily.InterNetwork,
                SocketType.Stream, ProtocolType.Tcp);
        newsock.Bind(ipep);
        newsock.Listen(10);
        Console.WriteLine("Dang cho Client ket noi den...");
        Socket client = newsock.Accept();
        IPEndPoint newclient = (IPEndPoint)client.RemoteEndPoint;
        Console.WriteLine("Da ket noi voi client {0} tai port {1}",
                newclient.Address, newclient.Port);
        string welcome = "Hello client";
        data = Encoding.ASCII.GetBytes(welcome);
        int sent = SendVarData(client, data);
        for (int i = 0; i < 5; i++)
        {
            data = ReceiveVarData(client);
            Console.WriteLine(Encoding.ASCII.GetString(data));
        }
        Console.WriteLine("Dong ket noi voi Client {0}", newclient.Address);
        client.Close();
        newsock.Close();
    }
}
```

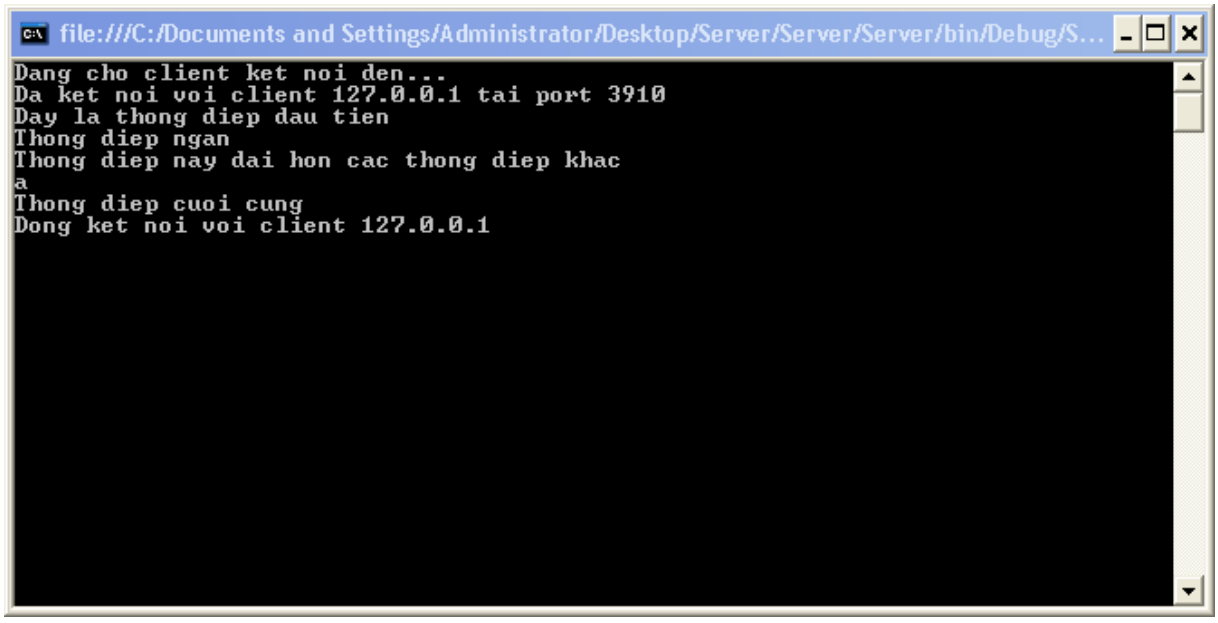
Chương trình Client gởi và nhận thông điệp cùng với kích thước

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
class VarTcpClient
{
```

```
private static int SendVarData(Socket s, byte[] data)
{
    int total = 0;
    int size = data.Length;
    int dataleft = size;
    int sent;
    byte[] datasize = new byte[4];
    datasize = BitConverter.GetBytes(size);
    sent = s.Send(datasize);
    while (total < size)
    {
        sent = s.Send(data, total, dataleft, SocketFlags.None);
        total += sent;
        dataleft -= sent;
    }
    return total;
}
private static byte[] ReceiveVarData(Socket s)
{
    int total = 0;
    int recv;
    byte[] datasize = new byte[4];
    recv = s.Receive(datasize, 0, 4, 0);
    int size = BitConverter.ToInt32(datasize, 0);
    int dataleft = size;
    byte[] data = new byte[size];
    while (total < size)
    {
        recv = s.Receive(data, total, dataleft, 0);
        if (recv == 0)
        {
            data = Encoding.ASCII.GetBytes("exit ");
            break;
        }
        total += recv;
        dataleft -= recv;
    }
    return data;
}
public static void Main()
{
    byte[] data = new byte[1024];
    int sent;
```

```
        IPEndPoint ipep = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 5000);
        Socket server = new Socket(AddressFamily.InterNetwork,
                SocketType.Stream, ProtocolType.Tcp);
        try
        {
            server.Connect(ipep);
        }
        catch (SocketException e)
        {
            Console.WriteLine("Khong the ket noi voi server");
            Console.WriteLine(e.ToString());
            return;
        }
        data = ReceiveVarData(server);
        string stringData = Encoding.ASCII.GetString(data);
        Console.WriteLine(stringData);
        string message1 = "Day la thong diep dau tien";
        string message2 = "Thong diep ngan";
        string message3 = "Thong diep nay dai hon cac thong diep khac";
        string message4 = "a";
        string message5 = "Thong diep cuoi cung";
        sent = SendVarData(server, Encoding.ASCII.GetBytes(message1));
        sent = SendVarData(server, Encoding.ASCII.GetBytes(message2));
        sent = SendVarData(server, Encoding.ASCII.GetBytes(message3));
        sent = SendVarData(server, Encoding.ASCII.GetBytes(message4));
        sent = SendVarData(server, Encoding.ASCII.GetBytes(message5));
        Console.WriteLine("Dang ngat ket noi voi server...");
        server.Shutdown(SocketShutdown.Both);
        server.Close();
    }
}
```

Kết quả



Hình II.8: Kết quả gởi và thông điệp cùng với kích thước

II.4.6.3. Sử dụng các hệ thống đánh dấu để phân biệt các thông điệp

Một cách khác để gởi các thông điệp với kích thước khác nhau là sử dụng các hệ thống đánh dấu. Hệ thống này sẽ chia các thông điệp bởi các ký tự phân cách để báo hiệu kết thúc thông điệp. Khi dữ liệu được nhận từ Socket, dữ liệu được kiểm tra từng ký tự một để phát hiện các ký tự phân cách, khi các ký tự phân cách được phát hiện thì dữ liệu trước ký tự phân cách chính là một thông điệp và dữ liệu sau ký tự phân cách sẽ bắt đầu một thông điệp mới.

Phương pháp này có một số hạn chế, nếu thông điệp lớn nó sẽ làm giảm tốc độ của hệ thống vì toàn bộ các ký tự của thông điệp đều phải được kiểm tra. Cũng có trường hợp một số ký tự trong thông điệp trùng với các ký tự phân cách và thông điệp này sẽ bị tách ra thành các thông điệp con, điều này làm cho chương trình chạy bị sai lệch.

II.4.7. Sử dụng C# Stream với TCP

Điều khiển thông điệp dùng giao thức TCP thường gây ra khó khăn cho các lập trình viên nên .NET Framework cung cấp một số lớp để giảm gánh nặng lập trình. Một trong những lớp đó là NetworkStream, và hai lớp dùng để gởi và nhận text sử dụng giao thức TCP là StreamWriter và StreamReader

II.4.7.1. Lớp NetworkStream

Lớp NetworkStream nằm trong namespace System.Net.Socket, lớp này có nhiều phương thức tạo lập để tạo một thể hiện của lớp NetworkStream nhưng phương thức tạo lập sau hay được dùng nhất:

```
Socket server = new Socket(AddressFamily.InterNetwork,
                SocketType.Stream, ProtocolType.Tcp);
NetworkStream ns = new NetworkStream(server);
```

Một số thuộc tính của lớp NetworkStream:

Thuộc Tính	**Mô Tả**
CanRead	true nếu NetworkStream hỗ trợ đọc
CanSeek	Luôn luôn false
CanWrite	true nếu NetworkStream hỗ trợ ghi
DataAvailable	true nếu có dữ liệu để đọc

Một số phương thức của lớp NetworkStream:

Phương Thức	**Mô Tả**
BeginRead()	Bắt đầu đọc NetworkStream bất đồng bộ
BeginWrite()	Bắt đầu ghi NetworkStream bất đồng bộ
Close()	Đóng đối tượng NetworkStream
CreateObjRef()	Tạo ra một đối tượng dùng như là proxy cho NetworkStream
EndRead()	Kết thúc đọc NetworkStream bất đồng bộ
EndWrite()	Kết thúc ghi NetworkStream bất đồng bộ
Equals()	So sánh hai đối tượng NetworkStreams
Flush()	Đẩy tất cả dữ liệu từ NetworkStream đi
GetHashCode()	Lấy hash code cho NetworkStream
GetLifetimeService()	Lấy đối tượng lifetime service cho NetworkStream
GetType()	Lấy kiểu NetworkStream
InitializeLifetimeService()	Lấy đối tượng lifetime service object để điều khiển chính sách lifetime choNetworkStream
Read()	Đọc dữ liệu từ NetworkStream
ReadByte()	Đọc một byte dữ liệu từ NetworkStream
ToString()	Trả về chuỗi mô tả NetworkStream
Write()	Ghi dữ liệu từ NetworkStream

Phương thức Read() được dùng để đọc các khối dữ liệu từ NetworkStream. Định dạng của phương thức này:

int Read(byte[] buffer, int offset, int size)

Trong đó:

- ✓ buffer: mảng các byte được đọc vào
- ✓ offset: vị trí bắt đầu để đọc vào trong bộ đệm
- ✓ size: số byte tối đa đọc được

Phương thức này trả về một giá trị interger mô tả số byte thực sự đọc được từ NetworkStream và dặt dữ liệu đọc được vào buffer.

Phương thức Write() dùng để gởi các khối dữ liệu đi cũng có định dạng tương tự:

void Write(byte[] buffer, int offset, int size)

Trong đó:

- ✓ buffer: mảng các byte để ghi
- ✓ offset: vị trí bắt đầu để ghi trong bộ đệm
- ✓ size: số byte tối đa được ghi bắt đầu tại vị trí offset

Chương trình TCP Client NetworkStream

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
class NetworkStreamTcpClient
{
    public static void Main()
    {
        byte[] data = new byte[1024];
        string input, stringData;
        int recv;
        IPEndPoint ipep = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 500);
        Socket server = new Socket(AddressFamily.InterNetwork,
                SocketType.Stream, ProtocolType.Tcp);
        try
        {
            server.Connect(ipep);
        }
        catch (SocketException e)
        {
```

```
            Console.WriteLine("Khong the ket noi den server");
            Console.WriteLine(e.ToString());
            return;
        }
        NetworkStream ns = new NetworkStream(server);
        if (ns.CanRead)
        {
            recv = ns.Read(data, 0, data.Length);
            stringData = Encoding.ASCII.GetString(data, 0, recv);
            Console.WriteLine(stringData);
        }
        else
        {
            Console.WriteLine("Error: Can't read from this Socket");
            ns.Close();
            server.Close();
            return;
        }
        while (true)
        {
            input = Console.ReadLine();
            if (input == "exit")
                break;
            if (ns.CanWrite)
            {
                ns.Write(Encoding.ASCII.GetBytes(input), 0, input.Length);
                ns.Flush();
            }
            recv = ns.Read(data, 0, data.Length);
            stringData = Encoding.ASCII.GetString(data, 0, recv);
            Console.WriteLine(stringData);
        }
        Console.WriteLine("Dang ngat ket noi voi server...");
        ns.Close();
        server.Shutdown(SocketShutdown.Both);
        server.Close();
    }
}
```

Chương trình này tạo ra một đối tượng NetworkStream từ đối tượng Socket:

```
NetworkStream ns = new NetworkStream(server);
```

Khi đối tượng NetworkStream được tạo ra, đối tượng Socket sẽ không được tham chiếu đến nữa cho đến khi nó bị đóng lại vào cuối chương trình, tất cả các thông tin liên lạc với Server ở xa được thực hiện thông đối tượng NetworkStream:

```
recv = ns.Read(data, 0, data.Length);
ns.Write(Encoding.ASCII.GetBytes(input), 0, input.Length);
ns.Flush();
```

Phương thức Flush() được dùng sau mỗi phương thức Write() để đảm bảo dữ liệu đặt vào NetworkStream sẽ lập tức được gởi đến hệ thống ở xa. Mặc dù đối tượng NetworkStream có thêm một số chức năng của Socket nhưng vẫn còn tồn tại vấn đề với biên thông điệp. Vấn đề này được giải quyết thông qua hai lớp hỗ trợ là StreamReader và StreamWriter.

Ta có thể kiểm tra chương trình này với chương trình TCP Server đơn giản ở trên.

II.4.7.2. Lớp StreamReader và StreamWriter

Namespcace System.IO chứa hai lớp StreamReader và StreamWriter điều khiển việc đọc và ghi các thông điệp text từ mạng. Cả hai lớp đều có thể được triển khai với một đối tượng NetworkStream để xác định các hệ thống đánh dấu cho các thông điệp TCP.

Lớp StreamReader có nhiều phương thức tạo lập, trong đó phương thức tạo lập đơn giản nhất của lớp StreamReader:

```
public StreamReader(Stream stream);
```

Biến stream có thể được tham chiếu đến bất kỳ kiểu đối tượng Stream nào kể cả đối tượng NetworkStream. Có nhiều phương thức và thuộc tính có thể được dùng với đối tượng StreamReader sau khi nó được tạo ra như trong bảng sau:

Phương Thức	**Mô Tả**
Close()	Đóng đối tượng StreamReader
CreateObjRef()	Tạo ra một đối tượng được dùng như là một proxy cho StreamReader
DiscardBufferedData()	Bỏ dữ liệu hiện tại ở StreamReader
Equals()	So sánh hai đối tượng StreamReader
GetHashCode()	Lấy hash code cho đối tượng StreamReader
GetLifetimeService()	Lấy đối tượng lifetime service object cho StreamReader

Phương Thức	**Mô Tả**
GetType()	Lấy kiểu của đối tượng StreamReader
InitializeLifetimeService()	Tạo ra một đối tượng lifetime service cho StreamReader
Peek()	Trả về byte dữ liệu hợp lệ tiếp theo từ mà không gỡ bỏ nó khỏi stream
Read()	Đọc một hoặc nhiều byte dữ liệu từ StreamReader
ReadBlock()	Đọc một nhóm các byte từ stream StreamReader và đặt nó vào một bộ đệm
ReadLine()	Đọc dữ liệu từ bắt đầu đối tượng StreamReader trở lên cho đến khi gặp ký tự xuống dòng đầu tiên
ReadToEnd()	Đọc dữ liệu cho đến khi hết stream
ToString()	Tạo ra một chuỗi mô tả đối tượng StreamReader

Tương tự đối tượng StreamReader, đối tượng StreamWriter có thể được tạo ra từ một đối tượng NetworkStream:

```
public StreamWriter(Stream stream);
```

StreamWriter cũng có nhiều phương thức và thuộc tính kết hợp với nó, một số phương thức và thuộc tính của lớp StreamReader cũng có trong đối tượng StreamWriter, ngoài ra nó còn có một số phương thức và thuộc tính riêng:

Phương Thức	**Mô Tả**
Flush()	Gởi tất cả dữ liệu trong bộ đệm StreamWriter ra stream
Write()	Gởi một hoặc nhiều byte dữ liệu ra stream
WriteLine()	Gởi dữ liệu cùng với ký tự xuống dòng ra stream

Phương thức ReadLine() là phương thức hay nhất của lớp StreamReader. Nó đọc các ký tự từ stream cho tới khi nó gặp ký tự xuống dòng. Tính năng này cho phép sử dụng ký tự xuống dòng như là một ký tự phân tách các thông điệp. Phương thức WriteLine() của lớp StreamWriter sẽ so khớp với phương thức ReadLine của lớp StreamReader do đó việc xử lý các thông điệp TCP trở nên dễ dàng hơn.

Chương trình Stream TCP Server

```
using System;
using System.IO;
using System.Net;
using System.Net.Sockets;
using System.Text;
```

```
class StreamTcpSrvr
{
    public static void Main()
    {
        string data;
        IPEndPoint ipep = new IPEndPoint(IPAddress.Any, 5000);
        Socket newsock = new Socket(AddressFamily.InterNetwork,
                SocketType.Stream, ProtocolType.Tcp);
        newsock.Bind(ipep);
        newsock.Listen(10);
        Console.WriteLine("Dang cho Client ket noi toi...");
        Socket client = newsock.Accept();
        IPEndPoint newclient = (IPEndPoint)client.RemoteEndPoint;
        Console.WriteLine("Da ket noi voi Client {0} tai port {1}",
                newclient.Address, newclient.Port);
        NetworkStream ns = new NetworkStream(client);
        StreamReader sr = new StreamReader(ns);
        StreamWriter sw = new StreamWriter(ns);
        string welcome = "Hello Client";
        sw.WriteLine(welcome);
        sw.Flush();
        while (true)
        {
            try
            {
                data = sr.ReadLine();
            }
            catch (IOException)
            {
                break;
            }

            Console.WriteLine(data);
            sw.WriteLine(data);
            sw.Flush();
        }
        Console.WriteLine("Da dong ket noi voi Client {0}", newclient.Address);
        sw.Close();
        sr.Close();
        ns.Close();
    }
}
```

Chương trình StreamTcpSrvr dùng phương thức WriteLine() của lớp StreamWriter để gởi các thông điệp text và kết thúc bằng ký tự xuống dòng. Đối với đối tượng NetworkStream, tốt hơn hết là ta phương thức Flush() sau khi gọi phương thức WriteLine() để đảm bảo rằng tất cả dữ liệu được gởi từ bộ đệm TCP.

Điểm khác biệt của chương trình này với chương trình TCP Server đơn giản ở trên là cách chương trình StreamTcpSrvr biết khi nào Client ngắt kết nối. Bởi vì phương thức ReadLine() hoạt động trên stream chứ không phải là Socket nên nó không thể trả về giá trị 0 khi Client ngắt kết nối. Thay vì vậy, phương thức ReadLine() sẽ phát sinh ra một biệt lệ nếu Client ngắt kết nối và ta phải dùng catch để bắt biệt lệ này và xử lý khi Client ngắt kết nối:

```
try
{
      data = sr.ReadLine();
}
catch (IOException)
{
      break;
}
```

Chương trình Stream TCP Client

```
using System;
using System.IO;
using System.Net;
using System.Net.Sockets;
using System.Text;
class StreamTcpClient
{
    public static void Main()
    {
        string data;
        string input;
        IPEndPoint ipep = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 5000);
        Socket server = new Socket(AddressFamily.InterNetwork,
                SocketType.Stream, ProtocolType.Tcp);
        try
        {
            server.Connect(ipep);
        }
        catch (SocketException e)
```

```
        {
            Console.WriteLine("Khong the ket noi den server");
            Console.WriteLine(e.ToString());
            return;
        }
        NetworkStream ns = new NetworkStream(server);
        StreamReader sr = new StreamReader(ns);
        StreamWriter sw = new StreamWriter(ns);
        data = sr.ReadLine();
        Console.WriteLine(data);
        while (true)
        {
            input = Console.ReadLine();
            if (input == "exit")
                break;
            sw.WriteLine(input);
            sw.Flush();
            data = sr.ReadLine();
            Console.WriteLine(data);
        }
        Console.WriteLine("Dang dong ket noi voi server...");
        sr.Close();
        sw.Close();
        ns.Close();
        server.Shutdown(SocketShutdown.Both);
        server.Close();
    }
}
```

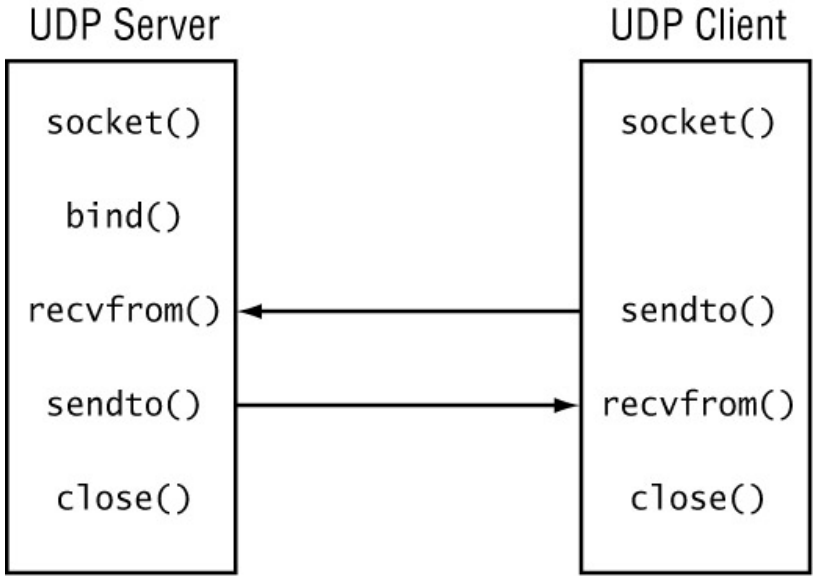
CHƯƠNG III: LẬP TRÌNH SOCKET PHI KẾT NỐI

III.1. Tổng quan

Các Socket phi kết nối cho phép gởi các thông điệp mà không cần phải thiết lập kết nối trước. Một phương thức đọc sẽ đọc toàn bộ thông điệp được gởi bởi một phương thức gởi, điều này làm tránh được các rắc rối, phức tạp với biên dữ liệu. Thật không may mắn là giao thức phi kết nối UDP không đảm bảo dữ liệu được truyền tới đích. Nhiều yếu tố như mạng bận, mạng bị đứt nữa chừng có thể ngăn cản các gói tin được truyền tới đích.

Nếu một thiết bị chờ dữ liệu từ một thiết bị ở xa, nó phải được gán một địa chỉ và port cục bộ, dùng hàm Bind() để gán. Một khi đã thực hiện xong, thiết bị có thể dùng Socket để gởi dữ liệu ra ngoài hay nhận dữ liệu từ Socket.

Bởi vì thiết bị Client không tạo ra kết nối đến một địa chỉ Server cụ thể do đó phương thức Connect() không cần dùng trong chương trình UDP Client. Mô hình bên dưới mô tả các bước lập trình Socket phi kết nối:



Hình V.1: Mô hình lập trình Socket phi kết nối

Khi kết nối không được thành lập thì phương thức Send() và Receive() không được dùng bởi vì trong hai phương thức trên đều không chỉ ra địa chỉ đích của dữ liệu.

Thay vào đó, Socket phi kết nối cung cấp hai phương thức để thực hiện việc này là SendTo() và ReceiveFrom()

III.2. Lập trình phía Server

UDP là một giao thức phi kết nối do đó các lập trình viên chỉ phải làm hai việc để tạo ra một ứng dụng Server gởi và nhận dữ liệu:

- ✓ Tạo ra Socket
- ✓ Kết nối Socket đến một IPEndPoint cục bộ

```
IPEndPoint ipep = new IPEndPoint(IPAddress.Any, 5000);
Socket newsock = new Socket(AddressFamily.InterNetwork,
                SocketType.Dgram, ProtocolType.Udp);
newsock.Bind(ipep);
```

Để thực hiện truyền thông phi kết nối, chúng ta phải chỉ ra SocketType là Dgram và ProtocolType là Udp.

Sau khi thực hiện xong hai bước trên, Socket có thể được dùng hoặc để chấp nhận các gói tin UDP đến trên IPEndPoint hoặc gởi các gói tin udp đến các thiết bị nhận khác trên mạng.

Phương thức SendTo() dùng để gởi dữ liệu, phương thức này chỉ ra dữ liệu để gởi và IPEndPoint của thiết bị nhận. Có nhiều quá tải hàm của phương thức SendTo() có thể được dùng tùy vào yêu cầu cụ thể.

```
SendTo(byte[] data, EndPoint Remote)
```

Phương thức trên gởi một mảng dữ liệu đến một EndPoint được chỉ ra bởi Remote. Một quá tải hàm khác phức tạp hơn của phương thức SendTo()

```
SendTo(byte[] data, SocketFlags Flags, EndPoint Remote)
```

Phương thức này cho phép thêm cờ SocketFlag, nó chỉ ra các tùy chọn UDP được sử dụng. Để chỉ ra số byte được gởi từ mảng byte ta sử dụng quá tải hàm sau của phương thức SendTo():

```
SendTo(byte[] data, int Offset, int Size, SocketFlags Flags,
EndPoint Remote)
```

Phương thức ReceiveFrom() có dùng định dạng với phương thức SendTo(), chỉ có một điểm khác biệt sau ở cách EndPoint được khai báo. Phương thức ReceiveFrom() đơn giản được định nghĩa như sau:

```
ReceiveFrom(byte[] data, ref EndPoint Remote)
```

Cũng như thông thường, tham số thứ nhất là một mảng byte được định nghĩa để nhận dữ liệu, tham số thứ hai ra phải truyền tham chiếu của đối tượng EndPoint. Tham chiếu này tham chiếu đến vị trí bộ nhớ nơi biến được lưu trữ. Phương thức ReceiveFrom() sẽ đặt thông tin EndPoint từ thiết bị ở xa vào vùng bộ nhớ của đối tượng EndPoint tham chiếu đến. Bằng việc sử dụng đối số thứ hai là tham chiếu ta sẽ lấy được địa chỉ IP và port của máy ở xa.

Chương trình UDP đơn giản

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
class SimpleUdpSrvr
{
    public static void Main()
    {
        int recv;
        byte[] data = new byte[1024];
        IPEndPoint ipep = new IPEndPoint(IPAddress.Any, 5000);
        Socket newsock = new Socket(AddressFamily.InterNetwork,
                SocketType.Dgram, ProtocolType.Udp);
        newsock.Bind(ipep);
        Console.WriteLine("Dang cho Client ket noi den...");
        IPEndPoint sender = new IPEndPoint(IPAddress.Any, 0);
        EndPoint Remote = (EndPoint)(sender);
        recv = newsock.ReceiveFrom(data, ref Remote);
        Console.WriteLine("Thong diep duoc nhan tu {0}:", Remote.ToString());
        Console.WriteLine(Encoding.ASCII.GetString(data, 0, recv));
        string welcome = "Hello Client";
        data = Encoding.ASCII.GetBytes(welcome);
        newsock.SendTo(data, data.Length, SocketFlags.None, Remote);
        while (true)
        {
            data = new byte[1024];
            recv = newsock.ReceiveFrom(data, ref Remote);

            Console.WriteLine(Encoding.ASCII.GetString(data, 0, recv));
            newsock.SendTo(data, recv, SocketFlags.None, Remote);
        }
    }
```

```
}
```

Để chương trình UDP chấp nhận các thông điệp UDP đến, nó phải được gắn với một port trên hệ thống. Việc này được thực hiện bằng cách tạo ra một đối tượng IPEndPoint sử dụng một địa chỉ IP cục bộ thích hợp, trong trường hợp này ta chỉ ra IPAddresss.Any để có thể dùng bất kỳ địa chỉ IP nào trên máy cục bộ để lắng nghe.

Sau khi gắn Socket vào một IPEndPoint, Server sẽ chờ Client kết nối đến, khi Client kết nối đến, Client sẽ gởi thông điệp đến Server. Server sau khi nhận được thông điệp từ Client nó sẽ gởi câu chào ngược lại cho Client:

```
recv = newsock.ReceiveFrom(data, ref Remote);
Console.WriteLine("Thong diep duoc nhan tu {0}:",Remote.ToString());
Console.WriteLine(Encoding.ASCII.GetString(data, 0, recv));
string welcome = "Hello client";
```

Khi gởi câu chào cho Client xong, Server sẽ bắt đầu nhận và gởi thông điệp

III.3. Lập trình phía Client

Bởi vì Client không cần chờ trên một port UDP định sắn nên nó cũng chẳng cần dùng phương thức Bind(), thay vì vậy nó sẽ lấy một port ngẫu nhien trên hệ thống khi dữ liệu được gởi và nó giữa port này để nhận dữ liệu trả về. Chương trình UDP Client cũng tương tự chương trình UDP Server:

Chương trình UDP Client đơn giản

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
class SimpleUdpClient
{
    public static void Main()
    {
        byte[] data = new byte[1024];
        string input, stringData;
        IPEndPoint ipep = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 5000);
        Socket server = new Socket(AddressFamily.InterNetwork,
                SocketType.Dgram, ProtocolType.Udp);
        string welcome = "Hello server";
        data = Encoding.ASCII.GetBytes(welcome);
        server.SendTo(data, data.Length, SocketFlags.None, ipep);
```

```
        IPEndPoint sender = new IPEndPoint(IPAddress.Any, 0);
        EndPoint Remote = (EndPoint)sender;
        data = new byte[1024];
        int recv = server.ReceiveFrom(data, ref Remote);
        Console.WriteLine("Thong diep duoc nhan tu {0}:", Remote.ToString());
        Console.WriteLine(Encoding.ASCII.GetString(data, 0, recv));
        while (true)
        {
            input = Console.ReadLine();
            if (input == "exit")
                break;
            server.SendTo(Encoding.ASCII.GetBytes(input), Remote);
            data = new byte[1024];
            recv = server.ReceiveFrom(data, ref Remote);
            stringData = Encoding.ASCII.GetString(data, 0, recv);
            Console.WriteLine(stringData);
        }
        Console.WriteLine("Dang dong client");
        server.Close();
    }
}
```

Chương trình UDP Client đầu tiên định nghĩa một IPEndPoint mà UDP Server sẽ gởi các gói tin:

```
IPEndPoint ipep = new IPEndPoint(IPAddress.Parse("127.0.0.1"),
5000);
```

Chương trình Client gởi thông điệp đến Server và chờ câu chào trả về từ Server. Bởi vì Client không cần chấp nhận các thông điệp UDP trên một port định trước nên Client không dùng phương thức Bind(). Nó sẽ nhận các thông điệp UDP trên cùng port mà nó đã gởi.

Chương trình SimpleUdpClient đọc dữ liệu nhập vào từ bàn phím rồi gởi đến và chờ dữ liệu từ Server gởi trả về. Khi Server gởi trả dữ liệu về, Client sẽ lấy thông điệp đó ra và hiển thị lên màn hình. Nếu người dùng nhận vào "exit" thì vòng lặp sẽ thoát và kết nối bị đóng lại.

Không giống như chương trình TCP Server, chương trình UDP Server sẽ không biết khi nào Client ngắt kết nối do đó khi Client ngắt kết nối thì nó phải gởi thông điệp ngắt kết nối cho Server biết.

III.3.1. Sử dụng phương thức Connect() trong chương trình UDP Client

Các phương thức UDP được thiết kế để cho phép các lập trình viên gởi các gói tin đến bất kỳ máy nào trên mạng bất cứ lúc nào. Bởi vì giao thức UDP không yêu cầu kết nối trước khi gởi dữ liệu nên phải chỉ ra địa chỉ của máy nhận trong phương thức SendTo() và phương thức ReceiveFrom(). Nếu chương trình của chúng ta chỉ cần gởi và nhận dữ liệu từ một máy, chúng ta có thể dùng phương thức Connect().

Sau khi UDP socket được tạo ra, chúng ta có thể dùng phương thức Connect() giống như trong chương trình TCP để chỉ ra udp Server ở xa. Sau khi dùng phương thức Connect() xong ta có thể dùng phương thức Send() và Receive() để truyền tải dữ liệu giữa các thiết bị với nhau. Kỹ thuật này được minh họa trong chương trình UDP Client sau:

Chương trình udp Client dùng phương thức Connect()

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
class OddUdpClient
{
    public static void Main()
    {
        byte[] data = new byte[1024];
        string input, stringData;
        IPEndPoint ipep = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 9050);
        Socket server = new Socket(AddressFamily.InterNetwork,
                SocketType.Dgram, ProtocolType.Udp);
        server.Connect(ipep);
        string welcome = "Xin chao server";
        data = Encoding.ASCII.GetBytes(welcome);
        server.Send(data);
        data = new byte[1024];
        int recv = server.Receive(data);
        Console.WriteLine("Nhan thong diep tu {0}:", ipep.ToString());
        Console.WriteLine(Encoding.ASCII.GetString(data, 0, recv));
        while (true)
        {
            input = Console.ReadLine();
            if (input == "exit")
                break;
```

```
            server.Send(Encoding.ASCII.GetBytes(input));
            data = new byte[1024];
            recv = server.Receive(data);
            stringData = Encoding.ASCII.GetString(data, 0, recv);
            Console.WriteLine(stringData);
        }
        Console.WriteLine("Dang dong client");
        server.Close();
    }
}
```

III.3.2. Phân biệt các thông điệp UDP

Một trong những tính năng quan trọng của UDP mà TCP không có được đó là khả năng xử lý thông điệp mà không cần quan tâm đến biên thông điệp. UDP bảo vệ biên thông điệp của tất cả các thông điệp được gởi. Mỗi lần gọi phương thức ReceiveFrom() nó chỉ đọc dữ liệu được gởi từ một phương thức SendTo().

Khi UDP Socket được tạo ra, nó có thể nhận thông điệp từ bất kỳ Client nào. Để udp Socket phân biệt được Client nào gởi dữ liệu nó bắt buộc mỗi thông điệp phải được chứa trong một gói tin riêng và được đánh dấu bởi thông tin IP của thiết bị gởi. Điều này cho phép thiết bị nhận phân biệt được các thông điệp và thiết bị gởi.

Chương trình Client và Server sau sẽ minh họa điều này.

Chương trình UDP Server

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
class TestUdpSrvr
{
    public static void Main()
    {
        int recv;
        byte[] data = new byte[1024];
        IPEndPoint ipep = new IPEndPoint(IPAddress.Any, 5000);
        Socket newsock = new Socket(AddressFamily.InterNetwork,
                SocketType.Dgram, ProtocolType.Udp);
        newsock.Bind(ipep);
        Console.WriteLine("Dang cho client ket noi den...");
        IPEndPoint sender = new IPEndPoint(IPAddress.Any, 0);
        EndPoint tmpRemote = (EndPoint)(sender);
```

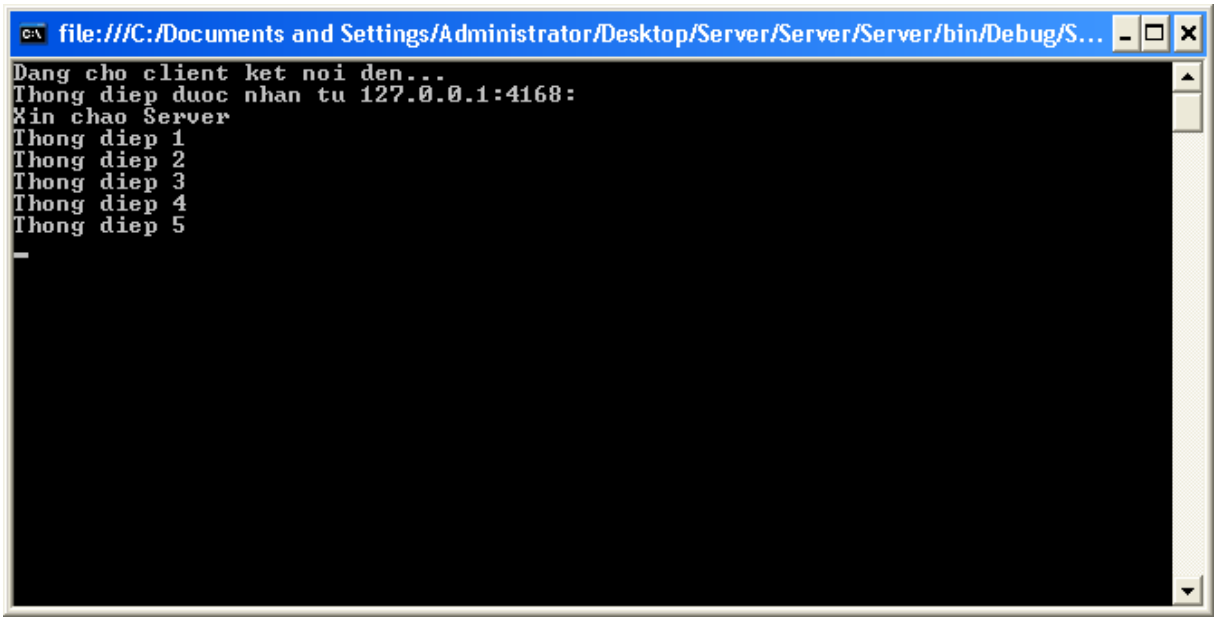
```
        recv = newsock.ReceiveFrom(data, ref tmpRemote);
        Console.WriteLine("Thong diep duoc nhan tu {0}:",
tmpRemote.ToString());
        Console.WriteLine(Encoding.ASCII.GetString(data, 0, recv));
        string welcome = "Xin chao client";
        data = Encoding.ASCII.GetBytes(welcome);
        newsock.SendTo(data, data.Length, SocketFlags.None, tmpRemote);
        for (int i = 0; i < 5; i++)
        {
            data = new byte[1024];
            recv = newsock.ReceiveFrom(data, ref tmpRemote);
            Console.WriteLine(Encoding.ASCII.GetString(data, 0, recv));
        }
        newsock.Close();
    }
}
```

Chương trình UDP Client

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
class TestUdpClient
{
    public static void Main()
    {
        byte[] data = new byte[1024];
        IPEndPoint ipep = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 5000);
        Socket server = new Socket(AddressFamily.InterNetwork,
                SocketType.Dgram, ProtocolType.Udp);
        string welcome = "Xin chao Server";
        data = Encoding.ASCII.GetBytes(welcome);
        server.SendTo(data, data.Length, SocketFlags.None, ipep);
        IPEndPoint sender = new IPEndPoint(IPAddress.Any, 0);
        EndPoint tmpRemote = (EndPoint)sender;
        data = new byte[1024];
        int recv = server.ReceiveFrom(data, ref tmpRemote);
        Console.WriteLine("Thong diep duoc nhan tu {0}:",
tmpRemote.ToString());
        Console.WriteLine(Encoding.ASCII.GetString(data, 0, recv));
        server.SendTo(Encoding.ASCII.GetBytes("Thong diep 1"), tmpRemote);
        server.SendTo(Encoding.ASCII.GetBytes("Thong diep 2"), tmpRemote);
```

```
            server.SendTo(Encoding.ASCII.GetBytes("Thong diep 3"), tmpRemote);
            server.SendTo(Encoding.ASCII.GetBytes("Thong diep 4"), tmpRemote);
            server.SendTo(Encoding.ASCII.GetBytes("Thong diep 5"), tmpRemote);
            Console.WriteLine("Dang dong client");
            server.Close();
        }
}
```

Kết quả ở Server



Hình V.2: UDP Server nhận biết được các thông điệp riêng rẽ

III.4. Ngăn cản mất dữ liệu

Một thuận lợi của việc truyền thông dùng giao thức TCP là giao thức TCP sử dụng bộ đệm TCP. Tất cả dữ liệu được gởi bởi TCP Socket được đặt vào bộ đệm TCP trước khi được gởi ra ngoài mạng. Cũng giống như vậy, tất cả dữ liệu nhận từ Socket được đặt vào bộ đệm TCP trước khi được đọc bởi phương thức Receive(). Khi phương thức Receive() cố gắng đọc dữ liệu từ bộ đệm, nếu nó không đọc hết dữ liệu thì phần còn lại vẫn nằm trong bộ đệm và chờ lần gọi phương thức Receive() kế tiếp.

Vì UDP không quan tâm đến việc gởi lại các gói tin nên nó không dùng bộ đệm. Tất cả dữ liệu được gởi từ Socket đều được lập tức gởi ra ngoài mạng và tất cả dữ liệu được nhận từ mạng lập tức được chuyển cho phương thức ReceiveFrom() trong lần gọi tiếp theo. Khi phương thức ReceiveFrom() được dùng trong chương trình, các lập trình viên phải đảm bảo rằng bộ đệm phải đủ lớn để chấp nhận hết dữ liệu từ UDP

Socket. Nếu bộ đệm quá nhỏ, dữ liệu sẽ bị mất. Để thấy được điều này, ta tiến hành thay đổi kích thước bộ đệm trong chương trình UDP đơn giản trên:

Chương trình BadUDPClient

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
class BadUdpClient
{
    public static void Main()
    {
        byte[] data = new byte[30];
        string input, stringData;
        IPEndPoint ipep = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 5000);
        Socket server = new Socket(AddressFamily.InterNetwork,
                SocketType.Dgram, ProtocolType.Udp);
        string welcome = "Xin chao serveer";
        data = Encoding.ASCII.GetBytes(welcome);
        server.SendTo(data, data.Length, SocketFlags.None, ipep);
        IPEndPoint sender = new IPEndPoint(IPAddress.Any, 0);
        EndPoint tmpRemote = (EndPoint)sender;
        data = new byte[30];
        int recv = server.ReceiveFrom(data, ref tmpRemote);
        Console.WriteLine("Thong diep duoc nhan tu {0}:",
tmpRemote.ToString());
        Console.WriteLine(Encoding.ASCII.GetString(data, 0, recv));
        while (true)
        {
            input = Console.ReadLine();
            if (input == "exit")
                break;
            server.SendTo(Encoding.ASCII.GetBytes(input), tmpRemote);
            data = new byte[30];
            recv = server.ReceiveFrom(data, ref tmpRemote);
            stringData = Encoding.ASCII.GetString(data, 0, recv);
            Console.WriteLine(stringData);
        }
        Console.WriteLine("Dang dong client");
        server.Close();
    }
}
```

Ta có thể test chương trình này với chương trình UDP Server đơn giản ở trên. Khi ta nhận dữ liệu ít hơn 10 byte thì chương trình vẫn chạy bình thường nhưng khi ta nhập dữ liệu lớn hơn 10 byte thì chương trình BadUdpClient sẽ phát sinh ra một biệt lệ. Mặc dầu ta không thể lấy lại dữ liệu đã bị mất nhưng ta có thể hạn chế mất dữ liệu bằng cách đặt phương thức ReceiveFrom() trong khối try-catch, khi dữ liệu bị mất bởi kích thước bộ đệm nhỏ, ta có thể tăng kích thước bộ đệm vào lần kế tiếp nhận dữ liệu. Chương trình BetterUdpClient sau minh họa việc này:

Chương trình BetterUdpClient

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
class BetterdUdpClient
{
    public static void Main()
    {
        byte[] data = new byte[30];
        string input, stringData;
        IPEndPoint ipep = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 5000);
        Socket server = new Socket(AddressFamily.InterNetwork,
                SocketType.Dgram, ProtocolType.Udp);
        string welcome = "Xin chao server";
        data = Encoding.ASCII.GetBytes(welcome);
        server.SendTo(data, data.Length, SocketFlags.None, ipep);
        IPEndPoint sender = new IPEndPoint(IPAddress.Any, 0);
        EndPoint tmpRemote = (EndPoint)sender;
        data = new byte[30];
        int recv = server.ReceiveFrom(data, ref tmpRemote);
        Console.WriteLine("Thong diep duoc nhan tu {0}:",
tmpRemote.ToString());
        Console.WriteLine(Encoding.ASCII.GetString(data, 0, recv));
        int i = 30;
        while (true)
        {
            input = Console.ReadLine();
            if (input == "exit")
                break;
            server.SendTo(Encoding.ASCII.GetBytes(input), tmpRemote);
            data = new byte[i];
            try
```

```
            {
                recv = server.ReceiveFrom(data, ref tmpRemote);
                stringData = Encoding.ASCII.GetString(data, 0, recv);
                Console.WriteLine(stringData);
            }
            catch (SocketException)
            {
                Console.WriteLine("Canh bao: du lieu bi mat, hay thu lai");
                i += 10;
            }
        }
        Console.WriteLine("Dang dong client");
        server.Close();
    }
}
```

Thay vì sử dụng mảng buffer với chiều dài cố định, chương trình BetterUdpClient dùng một biết có thể thiết lập giá trị khác nhau mỗi lần phương thức ReceiveFrom() được dùng.

```
data = new byte[i];
try
{
      recv = server.ReceiveFrom(data, ref tmpRemote);
      stringData = Encoding.ASCII.GetString(data, 0, recv);
      Console.WriteLine(stringData);
}
catch (SocketException)
{
      Console.WriteLine("Canh bao: du lieu bi mat, hay thu lai");
      i += 10;
}
```

III.5. Ngăn cản mất gói tin

Một khó khăn khác khi lập trình với giao thức udp là khả năng bị mất gói tin bởi vì udp là một giao thức phi kết nối nên không có cách nào mà thiết bị gởi biết được gói tin gởi có thực sự đến được đích hay không. Cách đơn giản nhất để ngăn chặn việc mất các gói tin là phải có cơ chế hồi báo giống như giao thức TCP. Các gói tin được gởi thành công đến thiết bị nhận thì thiết bị nhận phải sinh ra gói tin hồi báo

cho thiết bị gởi biết đã nhận thành công. Nếu gói tin hồi báo không được nhận trong một khoảng thời gian nào đó thì thiết bị nhận sẽ cho là gói tin đó đã bị mất và gởi lại gói tin đó.

Có hai kỹ thuật dùng để truyền lại các gói tin UDP:

- ✓ Sử dụng Socket bất đồng bộ và một đối tượng Timer. Kỹ thuật này yêu cầu sử dụng một Socket bất đồng bộ mà nó có thể lắng nghe các gói tin đến không bị block. Sau khi Socket được thiết lập đọc bất đồng bộ, một đối tượng Timer có thể được thiết lập, nếu đối tượng Timer tắt trước khi hành động đọc bất đồng bộ kết thúc thì việc gởi lại dữ liệu diễn ra.
- ✓ Sử dụng Socket đồng bộ và thiết lập giá trị Socket time-out. Để làm được việc này, ta dùng phương thức SetSocketOption().

III.5.1. Sử dụng Soket Time-out

Phương thức ReceiveFrom() là phương thức bị block. Nó sẽ block chương trình lại cho đến khi chương trình nhận dữ liệu. Nếu dữ liệu không bao giờ nhận, chương trình sẽ block mã mãi. Mặc định phương thức ReceiveFrom() sẽ bị block mãi mãi nếu không có dữ liệu được đọc. phương thức SetSocketOption() cung cấp nhiều tùy chọn cho các Socket đã được tạo, một trong những tùy chọn đó là ReceiveTimeout. Nó sẽ thiết lập khoảng thời gian Socket sẽ chờ dữ liệu đến trước khi phát ra tín hiệu time-out.

Định dạng của phương thức SetSocketOption() như sau:

```
SetSocketOption(SocketOptionLevel so, SocketOptionName sn, int
value)
```

SocketOptionLevel chỉ ra kiểu tùy chọn Socket để thực thi. SocketOptionName định nghĩa tùy chọn được thiết lập, và tham số cuối cùng thiết lập giá trị cho tùy chọn. Để chỉ ra giá trị TimeOut ta dùng như sau:

```
server.SetSocketOption(SocketOptionLevel.Socket,
    SocketOptionName.ReceiveTimeout, 3000);
```

Trong đó giá trị của tham số cuối cùng chỉ ra số miligiây tối đa hàm ReceiveFrom() sẽ chờ cho đến khi có dữ liệu để đọc. Chương trình sau sẽ minh họa cách dùng TimeOut:

Chương trình TimeOutUdpClient

```
using System;
using System.Net;
```

```
using System.Net.Sockets;
using System.Text;
class TimeoutUdpClient
{
    public static void Main()
    {
        byte[] data = new byte[1024];
        string input, stringData;
        int recv;
        IPEndPoint ipep = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 5000);
        Socket server = new Socket(AddressFamily.InterNetwork,
                SocketType.Dgram, ProtocolType.Udp);
        int sockopt = (int)server.GetSocketOption(SocketOptionLevel.Socket,
        SocketOptionName.ReceiveTimeout);
        Console.WriteLine("Gia tri timeout mac dinh: {0}", sockopt);
        server.SetSocketOption(SocketOptionLevel.Socket,
        SocketOptionName.ReceiveTimeout, 3000);
        sockopt = (int)server.GetSocketOption(SocketOptionLevel.Socket,
        SocketOptionName.ReceiveTimeout);
        Console.WriteLine("Gia tri timeout moi: {0}", sockopt);
        string welcome = "Xin chao server";
        data = Encoding.ASCII.GetBytes(welcome);
        server.SendTo(data, data.Length, SocketFlags.None, ipep);
        IPEndPoint sender = new IPEndPoint(IPAddress.Any, 0);
        EndPoint tmpRemote = (EndPoint)sender;
        data = new byte[1024];
        recv = server.ReceiveFrom(data, ref tmpRemote);
        Console.WriteLine("Thong diep duoc nhan tu {0}:", mpRemote.ToString());
        Console.WriteLine(Encoding.ASCII.GetString(data, 0, recv));
        while (true)
        {
            input = Console.ReadLine();
            if (input == "exit")
                break;
            server.SendTo(Encoding.ASCII.GetBytes(input), tmpRemote);
            data = new byte[1024];
            recv = server.ReceiveFrom(data, ref tmpRemote);
            stringData = Encoding.ASCII.GetString(data, 0, recv);
            Console.WriteLine(stringData);
        }
        Console.WriteLine("Dang dong client");
        server.Close();
    }
```

```
}
```

Chương trình TimeoutUdpClient đầu tiên lấy giá trị ReceiveTimeout ban đầu từ Socket và hiển thị nó, sau đó thiết lập giá trị này thành 3 giây:

```
int sockopt = (int)server.GetSocketOption(SocketOptionLevel.Socket,
SocketOptionName.ReceiveTimeout);
Console.WriteLine("Gia tri timeout mac dinh: {0}", sockopt);
server.SetSocketOption(SocketOptionLevel.Socket,
SocketOptionName.ReceiveTimeout, 3000);
```

Phương thức GetSocketOption() trả về một đối tượng Object, vì thế nó phải được ép kiểu thành kiểu interger. Sau khi biên dịch và chạy chương trình với chương trình SimpleUdpServer ở trên, kết quả xuất ra như sau:

```
C:\>TimeoutUdpClient
Gia tri timeout mac dinh: 0
Gia tri timeout moi: 3000
Unhandled Exception: System.Net.Sockets.SocketException: An existing
connection
was forcibly closed by the remote host
  at System.Net.Sockets.Socket.ReceiveFrom(Byte[] buffer, Int32 offset,
Int32
size, SocketFlags socketFlags, EndPoint& remoteEP)
  at System.Net.Sockets.Socket.ReceiveFrom(Byte[] buffer, EndPoint&
remoteEP)
  at TimeoutUdpClient.Main()
C:\>
```

Giá trị ban đầu của ReceiveTimeout được thiết lập là 0 cho biết nó sẽ chờ dữ liệu mãi mãi. Sau khi thêm phương thức SetSocketOpition() và được thiết lập giá trị 3000 mili giây thì hàm ReceiveFrom() sẽ đợi dữ liệu trong 3 giây, sau 3 giây nếu không có dữ liệu để đọc thì nó sẽ phát sinh ra biệt lệ do đó ta phải đặt hàm này trong khối try – catch để xử lý biệt lệ.

III.6. Điều khiển việc truyền lại các gói tin

Có nhiều lý do các gói tin UDP không thể đến được đích, có thể lần đầu tiên gởi không tới được đích nhưng khi gởi lại lần thứ hai, ba thì tới được đích. Hầu hết các ứng dụng udp đều cho phép gởi lại các gói tin một số lần trước khi loại bỏ nó. Khi gởi một gói tin mà không có thông điệp trả về, chúng ta có thể gởi lại thông điệp ban đầu nhiều lần, nếu sau khi gởi lại một số lần thông điệp đã đến được đích thì ta tiếp tục với phần còn lại của chương trình ngược lại ta sẽ phát sinh ra một thông báo lỗi.

Cách đơn giản nhất để thực hiện việc truyền lại là tạo ra một phương thức riêng trong lớp để điều khiển tất cả việc gởi và nhận các thông điệp. Các bước thực hiện như sau:

1) Gởi một thông điệp đến máy ở xa

2) Chờ câu trả lời từ máy ở xa

3) Nếu câu trả lời được nhận, chấp nhận nó và thoát khỏi phương thức với dữ liệu nhận và kích thước của dữ liệu.

4) Nếu không nhận được câu trả lời nào trong khoảng thời gian time-out, tăng biến đếm thử lên

5) Kiểm tra biến đếm thử, nếu nó nhỏ hơn số lần đã định nghĩa trước thì quay lại bước 1, nếu nó bằng số lần đã định nghĩa trước, không truyền lại nữa và thông báo lỗi.

Một khi phương thức để gởi và nhận các gói tin udp đã được tạo ra, nó có thể được dùng bất cứ đâu trong chương trình nơi có dữ liệu được gởi tới thiết bị ở xa và chờ câu trả lời. Phương thức này được cài đặt như sau:

```
private int SndRcvData(Socket s, byte[] message, EndPoint rmtdevice)
{
      int recv;
      int retry = 0;
      while (true)
{
Console.WriteLine("Truyen lai lan thu: #{0}", retry);
try
{
      s.SendTo(message, message.Length, SocketFlags.None, rmtdevice);
      data = new byte[1024];
      recv = s.ReceiveFrom(data, ref Remote);
}
catch (SocketException)
{
      recv = 0;
}
if (recv > 0)
{
```

```
        return recv;
}
else
{
        retry++;
        if (retry > 4)
        {
                return 0;
        }
}
```

Phương thức này yêu cầu ba tham số:

- ✓ Một đối tượng socket đã được thành lập
- ✓ Mảng dữ liệu chứa thông điệp để gởi tới thiết bị ở xa
- ✓ Một đối tượng EndPoint chứa địa chỉ IP và port của thiết bị ở xa

Đối tượng Socket được truyền vào phương thức trên phải được khởi tạo trước và giá trị ReceiveTimeout đã được thiết lập một khoảng thời gian chờ câu trả lời từ thiết bị ở xa.

Phương thức SndRcvData() đầu tiên gởi dữ liệu đến thiết bị ở xa dùng phương thức SendTo() truyền thống. Sau khi gởi thông điệp, phương thức SndRcvData() sẽ block ở phương thức ReceiveFrom() và chờ thông điệp trả về. Nếu thông điệp được nhận từ thiết bị ở xa trong khoảng giá trị ReceiveTimeout thì phương thức SndRcvData() sẽ đặt dữ liệu vào mảng byte đã được định nghĩa trong lớp và trả về số byte đọc được. Nếu không có thông điệp trả về vào lúc kết thúc giá trị ReceiveTimeout, một biệt lệ sẽ được phát ra và khối catch được xử lý. Trong khối catch, giá trị recv được thiết lập về 0. Sau khối try-catch, giá trị recv sẽ được kiểm tra. Nếu giá trị đó là số dương thì thông điệp đã được nhận thành công, nếu là số 0 thì không có thông điệp nào được nhận và giá trị này được tăng lên, sau đó kiểm tra nó đã đạt tới giá trị tối đa hay chưa, nếu chưa đạt tới giá trị tối đa toàn bộ quá trình sẽ được lặp lại và bắt đầu gởi lại thông điệp, nếu đã tới giá trị tối đa rồi thì phương thức SndRcvData() sẽ trả về 0.

Chương trình RetryUdpClient

```
using System;
using System.Net;
```

```
using System.Net.Sockets;
using System.Text;
class RetryUdpClient
{
    private byte[] data = new byte[1024];
    private static IPEndPoint sender = new IPEndPoint(IPAddress.Any, 0);
    private static EndPoint Remote = (EndPoint)sender;
    private int SndRcvData(Socket s, byte[] message, EndPoint rmtdevice)
    {
        int recv;
        int retry = 0;
        while (true)
        {
            Console.WriteLine("Truyen lai lan thu: #{0}", retry);
            try
            {
                s.SendTo(message, message.Length, SocketFlags.None, rmtdevice);
                data = new byte[1024];
                recv = s.ReceiveFrom(data, ref Remote);
            }
            catch (SocketException)
            {
                recv = 0;
            }
            if (recv > 0)
            {
                return recv;
            }
            else
            {
                retry++;
                if (retry > 4)
                {
                    return 0;
                }
            }
        }
    }
    public RetryUdpClient()
    {
        string input, stringData;
        int recv;
        IPEndPoint ipep = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 5000);
```

```
        Socket server = new Socket(AddressFamily.InterNetwork,
                SocketType.Dgram, ProtocolType.Udp);
        int sockopt = (int)server.GetSocketOption(SocketOptionLevel.Socket,
         SocketOptionName.ReceiveTimeout);
        Console.WriteLine("Gia tri timeout mac dinh: {0}", sockopt);
        server.SetSocketOption(SocketOptionLevel.Socket,
         SocketOptionName.ReceiveTimeout, 3000);
        sockopt = (int)server.GetSocketOption(SocketOptionLevel.Socket,
         SocketOptionName.ReceiveTimeout);
        Console.WriteLine("Gia tri timeout moi: {0}", sockopt);
        string welcome = "Xin chao Server";
        data = Encoding.ASCII.GetBytes(welcome);
        recv = SndRcvData(server, data, ipep);
        if (recv > 0)
        {
            stringData = Encoding.ASCII.GetString(data, 0, recv);
            Console.WriteLine(stringData);
        }
        else
        {
            Console.WriteLine("Khong the lien lac voi thiet bi o xa");
            return;
        }
        while (true)
        {
            input = Console.ReadLine();
            if (input == "exit")
                break;
            recv = SndRcvData(server, Encoding.ASCII.GetBytes(input), ipep);
            if (recv > 0)
            {
                stringData = Encoding.ASCII.GetString(data, 0, recv);
                Console.WriteLine(stringData);
            }
            else
                Console.WriteLine("Khong nhan duoc cau tra loi");
        }
        Console.WriteLine("Dang dong client");
        server.Close();
    }
    public static void Main()
    {
        RetryUdpClient ruc = new RetryUdpClient();
```

```
        }
}
```

CHƯƠNG IV: SỬ DỤNG CÁC LỚP HELPER CỦA C# SOCKET

IV.1. Lớp TCP Client

Lớp TcpClient nằm ở namespace System.Net.Sockets được thiết kế để hỗ trợ cho việc viết các ứng dụng TCP Client được dễ dàng.

Lớp TcpClient cho phép tạo ra một đối tượng Tcp Client sử dụng một trong ba phương thức tạo lập sau:

TcpClient(): là phương thức tạo lập đầu tiên, đối tượng được tạo ra bởi phương thức tạo lập này sẽ gắn kết với một địa chỉ cục bộ và một port TCP ngẫu nhiên. Sau khi đối tượng TcpClient được tạo ra, nó phải được kết nối đến thiết bị ở xa thông qua phương thức Connect() như ví dụ dưới đây:

```
TcpClient newcon = new TcpClient();
newcon.Connect("192.168.6.1", 8000);
```

TcpClient(IPEndPoint localEP): phương thức tạo lập này cho phép chúng ta chỉ ra địa chỉ IP cục bộ cùng với port được dùng. Đây là phương thức tạo lập thường được sử dụng khi thiết bị có nhiều hơn một card mạng và chúng ta muốn dữ liệu được gởi trên card mạng nào. Phương thức Connect() cũng được dùng để kết nối với thiết bị ở xa:

```
IPEndPoint iep = new IPEndPoint(IPAddress,Parse("192.168.6.1"),
8000);
TcpClient newcon = new TcpClient(iep);
newcon.Connect("192.168.6.2", 8000);
```

TcpClient(String host, int port): phương thức tạo lập thứ ba này thường được sử dụng nhất, nó cho phép chỉ ra thiết bị nhận trong phương thức tạo lập và không cần phải dùng phương thức Connect(). Địa chỉ của thiết bị ở xa có thể là một chuỗi hostname hoặc một chuỗi địa chỉ IP. Phương thức tạo lập của TcpClient sẽ tự động phân giải hostname thành địa chỉ IP. Ví dụ:

```
TcpClient newcon = new TcpClient("www.isp.net", 8000);
```

Mỗi khi đối tượng TcpClient được tạo ra, nhiều thuộc tính và phương thức có thể được dùng để xử lý việc truyền dữ liệu qua lại giữa các thiết bị.

Phương Thức	**Mô Tả**
Close()	Đóng kết nối TCP
Connect()	Thành lập kết nối TCP với thiết bị ở xa
Equals()	So sánh hai đối tượng TcpClient
GetHashCode()	Lấy mã hash code
GetStream()	Lấy đối tượng Stream nó có thể dùng để gởi và nhận dữ liệu
GetType()	Lấy kiểu của thể hiện hiện tại
ToString()	Chuyển thể hiện hiện tại sang kiểu chuỗi

Phương thức Connect() dùng để kết nối đối tượng TcpClient đến thiết bị ở xa. Mỗi khi kết nối được thành lập, phương thức GetStream() gán một đối tượng NetworkStream để gởi và nhận dữ liệu nhờ vào phương thức Read() và Write(). Lớp TcpClient còn có nhiều thuộc tính được mô tả trong bảng sau:

Thuộc Tính	**Mô Tả**
LingerState	Lấy hoặc thiết lập thời gian kết nối TCP vẫn còn sau khi gọi phương thức Close()
NoDelay	Lấy hoặc thiết lập thời gian trễ được dùng để gởi hoặc nhận ở bộ đệm TCP
ReceiveBufferSize	Lấy hoặc thiết lập kích thước bộ đệm TCP nhận
ReceiveTimeout	Lấy hoặc thiết lập thời gian timeout của Socket
SendBufferSize	Lấy hoặc thiết lập kích thước bộ đệm TCP gởi
SendTimeout	Lấy hoặc thiết lập giá trị timeout của Socket

Chương trình TCPClient đơn giản

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
class TcpClientSample
{
    public static void Main()
    {
        byte[] data = new byte[1024];
        string input, stringData;
        TcpClient server;
```

```
        try
        {
            server = new TcpClient("127.0.0.1", 5000);
        }
        catch (SocketException)
        {
            Console.WriteLine("Khong the ket noi den server");
            return;
        }
        NetworkStream ns = server.GetStream();
        int recv = ns.Read(data, 0, data.Length);
        stringData = Encoding.ASCII.GetString(data, 0, recv);
        Console.WriteLine(stringData);
        while (true)
        {
            input = Console.ReadLine();
            if (input == "exit")
                break;
            ns.Write(Encoding.ASCII.GetBytes(input), 0,
input.Length);
            ns.Flush();
            data = new byte[1024];
            recv = ns.Read(data, 0, data.Length);
            stringData = Encoding.ASCII.GetString(data, 0, recv);
            Console.WriteLine(stringData);
        }
        Console.WriteLine("Dang ngat ket noi voi server...");
        ns.Close();
        server.Close();
    }
}
```

Trong chương trình trên phương thức tạo lập sẽ tự động kết nối đến Server được chỉ ra ở xa, nó nên được đặt ở trong khối try-catch để phòng trường hợp Server không hợp lệ.

Sau khi đối tượng NetworkStream được tạo ra, ta có thể dùng phương thức Read() và Write() để nhận và gởi dữ liệu:

```
while (true)
{
      input = Console.ReadLine();
      if (input == "exit")
      break;
      ns.Write(Encoding.ASCII.GetBytes(input), 0, input.Length);
      ns.Flush();
      data = new byte[1024];
      recv = ns.Read(data, 0, data.Length);
      stringData = Encoding.ASCII.GetString(data, 0, recv);
      Console.WriteLine(stringData);
}
```

Phương thức Read() yêu cầu 3 tham số:

- ✓ Mảng các byte để đặt dữ liệu nhận vào
- ✓ Vị trí offset trong bộ đệm mà tại đó ta muốn đặt dữ liệu
- ✓ Chiều dài của bộ đệm dữ liệu

Cũng giống như phương thức Receive() của Socket, phương thức Read() sẽ đọc một lượng dữ liệu có độ lớn tối đa đúng bằng độ lớn bộ đệm. Nếu bộ đệm quá nhỏ, phần dữ liệu còn lại sẽ nằm ở trong stream và đợi lần gọi phương thức Read() tiếp theo.

Phương thức Write() cũng yêu cầu ba tham số:

- ✓ Mảng các byte để gởi dữ liệu
- ✓ Vị trí offset trong bộ đệm mà tại đó ta muốn gởi dữ liệu
- ✓ Chiều dài của dữ liệu được gởi

Cần chú ý rằng TCP không bảo vệ các biên thông điệp. Điều này cũng áp dụng cho lớp TcpClient, do đó ta cần phải xử lý vấn đề biên thông điệp giống như phương thức Receive() của lớp Socket bằng cách tạo ra vòng lặp để đảm bảo tất cả dữ liệu đều được đọc từ stream.

Ta có thể test chương trình này với chương trình TCP Server đơn giản ở phần trên.

IV.2. Lớp TCPListener

Cũng giống như lớp TcpClient, lớp TcpListener cũng cho phép chúng ta tạo ra các chương trình TCP Server một cách đơn giản

Lớp TcpListener có ba phương thức tạo lập:

- ✓ TcpListener(int port): gắn một đối tượng TcpListener vào một port được chỉ ra trên máy cục bộ.
- ✓ TcpListener(IPEndPoint ie): gắn một đối tượng TcpListener vào một đối tượng EndPoint cục bộ
- ✓ TcpListener(IPAddress addr, int port): gắn một đối tượng TcpListener vào một đối tượng IPAddress và một port

Không giống như lớp TcpClient, các phương thức tạo lập của lớp TcpListener yêu cầu ít nhất một tham số: số port mà Server lắng nghe kết nối. Nếu Server có nhiều card mạng và ta muốn lắng nghe trên một card mạng nào đó thì ta có thể dùng một đối tượng IPEndPoint để chỉ ra địa chỉ IP của card cùng với số port dùng để lắng nghe.

Lớp TcpListener có một số phương thức sau:

Phương Thức	**Mô Tả**
AcceptSocket()	Chấp nhận kết nối trên port và gán kết nối cho một đối tượng Socket
AcceptTcpClient()	Chấp nhận kết nối trên port và gán kết nối cho một đối tượng TCPClient
Equals()	So sánh hai đối tượng TcpListener
GetHashCode()	Lấy hash code
GetType()	Lấy kiểu của thể hiện hiện tại
Pending()	Kiểm tra xem có yêu cầu đang chờ kết nối hay không
Start()	Bắt đầu lắng nghe kết nối
Stop()	Ngừng lắng nghe kết nối
ToString()	Chuyển đối tượng TcpListener thành chuỗi

Phương thức Start() tương tự như phương thứcBind() và Listen() được dùng ở lớp socket. Phương thức Start() kết nối Socket đến EndPoint được định nghĩa ở phương thức tạo lập của lớp TcpListener và đặt TCP port vào chế độ lắng nghe, sẵng sàng chấp nhận kết nối. Phương thức AcceptTcpClient() có thể so sánh với phương thức Accept() của Socket, chấp nhận kết nối và gán nó cho một đối tượng TcpClient.

Sau khi đối tượng TcpClient được tạo ra, tất cả các truyền thông với thiết bị ở xa được thực hiện với đối tượng TcpClient mới chứ không phải với đối tượng

TcpListener ban đầu, do đó đối tượng TcpListener có thể được dùng để chấp nhận kết nối khác. Để đóng đối tượng TcpListener ta dùng phương thức Stop().

Chương trình TCPListener

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
class TcpListenerSample
{
    public static void Main()
    {
        int recv;
        byte[] data = new byte[1024];
        TcpListener newsock = new TcpListener(5000);
        newsock.Start();
        Console.WriteLine("Dan cho client ket noi den...");
        TcpClient client = newsock.AcceptTcpClient();
        NetworkStream ns = client.GetStream();
        string welcome = "Hello Client";
        data = Encoding.ASCII.GetBytes(welcome);
        ns.Write(data, 0, data.Length);
        while (true)
        {
            data = new byte[1024];
            recv = ns.Read(data, 0, data.Length);
            if (recv == 0)
                break;

            Console.WriteLine(Encoding.ASCII.GetString(data, 0, recv));
            ns.Write(data, 0, recv);
        }
        ns.Close();
        client.Close();
        newsock.Stop();
    }
}
```

Chương trình TcpListenerSample đầu tiên tạo ra một đối tượng TcpListener, sử dụng port 5000 và dùng phương thức Start() để đặt đối tượng mới tạo ra vào chế độ

lắng nghe. Sau đó phương thức AcceptTcpClient() chờ kết nối TCP đến và gán kết nối đến này vào một đối tượng TcpClient:

```
TcpListener newsock = new TcpListener(5000);
newsock.Start();
Console.WriteLine("Dan cho client ket noi den...");
TcpClient client = newsock.AcceptTcpClient();
NetworkStream ns = client.GetStream();
```

Khi đối tượng TcpClient được thành lập, một đối tượng NetworkStream được gán vào để truyền thông với máy ở xa. Tất cả các thông tin liên lạc đều được thực hiện bằng cách sử dụng phương thức Read() và Write()

IV.3. Lớp UdpClient

Lớp UdpClient được tạo ra để giúp cho việc lập trình mạng với giao thức UDP được đơn giản hơn. Lớp UdpClient có bốn phương thức tạo lập:

- ✓ UdpClient(): tạo ra một đối tượng UdpClient nhưng không gắn vào bất kỳ địa chỉ hay port nào.
- ✓ UdpClient(int port): gắn đối tượng UdpClient mới tạo vào một port/
- ✓ UdpClient(IPEndPoint iep): gắn đối tượng UdpClient mới tạo vào một địa chỉ Ip cục bộ và một port.
- ✓ UdpClient(string host, int port): gắn đối tượng UdpClient mới tạo vào một địa chỉ IP và một port bất kỳ và kết hợp nó với một địa chỉ IP và port ở xa.

Các phương thức tạo lập của lớp UdpClient tương tự như các phương thức tạo lập của lớp TcpClient, chúng ta có thể để cho hệ thống chọn port thích hợp cho ứng dụng hoặc ta có thể chỉ ra port được dùng trong ứng dụng. Nếu ứng dụng UDP phải chấp nhận dữ liệu trên một port nào đó, ta phải định nghĩa port đó trong phương thức tạo lập của lớp UdpClient.

Một số phương thức của lớp UdpClient:

Phương Thức	**Mô Tả**
Close()	Đóng Socket ở bên dưới
Connect()	Cho phép chỉ ra IP endpoint ở xa để gởi và nhận dữ liệu
DropMulticastGroup()	Gỡ bỏ Socket từ 1 nhóm UDP multicast
Equals()	So sánh hai đối tượng UdpClient

Phương Thức	**Mô Tả**
GetHashCode()	Lấy hash code
GetType()	Lấy kiểu của đối tượng hiện tại
JoinMulticastGroup()	Thêm Socket vào một nhóm UDP multicast
Receive()	Nhận dữ liệu từ Socket
Send()	Gởi dữ liệu đến thiết bị ở xa từ Socket
ToString()	Chuyển đối tượng UdpClient thành chuỗi

Có nhiều chỗ khác nhau giữa phương thức Send(), Receive() của lớp UdpClient và phương thức SendTo(), ReceiveFrom() của Socket.

Phương thức Receive()

Lớp UdpClient sử dụng phương thức Receive() để chấp nhận các gói tin trên một card mạng và một port. Chỉ có một cách sử dụng của phương thức Receive():

```
byte[] Receive(ref IPEndPoint iep)
```

Khi dữ liệu được nhận từ Socket, nó không đặt vào mảng byte trong phương thức như trong phương thức ReceiveFrom() mà nó sẽ trả về một mảng byte. Sự khác nhau thứ hai là phương thức ReceiveFrom() đặt thông tin của máy ở xa vào một đối tượng EndPoint còn phương thức Receive() đặt thông tin của máy ở xa vào một đối tượng IPEndPoint, việc này có thể làm cho lập trình viên cảm thấy dễ lập trình hơn.

Khi nhiều dữ liệu được nhận hơn kích thước bộ đệm, thay vì phát sinh ra một biệt lệ như trong phương thức ReceiveFrom() của Socket, UdpClient tả về một bộ đệm dữ liệu đủ lớn để chứa dữ liệu nhận đây là một tính năng rất hay của phương thức Receive().

Phương thức Send()

Phương thức Send() có ba quá tải hàm để gởi dữ liệu tới thiết bị ở xa:

- ✓ Send(byte[] data, int sz): gởi một mảng dữ liệu với kích thước là sz đến thiết bị ở xa mặc định. Để dùng quá tải hàm này, ta phải chỉ ra thiết bị ở xa mặc định bằng cách hoặc sử dụng phương thức tạo lập của lớp UdpClient hoặc dùng phương thức Connect().
- ✓ Send(byte[] data, int sz, IPEndPoint iep): cho phép gởi mảng dữ liệu có kích thước sz đến thiết bị ở xa được chỉ ra bởi iep.

- ✓ Send(byte[] data, int sz, string host, int port): gởi mảng dữ liệu kích thước sz đến máy ở xa và port được chỉ ra.

Chương trình UdpClient Server

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
class UdpSrvrSample
{
    public static void Main()
    {
        byte[] data = new byte[1024];
        IPEndPoint ipep = new IPEndPoint(IPAddress.Any, 5000);
        UdpClient newsock = new UdpClient(ipep);
        Console.WriteLine("Dang cho client ket noi den...");
        IPEndPoint sender = new IPEndPoint(IPAddress.Any, 0);
        data = newsock.Receive(ref sender);
        Console.WriteLine("Thong diep duoc nhan tu {0}:", sender.ToString());
        Console.WriteLine(Encoding.ASCII.GetString(data, 0, data.Length));
        string welcome = "Xin chao client";
        data = Encoding.ASCII.GetBytes(welcome);
        newsock.Send(data, data.Length, sender);
        while (true)
        {
            data = newsock.Receive(ref sender);

            Console.WriteLine(Encoding.ASCII.GetString(data, 0, data.Length));
            newsock.Send(data, data.Length, sender);
        }
    }
}
```

Chương trình UdpClient Client

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
class UdpClientSample
{
    public static void Main()
    {
        byte[] data = new byte[1024];
```

```
        string input, stringData;
        UdpClient server = new UdpClient("127.0.0.1", 5000);
        IPEndPoint sender = new IPEndPoint(IPAddress.Any, 0);
        string welcome = "Xin chao server";
        data = Encoding.ASCII.GetBytes(welcome);
        server.Send(data, data.Length);
        data = server.Receive(ref sender);
        Console.WriteLine("Thong diep duoc nhan tu {0}:", sender.ToString());
        stringData = Encoding.ASCII.GetString(data, 0, data.Length);
        Console.WriteLine(stringData);
        while (true)
        {
            input = Console.ReadLine();
            if (input == "exit")
                break;

            server.Send(Encoding.ASCII.GetBytes(input), input.Length);
            data = server.Receive(ref sender);
            stringData = Encoding.ASCII.GetString(data, 0, data.Length);
            Console.WriteLine(stringData);
        }
        Console.WriteLine("Dang dong client");
        server.Close();
    }
}
```

CHƯƠNG V: ĐA NHIỆM TIỂU TRÌNH

V.1. Khái niệm tiến trình và tiểu trình của Windows

Tiến trình là một thể hiện của chương trình đang hoạt động. Một tiến trình luôn sở hữu một không gian địa chỉ có kích thước 4GB chứa mã chương trình, các dữ liệu, sở hữu tài nguyên của hệ thống như tập tin, đối tượng đồng bộ hóa....

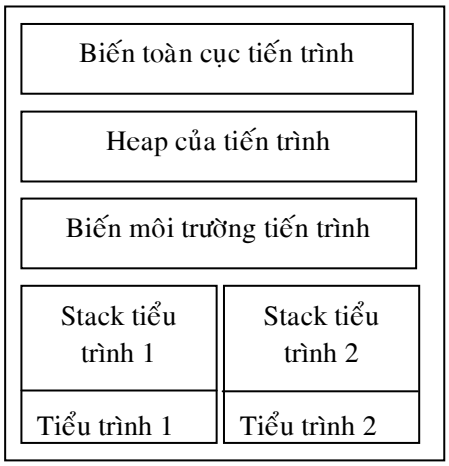
Mỗi tiến trình khi mới tạo lập đều chỉ có một tiểu trình chính nhưng sau đó có thể tạo lập nhiều tiến trình khác.

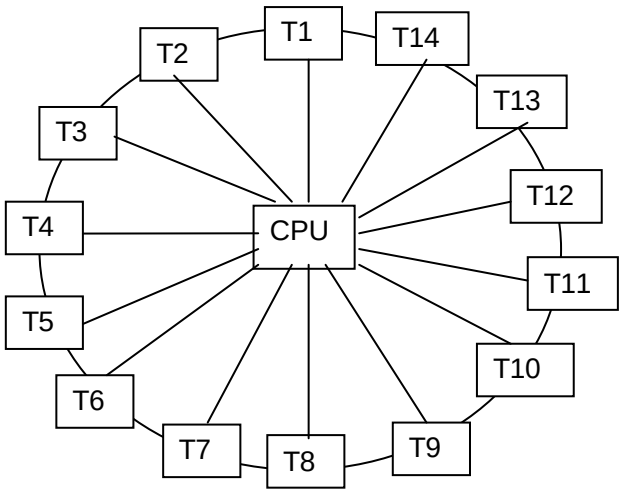
Tiểu trình là một thành phần đơn vị của tiến trình có thể thực hiện các chỉ thị ứng với một đoạn mã nào đó của chương trình.

Hệ điều hành Windows cho phép các tiểu trình hoạt động độc lập và tổ chức điều phối (lập lịch tiến trình) CPU để các tiểu trình hoạt động đồng thời. Hệ điều hành phân chia thời gian sử dụng CPU cho mỗi tiến trình rất mịn theo kiểu xoay vòng. Mỗi tiểu trình có thể có 1 trong 3 trạng thái : Running, Ready, Blocked.

Các tiểu trình trong một tiến trình có thể cùng truy xuất đến các biến toàn cục của tiến trình.

V.2. Mô hình





Các ứng dụng cài đặt theo mô hình đa tiến trình hay đa tiểu trình đều phải đối diện với các vấn đề sau :

- Hệ thống tiêu thụ thêm bộ nhớ để lưu trữ các cấu trúc mô tả tiến trình hay tiểu trình.

- Hệ thống tốn thêm thời gian để theo vết chương trình, quản lý các tiểu trình.
- Nhiều tiến trình tranh chấp tài nguyên dùng chung đòi hỏi thực hiện đồng bộ hóa.

V.3. Các kỹ thuật trong .NET tạo tiểu trình

Thư viện lớp .NET Framework cung cấp một số phương pháp tạo tiểu trình mới:

- ✓ Thực thi một phương phức bằng tiểu trình trong Thread-pool
- ✓ Thực thi phương thức một cách bất đồng bộ
- ✓ Thực thi một phương thức bằng tiểu trình theo chu kỳ hay ở một thời điểm xác định
- ✓ Thực thi phương thức bằng cách ra hiệu đối tượng WaitHandle

V.3.1. Tạo tiểu trình trong Thread-pool

Cách tạo:

- ✓ Khai báo một phương thức chứa mã lệnh cần thực thi
- ✓ Phương thức này phải trả về void và chỉ nhận một đối số.
- ✓ Tạo một thể hiện của ủy nhiệm System.Threading.WaitCallback tham chiếu đến phương thức này.
- ✓ Gọi phương thức tĩnh QueueUserWorkItem của lớp System.Threading.ThreadPool,
- ✓ Truyền thể hiện ủy nhiệm đã tạo làm đối số.
- ✓ Bộ thực thi sẽ xếp thể hiện ủy nhiệm này vào hàng đợi và thực thi nó khi một tiểu trình trong thread-pool sẵn sàng.

Ví dụ thực thi một phương thức có tên là DisplayMessage:

Truyền DisplayMessage đến thread-pool hai lần

Lần đầu không có đối số

Lần sau có đối số là đối tượng MessageInfo

Chương trình ThreadPoolExample

```
using System;
using System.Threading;
// Lớp dùng để truyền dữ liệu cho phương thức DisplayMessage
// khi nó được thực thi bằng thread-pool.
```

```
public class MessageInfo
{
    private int iterations;
    private string message;
    // Phương thức khởi dựng nhận các thiết lập cấu hình cho tiểu trình.
    public MessageInfo(int iterations, string message)
    {
        this.iterations = iterations;
        this.message = message;
    }
    // Các thuộc tính dùng để lấy các thiết lập cấu hình.
    public int Iterations { get { return iterations; } }
    public string Message { get { return message; } }
}
public class ThreadPoolExample
{
    // Hiển thị thông tin ra cửa sổ Console.
    public static void DisplayMessage(object state)
    {
        // Ép đối số state sang MessageInfo.
        MessageInfo config = state as MessageInfo;
        // Nếu đối số config là null, không có đối số nào truyền cho phương
thức
        //               ThreadPool.QueueUserWorkItem;
        // sử dụng các giá trị mặc định.
        if (config == null)
        {
            // Hiển thị một thông báo ra cửa sổ Console ba lần.
            for (int count = 0; count < 3; count++)
            {
                Console.WriteLine("A thread-pool example.");
                // Vào trạng thái chờ, dùng cho mục đích minh họa.
                // Tránh đưa các tiểu trình của thread-pool
                // vào trạng thái chờ trong các ứng dụng thực tế.
                Thread.Sleep(1000);
            }
        }
        else
        {
            // Hiển thị một thông báo được chỉ định trước
            // với số lần cũng được chỉ định trước.
            for (int count = 0; count < config.Iterations; count++)
            {
```

```
                Console.WriteLine(config.Message);
                // Vào trạng thái chờ, dùng cho mục đích minh họa.
                // Tránh đưa các tiểu trình của thread-pool
                // vào trạng thái chờ trong các ứng dụng thực tế.
                Thread.Sleep(1000);
            }
        }
    }
    public static void Main()
    {
        // Tạo một đối tượng ủy nhiệm, cho phép chúng ta
        // truyền phương thức DisplayMessage cho thread-pool.
        WaitCallback workMethod = new
            WaitCallback(ThreadPoolExample.DisplayMessage);
        // Thực thi DisplayMessage bằng thread-pool (không có đối số).
        ThreadPool.QueueUserWorkItem(workMethod);
        // Thực thi DisplayMessage bằng thread-pool (truyền một
        // đối tượng MessageInfo cho phương thức DisplayMessage).
        MessageInfo info = new MessageInfo(5, "A thread-pool example with
arguments.");
        ThreadPool.QueueUserWorkItem(workMethod, info);
        // Nhấn Enter để kết thúc.
        Console.WriteLine("Main method complete. Press Enter.");
        Console.ReadLine();
    }
}
```

Tình huống sử dụng:

Khi một tiểu trình trong thread-pool sẵn sàng, nó nhận công việc kế tiếp từ hàng đợi và thực thi công việc này. Khi đã hoàn tất công việc, thay vì kết thúc, tiểu trình này quay về thread-pool và nhận công việc kế tiếp từ hàng đợi.

Việc sử dụng thread-pool giúp đơn giản hóa việc lập trình hỗ-trợ-đa-tiểu-trình. Tuy nhiên, cần lưu ý khi quyết định sử dụng thread-pool, cần xem xét các điểm sau:

- Không nên sử dụng thread-pool để thực thi các tiến trình chạy trong một thời gian dài. Vì số tiểu trình trong thread-pool là có giới hạn. Đặc biệt, tránh đặt các tiểu trình trong thread-pool vào trạng thái đợi trong một thời gian quá dài.

- Không thể điều khiển lịch trình của các tiểu trình trong thread-pool, cũng như không thể thay đổi độ ưu tiên của các công việc. Thread-pool xử lý các công việc theo thứ tự thêm chúng vào hàng đợi.

V.3.2. Tạo tiểu trình bất đồng bộ

Khi cho gọi một phương thức,thường thực hiện một cách đồng bộ; nghĩa là mã lệnh thực hiện lời gọi phải đi vào trạng thái dừng (block) cho đến khi phương thức được thực hiện xong.

Trong một số trường hợp, cần thực thi phương thức một cách bất đồng bộ; nghĩa là cho thực thi phương thức này trong một tiểu trình riêng trong khi vẫn tiếp tục thực hiện các công việc khác. Sau khi phương thức đã hoàn tất, cần lấy trị trả về của nó .

Nguyên tắc hoạt động:

NET Framework hỗ trợ chế độ thực thi bất đồng bộ, cho phép thực thi bất kỳ phương thức nào một cách bất đồng bộ bằng một ủy nhiệm.

Khi khai báo và biên dịch một ủy nhiệm, trình biên dịch sẽ tự động sinh ra hai phương thức hỗ trợ chế độ thực thi bất đồng bộ: BeginInvoke và EndInvoke. Khi gọi phương thức BeginInvoke của một thể hiện ủy nhiệm, phương thức được tham chiếu bởi ủy nhiệm này được xếp vào hàng đợi để thực thi bất đồng bộ.

Quyền kiểm soát quá trình thực thi được trả về cho mã gọi BeginInvoke ngay sau đó, và phương thức được tham chiếu sẽ thực thi trong ngữ cảnh của tiểu trình sẵn sàng trước tiên trong thread-pool.

Các bước thực hiện:

Khai báo một ủy nhiệm có chữ ký giống như phương thức cần thực thi.

Tạo một thể hiện của ủy nhiệm tham chiếu đến phương thức này.

Gọi phương thức BeginInvoke của thể hiện ủy nhiệm để thực thi phương thức

Sử dụng phương thức EndInvoke để kiểm tra trạng thái của phương thức cũng như thu lấy trị trả về của nó nếu đã hoàn tất .

Các đối số của phương thức BeginInvoke gồm các đối số được chỉ định bởi ủy nhiệm, cộng với hai đối số dùng khi phương thức thực thi bất đồng bộ kết thúc:

- ✓ Một thể hiện của ủy nhiệm System.AsyncCallback tham chiếu đến phương thức mà bộ thực thi sẽ gọi khi phương thức thực thi bất đồng bộ

kết thúc. Phương thức này sẽ được thực thi trong ngữ cảnh của một tiểu trình trong thread-pool. Truyền giá trị null cho đối số này nghĩa là không có phương thức nào được gọi và phải sử dụng một cơ chế khác để xác định khi nào phương thức thực thi bất bộ kết thúc.

- ✓ Một tham chiếu đối tượng mà bộ thực thi sẽ liên kết với quá trình thực thi bất đồng bộ. Phương thức thực thi bất đồng bộ không thể sử dụng hay truy xuất đến đối tượng này, nhưng mã lệnh có thể sử dụng nó khi phương thức này kết thúc, cho phép liên kết thông tin trạng thái với quá trình thực thi bất đồng bộ.

Ví dụ, đối tượng này cho phép ánh xạ các kết quả với các thao tác bất đồng bộ đã được khởi tạo trong trường hợp khởi tạo nhiều thao tác bất đồng bộ nhưng sử dụng chung một phương thức callback để xử lý việc kết thúc.

Phương thức EndInvoke cho phép lấy trị trả về của phương thức thực thi bất đồng bộ, nhưng trước hết phải xác định khi nào nó kết thúc.

Có bốn kỹ thuật dùng để xác định một phương thức thực thi bất đồng bộ đã kết thúc hay chưa:

Blocking: dừng quá trình thực thi của tiểu trình hiện hành cho đến khi phương thức thực thi bất đồng bộ kết thúc. Điều này rất giống với sự thực thi đồng bộ. Tuy nhiên, nếu linh hoạt chọn thời điểm chính xác để đưa mã lệnh vào trạng thái dừng (block) thì vẫn còn cơ hội thực hiện thêm một số việc trước khi mã lệnh đi vào trạng thái này.

Polling: lặp đi lặp lại việc kiểm tra trạng thái của phương thức thực thi bất đồng bộ để xác định nó kết thúc hay chưa. Đây là một kỹ thuật rất đơn giản, nhưng nếu xét về mặt xử lý thì không được hiệu quả. Nên tránh các vòng lặp chặt làm lãng phí thời gian của bộ xử lý; tốt nhất là nên đặt tiểu trình thực hiện polling vào trạng thái nghỉ (sleep) trong một khoảng thời gian bằng cách sử dụng Thread.Sleep giữa các lần kiểm tra trạng thái. Bởi kỹ thuật polling đòi hỏi phải duy trì một vòng lặp nên hoạt động của tiểu trình chờ sẽ bị giới hạn, tuy nhiên có thể dễ dàng cập nhật tiến độ công việc

Waiting: sử dụng một đối tượng dẫn xuất từ lớp System.Threading.WaitHandle để báo hiệu khi phương thức thực thi bất đồng bộ kết thúc. Waiting là một cải tiến của kỹ thuật polling, nó cho phépchờ nhiều phương thức thực thi bất đồng bộ kết thúc.

Có thể chỉ định các giá trị time-out cho phép tiểu trình thực hiện waiting dừng lại nếu phương thức thực thi bất đồng bộ đã diễn ra quá lâu, hoặc muốn cập nhật định kỳ bộ chỉ trạng thái.

Callback: Callback là một phương thức mà bộ thực thi sẽ gọi khi phương thức thực thi bất đồng bộ kết thúc. Mã lệnh thực hiện lời gọi không cần thực hiện bất kỳ thao tác kiểm tra nào, nhưng vẫn có thể tiếp tục thực hiện các công việc khác.
Callback rất linh hoạt nhưng cũng rất phức tạp, đặc biệt khi có nhiều phương thức thực thi bất đồng bộ chạy đồng thời nhưng sử dụng cùng một callback. Trong những trường hợp như thế, phải sử dụng các đối tượng trạng thái thích hợp để so trùng các phương thức đã hoàn tất với các phương thức đã khởi tạo.

Ví dụ:

Lớp AsyncExecutionExample trong mô tả cơ chế thực thi bất đồng bộ. Nó sử dụng một ủy nhiệm có tên là AsyncExampleDelegate để thực thi bất đồng bộ một phương thức có tên là LongRunningMethod.

Phương thức LongRunningMethod sử dụng Thread.Sleep để mô phỏng một phương thức có thời gian thực thi dài:

```
    // Ủy nhiệm cho phép bạn thực hiện việc thực thi bất đồng bộ
    //của AsyncExecutionExample.LongRunningMethod.
    public delegate DateTime AsyncExampleDelegate(int delay, string
name);
    // Phương thức có thời gian thực thi dài.
    public static DateTime LongRunningMethod(int delay, string name)
    {
        Console.WriteLine("{0} : {1} example - thread starting.",
                    DateTime.Now.ToString("HH:mm:ss.ffff"), name);
        // Mô phỏng việc xử lý tốn nhiều thời gian.
        Thread.Sleep(delay);
        Console.WriteLine("{0}:{1}example-thread finishing.",
                    DateTime.Now.ToString("HH:mm:ss.ffff"), name);
        // Trả về thời gian hoàn tất phương thức.
        return DateTime.Now;
    }
```

AsyncExecutionExample chứa 5 phương thức diễn đạt các cách tiếp cận khác nhau về việc kết thúc phương thức thực thi bất đồng bộ.

V.3.2.1. Phương thức BlockingExample

Phương thức BlockingExample thực thi bất đồng bộ phương thức LongRunningMethod và tiếp tục thực hiện công việc của nó trong một khoảng thời gian. Khi xử lý xong công việc này, BlockingExample chuyển sang trang thái dừng (block) cho đến khi phương thức LongRunningMethod kết thúc.

Để vào trạng thái dừng, BlockingExample thực thi phương thức EndInvoke của thể hiện ủy nhiệm AnsyncExampleDelegate. Nếu phương thức LongRunningMethod kết thúc, EndInvoke trả về ngay lập tức, nếu không, BlockingExample chuyển sang trạng thái dừng cho đến khi phương thức LongRunningMethod kết thúc.

```
public static void BlockingExample()
    {
        Console.WriteLine(Environment.NewLine + "*** Running
Blocking Example ***");
        // Gọi LongRunningMethod một cách bất đồng bộ. Truyền null
cho
        // cả ủy nhiệm callback và đối tượng trạng thái bất đồng bộ.
        AsyncExampleDelegate longRunningMethod = new
AsyncExampleDelegate(LongRunningMethod);
        IAsyncResult asyncResult =
longRunningMethod.BeginInvoke(2000, "Blocking", null, null);
        // Thực hiện công việc khác cho đến khi
        // sẵn sàng đi vào trạng thái dừng.
        for (int count = 0; count < 3; count++)
        {
            Console.WriteLine("{0} : Continue processing until " +
"ready to block...",
                    DateTime.Now.ToString("HH:mm:ss.ffff"));
            Thread.Sleep(200);
        }
        // Đi vào trạng thái dừng cho đến khi phương thức
        // thực thi bất đồng bộ kết thúc và thu lấy kết quả.
        Console.WriteLine("{0} : Blocking until method is
complete...", DateTime.Now.ToString("HH:mm:ss.ffff"));
```

```
        DateTime completion =
                                longRunningMethod.EndInvoke(asyncResult);
        // Hiển thị thông tin kết thúc.
        Console.WriteLine("{0} : Blocking example complete.",
completion.ToString("HH:mm:ss.ffff"));
    }
```

V.3.2.2. Phương thức PollingExample

Phương thức PollingExample thực thi bất đồng bộ phương thức LongRunningMethod và sau đó thực hiện vòng lặp polling cho đến khi LongRunningMethod kết thúc.

PollingExample kiểm tra thuộc tính IsComplete của thể hiện IAsyncResult (được trả về bởi BeginInvoke) để xác định phương thức LongRunningMethod đã kết thúc hay chưa, nếu chưa, PollingExample sẽ gọi Thread.Sleep.

```
public static void PollingExample()
    {
        Console.WriteLine(Environment.NewLine + " Running Polling
Example");
        // Gọi LongRunningMethod một cách bất đồng bộ. Truyền null
cho
        // cả ủy nhiệm callback và đối tượng trạng thái bất đồng bộ.
AsyncExampleDelegate longRunningMethod =  new
     AsyncExampleDelegate(LongRunningMethod);

        IAsyncResult asyncResult =
longRunningMethod.BeginInvoke(2000, "Polling", null, null);
        // Thực hiện polling để kiểm tra phương thức thực thi
        // bất đồng bộ kết thúc hay chưa. Nếu chưa kết thúc thì đi
vào
        // trạng thái chờ trong 300 mini-giây trước khi thực hiện
polling lần nữa.
        Console.WriteLine("{0} : Poll repeatedly until method is
complete...", DateTime.Now.ToString("HH:mm:ss.ffff")); while
(!asyncResult.IsCompleted)
        {
            Console.WriteLine("{0} : Polling...",
```

```
                DateTime.Now.ToString("HH:mm:ss.ffff"));
            Thread.Sleep(300);
        }
        // Thu lấy kết quả của phương thức thực thi bất đồng bộ.
        DateTime completion =
longRunningMethod.EndInvoke(asyncResult);
        // Hiển thị thông tin kết thúc.
        Console.WriteLine("{0} : Polling example complete.",
completion.ToString("HH:mm:ss.ffff"));
    }
```

V.3.2.3. Phương thức WaitingExample

Phương thức WaitingExample thực thi bất đồng bộ phương thức LongRunningExample và sau đó chờ cho đến khi LongRunningMethod kết thúc. WaitingExample sử dụng thuộc tính AsyncWaitHandle của thể hiện IAsyncResult (được trả về bởi BeginInvoke) để có được một WaitHandle.

Gọi phương thức WaitOne của WaitHandle. Việc sử dụng giá trị time-out cho phép WaitingExample dừng quá trình đợi để thực hiện công việc khác hoặc dừng hoàn toàn nếu phương thức thực thi bất đồng bộ diễn ra quá lâu.

```
  public static void WaitingExample()
    {
        Console.WriteLine(Environment.NewLine + "*** Running Waiting
Example ***");
        // Gọi LongRunningMethod một cách bất đồng bộ. Truyền null
cho
        // cả ủy nhiệm callback và đối tượng trạng thái bất đồng bộ.
        AsyncExampleDelegate longRunningMethod = new
                AsyncExampleDelegate(LongRunningMethod);
        IAsyncResult asyncResult =
                    longRunningMethod.BeginInvoke(2000, "Waiting",
null, null);
        // Đợi phương thức thực thi bất đồng bộ kết thúc.
        // Time-out sau 300 mili-giây và hiển thị trạng thái ra
        // cửa sổ Console trước khi tiếp tục đợi.
        Console.WriteLine("{0} : Waiting until method is
complete...", DateTime.Now.ToString("HH:mm:ss.ffff"));
```

```
        while (!asyncResult.AsyncWaitHandle.WaitOne(300, false))
        {
            Console.WriteLine("{0} : Wait timeout...",
DateTime.Now.ToString("HH:mm:ss.ffff"));
        }
        // Thu lấy kết quả của phương thức thực thi bất đồng bộ.
        DateTime completion =
longRunningMethod.EndInvoke(asyncResult);
        // Hiển thị thông tin kết thúc.
        Console.WriteLine("{0} : Waiting example complete.",
completion.ToString("HH:mm:ss.ffff"));
    }
```

V.3.2.4. Phương thức WaitAllExample

Phương thức WaitAllExample thực thi bất đồng bộ phương thức LongRunningMethod nhiều lần và sau đó sử dụng mảng các đối tượng WaitHandle để đợi cho đến khi tất cả các phương thức kết thúc.

```
public static void WaitAllExample()
    {
        Console.WriteLine(Environment.NewLine + "*** Running WaitAll
Example ***");
        // Một ArrayList chứa các thể hiện IAsyncResult
        // cho các phương thức thực thi bất đồng bộ.
        ArrayList asyncResults = new ArrayList(3);
        // Gọi ba lần LongRunningMethod một cách bất đồng bộ.
        // Truyền null cho cả ủy nhiệm callback và đối tượng
        // trạng thái bất đồng bộ. Thêm thể hiện IAsyncResult
        // cho mỗi phương thức vào ArrayList.
        AsyncExampleDelegate longRunningMethod = new
AsyncExampleDelegate(LongRunningMethod);
        asyncResults.Add(longRunningMethod.BeginInvoke(3000,
"WaitAll 1", null, null));
        asyncResults.Add(longRunningMethod.BeginInvoke(2500,
"WaitAll 2", null, null));
        asyncResults.Add(longRunningMethod.BeginInvoke(1500,
"WaitAll 3", null, null));
        // Tạo một mảng các đối tượng WaitHandle,
```

```
        // sẽ được sử dụng để đợi tất cả các phương thức
        // thực thi bất đồng bộ kết thúc.
        WaitHandle[] waitHandles = new WaitHandle[3];
        for (int count = 0; count < 3; count++)
        {
            waitHandles[count] =
            ((IAsyncResult)asyncResults[count]).AsyncWaitHandle;
        }
        // Đợi cả ba phương thức thực thi bất đồng bộ kết thúc.
        // Time-out sau 300 mili-giây và hiển thị trạng thái ra
        // cửa sổ Console trước khi tiếp tục đợi.
        Console.WriteLine("{0} : Waiting until all 3 methods are
complete...", DateTime.Now.ToString("HH:mm:ss.ffff")); while
(!WaitHandle.WaitAll(waitHandles, 300, false)) {
Console.WriteLine("{0} : WaitAll timeout...",
DateTime.Now.ToString("HH:mm:ss.ffff")); }
        // Kiểm tra kết quả của mỗi phương thức và xác định
        // thời gian phương thức cuối cùng kết thúc.
        DateTime completion = DateTime.MinValue;
        foreach (IAsyncResult result in asyncResults)
        {
            DateTime time = longRunningMethod.EndInvoke(result);
            if (time > completion) completion = time;
        }
        // Hiển thị thông tin kết thúc.
        Console.WriteLine("{0} : WaitAll example complete.",
completion.ToString("HH:mm:ss.ffff"));
    }
```

V.3.2.5. Phương thức CallbackExample

Phương thức CallbackExample thực thi bất đồng bộ phương thức LongRunningMethod và truyền một thể hiện ủy nhiệm AsyncCallback (tham chiếu đến phương thức CallbackHandler) cho phương thức BeginInvoke.

Phương thức CallbackHandler sẽ được gọi một cách tự động khi phương thức LongRunningMethod kết thúc. Kết quả là phương thức CallbackExample vẫn tiếp tục thực hiện công việc.

```
public static void CallbackExample()
```

```
    {
        Console.WriteLine(Environment.NewLine +
                "*** Running Callback Example ***");
        // Gọi LongRunningMethod một cách bất đồng bộ. Truyền một
        // thể hiện ủy nhiệm AsyncCallback tham chiếu đến
        // phương thức CallbackHandler. CallbackHandler sẽ
        // tự động được gọi khi phương thức thực thi bất đồng bộ
        // kết thúc. Truyền một tham chiếu đến thể hiện ủy nhiệm
        // AsyncExampleDelegate như một trạng thái bất đồng bộ;
        // nếu không, phương thức callback không thể truy xuất
        // thể hiện ủy nhiệm để gọi EndInvoke.
        AsyncExampleDelegate longRunningMethod = new
                    AsyncExampleDelegate(LongRunningMethod);
        IAsyncResult asyncResult =
longRunningMethod.BeginInvoke(2000, "Callback", new
AsyncCallback(CallbackHandler), longRunningMethod);
        // Tiếp tục với công việc khác.
        for (int count = 0; count < 15; count++)
        {
            Console.WriteLine("{0} : Continue processing...",
                DateTime.Now.ToString("HH:mm:ss.ffff"));
            Thread.Sleep(200);
        }
    }
    // Phương thức xử lý việc kết thúc bất đồng bộ bằng
callbacks.public
    static void CallbackHandler(IAsyncResult result)
    {
        // Trích tham chiếu đến thể hiện AsyncExampleDelegate
        // từ thể hiện IAsyncResult.
        AsyncExampleDelegate longRunningMethod =
(AsyncExampleDelegate)result.AsyncState;
        // Thu lấy kết quả của phương thức thực thi bất đồng bộ.
        DateTime completion = longRunningMethod.EndInvoke(result);
        // Hiển thị thông tin kết thúc.
        Console.WriteLine("{0} : Callback example complete.",
completion.ToString("HH:mm:ss.ffff"));
```

```
}
```

V.3.3. Thực thi phương thức bằng Timer

Kỹ thuật này giúp thực thi một phương thức trong một tiểu trình riêng theo chu kỳ hay ở một thời điểm xác định. Thông thường, rất hữu ích khi thực thi một phương thức ở một thời điểm xác định hay ở những thời khoảng xác định. Ví dụ, cần sao lưu dữ liệu lúc 1:00 AM mỗi ngày hay xóa vùng đệm dữ liệu mỗi 20 phút.

Lớp Timer giúp việc định thời thực thi một phương thức trở nên dễ dàng, cho phép thực thi một phương thức được tham chiếu bởi ủy nhiệm TimerCallback ở những thời khoảng nhất định. Phương thức được tham chiếu sẽ thực thi trong ngữ cảnh của một tiểu trình trong thread-pool.

Cách thực hiện

Khai báo một phương thức trả về void và chỉ nhận một đối tượng làm đối số. Sau đó, tạo một thể hiện ủy nhiệm System.Threading.TimerCallback tham chiếu đến phương thức này.

Tiếp theo, tạo một đối tượng System.Threading.Timer và truyền nó cho thể hiện ủy nhiệm TimerCallback cùng với một đối tượng trạng thái mà Timer sẽ truyền cho phương thức khi Timer hết hiệu lực.

Bộ thực thi sẽ chờ cho đến khi Timer hết hiệu lực và sau đó gọi phương thức bằng một tiểu trình trong thread-pool.

Khi tạo một đối tượng Timer, cần chỉ định hai thời khoảng (thời khoảng có thể được chỉ định là các giá trị kiểu int, long, uint, hay System.TimeSpan):

- ✓ Giá trị đầu tiên là thời gian trễ (tính bằng mili-giây) để phương thức được thực thi lần đầu tiên. Chỉ định giá trị 0 để thực thi phương thức ngay lập tức, và chỉ định System.Threading.Timeout.Infinite để tạo Timer ở trạng thái chưa bắt đầu (unstarted).
- ✓ Giá trị thứ hai là khoảng thời gian mà Timer sẽ lặp lại việc gọi phương thức sau lần thực thi đầu tiên. Nếu chỉ định giá trị 0 hay Timeout.Infinite thì Timer chỉ thực thi phương thức một lần duy nhất (với điều kiện thời gian trễ ban đầu không phải là Timeout.Infinite). Đối số thứ hai có thể cung cấp bằng các trị kiểu int, long, uint, hay System.TimeSpan.

Sau khi tạo đối tượng Timer, cũng có thể thay đổi các thời khoảng được sử dụng bởi Timer bằng phương thức Change, nhưng không thể thay đổi phương thức sẽ được gọi.

Khi đã dùng xong Timer, nên gọi phương thức Timer.Depose để giải phóng tài nguyên hệ thống bị chiếm giữ bởi Timer. Việc hủy Timer cũng hủy luôn phương thức đã được định thời thực thi.

Ví dụ

Lớp TimerExample dưới đây trình bày cách sử dụng Timer để gọi một phương thức có tên là TimerHandler. Ban đầu, Timer được cấu hình để gọi TimerHandler sau hai giây và lặp lại sau một giây. Ví dụ này cũng trình bày cách sử dụng phương thức Timer.Change để thay đổi các thời khoảng.

Chương trình TimerExample

```
using System;
using System.Threading;
public class TimerExample
{
    // Phương thức sẽ được thực khi Timer hết hiệu lực.
    // Hiển thị một thông báo ra cửa sổ Console.
    private static void TimerHandler(object state)
    {
        Console.WriteLine("{0} : {1}", DateTime.Now.ToString("HH:mm:ss.ffff"),
state);
    }
    public static void Main()
    {
        // Tạo một thể hiện ủy nhiệm TimerCallback mới
        // tham chiếu đến phương thức tĩnh TimerHandler.
        // TimerHandler sẽ được gọi khi Timer hết hiệu lực.
        TimerCallback handler = new TimerCallback(TimerHandler);
        // Tạo một đối tượng trạng thái, đối tượng này sẽ được
        // truyền cho phương thức TimerHandler.
        // Trong trường hợp này, một thông báo sẽ được hiển thị.
        string state = "Timer expired.";
        Console.WriteLine("{0} : Creating Timer.",
DateTime.Now.ToString("HH:mm:ss.ffff"));
        // Tạo một Timer, phát sinh lần đầu tiên sau hai giây
        // và sau đó là mỗi giây.
        using (Timer timer = new Timer(handler, state, 2000, 1000))
```

```
        {
            int period;
            // Đọc thời khoảng mới từ Console cho đến khi
            // người dùng nhập 0. Các giá trị không hợp lệ
            // sẽ sử dụng giá trị mặc định là 0 (dừng ví dụ).
            do
            {
                try
                {
                    period = Int32.Parse(Console.ReadLine());
                }
                catch { period = 0; }
                // Thay đổi Timer với thời khoảng mới.
                if (period > 0) timer.Change(0, period);
            } while (period > 0);
        }
        // Nhấn Enter để kết thúc.
        Console.WriteLine("Main method complete. Press Enter.");
        Console.ReadLine();
    }
}
```

V.3.4. Thực thi phương thức bằng tiểu trình mới

Thực thi mã lệnh trong một tiểu trình riêng, và muốn kiểm soát hoàn toàn quá trình thực thi và trạng thái của tiểu trình đó.

Để tăng độ linh hoạt và mức độ kiểm soát khi hiện thực các ứng dụng hỗ-trợ-đa-tiểu-trình, bạn phải trực tiếp tạo và quản lý các tiểu trình. Lớp Thread cung cấp một cơ chế mà qua đó bạn có thể tạo và kiểm soát các tiểu trình.

Các bước thực hiện:

Tạo một đối tượng ủy nhiệm ThreadStart tham chiếu đến phương thức chứa mã lệnh mà muốn dùng một tiểu trình mới để chạy nó. Giống như các ủy nhiệm khác, ThreadStart có thể tham chiếu đến một phương thức tĩnh hay phương thức của một đối tượng. Phương thức được tham chiếu phải trả về void và không có đối số.

Tạo một đối tượng Thread, và truyền thể hiện ủy nhiệm ThreadStart cho phương thức khởi dựng của nó. Tiểu trình mới có trạng thái ban đầu là Unstarted (một thành viên thuộc kiểu liệt kê System.Threading.ThreadState).

Gọi thực thi phương thức Start của đối tượng Thread để chuyển trạng thái của nó sang ThreadState.Running và bắt đầu thực thi phương thức được tham chiếu bởi thể hiện ủy nhiệm ThreadStart.

Vì ủy nhiệm ThreadStart khai báo không có đối số, nên không thể truyền dữ liệu trực tiếp cho phương thức được tham chiếu. Để truyền dữ liệu cho tiểu trình mới, cần phải cấu hình dữ liệu là khả truy xuất đối với mã lệnh đang chạy trong tiểu trình mới.

Cách tiếp cận thông thường là tạo một lớp đóng gói cả dữ liệu cần cho tiểu trình và phương thức được thực thi bởi tiểu trình. Khi muốn chạy một tiểu trình mới, thì tạo một đối tượng của lớp này, cấu hình trạng thái cho nó, và rồi chạy tiểu trình.

Ví dụ

```
using System;
using System.Threading;
public class ThreadExample
{
    // Các biến giữ thông tin trạng thái.
    private int iterations;
    private string message;
    private int delay;
    public ThreadExample(int iterations, string message, int delay)
    {
        this.iterations = iterations;
        this.message = message;
        this.delay = delay;
    }
    public void Start()
    {
        // Tạo một thể hiện ủy nhiệm ThreadStart
        // tham chiếu đến DisplayMessage.
        ThreadStart method = new ThreadStart(this.DisplayMessage);
        // Tạo một đối tượng Thread và truyền thể hiện ủy nhiệm
        // ThreadStart cho phương thức khởi dựng của nó.
        Thread thread = new Thread(method);
        Console.WriteLine("{0} : Starting new thread.",
                DateTime.Now.ToString("HH:mm:ss.ffff"));
        // Khởi chạy tiểu trình mới.
        thread.Start();
    }
    private void DisplayMessage()
```

```
    {
        // Hiển thị thông báo ra cửa sổ Console với số lần
        // được chỉ định (iterations), nghỉ giữa mỗi thông báo
        // một khoảng thời gian được chỉ định (delay).
        for (int count = 0; count < iterations; count++)
        {
            Console.WriteLine("{0} : {1}",
DateTime.Now.ToString("HH:mm:ss.ffff"), message);
            Thread.Sleep(delay);
        }
    }
    public static void Main()
    {
        // Tạo một đối tượng ThreadExample.
        ThreadExample example = new
            ThreadExample(5, "A thread example.", 500);
        // Khởi chạy đối tượng ThreadExample.
        example.Start();
        // Tiếp tục thực hiện công việc khác.
        for (int count = 0; count < 13; count++)
        {
            Console.WriteLine("{0} : Continue processing...",
DateTime.Now.ToString("HH:mm:ss.ffff"));
            Thread.Sleep(200);
        }
        // Nhấn Enter để kết thúc.
        Console.WriteLine("Main method complete. Press Enter.");
        Console.ReadLine();
    }
}
```

V.3.5. Điều khiển quá trình thực thi của một tiểu trình

Cần nắm quyền điều khiển khi một tiểu trình chạy và dừng, có thể tạm dừng quá trình thực thi của một tiểu trình

Các phương thức của lớp Thread: Abort, Interrupt, Resume, Start, và Suspend cung cấp một cơ chế điều khiển mức cao lên quá trình thực thi của một tiểu trình. Mỗi phương thức này trở về tiểu trình đang gọi ngay lập tức. Kết quả là phải viết mã để bắt và thụ lý các ngoại lệ có thể bị ném khi ta cố điều khiển quá trình thực thi của một Thread.

Phương Thức	**Mô Tả**
Abort()	- Kết thúc một tiểu trình bằng cách ném ngoại lệ System.Threading. ThreadAbortException trong mã lệnh đang được chạy. - Mã lệnh của tiểu trình bị hủy có thể bắt ngoại lệ ThreadAbortException để thực hiện việc dọn dẹp, nhưng bộ thực thi sẽ tự động ném ngoại lệ này lần nữa để bảo đảm tiểu trình kết thúc, trừ khi ResetAbort được gọi.
Interrupt()	- Ném ngoại lệ - System.Threading.ThreadInterruptedException (trong mã lệnh đang được chạy) lúc tiểu trình đang ở trạng thái WaitSleepJoin. Điều này nghĩa là tiểu trình này đã gọi Sleep, Join hoặc đang đợi WaitHandle - Nếu tiểu trình này không ở trạng thái WaitSleepJoin, ThreadInterruptedException sẽ bị ném sau khi tiểu trình đi vào trạng thái WaitSleepJoin.
Resume	- Phục hồi quá trình thực thi của một tiểu trình đã bị tạm ngưng . - Việc gọi Resume trên một tiểu trình chưa bị tạm hoãn sẽ sinh ra ngoại lệ System.Threading.ThreadStateException trong tiểu trình đang gọi.
Start	Khởi chạy tiểu trình mới;
Suspend	- Tạm hoãn quá trình thực thi của một tiểu trình cho đến khi phương thức Resume được gọi. - Việc tạm hoãn một tiểu trình đã bị tạm hoãn sẽ không có hiệu lực - Việc gọi Suspend trên một tiểu trình chưa khởi chạy hoặc đã kết thúc sẽ sinh ra ngoại lệ ThreadStateException trong tiểu trình đang gọi.

Ví dụ:

Lớp ThreadControlExample:

- ✓ Ví dụ này khởi chạy một tiểu trình thứ hai, hiển thị định kỳ một thông báo ra cửa sổ Console và rồi đi vào trạng thái nghỉ (sleep).
- ✓ Bằng cách nhập các lệnh tại dấu nhắc lệnh, ta có thể gián đoạn, tạm hoãn, phục hồi, và hủy bỏ tiểu trình thứ hai.

Chương trình ThreadControlExample

```
using System;
using System.Threading;
public class ThreadControlExample
{
    private static void DisplayMessage()
    {
        // Lặp đi lặp lại việc hiển thị một thông báo ra cửa sổ Console.
        while (true)
        {
            try
            {
                Console.WriteLine("{0} : Second thread running. Enter
(S)uspend, (R)esume, (I)nterrupt, or (E)xit.",
DateTime.Now.ToString("HH:mm:ss.ffff"));
                // Nghỉ 2 giây.
                Thread.Sleep(2000);
            }
            catch (ThreadInterruptedException)
            {
                // Tiểu trình đã bị gián đoạn. Việc bắt ngoại lệ
                // ThreadInterruptedException cho phép ví dụ này
                // thực hiện hành động phù hợp và tiếp tục thực thi.

                Console.WriteLine("{0} : Second thread interrupted.",
DateTime.Now.ToString("HH:mm:ss.ffff"));
            }
            catch (ThreadAbortException abortEx)
            {
                // Đối tượng trong thuộc tính
                // ThreadAbortException.ExceptionState được cung cấp
                // bởi tiểu trình đã gọi Thread.Abort.
                // Trong trường hợp này, nó chứa một chuỗi
                // mô tả lý do của việc hủy bỏ.
                Console.WriteLine("{0} : Second thread aborted ({1})",
DateTime.Now.ToString("HH:mm:ss.ffff"), abortEx.ExceptionState);
```

```
                // Mặc dù ThreadAbortException đã được thụ lý,
                // bộ thực thi sẽ ném nó lần nữa để bảo đảmtiểu trình kết thúc
            }
        }
    }
    public static void Main()
    {
        // Tạo một đối tượng Thread và truyền cho nó một thể hiện
        // ủy nhiệm ThreadStart tham chiếu đến DisplayMessage.
        Thread thread = new Thread(new ThreadStart(DisplayMessage));
        Console.WriteLine("{0} : Starting second thread.",
DateTime.Now.ToString("HH:mm:ss.ffff"));
        // Khởi chạy tiểu trình thứ hai.
        thread.Start();
        // Lặp và xử lý lệnh do người dùng nhập.
        char command = ' ';
        do
        {
            string input = Console.ReadLine();
            if (input.Length > 0)
                command = input.ToUpper()[0];
            else command = ' ';
            switch (command)
            {
                case 'S':
                    // Tạm hoãn tiểu trình thứ hai.

                    Console.WriteLine("{0} : Suspending second thread.",
DateTime.Now.ToString("HH:mm:ss.ffff"));
                    thread.Suspend();
                    break;
                case 'R':
                    // Phục hồi tiểu trình thứ hai.
                    try
                    {
                        Console.WriteLine("{0} : Resuming second thread.",
DateTime.Now.ToString("HH:mm:ss.ffff"));
                        thread.Resume();
                    }
                    catch (ThreadStateException)
                    {
                        Console.WriteLine("{0} : Thread wasn't suspended.",
DateTime.Now.ToString("HH:mm:ss.ffff"));
```

```
                    } break;
                case 'I':
                    // Gián đoạn tiểu trình thứ hai.
                    Console.WriteLine("{0} : Interrupting second thread.",
DateTime.Now.ToString("HH:mm:ss.ffff"));
                    thread.Interrupt();
                    break;
                case 'E':
                    // Hủy bỏ tiểu trình thứ hai và truyền một đối tượng
                    // trạng thái cho tiểu trình đang bị hủy,
                    // trong trường hợp này là một thông báo.
                    Console.WriteLine("{0} : Aborting second thread.",
DateTime.Now.ToString("HH:mm:ss.ffff"));
                    thread.Abort("Terminating example.");
                    // Đợi tiểu trình thứ hai kết thúc.
                    thread.Join();
                    break;
            }
        }
        while (command != 'E');
        // Nhấn Enter để kết thúc.

        Console.WriteLine("Main method complete. Press Enter.");
        Console.ReadLine();
    }
}
```

V.3.6. Nhận biết khi nào một tiểu trình kết thúc

Để kiểm tra một tiểu trình đã kết thúc hay chưa là kiểm tra thuộc tính Thread.IsAlive. Thuộc tính này trả về true nếu tiểu trình đã được khởi chạy nhưng chưa kết thúc hay bị hủy.

Thông thường,cần một tiểu trình để đợi một tiểu trình khác hoàn tất việc xử lý của nó. Thay vì kiểm tra thuộc tính IsAlive trong một vòng lặp, có thể sử dụng phương thức Thread.Join. Phương thức này khiến tiểu trình đang gọi dừng lại (block) cho đến khi tiểu trình được tham chiếu kết thúc.

Có thể tùy chọn chỉ định một khoảng thời gian (giá trị int hay TimeSpan) mà sau khoảng thời gian này, Join sẽ hết hiệu lực và quá trình thực thi của tiểu trình đang

gọi sẽ phục hồi lại. Nếu chỉ định một giá trị time-out, Join trả về true nếu tiểu trình đã kết thúc, và false nếu Join đã hết hiệu lực.

Ví dụ

Ví dụ thực thi một tiểu trình thứ hai và rồi gọi Join để đợi tiểu trình thứ hai kết thúc. Vì tiểu trình thứ hai mất 5 giây để thực thi, nhưng phương thức Join chỉ định giá trị time-out là 3 giây, nên Join sẽ luôn hết hiệu lực và ví dụ này sẽ hiển thị một thông báo ra cửa sổ Console.

Chương trình ThreadFinishExample

```
using System;
using System.Threading;
public class ThreadFinishExample
{
    private static void DisplayMessage()
    {
        // Hiển thị một thông báo ra cửa sổ Console 5 lần.
        for (int count = 0; count < 5; count++)
        {
            Console.WriteLine("{0} : Second thread",
            DateTime.Now.ToString("HH:mm:ss.ffff"));
            // Nghỉ 1 giây.
            Thread.Sleep(1000);
        }
    }
    public static void Main()
    {
        // Tạo một thể hiện ủy nhiệm ThreadStart
        // tham chiếu đến DisplayMessage.
        ThreadStart method = new ThreadStart(DisplayMessage);
        // Tạo một đối tượng Thread và truyền thể hiện ủy nhiệm
        // ThreadStart cho phương thức khởi dựng của nó.
        Thread thread = new Thread(method);
        Console.WriteLine("{0} : Starting second thread.",
DateTime.Now.ToString("HH:mm:ss.ffff"));
        // Khởi chạy tiểu trình thứ hai.
        thread.Start();
        // Dừng cho đến khi tiểu trình thứ hai kết thúc,
        // hoặc Join hết hiệu lực sau 3 giây.
        if (!thread.Join(3000))
        {
```

```
            Console.WriteLine("{0} : Join timed out !!",
DateTime.Now.ToString("HH:mm:ss.ffff"));
        }
        // Nhấn Enter để kết thúc.
        Console.WriteLine("Main method complete. Press Enter.");
        Console.ReadLine();
    }
}
```

V.3.7. Khởi chạy một tiến trình mới

Lớp Process cung cấp một dạng biểu diễn được quản lý cho một tiến trình của hệ điều hành. Lớp Process hiện thực bốn quá tải hàm phương thức Start. Hai trong số này là các phương thức tĩnh, cho phép chỉ định tên và các đối số cho tiến trình mới. Ví dụ, hai lệnh dưới đây đều thực thi Notepad trong một tiến trình mới:

```
// Thực thi notepad.exe, không có đối số.
        Process.Start("notepad.exe");
// Thực thi notepad.exe, tên file cần mở là đối số.
        Process.Start("notepad.exe", "SomeFile.txt");
```

Hai dạng khác của phương thức Start yêu cầu tạo đối tượng ProcessStartInfo được cấu hình với các chi tiết của tiến trình cần chạy.Việc sử dụng đối tượng ProcessStartInfo cung cấp một cơ chế điều khiển tốt hơn trên các hành vi và cấu hình của tiến trình mới.

Tóm tắt một vài thuộc tính thông dụng của lớp ProcessStartInfo:

- ✓ Arguments: Các đối số dùng để truyền cho tiến trình mới
- ✓ ErrorDialog :Nếu Process.Start không thể khởi chạy tiến trình đã được chỉ định, nó sẽ ném ngoại lệ System.ComponentModel.Win32Exception. Nếu ErrorDialog là true, Start sẽ hiển thị một thông báo lỗi trước khi ném ngoại lệ
- ✓ FileName : Tên của ứng dụng.
- ✓ WindowStyle :Điều khiển cách thức hiển thị của cửa sổ. Các giá trị hợp lệ bao gồm: Hidden, Maximized, Minimized, và Normal
- ✓ WorkingDirectory : Tên đầy đủ của thư mục làm việc

Khi đã hoàn tất với một đối tượng Process, bạn nên hủy nó để giải phóng các tài nguyên hệ thống, gọi phương thức Close(), Dispose().

Ví dụ sau sử dụng Process để thực thi Notepad trong một cửa sổ ở trạng thái phóng to và mở một file có tên là C:\Temp\file.txt. Sau khi tạo, ví dụ này sẽ gọi phương thức Process.WaitForExit để dừng tiểu trình đang chạy cho đến khi tiến trình kết thúc hoặc giá trị time-out hết hiệu lực.

Chương trình StartProcessExample

```
using System;
using System.Diagnostics;
public class StartProcessExample
{
    public static void Main()
    {
        // Tạo một đối tượng ProcessStartInfo và cấu hình cho nó
        // với các thông tin cần thiết để chạy tiến trình mới.
        ProcessStartInfo startInfo = new ProcessStartInfo();
        startInfo.FileName = "notepad.exe";
        startInfo.Arguments = "file.txt";
        startInfo.WorkingDirectory = "C:\\Temp";
        startInfo.WindowStyle = ProcessWindowStyle.Maximized;
        startInfo.ErrorDialog = true;
        // Tạo một đối tượng Process mới.
        using (Process process = new Process())
        {
            // Gán ProcessStartInfo vào Process.
            process.StartInfo = startInfo;
            try
            {
                // Khởi chạy tiến trình mới.
                process.Start();
                // Đợi tiến trình mới kết thúc trước khi thoát.
                Console.WriteLine("Waiting 30 seconds for process to finish.");
                process.WaitForExit(30000);
            }
            catch (Exception ex)
            {
                Console.WriteLine("Could not start process.");
                Console.WriteLine(ex);
            }
        }
        // Nhấn Enter để kết thúc.
        Console.WriteLine("Main method complete. Press Enter.");
        Console.ReadLine();
```

```
    }
}
```

V.3.8. Kết thúc một tiến trình

Nếu khởi chạy một tiến trình mới từ mã lệnh được quản lý bằng lớp Process , có thể kết thúc tiến trình mới bằng đối tượng Process mô tả tiến trình này. Một khi đã có đối tượng Process mô tả tiến trình cần kết thúc, cần gọi phương thức CloseMainWindow hay phương thức Kill().

Phương thức CloseMainWindow gửi một thông điệp đến cửa sổ chính của ứng dụng. CloseMainWindow sẽ không kết thúc các ứng dụng không có cửa sổ chính hoặc các ứng dụng có cửa sổ chính bị vô hiệu. Với những tình huống như thế, CloseMainWindow sẽ trả về false. CloseMainWindow trả về true nếu thông điệp được gửi thành công, nhưng không bảo đảm tiến trình thật sự kết thúc.

Phương thức Kill() kết thúc một tiến trình ngay lập tức; người dùng không có cơ hội dừng việc kết thúc, và tất cả các dữ liệu chưa được lưu sẽ bị mất.

Ví dụ sau khởi chạy một thể hiện mới của Notepad, đợi 5 giây, sau đó kết thúc tiến trình Notepad.

Trước tiên, ví dụ này kết thúc tiến trình bằng CloseMainWindow. Nếu CloseMainWindow trả về false, hoặc tiến trình Notepad vẫn cứ chạy sau khi CloseMainWindow được gọi, ví dụ này sẽ gọi Kill() và buộc tiến trình Notepad kết thúc. Có thể buộc CloseMainWindow trả về false bằng cách bỏ mặc hộp thoại File Open mở.

Chương trình TerminateProcessExample

```
using System;
using System.Threading;
using System.Diagnostics;
public class TerminateProcessExample
{
    public static void Main()
    {
        // Tạo một Process mới và chạy notepad.exe.
        using (Process process = Process.Start("notepad.exe"))
        {
            // Đợi 5 giây và kết thúc tiến trình Notepad.
            Thread.Sleep(5000);
```

```
            // Kết thúc tiến trình Notepad.
            // Gửi một thông điệp đến cửa sổ chính.
            if (!process.CloseMainWindow())
            {
                // Không gửi được thông điệp. Kết thúc Notepad bằng Kill.

                process.Kill();
            }
            else
            {
                // Thông điệp được gửi thành công; đợi 2 giây
                // để chứng thực việc kết thúc trước khi viện đến Kill.
                if (!process.WaitForExit(2000))
                {
                    process.Kill();

                }

                // Nhấn Enter để kết thúc.
                Console.WriteLine("Main method complete. Press Enter.");
                Console.ReadLine();
            }


        }
    }
}
```

V.4. Thực thi phương thức bằng cách ra hiệu đối tượng WaitHandle

Sử dụng các lớp dẫn xuất từ WaitHandle để gọi thực thi một phương thức. Bằng phương thức RegisterWaitForSingleObject của lớp ThreadPool, có thể đăng ký thể hiện ủy nhiệm WaitOrTimerCallback với thread-pool khi một đối tượng dẫn xuất từ WaitHandle đi vào trạng thái signaled.

Có thể cấu hình thread-pool để thực thi phương thức chỉ một lần hay tự động đăng ký lại phương thức mỗi khi WaitHandle đi vào trạng thái signaled. Nếu WaitHandle đã ở trạng thái signaled khi gọi RegisterWaitForSingleObject, phương thức sẽ thực thi ngay lập tức. Phương thức Unregister của đối tượng System.Threading.RegisteredWaitHandle (được trả về bởi phương thức RegisterWaitForSingleObject) được sử dụng để hủy bỏ việc đăng ký.

Lớp thường được dùng làm bộ kích hoạt là AutoResetEvent, nó sẽ tự động chuyển sang trạng thái unsignaled sau khi ở trạng thái signaled. Tuy nhiên, cũng có thể thay đổi trạng thái signaled theo ý muốn bằng lớp ManualResetEvent hay Mutex.

CHƯƠNG VI: ĐỒNG BỘ HÓA

VI.1. Lý do đồng bộ hóa

Trong các hệ điều hành đa nhiệm cho phép nhiều công việc được thực hiện đồng thời. Việc tồn tại cùng lúc nhiều tiểu trình trong môi trường có thể dẫn đến sự tranh chấp, ngăn cản hoạt động lẫn nhau giữa các tiểu trình.

Ví dụ : Với một tiến trình mới đầu tạo lập 4 tiểu trình cùng có nội dung xử lý đồng nhất.: Mỗi tiểu trình sẽ tăng giá trị của biến toàncục Count lên 250.000 lần . Do vậy Count sẽ tăng lên 1.000.000 lần trong tòan bộ tiến trình.

Để hạn chế các trình trạng tranh chấp tài nguyên cần có cơ chế điều khiển các tiểu trình truy xuất tài nguyên một cách tuần tự . đó chính là thực hiện đồng bộ hóa các tiểu trình.

Các trường hợp cần thực hiện đồng bộ hóa.

Khi một tiểu trình truy xuất đến một tài nguyên dùng chung , cần chú ý thực hiện đồng bộ hóa việc truy xuất tài nguyên để tránh xảy ra tranh chấp. Các cơ chế đồng bộ hóa đều dựa trên ý tưởng chỉ cho phép một tiểu trình được truy xuất tài nguyên khi thỏa điều kiện không có tranh chấp trên đó. Những tiểu trình không hội đủ điều kiện để sử dụng tài nguyên thì được hệ điều hành đặt vào trạng thái chờ (Không chiếm CPU), và hệ điều hành sẽ luôn kiểm sóat tình trạng truy xuất tài nguyên của tiểu trình khác để có thể giải phóng kịp thời các tiểu trình đang chờ vào thời điểm thích hợp.

VI.2. Các phương pháp đồng bộ hóa

Để thực hiện được cơ chế đồng bộ hóa , hệ điều hành sử dụng các đối tượng đồng bộ hóa gắn liền với các tài nguyên để phản ánh tình trạng truy xuất trên các tài nguyên.

Các đối tượng đồng bộ hóa có thể nhận trạng thái TRUE hoặc FALSE. Khi đối tượng đồng bộ hóa ở trạng thái TRUE tiểu trình được phép truy cập tài nguyên, ngược lại thì không.

VI.3. Phương pháp Semaphore

Khái niệm: Semaphore là một đối tượng đồng bộ hóa lưu trữ một biến đếm có giá trị từ 0 đến Max, semaphore nhận trạng thái TRUE khi giá trị của biến đếm > 0 và nhận trạng thái FALSE nếu có giá trị biến đếm =0

Tình huống sử dụng: Sử dụng semaphore để kiểm soát việc cho phép một số hữu hạn tiểu trình cùng lúc truy xuất một tài nguyên dùng chung.

Biến đếm của đối tượng semaphore sẽ cho biết số tiểu trình đang truy xuất tài nguyên mà semaphore bảo vệ, biến đếm sẽ giảm 1 nếu có thêm một tiểu trình truy xuất tài nguyên, ngược lại sẽ tăng giá trị lên 1 nếu có một tiểu trình chấm dứt truy xuất tài nguyên. Khi biến đếm đạt giá trị 0 tài nguyên được bảo vệ, không tiểu trình nào ngòai Max tiểu trình đã đã đăng ký được truy xuất. Có thể dùng semaphore để đồng bộ hóa các tiểu trình trong cùng hoặc khác tiến trình.

Cách tạo đối tượng Semaphore

Class Semaphore

Semaphore(int InitCount, int MaxCount)

Đăng ký truy cập tài nguyên chung

Phương thức WaitOne( )

Kết thúc truy cập tài nguyên chung

Phương thức Release( )

Ví dụ: Tạo ra 10 tiểu trình nhưng tại một thời điểm chỉ có 3 tiểu trình truy cập tài nguyên chung:

```
class SemaphoreTest
{
    static Semaphore s = new Semaphore(3, 3);
    // Available=3; Capacity=3
    static void Main()
    {
        for (int i = 0; i < 10; i++)
        {
            Thead thread = new Thread(new ThreadStart(Go));
            thread.Start();
        }
    }
```

```
    static void Go()
    {
        while (true)
        {
            s.WaitOne();
            Thread.Sleep(100);
            // Only 3 threads can get here at once
            s.Release();
        }
    }
}
```

VI.4. Phương pháp dùng lớp Monitor

Một cơ chế đồng bộ hóa khác là lớp Monitor. Lớp này cho phép một tiểu trình đơn thu lấy khóa (lock) trên một đối tượng bằng cách gọi phương thức tĩnh Monitor.Enter. Bằng cách thu lấy khóa trước khi truy xuất một tài nguyên hay dữ liệu dùng chung, ta chắc chắn rằng chỉ có một tiểu trình có thể truy xuất tài nguyên đó cùng lúc. Một khi đã hoàn tất với tài nguyên, tiểu trình này sẽ giải phóng khóa để tiểu trình khác có thể truy xuất nó bằng phương thức tĩnh Monitor.Exit.

Khối mã được gói trong lệnh lock tương đương với gọi Monitor.Enter khi đi vào khối mã này, và gọi Monitor.Exit khi đi ra khối mã này. Tiểu trình chủ có thể gọi Monitor.Wait để giải phóng lock và đặt tiểu trình này vào hàng chờ (wait queue).

Các tiểu trình trong hàng chờ cũng có trạng thái là WaitSleepJoin và sẽ tiếp tục block cho đến khi tiểu trình chủ gọi phương thức Pulse hay PulseAll của lớp Monitor. Phương thức Pulse di chuyển một trong các tiểu trình từ hàng chờ vào hàng sẵn sàng, còn phương thức PulseAll thì di chuyển tất cả các tiểu trình. Khi một tiểu trình đã được di chuyển từ hàng chờ vào hàng sẵn sàng, nó có thể thu lấy lock trong lần giải phóng kế tiếp.

Lớp ThreadSyncExample trình bày cách sử dụng lớp Monitor và lệnh lock.

Ví dụ này khởi chạy ba tiểu trình, mỗi tiểu trình (lần lượt) thu lấy lock của một đối tượng có tên là consoleGate. Kế đó, mỗi tiểu trình gọi phương thức Monitor.Wait. Khi người dùng nhấn Enter lần đầu tiên, Monitor.Pulse sẽ được gọi để giải phóng một tiểu trình đang chờ. Lần thứ hai người dùng nhấn Enter, Monitor.PulseAll sẽ được gọi để giải phóng tất cả các tiểu trình đang chờ còn lại.

Chương trình ThreadSyncExample

```
using System;
using System.Threading;
public class ThreadSyncExample
{
    private static object consoleGate = new Object();
    private static void DisplayMessage()
    {
        Console.WriteLine("{0} : Thread started, acquiring lock...",
DateTime.Now.ToString("HH:mm:ss.ffff"));
        // Thu lấy chốt trên đối tượng consoleGate.
        try
        {
            Monitor.Enter(consoleGate);
            Console.WriteLine("{0} : {1}",
            DateTime.Now.ToString("HH:mm:ss.ffff"), "Acquired consoleGate lock,
waiting...");
            // Đợi cho đến khi Pulse được gọi trên đối tượng consoleGate.
            Monitor.Wait(consoleGate);
            Console.WriteLine("{0} : Thread pulsed, terminating.",
DateTime.Now.ToString("HH:mm:ss.ffff"));
        }
        finally
        {
            Monitor.Exit(consoleGate);
        }
    }
    public static void Main()
    {
        // Thu lấy chốt trên đối tượng consoleGate.
        lock (consoleGate)
        {
            // Tạo và khởi chạy ba tiểu trình mới
            // (chạy phương thức DisplayMesssage).
            for (int count = 0; count < 3; count++)
            {
                (new Thread(new ThreadStart(DisplayMessage))).Start();
            }
        }
        Thread.Sleep(1000);
        // Đánh thức một tiểu trình đang chờ.
        Console.WriteLine("{0} : {1}", DateTime.Now.ToString("HH:mm:ss.ffff"),
"Press Enter to pulse one waiting thread.");
```

```
            Console.ReadLine();
            // Thu lấy chốt trên đối tượng consoleGate.
            lock (consoleGate)
            {
                // Pulse một tiểu trình đang chờ.
                Monitor.Pulse(consoleGate);
            }
            // Đánh thức tất cả các tiểu trình đang chờ.
            Console.WriteLine("{0} : {1}", DateTime.Now.ToString("HH:mm:ss.ffff"),
"Press Enter to pulse all waiting threads.");
            Console.ReadLine();
            // Thu lấy chốt trên đối tượng consoleGate.
            lock (consoleGate)
            {
                // Pulse tất cả các tiểu trình đang chờ.
                Monitor.PulseAll(consoleGate);
            }
            // Nhấn Enter để kết thúc.
            Console.WriteLine("Main method complete. Press Enter.");
            Console.ReadLine();
        }
}
```

VI.5. System.Threading.WaitHandle, bao gồm AutoResetEvent, ManualResetEvent

Các lớp thông dụng khác dùng để đồng bộ hóa tiểu trình là các lớp con của lớp System.Threading.WaitHandle, bao gồm AutoResetEvent, ManualResetEvent. Thể hiện của các lớp này có thể ở trạng thái signaled hay unsignaled.

Các tiểu trình có thể sử dụng các phương thức của các lớp được liệt kê để đi vào trạng thái WaitSleepJoin và đợi trạng thái của một hay nhiều đối tượng dẫn xuất từ WaitHandle biến thành signaled.

Phương Thức	**Mô Tả**
WaitAny()	Tiểu trình gọi phương thức tĩnh này sẽ đi vào trạng thái WaitSleepJoin và đợi bất kỳ một trong các đối tượng WaitHandle thuộc một mảng WaitHandle biến thành *signaled*. Cũng có thể chỉ định giá trị time-out.
WaitAll()	Tiểu trình gọi phương thức tĩnh này sẽ đi vào trạng thái

Phương Thức	**Mô Tả**
	WaitSleepJoin và đợi tất cả các đối tượng WaitHandle trong một mảng WaitHandle biến thành *signaled*. Bạn cũng có thể chỉ định giá trị time-out.
WaitOne()	Tiểu trình gọi phương thức này sẽ đi vào trạng thái WaitSleepJoin và đợi một đối tượng WaitHandle cụ thể biến thành *signaled*.

Điểm khác biệt chính giữa các lớp AutoResetEvent, ManualResetEvent, là cách thức chúng chuyển trạng thái từ signaled thành unsignaled.

Lớp AutoResetEvent và ManualResetEvent là cục bộ đối với một tiến trình. Để ra hiệu một AutoResetEvent, bạn hãy gọi phương thức Set của nó, phương thức này chỉ giải phóng một tiểu trình đang đợi sự kiện. AutoResetEvent sẽ tự động trở về trạng thái unsignaled.

Lớp ManualResetEvent phải được chuyển đổi qua lại giữa signaled và unsignaled bằng phương thức Set và Reset của nó.
Gọi Set trên một ManualResetEvent sẽ đặt trạng thái của nó là signaled, giải phóng tất cả các tiểu trình đang đợi sự kiện. Chỉ khi gọi Reset mới làm cho ManualResetEvent trở thành unsignaled.

Sử dụng các lớp dẫn xuất từ WaitHandle để gọi thực thi một phương thức. Bằng phương thức RegisterWaitForSingleObject của lớp ThreadPool, có thể đăng ký thể hiện ủy nhiệm WaitOrTimerCallback với thread-pool khi một đối tượng dẫn xuất từ WaitHandle đi vào trạng thái signaled.

Có thể cấu hình thread-pool để thực thi phương thức chỉ một lần hay tự động đăng ký lại phương thức mỗi khi WaitHandle đi vào trạng thái signaled.
Nếu WaitHandle đã ở trạng thái signaled khi gọi RegisterWaitForSingleObject, phương thức sẽ thực thi ngay lập tức.

Phương thức Unregister của đối tượng System.Threading.RegisteredWaitHandle (được trả về bởi phương thức RegisterWaitForSingleObject) được sử dụng để hủy bỏ việc đăng ký.

Lớp thường được dùng làm bộ kích hoạt là AutoResetEvent, nó sẽ tự động chuyển sang trạng thái unsignaled sau khi ở trạng thái signaled. Tuy nhiên, chúng ta cũng có thể thay đổi trạng thái signaled theo ý muốn bằng lớp ManualResetEvent hay Mutex.

Ví dụ dưới đây trình bày cách sử dụng một AutoResetEvent để kích hoạt thực thi một phương thức có tên là EventHandler:

Chương trình EventExecutionExample

```
using System.Threading;
public class EventExecutionExample
{
    // Phương thức sẽ được thực thi khi AutoResetEvent đi vào trạng
    // thái signaled hoặc quá trình đợi hết thời gian (time-out).
    private static void EventHandler(object state, bool timedout)
    {
        // Hiển thị thông báo thích hợp ra cửa sổ Console
        // tùy vào quá trình đợi đã hết thời gian hay
        // AutoResetEvent đã ở trạng thái signaled.
        if (timedout)
        {
            Console.WriteLine("{0} : Wait timed out.",
                DateTime.Now.ToString("HH:mm:ss.ffff"));
        }
        else
        {
            Console.WriteLine("{0} : {1}",
            DateTime.Now.ToString("HH:mm:ss.ffff"), state);
        }
    }
    public static void Main()
    {
        // Tạo một AutoResetEvent ở trạng thái unsignaled.
        AutoResetEvent[] autoEvent;
        autoEvent[0] = new AutoResetEvent(false);

        // Tạo một thể hiện ủy nhiệm WaitOrTimerCallback
        // tham chiếu đến phương thức tĩnh EventHandler.
        // EventHandler sẽ được gọi khi AutoResetEvent đi vào
        // trạng thái signaled hay quá trình đợi hết thời gian.
        WaitOrTimerCallback handler = new WaitOrTimerCallback(EventHandler);
        // Tạo đối tượng trạng thái (được truyền cho phương thức
```

```
        // thụ lý sự kiện khi nó được kích hoạt). Trong trường hợp
        // này, một thông báo sẽ được hiển thị.
        string state = "AutoResetEvent signaled.";
        // Đăng ký thể hiện ủy nhiệm để đợi AutoResetEvent đi vào
        // trạng thái signaled. Thiết lập giá trị time-out là 3 giây.
        RegisteredWaitHandle handle =
ThreadPool.RegisterWaitForSingleObject(autoEvent, handler, state, 3000, false);
        Console.WriteLine("Press ENTER to signal the AutoResetEvent or enter
\"Cancel\" to unregister the wait operation.");
        while (Console.ReadLine().ToUpper() != "CANCEL")
        {
            // Nếu "Cancel" không được nhập vào Console,
            // AutoResetEvent sẽ đi vào trạng thái signal,
            // và phương thức EventHandler được thực thi.
            // AutoResetEvent sẽ tự động trở về trạng thái unsignaled.
            autoEvent.Set();
        }
        // Hủy bỏ việc đăng ký quá trình đợi.
        Console.WriteLine("Unregistering wait operation.");
        handle.Unregister(null);
        // Nhấn Enter để kết thúc.
        Console.WriteLine("Main method complete. Press Enter.");
        Console.ReadLine();
    }
}
```

VI.6. Phương pháp Mutex

Mutex cung cấp một cơ chế để đồng bộ hóa quá trình thực thi của các tiểu trình vượt qua biên tiến trình. Một Mutex là signaled khi nó không thuộc sở hữu của bất kỳ tiểu trình nào. Một tiểu trình giành quyền sở hữu Mutex lúc khởi dựng hoặc sử dụng một trong các phương thức được liệt kê ở trên.

Quyền sở hữu Mutex được giải phóng bằng cách gọi phương thức Mutex.ReleaseMutex (ra hiệu Mutex và cho phép một tiểu trình khác thu lấy quyền sở hữu này).

Ví dụ dưới đây sử dụng một Mutex có tên là MutexExample để bảo đảm chỉ một thể hiện của ví dụ có thể thực thi.

Chương trình MutexExample

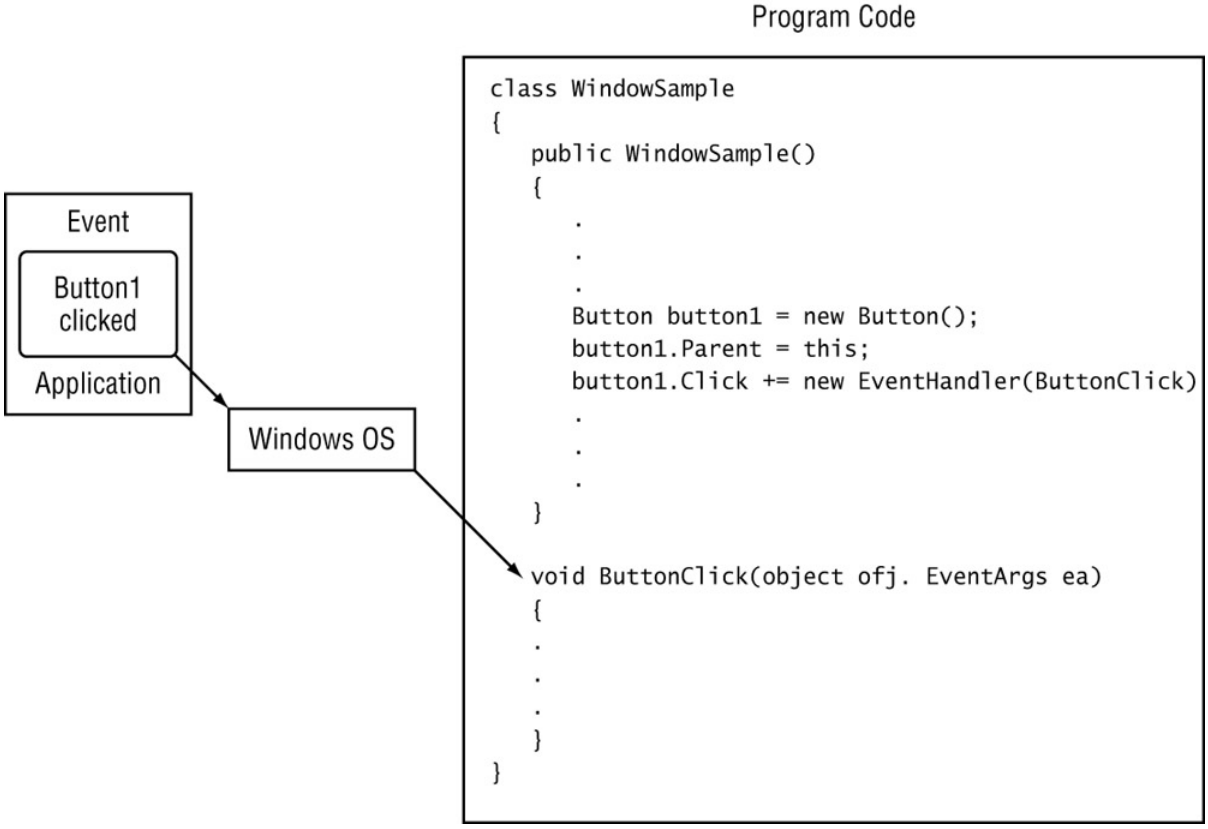
```
using System;
using System.Threading;
public class MutexExample
```

```
{
    public static void Main()
    {
        // Giá trị luận lý cho biết ứng dụng này
        // có quyền sở hữu Mutex hay không.
        bool ownsMutex;
        // Tạo và lấy quyền sở hữu một Mutex có tên là MutexExample.
        using (Mutex mutex = new Mutex(true, "MutexExample", out ownsMutex))
        {
            // Nếu ứng dụng sở hữu Mutex, nó có thể tiếp tục thực thi;
            // nếu không, ứng dụng sẽ thoát.
            if (ownsMutex)
            {
                Console.WriteLine("This application currently owns the mutex
named MutexExample. Additional instances of this application will not run until
you release the mutex by pressing Enter.");
                Console.ReadLine();
                // Giải phóng
                Mutex.mutex.ReleaseMutex();
            }
            else
            {
                Console.WriteLine("Another instance of this" + " application
already owns the mutex named" + " MutexExample. This instance of the" + "
application will terminate.");
            }
        }
        // Nhấn Enter để kết thúc.
        Console.WriteLine("Main method complete. Press Enter.");
        Console.ReadLine();
    }
}
```

CHƯƠNG VII: LẬP TRÌNH SOCKET BẤT ĐỒNG BỘ

VII.1. Lập trình sự kiện trong Windows

Trong lập trình sự kiện trong Windows, mỗi khi một sự kiện xảy ra, một phương thức được gọi để thực thi dựa trên sự kiện đó như trong hình dưới đây:



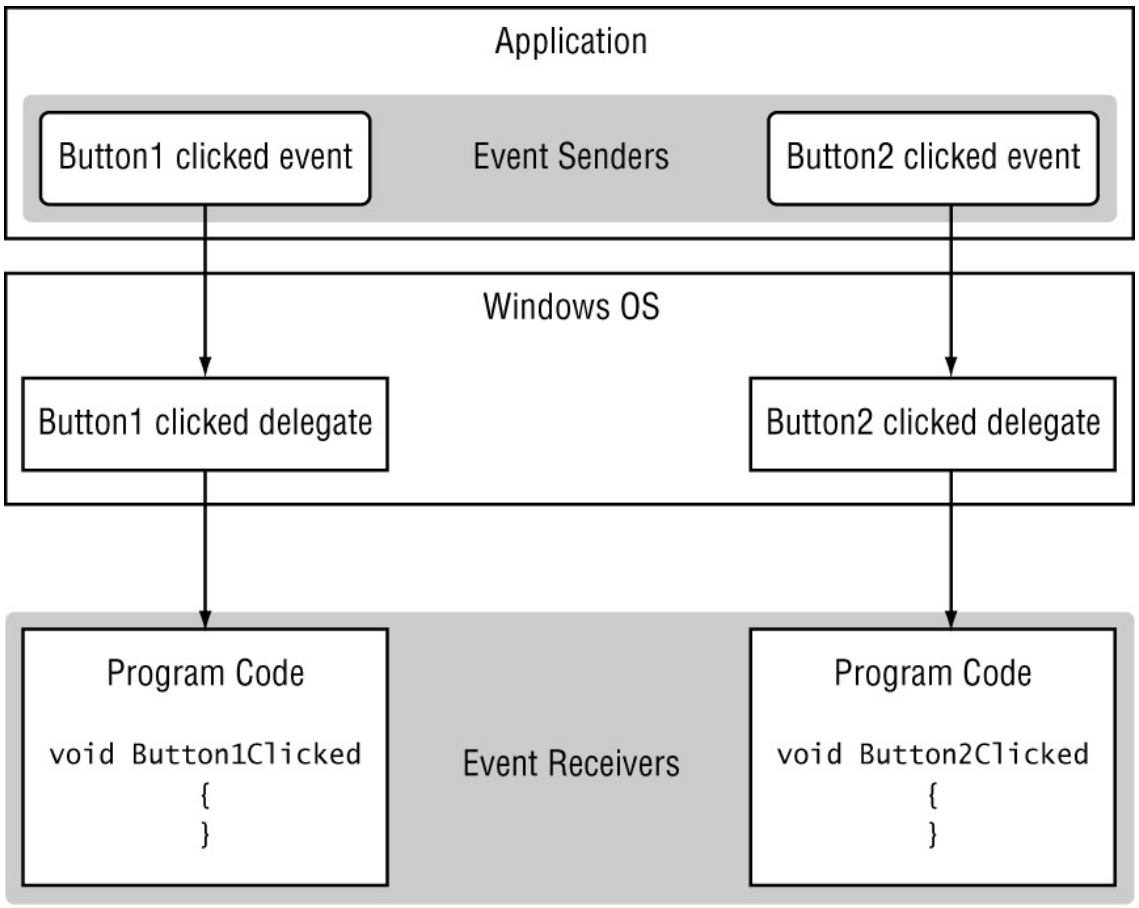
Hình VI.1: Lập trình sự kiện trên Windows

Trong các chương trước, chúng ta đã lập trình Socket trong chế độ blocking. Socket ở chế độ blocking sẽ chờ mãi mãi cho đến khi hoàn thành nhiệm vụ của nó. Trong khi nó bị blocking thì các chức năng khác của chương trình không thực hiện được.

Khi lập trình Windows thì lúc gọi một phương thức bị blocking thì toàn bộ chương trình sẽ đứng lại và không thực hiện các chức năng khác được. Do đó việc lập trình bất đồng bộ là cần thiết để cho chương trình khỏi bị đứnng.

VII.1.1. Sử dụng Event và Delegate

Event là một thông điệp được gởi bởi một đối tượng mô tả một hoạt động mà nó diễn ra. Thông điệp này xác định hoạt động và truyền các dữ liệu cần thiết cho hoạt động. Event có thể là mô tả hoạt động nào đó, chẳng hạn như hoạt động click một Button, hoạt động nhận và gởi dữ liệu trên Socket. Event sender không cần thiết phải biết đối tượng nào sẽ điều khiển thông điệp sự kiện mỗi khi nó được gởi thông qua hệ thống Windows. Nó để cho bộ nhận sự kiện đăng ký với hệ thống Windows và thông báo kiểu sự kiện mà bộ nhận sự kiện muốn nhận như hình minh họa sau:



Hình VI.2: Gởi và nhận sự kiện trong Windows

Bộ nhận sự kiện được xác định trong hệ thống Windows bởi một con trỏ lớp được gọi là delegate. Một delegate là một lớp nó giữ tham chiếu đến một phương thức mà phương thức này điều khiển sự kiện được nhận. Khi hệ thống Windows nhận sự kiện. nó kiểm tra coi thử có delegate nào đăng ký để xử lý nó không. Nếu có delegate đăng ký để xử lý sự kiện, thông điệp sự kiện được truyền vào phương thức được định

nghĩa bởi delegate. Sau khi phương thức hoàn tất, hệ thống Windows sẽ xử lý sự kiện tiếp theo xảy ra cho tới khi sự kiện kết thúc chương trình được phát ra.

Ví dụ đơn giản sau mô tả cách lập trình sự kiện trên Windows Forrm

Chương trình WindowSample

```
using System;
using System.Drawing;
using System.Windows.Forms;
class WindowSample : Form
{
    private TextBox data;
    private ListBox results;
    public WindowSample()
    {
        Text = "Sample Window Program";
        Size = new Size(400, 380);
        Label label1 = new Label();
        label1.Parent = this;
        label1.Text = "Enter text string:";
        label1.AutoSize = true;
        label1.Location = new Point(10, 10);
        data = new TextBox();
        data.Parent = this;
        data.Size = new Size(200, 2 * Font.Height);
        data.Location = new Point(10, 35);
        results = new ListBox();
        results.Parent = this;
        results.Location = new Point(10, 65);
        results.Size = new Size(350, 20 * Font.Height);
        Button checkit = new Button();
        checkit.Parent = this;
        checkit.Text = "test";
        checkit.Location = new Point(235, 32);
        checkit.Size = new Size(7 * Font.Height, 2 * Font.Height);
        checkit.Click += new EventHandler(checkit_OnClick);
    }
    void checkit_OnClick(object obj, EventArgs ea)
    {
        results.Items.Add(data.Text);
        data.Clear();
    }
    public static void Main()
    {
```

```
        Application.Run(new WindowSample());
    }
}
```

Điểm chính trong chương trình này là EvenHandler đăng ký phương thức ButtonOnClick() cho đối tượng Button với sự kiện click:

```
checkit.Click += new EventHandler(checkit_OnClick);
```

Khi người dùng click button, điều khiển chương trình sẽ được chyển đến phương thức ButtonOnClick()

```
void checkit_OnClick(object obj, EventArgs ea)
    {
        results.Items.Add(data.Text);
        data.Clear();
    }
```

VII.1.2. Lớp AsyncCallback trong lập trình Windows

Khi sự kiện kích hoạt delegate, .NET cung cấp một cơ chế để kích hoạt delegate. Lớp AsyncCallback cung cấp các phương thức để bắt đầu một chức năng bất đồng bộ và cung cấp một phương thức delegate để gọi khi chức năng bất đồng bộ kết thúc.

Tiến trình này khác với cách lập trình sự kiện cơ bản, sự kiện này không phát sinh ra từ đối tượng Windows mà nó xuất phát từ một phương thức khác trong chương trình. Phương thức này chính nó đăng ký một delegate AsyncCallback để gọi khi phương thức hoàn tất chức năng của nó.

Socket sử dụng phương thức được định nghĩa trong lớp AsyncCallback để cho phép các chức năng mạng hoạt động một cách bất đồng bộ. Nó phát tín hiệu cho hệ điều hành khi chức năng mạng hoàn tất. Trong môi trường lập trình Windows, những phương thức này giúp cho ứng dụng khỏi bị treo trong khi chờ các chức năng mạng hoàn tất.

VII.2. Sử dụng Socket bất đồng bộ

Đối tượng Socket có nhiều phương thức sử dụng lớp AsyncCallback để gọi các phương thức khác khi các chức năng mạng hoàn tất. Nó cho phép dùng tiếp tục xử lý các sự kiện khác trong khi chờ cho các chức năng mạng hòa thành công việc của nó.

Các phương thức bất đồng bộ của Socket chia các chức năng mạng làm hai phần:

- ✓ Một phương thức Begin bắt đầu các chức năng mạng và đăng ký phương thức AsyncCallback.
- ✓ Một phương thức End hoàn thành chức năng mạng khi phương thức AsyncCallback được gọi.

Bảng sau cho biết các phương thức bất đồng bộ có thể được giữa với Socket. Mỗi phương thức Begin được kết hợp với một phương thức End để hoàn tất chức năng của chúng:

Phương thức bắt đầu	**Mô tả**	**Phương thức kết thúc**
BeginAccept()	Chấp nhận kết nối	EndAccept()
BeginConnect()	Kết nối đến thiết bị ở xa	EndConnect()
BeginReceive()	Nhận dữ liệu từ Socket	EndReceive()
BeginReceiveFrom()	Nhận dữ liệu từ thiết bị ở xa	EndReceiveFrom()
BeginSend()	Gởi dữ liệu từ Socket	EndSend()
BeginSendTo()	Gởi dữ liệu đến thiết bị ở xa	EndSendTo

VII.2.1. Thành lập kết nối

Phương thức được dùng để thành lập kết nối với thiết bị ở xa phụ thuộc vào chương trình đóng vai trò là Server hay Client. Nếu là Server thì phương thức BeginAccept() được dùng, nếu là Client thì phương thức BeginConnect() được dùng.

VII.2.1.1. Phương thức BeginAccept() và EndAccept()

Để chấp nhận kết nối từ thiết bị ở xa, phương thức BeginAccetp() được dùng. Định dạng của nó như sau:

```
IAsyncResult BeginAccept(AsyncCallback callback, object state)
```

Phương thức BeginAccept() nhận hai đối số: tên của phương thức AsyncCallback dùng để kết thúc phương thức và một đối tượng state có thể được dùng để truyền thông tin giữa các phương thức bất đồng bộ. Phương thức này được dùng như sau:

```
Socket sock = new Socket(AddressFamily.InterNetwork,
SocketType.Stream,
 ProtocolType.Tcp);
IPEndPoint iep = new IPEndPoint(IPAddress.Any, 9050);
sock.Bind(iep);
```

```
sock.Listen(5);
sock.BeginAccept(new AsyncCallback(CallAccept), sock);
```

Đoạn code ở trên tạo ra một đối tượng Socket và gán nó đến một địa chỉ IP cục bộ và một port TCP cục bộ để chấp nhận kết nối. Phương thức BeginAccept() định nghĩa delegate được dùng khi có kết nối trên Socket. Tham số cuối cùng được truyền vào phương thức BeginAccept() là Socket ban đầu được tạo ra.

Sau khi phương thức BeginAccept() kết thúc, phương thức AsyncCallback định nghĩa sẽ được gọi khi kết nối xảy ra. Phương thức AsyncCallback phải bao gồm phương thức EndAccept() để kết thúc việc chấp nhận Socket. Sau đây là định dạng của phương thức EndAccept():

```
Socket EndAccept(IAsyncResult iar);
```

Đối tượng IasyncResult truyền giá trị bất đồng bộ từ phương thức BeginAccept() đến phương thức EndAccept(). Cũng giống như phương thức đồng bộ Accept(), phương thức EndAccept() trả về một đối tượng Socket được dùng để kết nối với Client. Tất cả các thông tin liên lạc vói thiết bị ở xa đều được thực hiện thông qua đối tượng Socket này.

```
private static void CallAccept(IAsyncResult iar)
{
  Socket server = (Socket)iar.AsyncState;
  Socket client = server.EndAccept(iar);
  .
  .
  .
}
```

Tên của phương thức AsyncCallback phải giống với tên của phương thức được dùng làm tham số của phương thức BeginAccept(). Bước đầu tiên trong phương thức là nhận Socket ban đầu của Server. Việc này được thực hiện bằng cách dùng property AsyncState của lớp IasyncResult. Property này có kiểu là object do đó nó phải được ép kiểu sang một đối tượng Socket.

Sau khi Socket ban đầu được lấy về, phương thức EndAccept() có thể lấy được đối tượng Socket mới để truyền thông với thiết bị ở xa. Đối số truyền vào của của phương thức EndAccept() phải giống với đối số truyền vào phương thức AsyncCallback.

Đối tượng Client Socket sau khi được tạo ra, nó có thể được dùng giống như bất kỳ đối tượng Socket nào khác, nó có thể được sử dụng các phương thức đồng bộ hoặc bất đồng bộ để gởi và nhận dữ liệu.

VII.2.1.2. Phương thức BeginConnect() và EndConnect()

Để ứng dụng Client kết nối đến Server ở xa bằng phương thức bất đồng bộ ta phải dùng phương thức BeginConnect(). Định dạng của phương thức này nhu sau:

```
IAsyncResult BeginConnect(EndPoint ep, AsyncCallback callback,
Object state)
```

Tham số truyền vào phương thức BeginConnect() là một EndPoint của thiết bị ở xa cần kết nối đến. Giống như phương thức BeginAccept(), phương thức BeginConnect() cũng chỉ ra tên phương thức do delegate AsyncCallback tham chiếu đến và phương thức này được gọi khi kết nối hoàn tất. Tham số cuối cùng là một đối tượng state, nó có thể được truyền vào phương thức EndConnect() để truyền tải các dữ liệu cần thiết.

Đây là đoạn code ví dụ của phương thức BeginConnect():

```
Socket newsock = new Socket(AddressFamily.InterNetwork,
SocketType.Stream,
ProtocolType.Tcp);
IPEndPoint iep = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 9050);
newsock.BeginConnect(iep, new AsyncCallback(Connected), newsock);
```

Đoạn code này tạo ra một đối tượng Socket newsock và một đối tượng IPEndPoint iep cho thiết bị ở xa. Phương thức BeginConnect() tham chiếu đến phương thức AsyncCallback (Connect) và truyền đối tượng Socket ban đầu newsock đến phương thức AsyncCallback.

Khi kết nối được thành lập phương thức do delegate AsyncCallback tham chiếu tới được gọi. Phương thức này dùng phương thức EndConnect() để hoàn thành việc kết nối. Định dạng của phương thức EndConnect() như sau:

```
EndConnect(IAsyncResult iar)
```

Đối tượng IasyncResult truyền giá trị từ phương thức BeginConnect(). Sau đây là ví dụ cách dùng phương thức này:

```
public static void Connected(IAsyncResult iar)
{
```

```
  Socket sock = (Socket)iar.AsyncState;
  try
  {
   sock.EndConnect(iar);
  } catch (SocketException)
  {
   Console.WriteLine("Unable to connect to host");
  }
}
```

Đầu tiên ta lấy Socket ban đầu được sử dụng bởi phương thức BeginConnect(), Socket này lấy được là nhờ vào thuộc tính object của đối tượng IAsyncResult được truyền vào trong phương thức tham chiếu bởi delegate AsyncCallback.

Sau khi Socket ban đầu được tạo ra, phương thức EndConnect() được gọi, đối số truyền vào phương thức EndConnect() là một đối tượng IAsyncResult chỏ ngược trở lại phương thức BeginConnect() ban đầu. Bởi vì thiết bị ở xa có thể kết nối được và cũng có thể không nên tốt nhất là đặt nó vào trong khối try – catch.

VII.2.2. Gởi dữ liệu

VII.2.2.1. Phương thức BeginSend() và phương thức EndSend()

Phương thức BeginSend() cho phép gởi dữ liệu đến một Socket đã được kết nối. Định dạng của phương thức này như sau:

```
IAsyncResult BeginSend(byte[] buffer, int offset, int size,
 SocketFlags sockflag, AsyncCallback callback, object state)
```

Hầu hết các đối số của phương thức này giống như các đối số của phương thức đồng bộ Send() chỉ có thêm hai đối số là AsyncCallback và object.

- ✓ Đối số AsyncCallback: xác định phương thức được gọi khi phương thức BeginSend() thực hiện thành công.
- ✓ Đối số object: gởi thông tin tình trạng đến phương thức EndSend()

Sau đây là một ví dụ của phương thức BeginSend():

```
sock.BeginSend(data, 0, data.Length, SocketFlags.None,
 new AsyncCallback(SendData), sock);
```

Trong ví dụ này toàn bộ dữ liệu trong bộ đệm data được gởi đi và phương thức SendData được gọi khi Socket sẵng sàng gởi dữ liệu. Đối tượng Socket sock sẽ được truyền đến phương thức do delegate AsyncCallback tham chiếu tới.

Phương thức EndSend() sẽ hoàn tất việc gởi dữ liệu. Định dạng của phương thức này như sau:

```
int EndSend(IAsyncResult iar)
```

Phương thức EndSend() trả về số byte được gởi thành công thừ Socket. Sau đây là một ví dụ của phương thức EndSend():

```
private static void SendData(IAsyncResult iar)
{
  Socket server = (Socket)iar.AsyncState;
  int sent = server.EndSend(iar);
}
```

Socket ban đầu được tạo lại bằng cách dùng thuộc tính AsyncState của đối tượng IAsyncResult

VII.2.2.2. Phương thức BeginSendTo() và EndSendTo()

Phương thức BeginSendTo() được dùng với Socket phi kết nối để bắt đầu truyền tải dữ liệu bất đồng bộ tói thiết bị ở xa. Định dạng của phương thức BeginSendTo() như sau:

```
IAsyncResult BeginSendTo(byte[] buffer, int offset, int size,
 SocketFlags sockflag, EndPoint ep, AsyncCallback callback, object
state)
```

Các đối số của phương thức BeginSendTo() cũng giống như các đối số của phương thức SendTo().

Sau đây là một ví dụ của phương thức BeginSendTo():

```
     IPEndPoint iep = new IPEndPoint(IPAddress.Parse("192.168.1.6"),
9050);
sock.BeginSendTo(data, 0, data.Length, SocketFlags.None, iep,
new AsynCallback(SendDataTo), sock);
```

Phương thức SendDataTo() do delegate AsyncCallback tham chiếu đến được truyền vào làm đối số của phương thức BeginSendTo(). Phương thức này sẽ được thực thi khi dữ liệu bắt đầu được gởi đi từ Socket. Phương thức EndSendTo() sẽ được thực thi khi quá trình gởi dữ liệu kết thúc. Định dạng của phương thức này như sau:

```
int EndSendTo(IAsyncResult iar)
```

Đối số truyền vào của phương thức EndSendTo() là một đối tượng IAsyncResult, đối tượng này mang giá trị được truyền từ phương thức BeginSendTo(). Khi quá trình gởi dữ liệu bất đồng bộ kết thúc thì phương thức EndSendTo() sẽ trả về số byte mà nó thực sự gởi được.

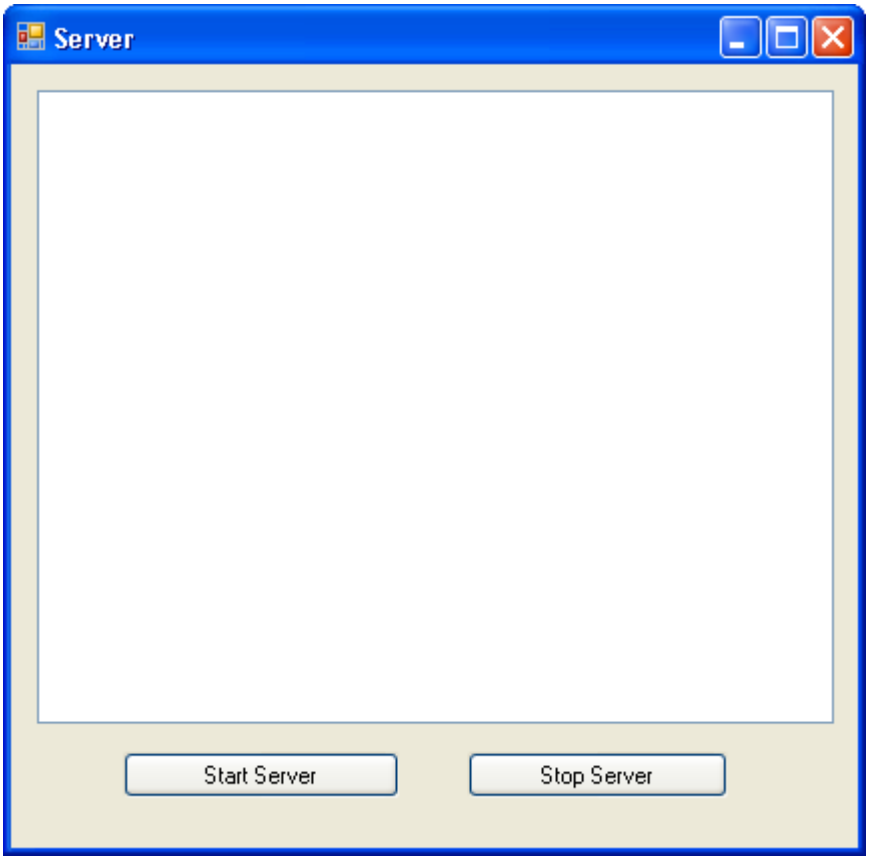
VII.2.3. Nhận dữ liệu

VII.2.3.1. Phương thức BeginReceive(), EndReceive, BeginReceiveFrom(), EndReceiveFrom()

Cách sử dụng của các phương thức này cũng tương tự như cách sử dụng của các phương thức: BeginSend(), EndSend(), BeginSendTo() và EndSendTo()

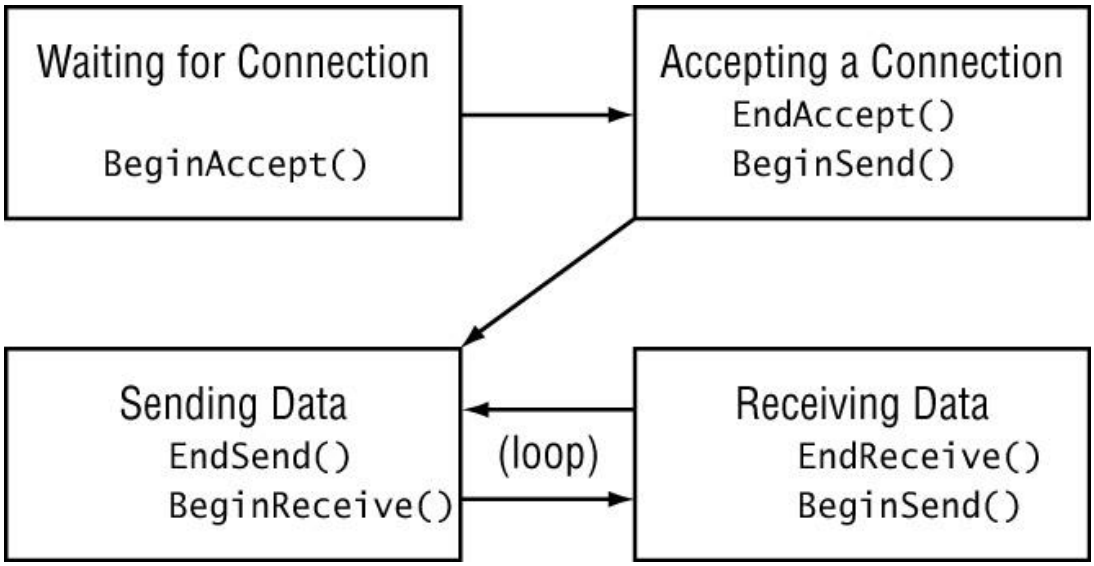
VII.2.4. Chương trình WinForm gởi và nhận dữ liệu giữa Client và Server

VII.2.4.1. Chương trình Server



Hình VI.3: Giao diện Server

VII.2.4.2. Mô hình chương trình Server



Hình VI.4: Mô hình chương trình Server

VII.2.4.3. Lớp ServerProgram

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Net;
using System.Net.Sockets;
namespace Server
{
    class ServerProgram
    {
        private IPAddress serverIP;
        public IPAddress ServerIP
        {
            get
            {
                return serverIP;
            }
            set
            {
                this.serverIP = value;
            }
        }
        private int port;
        public int Port
        {
            get
```

```
            {
                return this.port;
            }
            set
            {
                this.port = value;
            }
        }
        //delegate để set dữ liệu cho các Control
        //Tại thời điểm này ta chưa biết dữ liệu sẽ được hiển thị vào đâu nên
ta phải dùng delegate
        public delegate void SetDataControl(string Data);
        public SetDataControl SetDataFunction = null;
        Socket serverSocket = null ;
        IPEndPoint serverEP = null;
        Socket clientSocket = null;
        //buffer để nhận và gởi dữ liệu
        byte[] buff = new byte[1024];
        //Số byte thực sự nhận được
        int byteReceive = 0;
        string stringReceive = "";

        public ServerProgram(IPAddress ServerIP, int Port)
        {
            this.ServerIP = ServerIP;
            this.Port = Port;
        }
        //Lắng nghe kết nối
        public void Listen()
        {
            serverSocket = new Socket(AddressFamily.InterNetwork,
SocketType.Stream, ProtocolType.Tcp);
            serverEP = new IPEndPoint(ServerIP, Port);
            //Kết hợp Server Socket với Local Endpoint
            serverSocket.Bind(serverEP);
            //Lắng nghe kết nối trên Server Socket
            //-1: không giới hạn số lượng client kết nối đến
            serverSocket.Listen(-1);
            SetDataFunction("Dang cho ket noi");
            //Bắt đầu chấp nhận Client kết nối đến
            serverSocket.BeginAccept(new AsyncCallback(AcceptScoket),
serverSocket);
        }
```

```
        //Hàm callback chấp nhận Client kết nối
        private void AcceptScoket(IAsyncResult ia)
        {
            Socket s = (Socket)ia.AsyncState;
            //Hàm Accept sẽ block server lại và chờ Client kết nối đến
            //Sau khi Client kết nối đến sẽ trả về socket chứa thông tin của
Client
            clientSocket = s.EndAccept(ia);
            string hello = "Hello Client";
            buff = Encoding.ASCII.GetBytes(hello);
            SetDataFunction("Client " + clientSocket.RemoteEndPoint.ToString()
+ "da ket noi den");
            clientSocket.BeginSend(buff, 0, buff.Length, SocketFlags.None, new
AsyncCallback(SendData), clientSocket);
        }
        private void SendData(IAsyncResult ia)
        {
            Socket s = (Socket)ia.AsyncState;
            s.EndSend(ia);
            //khởi tạo lại buffer để nhận dữ liệu
            buff = new byte[1024];
            //Bắt đầu nhận dữ liệu
            s.BeginReceive(buff, 0, buff.Length, SocketFlags.None, new
AsyncCallback(ReceiveData), s);
        }
        public void Close()
        {
            clientSocket.Close();
            serverSocket.Close();
        }

        private void ReceiveData(IAsyncResult ia)
        {
            Socket s = (Socket)ia.AsyncState;
            try
            {
                //Hàm EmdReceive sẽ bị block cho đến khi có dữ liệu trong TCP
buffer
                byteReceive = s.EndReceive(ia);
            }
            catch
            {
                //Trường hợp lỗi xảy ra khi Client ngắt kết nối
```

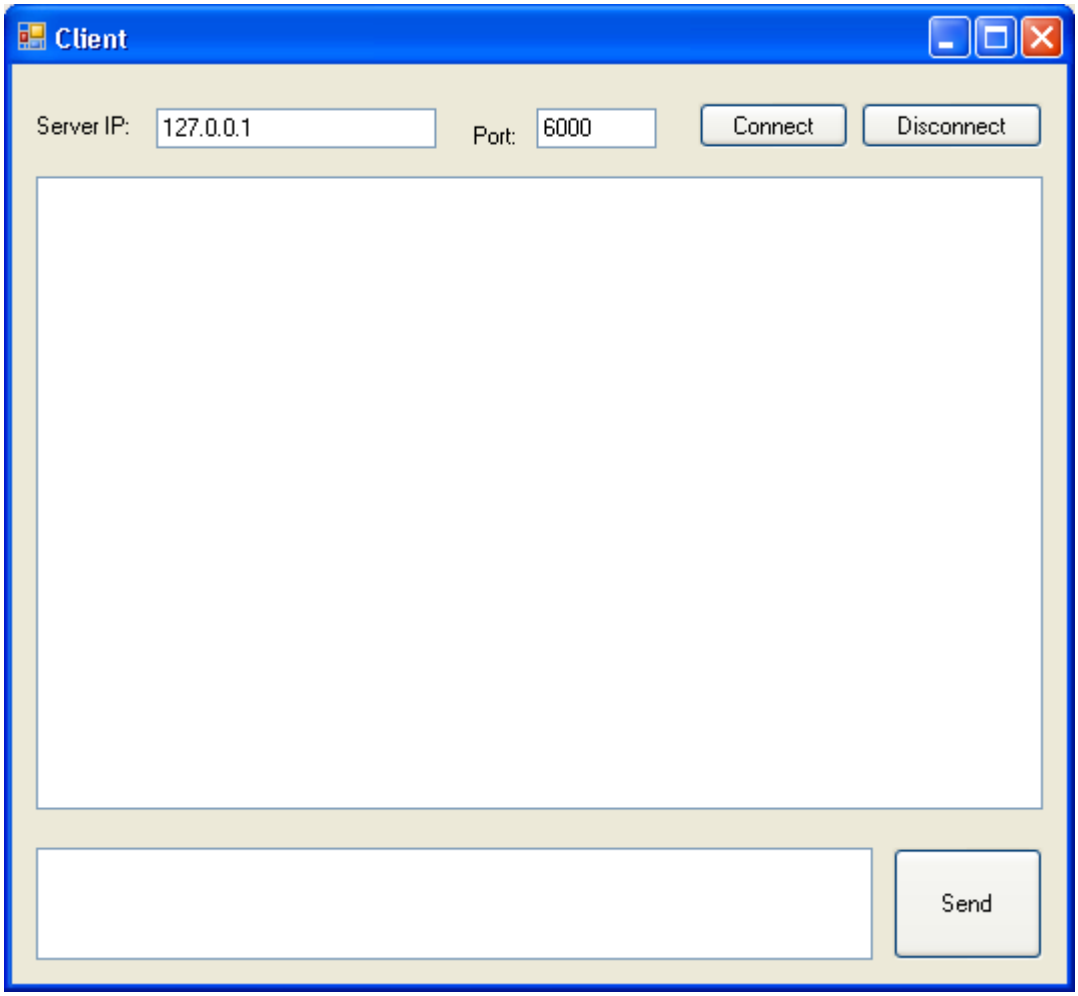
```
                this.Close();
                SetDataFunction("Client ngat ket noi");
                this.Listen();
                return;
            }

            //Nếu Client shutdown thì hàm EndReceive sẽ trả về 0
            if (byteReceive == 0)
            {
                Close();
                SetDataFunction("Clien dong ket noi");
            }
            else
            {
                stringReceive = Encoding.ASCII.GetString(buff, 0, byteReceive);
                SetDataFunction(stringReceive);

                //Sau khi Server nhận dữ liệu xong thì bắt đầu gởi dữ liệu
xuống cho Client
                s.BeginSend(buff, 0, buff.Length, SocketFlags.None, new
AsyncCallback(SendData), s);
            }
        }
    }
}
```

VII.2.4.4. Lớp ServerForm

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Net;
using System.Net.Sockets;

namespace Server
{
    public partial class ServerForm : Form
    {
        ServerProgram server = new ServerProgram(IPAddress.Any, 6000);
```

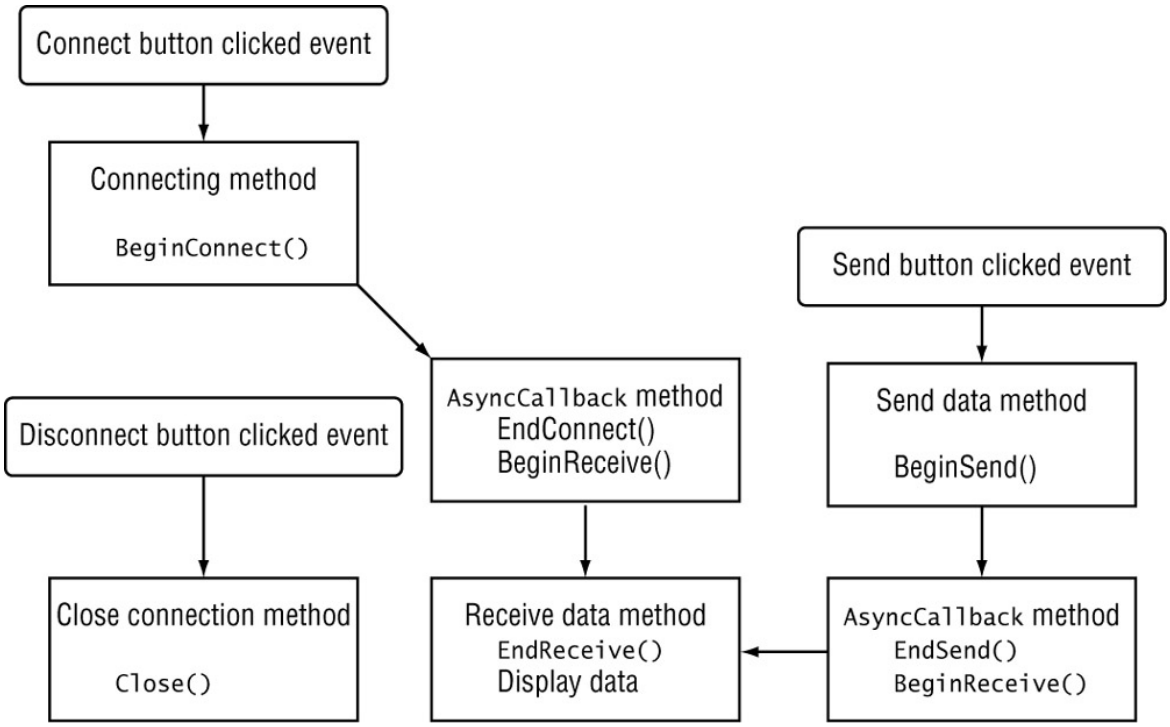
```
        public ServerForm()
        {
            InitializeComponent();
            CheckForIllegalCrossThreadCalls = false;
            server.SetDataFunction = new ServerProgram.SetDataControl(SetData);
        }

        private void SetData(string Data)
        {
            this.listBox1.Items.Add(Data);
        }
        private void cmdStart_Click(object sender, EventArgs e)
        {
            server.Listen();
        }
        private void cmdStop_Click(object sender, EventArgs e)
        {
            this.server.Close();
            SetData("Server dong ket noi");
        }
        private void ServerForm_Load(object sender, EventArgs e)
        {

        }
    }
}
```

VII.2.5. Chương trình Client



Hình VI.5: Giao diện Client

VII.2.5.1. Mô hình chương trình Client



Hình VI.6: Mô hình chương trình Client

VII.2.5.2. Lớp ClientProgram

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Net;
using System.Net.Sockets;
namespace Client
{
    class ClientProgram
    {
        //delegate để set dữ liệu cho các Control
        //Tại thời điểm này ta chưa biết dữ liệu sẽ được hiển thị vào đâu nên
ta phải dùng delegate
        public delegate void SetDataControl(string Data);
        public SetDataControl SetDataFunction = null;

        //buffer để nhận và gởi dữ liệu
        byte[] buff = new byte[1024];
        //Số byte thực sự nhận được
        int byteReceive = 0;
        //Chuỗi nhận được
        string stringReceive = "";
```

```
        Socket serverSocket = new Socket(AddressFamily.InterNetwork,
SocketType.Stream, ProtocolType.Tcp);
        IPEndPoint serverEP = null;
        //Lắng nghe kết nối
        public void Connect(IPAddress serverIP, int Port)
        {
            serverEP = new IPEndPoint(serverIP, Port);
            //Việc kết nối có thể mất nhiều thời gian nên phải thực hiện bất
đồng bộ
            serverSocket.BeginConnect( serverEP, new
AsyncCallback(ConnectCallback), serverSocket);
        }
        //Hàm callback chấp nhận Client kết nối
        private void ConnectCallback(IAsyncResult ia)
        {
            //Lấy Socket đang thực hiện việc kết nối bất đồng bộ
            Socket s = (Socket)ia.AsyncState;
            try
            {
                //Set dữ liệu cho Control
                SetDataFunction("Đang chờ kết nối");
                //Hàm EndConnect sẽ bị block cho đến khi kết nối thành công
                s.EndConnect(ia);
                SetDataFunction("Kết nối thành công");
            }
            catch
            {
                SetDataFunction("Kết nối thất bại");
                return;
            }
            //Ngay sau khi kết nối xong bắt đầu nhận câu chào từ Server gởi
xuống
            s.BeginReceive(buff, 0, buff.Length, SocketFlags.None, new
AsyncCallback(ReceiveData), s);
        }

        private void ReceiveData(IAsyncResult ia)
        {
            Socket s = (Socket)ia.AsyncState;
            byteReceive = s.EndReceive(ia);
            stringReceive = Encoding.ASCII.GetString(buff, 0, byteReceive);
            SetDataFunction(stringReceive);
        }
```

```
        private void SendData(IAsyncResult ia)
        {
            Socket s = (Socket)ia.AsyncState;
            s.EndSend(ia);
            //khởi tạo lại buffer để nhận dữ liệu
            buff = new byte[1024];
            //Bắt đầu nhận dữ liệu
            s.BeginReceive(buff, 0, buff.Length, SocketFlags.None, new
AsyncCallback(ReceiveData), s);
        }

        //Hàm ngắt kết nối
        public bool Disconnect()
        {
            try
            {
                //Shutdown Soket đang kết nối đến Server
                serverSocket.Shutdown(SocketShutdown.Both);
                serverSocket.Close();
                return true;
            }
            catch
            {
                return false;
            }
        }
        //Hàm gởi dữ liệu
        public void SendData(string Data)
        {
            buff = Encoding.ASCII.GetBytes(Data);
            serverSocket.BeginSend(buff, 0, buff.Length, SocketFlags.None, new
AsyncCallback(SendToServer), serverSocket);
        }
       //Hàm CallBack gởi dữ liệu
        private void SendToServer(IAsyncResult ia)
        {
            Socket s = (Socket)ia.AsyncState;
            s.EndSend(ia);
            buff = new byte[1024];
            s.BeginReceive(buff, 0, buff.Length, SocketFlags.None, new
AsyncCallback(ReceiveData), s);
        }
    }
```

```
}
```

VII.2.5.3. Lớp ClientForm

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Net;
using System.Net.Sockets;

namespace Client
{
    public partial class ClientForm : Form
    {
        ClientProgram client = new ClientProgram();
        public ClientForm()
        {
            InitializeComponent();
            CheckForIllegalCrossThreadCalls = false;
            client.SetDataFunction = new ClientProgram.SetDataControl(SetData);
        }
        private void SetData(string Data)
        {
            this.listBox1.Items.Add(Data);
        }

        private void cmdConnect_Click(object sender, EventArgs e)
        {
            client.Connect(IPAddress.Parse(this.txtServerIP.Text),
int.Parse(this.txtPort.Text));
        }
        private void cmdDisconnect_Click(object sender, EventArgs e)
        {
            client.Disconnect();
        }
        private void cmdSend_Click_1(object sender, EventArgs e)
        {
            client.SendData(this.txtInput.Text);
            this.txtInput.Text = "";
        }
    }
```

```
}
```

VII.3. Lập trình Socket bất đồng bộ sử dụng tiểu trình

VII.3.1. Lập trình sử dụng hàng đợi gởi và hàng đợi nhận thông điệp

Trong cách lập trình này, chúng ta sẽ dùng hai hàng đợi, một hàng đợi gởi và một hàng đợi nhận để thực hiện việc gởi và nhận dữ liệu. Lớp Queue nằm trong namespace System.Collections giúp ta thực hiện việc này.

```
Queue inQueue = new Queue();
Queue outQueue = new Queue();
```

Hai phương thức quan trọng của lớp Queue được dùng trong lập trình mạng là EnQueue() và DeQueue(). Phương thức EnQueue() sẽ đưa một đối tượng vào hàng đợi và phương thức DeQueue() sẽ lấy đối tượng đầu tiên từ hàng đợi ra.

Ý tưởng của phương pháp lập trình này là ta sẽ dùng hai vòng lặp vô hạn để kiểm tra hàng đợi gởi và hàng đợi nhận, khi hàng đợi gởi có dữ liệu thì dữ liệu đó được gởi ra ngoài mạng thông qua Socket, tương tự khi hàng đợi nhận có dữ liệu thì nó sẽ ngay lập tức lấy dữ liệu đó ra và xử lý.

```
private void Send()
{
	while (true)
	{
		if (OutQ.Count > 0)
		{
			streamWriter.WriteLine(OutQ.Dequeue().ToString());
			streamWriter.Flush();
		}
	}
}

private void Receive()
{
	string s;
	while (true)
	{
		s = streamReader.ReadLine();
		InQ.Enqueue(s);
	}
}
```

Để chạy song song hai vòng lặp vô hạn này ta phải tạo ra hai tiểu trình riêng, mỗi vòng lặp vô hạn sẽ chạy trong một tiểu trình riêng biệt do đó trong khi hai vòng lặp vô hạn này chạy thì tiến trình chính vẫn có thể làm được các công việc khác.

```
Thread tSend = new Thread(new ThreadStart(Send));
tSend.Start();

Thread tReceive = new Thread(new ThreadStart(Receive));
tReceive.Start();
```

Ngoài ra, ta còn sử dụng một tiểu trình khác thực hiện việc kết nối nhằm tránh gây treo tiến trình chính.

```
Thread tConnect = new Thread(new ThreadStart(WaitingConnect));
tConnect.Start();
```

Việc điều khiển kết nối được thực hiện trong một tiểu trình khác và được xử lý bằng phương thức WaitingConnect():

```
private void WaitingConnect()
{
      tcpListener = new TcpListener(1001);
      tcpListener.Start();

      socketForClient = tcpListener.AcceptSocket();

      if (socketForClient.Connected)
      {
            MessageBox.Show("Client Connected");
            netWorkStream = new NetworkStream(socketForClient);
            streamReader = new StreamReader(netWorkStream);
            streamWriter = new StreamWriter(netWorkStream);
            tSend = new Thread(new ThreadStart(Send));
            tSend.Start();
            tReceive = new Thread(new ThreadStart(Receive));
            tReceive.Start();
      }
      else
      {
            MessageBox.Show("Ket noi khong thanh cong");
      }
}
```

Việc nhập dữ liệu được thực hiện bằng phương thức InputData()

```
public void InputData(string s)
{
      InQ.Enqueue(s);
      OutQ.Enqueue(s);
}
```

Phương thức này đơn giản chỉ đưa dữ liệu nhập vào hàng đợi nhận để hiển thị dữ liệu lên màn hình và đưa dữ liệu nhập này vào hàng đợi gởi để gởi ra ngoài mạng.

Lớp ServerObject

```
using System;
using System.IO;
using System.Windows.Forms;
using System.Threading;
using System.Collections;
using System.Net.Sockets;
using System.Collections.Generic;
using System.Text;

namespace Server
{
    class ServerObject
    {
        Thread tSend, tReceive, tConnect;
        public Queue InQ = new Queue();
        public Queue OutQ = new Queue();

        private TcpListener tcpListener;
        Socket socketForClient;
        private NetworkStream netWorkStream;
        private StreamWriter streamWriter;
        private StreamReader streamReader;

        public void CreateConnection()
        {
            tConnect = new Thread(new ThreadStart(WaitingConnect));
            tConnect.Start();
        }

        private void WaitingConnect()
        {
            tcpListener = new TcpListener(1001);
```

```
        tcpListener.Start();

        socketForClient = tcpListener.AcceptSocket();

        if (socketForClient.Connected)
        {
            MessageBox.Show("Client Connected");
            netWorkStream = new NetworkStream(socketForClient);
            streamReader = new StreamReader(netWorkStream);
            streamWriter = new StreamWriter(netWorkStream);

            tSend = new Thread(new ThreadStart(Send));
            tSend.Start();

            tReceive = new Thread(new ThreadStart(Receive));
            tReceive.Start();
        }
        else
        {
            MessageBox.Show("Ket noi khong thanh cong");
        }
        //socketForClient.Close();
    }
    private void Send()
    {
        while (true)
        {
            if (OutQ.Count > 0)
            {
                streamWriter.WriteLine(OutQ.Dequeue().ToString());
                streamWriter.Flush();
            }
        }
    }
    private void Receive()
    {
        string s;
        while (true)
        {
            s = streamReader.ReadLine();
            InQ.Enqueue(s);
        }
    }
```

```
        public void InputData(string s)
        {
            InQ.Enqueue(s);
            OutQ.Enqueue(s);
        }

    }
}
```

Lớp ClientObject

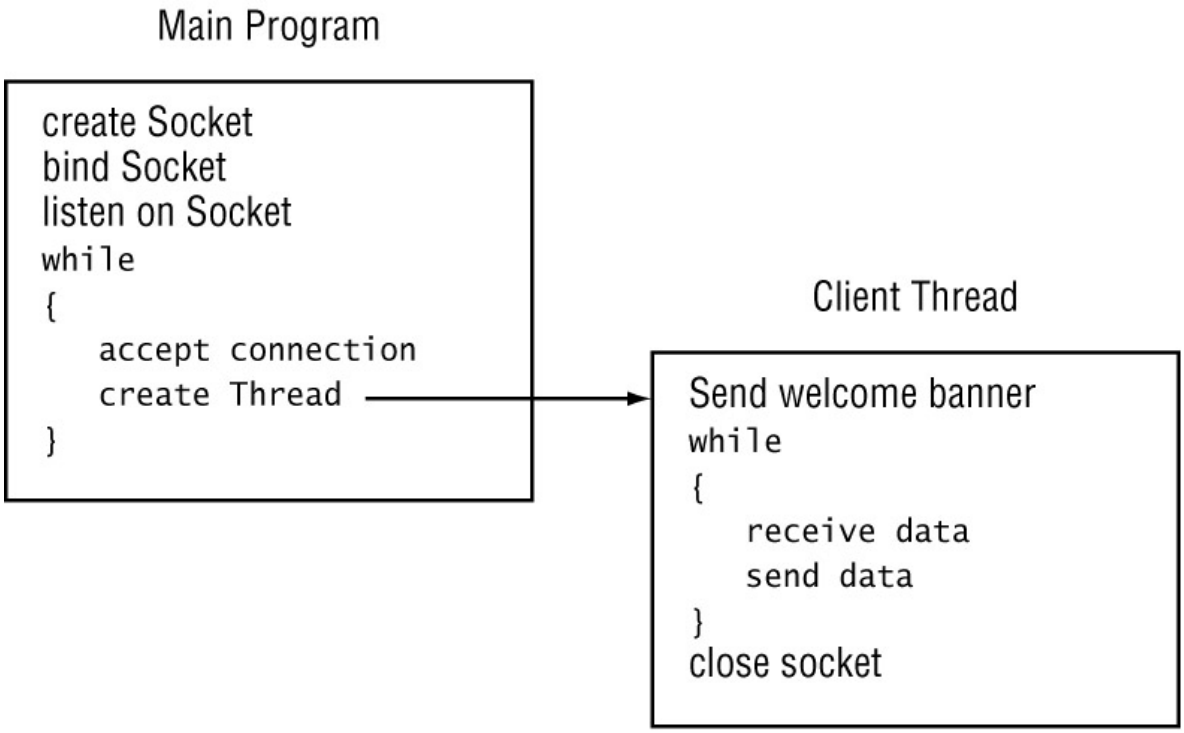
```
using System;
using System.Windows.Forms;
using System.Collections;
using System.Net.Sockets;
using System.Threading;
using System.IO;
using System.Collections.Generic;
using System.Text;

namespace Client
{
    class ClientObject
    {
        Thread tSend, tReceive, tConnect;

        public Queue InQ = new Queue();
        public Queue OutQ = new Queue();

        private TcpClient socketForServer;
        private NetworkStream networkStream;
        private StreamWriter streamWriter;
        private StreamReader streamReader;

        public void Connect()
        {
            tConnect = new Thread(new ThreadStart(WaitConnect));
            tConnect.Start();
        }
        public void WaitConnect()
        {
            try
            {
```

```
                socketForServer = new TcpClient("localhost", 1001);
            }
            catch {
                MessageBox.Show("Ket noi that bai");
            }
            networkStream = socketForServer.GetStream();
            streamReader = new StreamReader(networkStream);
            streamWriter = new StreamWriter(networkStream);

            tSend = new Thread(new ThreadStart(Send));
            tSend.Start();

            tReceive = new Thread(new ThreadStart(Receive));
            tReceive.Start();
        }
        private void Send()
        {
            while (true)
            {
                if (OutQ.Count > 0)
                {
                    streamWriter.WriteLine(OutQ.Dequeue().ToString());
                    streamWriter.Flush();
                }
            }
        }
        private void Receive()
        {
            string s;
            while (true)
            {
                s = streamReader.ReadLine();
                InQ.Enqueue(s);
            }
        }
        public void InputData(string s)
        {
            InQ.Enqueue(s);
            OutQ.Enqueue(s);
        }
    }
}
```

VII.3.2. Lập trình ứng dụng nhiều Client

Một trong những khó khăn lớn nhất của các nhà lập trình mạng đó là khả năng xử lý nhiều Client kết nối đến cùng một lúc. Để ứng dụng server xử lý được với nhiều Client đồng thời thì mỗi Client kết nối tới phải được xử lý trong một tiểu trình riêng biệt.

Mô hình xử lý như sau:



Hình VI.7: Mô hình xử lý một Server nhiều Client

Chương trình Server sẽ tạo ra đối tượng Socket chính trong chương trình chính, mỗi khi Client kết nối đến, chương trình chính sẽ tạo ra một tiểu trình riêng biệt để điều khiển kết nối. Bởi vì phương thức Accept() tạo ra một đối tượng Socket mới cho mỗi kết nối nên đối tượng mới này được sử dùng để thông tin liên lạc với Client trong tiểu trình mới. Socket ban đầu được tự do và có thể chấp nhận kết nối khác.

Chương trình ThreadedTcpSrvr

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
using System.Threading;
using System.Net;
using System.Net.Sockets;
```

```
class ThreadedTcpSrvr
{
    private TcpListener client;
    public ThreadedTcpSrvr()
    {
        client = new TcpListener(5000);
        client.Start();
        Console.WriteLine("Dang cho client...");
        while (true)
        {
            while (!client.Pending())
            {
                Thread.Sleep(1000);
            }
            ConnectionThread newconnection = new ConnectionThread();
            newconnection.threadListener = this.client;
            Thread newthread = new Thread(new
ThreadStart(newconnection.HandleConnection));
            newthread.Start();
        }
    }
    public static void Main()
    {
        ThreadedTcpSrvr server = new ThreadedTcpSrvr();
    }
}
class ConnectionThread
{
    public TcpListener threadListener;
    private static int connections = 0;
    public void HandleConnection()
    {
        int recv;
        byte[] data = new byte[1024];
        TcpClient client = threadListener.AcceptTcpClient();
        NetworkStream ns = client.GetStream();
        connections++;
        Console.WriteLine("Client moi duoc chap nhan, Hien tai co {0} ket noi",
connections);
        string welcome = "Xin chao client";
        data = Encoding.ASCII.GetBytes(welcome);
        ns.Write(data, 0, data.Length);
```

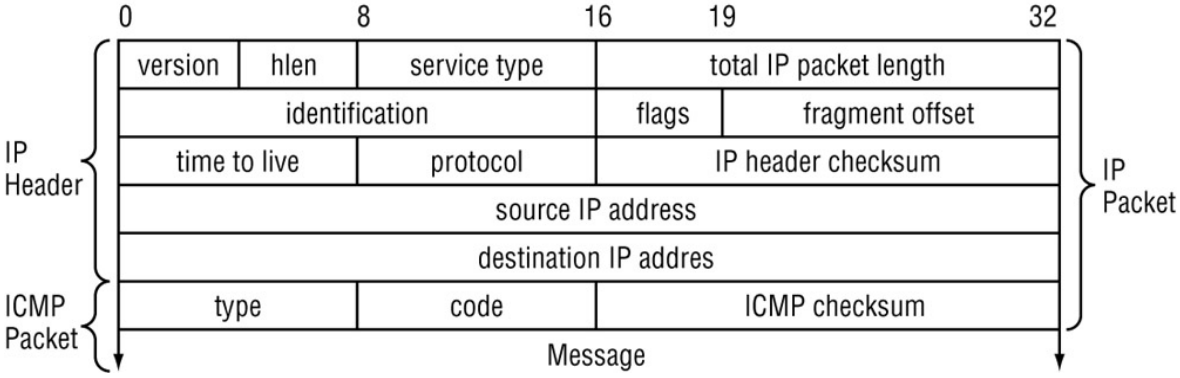
```
        while (true)
        {
            data = new byte[1024];
            recv = ns.Read(data, 0, data.Length);
            if (recv == 0)
                break;

            ns.Write(data, 0, recv);
        }
        ns.Close();
        client.Close();
        connection--;
        Console.WriteLine("Client disconnected: {0} active connections",
connections);
    }
}
```

CHƯƠNG VIII: LẬP TRÌNH VỚI CÁC GIAO THỨC

VIII.1. LẬP TRÌNH VỚI GIAO THỨC ICMP

VIII.1.1. Giao thức ICMP

ICMP được định nghĩa trong RFC 792, giao thức này giúp xác định lỗi của cac thiết bị mạng. ICMP sử dụng IP để truyền thông trên mạng. Mặc dù nó dùng IP nhưng ICMP hoàn toàn là một giao thức độc lập với TCP, UDP. Giao thức tầng kế tiếp của các gói tin IP được xác định trong phần dữ liệu sử dụng trường Type. Các gói tin ICMP được xác định bởi trường Type bằng 1 của gói tin IP, toàn bộ gói tin ICMP được kết hợp với phần dữ liệu của gói tin IP.



Định dạng của gói tin ICMP và IP

VIII.1.1.1. Định dạng của gói tin ICMP

Tương tự như TCP và UDP, ICMP dùng một đặc tả định dạng gói tin đặc biệt để xác định thông tin. Các gói tin ICMP gồm những trường sau:

Trường Type: kích thước 1 byte, xác định loại thông điệp ICMP trong gói tin. Nhiều loại gói tin ICMP được dùng để gởi thông điệp điều khiển đến các thiết bị ở xa. Mỗi loại thông điệp có định dạng riêng và các dữ liệu cần thiết.

Trường Code: kích thước 1 byte, các kiểu thông điệp ICMP khác nhau yêu cầu các tùy chọn dữ liệu và điều khiển riêng, những tùy chọn này được định nghĩa ở trường Code.

Trường Checksum: kích thước 2 byte, trường này đảm bảo gói tin ICMP đến đích mà không bị hư hại.

Trường Message: có nhiều byte, những byte này chứa các thành phần dữ liệu cần thiết cho mỗi kiểu thông điệp ICMP. Trường Message thường chứa đựng những thông tin được gởi và nhận từ thiết bị ở xa. Nhiều kiểu thông điệp ICMP định nghĩa 2 trường đầu tiên trong Message là Identifier và số Sequense. Các trường này dùng để định danh duy nhất gói tin ICMP đến các thiết bị.

VIII.1.1.2. Các tường Type của gói tin ICMP

Có nhiều kiểu gói tin ICMP, mỗi kiểu của gói tin ICMP được định nghĩa bởi 1 byte trong trường Type. Bảng sau liệt kê danh sách các kiểu ICMP ban đầu được định nghĩa trong RFC 792.

Type Code	**Mô Tả**
0	Echo reply
3	Destination unreachable
4	Source quench
5	Redirect
8	Echo request
11	Time exceeded
12	Parameter problem
13	Timestamp request
14	Timestamp reply
15	Information request
16	Information reply

Các trường Type của gói tin ICMP

Từ khi phát hành RFC 792 vào tháng 9 năm 1981, nhiều trường ICMP được tạo ra. Các gói tin ICMP được dùng cho việc thông báo lỗi mạng. Những mô tả sau hay dùng các gói tin ICMP:

VIII.1.1.3. Echo Request and Echo Reply Packets

Hai gói tin ICMP thường được sử dụng nhất là Echo Request và Echo Reply. Những gói tin này cho phép một thiết bị yêu cầu một trả lời ICMP từ một thiết bị ở xa

trên mạng. Đây chính là nhân của lệnh ping hay dùng để kiểm tra tình trạng của các thiết bị mạng.

Gói tin Echo Request dùng Type 8 của ICMP với giá trị code bằng 0. Phần dữ liệu của Message gồm các thành phần sau:

- 1 byte Indentifier: xác định duy nhất gói tin Echo Request
- 1 byte số Sequence: cung cấp thêm định danh cho gói tin gói tin ICMP trong một dòng các byte chứa dữ liệu được trả về gởi thiết bị nhận.

Khi một thiết bị nhận nhận một gói tin Echo Request, nó phải trả lời với một gói tin Echo Reply, trường Type ICMP bằng 0. Gói tin Echo Reply phải chứa cùng Identifier và số Sequence như gói tin Echo Request tương ứng. Phần giá trị dữ liệu cũng phải giống như của gói tin Echo Request.

VIII.1.1.4. Gói tin Destination Unreachable

Gói tin Destination Unreachable với trường Type bằng 3 thường trả về bởi thiết bị Router sau khi nó nhận được một gói tin IP mà nó không thể chuyển tới đích. Phần dữ liệu của gói tin Destination Unreachable chứa IP Header cộng với 64 bit giản đồ. Trong gói tin, trường Code xác định lý do gói tin không thể được chuyển đi bởi router. Bảng sau cho biết một số giá trị Code có thể gặp phải:

Code	**Mô Tả**
0	Network unreachable
1	Host unreachable
2	Protocol unreachable
3	Port unreachable
4	Fragmentation needed and DF flag set
5	Source route failed
6	Destination network unknown
7	Destination host unknown
8	Source host isolated
9	Communication with destination network prohibited
10	Communication with destination host prohibited
11	Network unreachable for type of service
12	Host unreachable for type of service

Các giá trị Code Destination Unreachable

VIII.1.1.5. Gói tin Time Exceeded

Gói tin Time Exceeded với trường Type của ICMP bằng 11 là một công cụ quan trọng để khắc phục các vấn đề mạng. Nó cho biết một gói tin IP đã vượt quá giá trị thời gian sống (TTL) được định nghĩa trong IP Header.

Mỗi khi một gói tin IP đến 1 router, giá trị TTL giảm đi một. Nếu giá trị TTL về 0 trước khi gói tin IP đến đích, router cuối cùng nhận được gói tin với giá trị TTL bằng 0 sẽ phải gởi một gói tin Time Exceeded cho thiết bị gởi. Lệnh tracert hay dùng gói tin này.

VIII.1.2. Sử dụng Raw Socket

Bởi vì các gói tin ICMP không dùng cả TCP và UDP, nên ta không dùng được các lớp hellper như TcpClient, UdpClient. Thay vì vậy, ta phải sử dụng Raw Socket, đây là một tính năng của lớp Socket. Raw Socket cho phép định nghĩa riêng các gói tin mạng phía trên tầng IP. Tất nhiên là tả phải làm tất cả mọi việc bằng tay như là tạo tất cả các trường của gói tin ICMP chứ không dùng các thư viện có sẵn của .NET như đã làm với TCP và UDP.

VIII.1.2.1. Định dạng của Raw Socket

Để tạo ra một Raw Socket, ta phải dùng SocketType.Raw khi Socket được tạo ra. Có nhiều giá trị ProtocolType ta có thể dùng với Raw Socket được liệt kê trong bảng sau:

Giá Trị	**Mô Tả**
Ggp	Gateway-to-Gateway Protocol
Icmp	Internet Control Message Protocol
Idp	IDP Protocol
Igmp	Internet Group Management Protocol
IP	A raw IP packet
Ipx	Novell IPX Protocol
ND	Net Disk Protocol
Pup	Xerox PARC Universal Protocol (PUP)
Raw	A raw IP packet
Spx	Novell SPX Protocol

Giá Trị	Mô Tả
SpxII	Novell SPX Version 2 Protocol
Unknown	An unknown protocol
Unspecified	An unspecified protocol

Các giá trị của Raw Socket ProtocolType

Những giao thức được liệt kê cho Raw Socket ở trên cho phép thư viện .NET tạo ra các gói tin IP bên dưới. Bằng cách sử dụng giá trị ProtocolType.Icmp, gói tin IP được tạo ra bởi Socket xác định giao thức tầng tiếp theo là ICMP (Trường Type bằng 1). Điều này cho phép thiết bị ở xa nhận ra ngay gói tin là 1 gói tin ICMP và xử lý nó một cách tương ứng. Sau đây là lệnh tạo ra một Socket cho các gói tin ICMP:

```
Socket sock = new Socket(AddressFamily.InterNetwork, SocketType.Raw, ProtocolType.Icmp);
```

VIII.1.2.2. Gởi các gói tin Raw

Bởi vì ICMP là một giao thức phi kết nối, nên ta không thể kết nối socket đến một port cục bộ để gởi một gói tin hoặc sử dụng phương thức Connect() để kết nối nó đến một thiết bị ở xa. Ta phải dùng phương thức SendTo() để chỉ ra đối tượng IPEndPoint của địa chỉ đích. ICMP không dùng các port vì thế giá trị port của đối tượng IPEndPoint không quan trọng. Ví dụ sau tạo một đối tượng IPEndPoint đích không có port và gởi một gói tin đến nó:

```
IPEndPoint iep = new IPEndPoint(IPAddress.Parse("192.168.1.2"), 0);
sock.SendTo(packet, iep);
```

VIII.1.2.3. Nhận các gói tin Raw

Nhận dữ liệu từ một Raw Socket khó hơn gởi dữ liệu từ một Raw Socket. Để nhận dữ liệu từ Raw Socket, ta phải dụng phương thức ReceiveFrom(). Bởi vì Raw Socket không định nghĩa giao thức tầng cao hơn, dữ liệu trả về từ một lần gọi phương thức ReceiveFrom() chứa toàn bộ nội dung của gói tin IP. Dữ liệu của gói tin IP bắt đầu từ byte thứ 20 do đó để lấy ra được dữ liệu và header của gói tin ICMP, ta phải bắt đầu đọc từ byte thứ 20 của dữ liệu nhận được.

VIII.1.3. Tạo ra một lớp ICMP

Như đã đề cập ở trước, Raw Socket không tự động định dạng gói tin ICMP vì vậy ta phải tự định dạng. Lớp ICMP phải định nghĩa mỗi biến cho mỗi thành phần trong gói tin ICMP. Các biến được định nghĩa mô tả một gói tin ICMP.

Các thành phần của một lớp ICMP điển hình

Biến Dữ Liệu	**Kích Thước**	**Kiểu**
Type	1 byte	Byte
Code	1 byte	Byte
Checksum	2 bytes	Unsigned 16-bit integer
Message	multibyte	Byte array

```
class ICMP
    {
        public byte Type;
        public byte Code;
        public UInt16 Checksum;
        public int MessageSize;
        public byte[] Message = new byte[1024];
        public ICMP()
        {
        }
    }
```

Sau khi gởi một gói tin ICMP thì hầu như sẽ nhận được một gói tin ICMP trả về từ thiết bị ở xa. Để dễ dàng giải mã nội dung của gói tin, chúng ta nên tạo một phương thức tạo lập khác của lớp ICMP nó có thể nhận một mảng byte Raw ICMP và đặt các giá trị vào phần dữ liệu thích hợp trong lớp:

```
public ICMP(byte[] data, int size)
  {
      Type = data[20];
      Code = data[21];
      Checksum = BitConverter.ToUInt16(data, 22);
      MessageSize = size - 24;
      Buffer.BlockCopy(data, 24, Message, 0, MessageSize);
  }
```

Raw Socket trả về toàn bộ gói tin IP do đó ta phải bỏ qua thông tin IP header trước khi lấy thông tin của gói tin ICMP vì thế phần Type ở tại vị trí 20 của mảng byte. Phần dữ liệu trong gói tin ICMP được lấy ra từng byte một và được đặt vào thành phần thích hợp của ICMP

Sua khi tạo ra một đối tượng ICMP mới với dữ liệu nhận được, ta có thể lấy các thành phần dữ liệu ra:

```
int recv = ReceiveFrom(data, ref ep);
ICMP response = new ICMP(data, recv);
Console.WriteLine("Received ICMP packet:");
Console.WriteLine(" Type {0}", response.Type);
Console.WriteLine(" Code: {0}", response.Code);
Int16 Identifier = BitConverter.ToInt16(response.Message, 0);
Int16 Sequence = BitConverter.ToInt16(response.Message, 2);
Console.WriteLine(" Identifier: {0}", Identifier);
Console.WriteLine(" Sequence: {0}", Sequence);
stringData = Encoding.ASCII.GetString(response.Message, 4, response.MessageSize
- 4);
Console.WriteLine(" data: {0}", stringData);
```

VIII.1.4. Tạo gói tin ICMP

Sau khi một đối tượng ICMP đã được tạo ra và phần dữ liệu của gói tin đã được định nghĩa, thì ta vẫn chưa thể gởi trực tiếp đối tượng ICMP bằng phương thức SendTo() được, nó phải chuyển thành 1 mảng byte, việc này được thực hiện nhờ vào phương thức Buffer.BlockCopy():

```
public byte[] getBytes()
{
      byte[] data = new byte[MessageSize + 9];
      Buffer.BlockCopy(BitConverter.GetBytes(Type), 0, data, 0, 1);
      Buffer.BlockCopy(BitConverter.GetBytes(Code), 0, data, 1, 1);
      Buffer.BlockCopy(BitConverter.GetBytes(Checksum), 0, data, 2, 2);
      Buffer.BlockCopy(Message, 0, data, 4, MessageSize);
      return data;
}
```

VIII.1.5. Tạo phương thức Checksum

Có lẽ phần khó khăn nhất của việc tạo một gói tin ICMP là tính toán giá trị checksum của gói tin. Cách dễ nhất để thực hiện điều này là tạo ra mọt phương thức để tính checksum rồi đặt đặt nó vào trong lớp ICMP để nó được sử dụng bởi chương trình ứng dụng ICMP.

```
public UInt16 getChecksum()
{
      UInt32 chcksm = 0;
      byte[] data = getBytes();
```

```
        int packetsize = MessageSize + 8;
        int index = 0;
        while (index < packetsize)
        {
              chcksm += Convert.ToUInt32(BitConverter.ToUInt16(data, index));
              index += 2;
        }
        chcksm = (chcksm >> 16) + (chcksm & 0xffff);
        chcksm += (chcksm >> 16);
        return (UInt16)(~chcksm);
}
```

Bởi vì giá trị checksum sử dụng mộ số 16 bit, thuật toán này đọc từng khúc 2 byte của gói tin ICMP một lúc (sử dụng phương thức ToUInt16() của lớp BitConverter) và thực hiện các phép toán số học cần thiết trên các byte. Giá trị trả về là một số nguyên dương 16 bit.

Để sử dụng giá trị checksum trong một chương trình ứng dụng ICMP, đầu tiên điền tất cả các thành phần dữ liệu, thiết lập thành phần Checksum thành 0, tiếp theo gọi phương thức getChecksum() để tính toán checksum của gói tin ICMP và sau đó đặt kết quả vào thành phần Checksum của gói tin:

```
packet.Checksum = 0;
packet.Checksum = packet.getChecksum();
```

Sau khi thành phần Checksum được tính toán, gói tin đã sẵn sàng được gởi ra ngoài thiết bị đích dùng phương thức SendTo().

Khi một gói tin ICMP nhận được từ một thiết bị ở xa, chúng ta phải lấy phần Checksum ra và so sánh với giá trị checksum tính được, nếu 2 giá trị đó không khớp nhau thì đã có lỗi xảy ra trong quá trình truyền và gói tin phải được truyền lại.

VIII.1.6. Lớp ICMP hoàn chỉnh

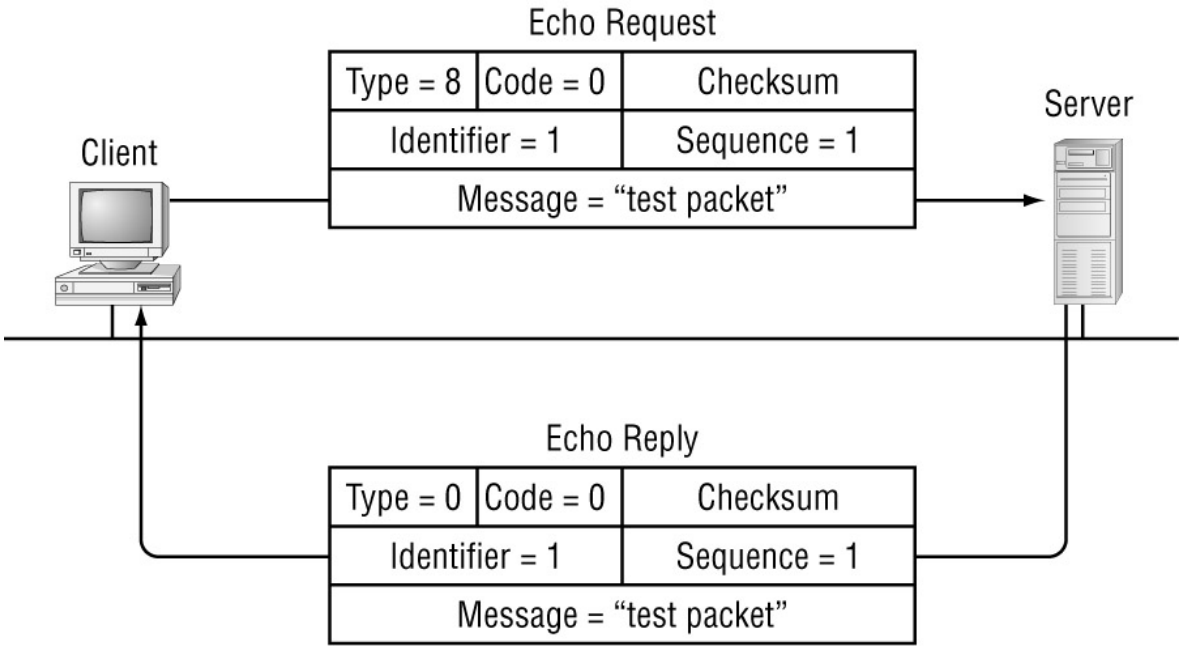
```
using System;
using System.Net;
using System.Text;
class ICMP
{
    public byte Type;
    public byte Code;
    public UInt16 Checksum;
```

```
    public int MessageSize;
    public byte[] Message = new byte[1024];
    public ICMP()
    {
    }
    public ICMP(byte[] data, int size)
    {
        Type = data[20];
        Code = data[21];
        Checksum = BitConverter.ToUInt16(data, 22);
        MessageSize = size - 24;
        Buffer.BlockCopy(data, 24, Message, 0, MessageSize);
    }
    public byte[] getBytes()
    {
        byte[] data = new byte[MessageSize + 9];
        Buffer.BlockCopy(BitConverter.GetBytes(Type), 0, data, 0, 1);
        Buffer.BlockCopy(BitConverter.GetBytes(Code), 0, data, 1, 1);
        Buffer.BlockCopy(BitConverter.GetBytes(Checksum), 0, data, 2, 2);
        Buffer.BlockCopy(Message, 0, data, 4, MessageSize);
        return data;
    }
    public UInt16 getChecksum()
    {
        UInt32 chcksm = 0;
        byte[] data = getBytes();
        int packetsize = MessageSize + 8;
        int index = 0;
        while (index < packetsize)
        {
            chcksm += Convert.ToUInt32(BitConverter.ToUInt16(data, index));
            index += 2;
        }
        chcksm = (chcksm >> 16) + (chcksm & 0xffff);
        chcksm += (chcksm >> 16);
        return (UInt16)(~chcksm);
    }
}
```

VIII.1.7. Chương trình ping đơn giản

Chương trình ping là một chương trình quan tronjng, nó là công cụ ban đầu để chuẩn đoán khả năng kết nối của một thiết bị mạng. Chương trình ping dùng các gói

tin ICMP Echo Request (Type 8) để gởi một thông điệp đơn giản đến thiết bị ở xa. Khi thiết bị ở xa nhận thông điệp, nó trả lời lại với một gói tin ICMP Echo Reply (Type 0) chứa thông điệp ban đầu



Thông điệp ICMP được dùng trong lệnh ping

Chương trình ping đơn giản:

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
class SimplePing
{
    public static void Main(string[] argv)
    {
        byte[] data = new byte[1024];
        int recv;
        Socket host = new Socket(AddressFamily.InterNetwork, SocketType.Raw,
ProtocolType.Icmp);
        IPEndPoint iep = new IPEndPoint(IPAddress.Parse(argv[0]), 0);
        EndPoint ep = (EndPoint)iep;
        ICMP packet = new ICMP();
        packet.Type = 0x08;
        packet.Code = 0x00;
        packet.Checksum = 0;
```

```
        Buffer.BlockCopy(BitConverter.GetBytes((short)1), 0, packet.Message, 0,
2);
        Buffer.BlockCopy(BitConverter.GetBytes((short)1), 0, packet.Message, 2,
2);
        data = Encoding.ASCII.GetBytes("goi tin test");
        Buffer.BlockCopy(data, 0, packet.Message, 4, data.Length);
        packet.MessageSize = data.Length + 4;
        int packetsize = packet.MessageSize + 4;
        UInt16 chcksum = packet.getChecksum();
        packet.Checksum = chcksum;
        host.SetSocketOption(SocketOptionLevel.Socket,
SocketOptionName.ReceiveTimeout, 3000);
        host.SendTo(packet.getBytes(), packetsize, SocketFlags.None, iep);
        try
        {
            data = new byte[1024];
            recv = host.ReceiveFrom(data, ref ep);
        }
        catch (SocketException)
        {
            Console.WriteLine("Khong co tra loi tu thiet bi o xa");
            return;
        }
        ICMP response = new ICMP(data, recv);
        Console.WriteLine("tra loi tu: {0}", ep.ToString());
        Console.WriteLine(" Type {0}", response.Type);
        Console.WriteLine(" Code: {0}", response.Code);
        int Identifier = BitConverter.ToInt16(response.Message, 0);
        int Sequence = BitConverter.ToInt16(response.Message, 2);
        Console.WriteLine(" Identifier: {0}", Identifier);
        Console.WriteLine(" Sequence: {0}", Sequence);
        string stringData = Encoding.ASCII.GetString(response.Message, 4,
response.MessageSize - 4);
        Console.WriteLine(" data: {0}", stringData);

        host.Close();
    }
}
```

Chương trình ping đơn giản này chỉ yêu cầu một địa chỉ IP được dùng ở command line. Trong phần đầu tiên của chương trình này, một gói tin ICMP được tạo ra, trường Type có giá trị là 8 và Code có giá trị 0. Nó tạo ra một gói tin Echo Request

sử dụng trường Identifier và Sequence để theo dõi các gói tin riêng biệt và cho phép nhập text vào phần dữ liệu.

Cũng giống như các chương trình hướng phi kết nối như UDP, một giá trị time-out được thiết lập cho Socket dùng phương thức SetSocketOption(). Nếu không có gói tin ICMP trả về trong 3 giây, một biệt lệ sẽ được ném ra và chương trình sẽ thoát.

Gói tin ICMP trả về tạo ra một đối tượng ICMP mới, đối tượng này có thể dùng để quyết định nếu gói tin nhận được khớp với gói tin ICMP gởi. Trường Identifier, Sequence và phần dữ liệu của gói tin nhận phải khớp giá trị của gói tin gởi.

VIII.1.8. Chương trình TraceRoute đơn giản

Chương trình traceroute gởi các ICMP Echo Request đến máy ở xa để biết được các router mà nó sẽ đi qua cho đến đích. Bằng cách thiết lập trường TTL (time to live) của gói tin IP tăng lên 1 giá trị khi đi qua mỗi router, chương trình traceroute có thể bắt buộc gói tin ICMP bị loại bỏ tại các router khác nhau trên đường nó tới đích. Mỗi lần trường TTL hết hạn, router cuối cùng sẽ gởi trả về một gói tin (Type 11) cho thiết bị gởi.

Bằng cách đặt giá trị khởi đầu cho trường TTL bằng 1, khi gói tin IP đi qua mỗi router thì giá trị TTL sẽ giảm đi một. Sau mỗi lần thiết bị gởi nhận được gói tin ICMP Time Exceeded trường TTL sẽ được tăng lên 1. Bằng cách hiển thị các địa chỉ gởi các gói tin ICMP Time Exceeded, ta có thể xác định được chi tiết đường đi của một gói tin tới đích.

Chương trình traceroute sử dụng phương thức SetSocketOption() và tùy chọn IPTimeToLive của Socket để thay đổi giá trị TTL của gói tin IP. Bởi vì chỉ có mỗi trường TTL của gói tin IP bị thay đổi, gói tin ICMP có thể được tạo ra một lần và được sử dụng trong các lần tiếp theo mỗi khi sử dụng gói tin IP này.

Chương trình traceroute đơn giản:

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
class TraceRoute
{
    public static void Main(string[] argv)
    {
```

```
        byte[] data = new byte[1024];
        int recv, timestart, timestop;
        Socket host = new Socket(AddressFamily.InterNetwork, SocketType.Raw,
ProtocolType.Icmp);
        IPHostEntry iphe = Dns.Resolve(argv[0]);
        IPEndPoint iep = new IPEndPoint(iphe.AddressList[0], 0);
        EndPoint ep = (EndPoint)iep;
        ICMP packet = new ICMP();
        packet.Type = 0x08;
        packet.Code = 0x00;
        packet.Checksum = 0;
        Buffer.BlockCopy(BitConverter.GetBytes(1), 0, packet.Message, 0, 2);
        Buffer.BlockCopy(BitConverter.GetBytes(1), 0, packet.Message, 2, 2);
        data = Encoding.ASCII.GetBytes("goi tin test");
        Buffer.BlockCopy(data, 0, packet.Message, 4, data.Length);
        packet.MessageSize = data.Length + 4;
        int packetsize = packet.MessageSize + 4;
        UInt16 chcksum = packet.getCchecksum();
        packet.Checksum = chcksum;
        host.SetSocketOption(SocketOptionLevel.Socket,
                   SocketOptionName.ReceiveTimeout, 3000);
        int badcount = 0;
        for (int i = 1; i < 50; i++)
        {
            host.SetSocketOption(SocketOptionLevel.IP,
                  SocketOptionName.IpTimeToLive, i);
            timestart = Environment.TickCount;
            host.SendTo(packet.getBytes(), packetsize, SocketFlags.None, iep);
            try
            {
                data = new byte[1024];
                recv = host.ReceiveFrom(data, ref ep);
                timestop = Environment.TickCount;
                ICMP response = new ICMP(data, recv);
                if (response.Type == 11)
                    Console.WriteLine("Chang {0}: tra loi tu {1}, {2}ms", i,
ep.ToString(), timestop - timestart);
                if (response.Type == 0)
                {
                    Console.WriteLine("{0} den {1} chang, {2}ms.",
ep.ToString(), i, timestop - timestart);
                    break;
                }
                badcount = 0;
            }
            catch (SocketException)
            {
                Console.WriteLine("chang {0}: khong co tra loi tu thiet bi o
xa", i);
                badcount++;
                if (badcount == 5)
                {
                    Console.WriteLine("Khong the lien lac voi thiet bi o xa");
                    break;
                }
            }
        }

        host.Close();
    }
}
```

VIII.2. LẬP TRÌNH VỚI GIAO THỨC SMTP

VIII.2.1. Cơ bản về Email

Hầu hết mọi gói tin email trên Internet đều dùng mô hình email của Unix. Mô hình này trở nên phổ biến và được sử dụng rộng rãi để phân phá thư đến cả người dùng cục bộ và người dùng ở xa.

Mô hình email Unix chia các chức năng email ra làm 3 phần:

- The Message Transfer Agent (MTA)
- The Message Delivery Agent (MDA)
- The Message User Agent (MUA)

Mỗi một phần thực hiện một chức năng riêng biệt trong quá trình gởi, nhận và hiển thị thư

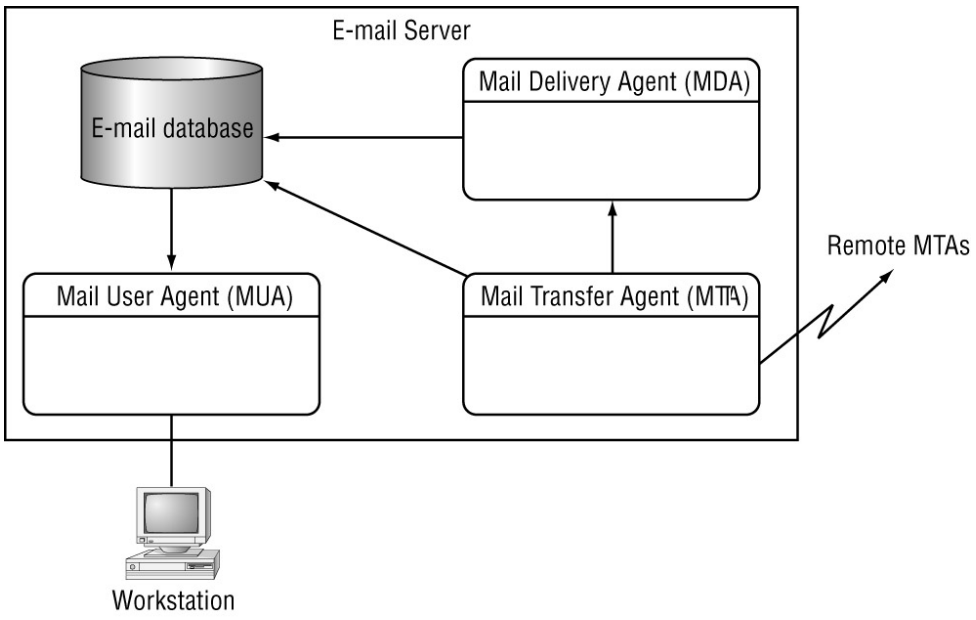
VIII.2.1.1. Hoạt động của MTA

MTA điều khiển cả các mail gởi đến và mail gởi đi, trước đây nó được chia ra làm 2 chức năng riêng biệt:

Quyết định nơi và cách gởi mail ra ngoài

Quyết định noi chuyển mail đến

Mỗi một chức năng trên yêu cầu một chức năng xử lý khác nhau trong MTA



Mô hình email Unix

VIII.2.1.2. Gởi mail ra ngoài

Với mỗi mail được gởi ra ngoài, MTA phải quyết định đích của địa chỉ nhận. Nếu đích là hệ thống cục bộ, MTA có thể hoặc là chuyển mail trực tiếp hệ thống mailbox cục bộ hoặc chuyển mail đến MDA để phát mail.

Tuy nhiên, nếu đích là một domain ở xa, MTA phải thực hiện 2 chức năng sau:

- Quyết định mail server của domain đó có sử dụng mục MX ở DNS hay không
- Thành lập một liên kết thông tin với mail server ở xa và chuyển mail.

Liên kết thông tin với mail server ở xa luôn luôn dùng SMTP (Simple Mail Transfer Protocol). Giao thức chuẩn này cung cấp một ký thuật giao tiếp chung để chuyển mail giữa 2 hệ thống mail khác nhau trên Internet

VIII.2.1.3. Nhận mail

MTA chịu trách nhiệm nhận các thư gởi đến cả từ hệ thống cục bộ và từ người dùng ở xa. Địa chỉ đích của thư phải được xem xét kỹ lưỡng và một quyết định phải được thực hiện cho biết thư có thể thực sự được gởi từ hệ thống cục bộ hay không.

Có 3 danh mục địa chỉ đích có thể được dùng trong thư: các tài khoản cục bộ của hệ thống, các tài khoản bí danh cục bộ, các tài khoản người dùng ở xa.

Các tài khoản cục bộ của hệ thống: mỗi hệ thống mail, dù là Window hay Unix hay ngay cả Macintosh cũng đều có một tập hợp các tài khoản cục bộ có thể truy cập vào hệ thống. MTA phải nhận ra các thư các mail có đích là tài khỏa người dùng và chuyển nó hoặc trực tiếp đến hộp thư của người dùng hoặc đến một MDA riêng biệt để chuyển đi.

Các tài khoản bí danh cục bộ: Nhiều MTA cho phép các tên bí danh được tạo ra. Tên bí danh chính nó không thể lưu giữ thư, thay vì vậy nó sử dụng con trỏ đến một hoặc nhiều người dùng của hệ thống thực sự nơi mà thư được lưu trữ. Mỗi khi MTA quyết định tên bí danh hợp lệ, nó chuyển đổi địa chỉ đích thành hệ thống tên tài khoản người dùng thực sự,

Các tài khoản người dùng ở xa: nhiều MTA cũng chấp nhận các thư gởi đến mà đích là các tài khoản người dùng không nằm trên hệ thống cục bộ. Đây là công việc

đòi hỏi phải có nhiều kỹ thuật, nhiều khó khăn để MTA điều khiển việc gởi mail đến một hệ thống các tài khoản ở xa.

Kỹ thuật này được gọi là relying. Một mail server chấp nhận một thư gởi đến mà đích là một tài khoản của máy ở xa và tự động chuyển thư đó đến máy ở xa. Nhiều ISP, tính năng này là cần thiết bởi khác hàng không có khả năng gởi thư trực tiếp đến các máy ở xa. Thay vì vậy, họ sẽ gởi thư đến mail server của ISP, và mail server của ISP sẽ tự động chuyển thư này đến hệ thống đích.

Thật không may mắn, việc chuyển tiếp mail có thể bị khai thác, người dùng có thể sử dụng các mail server chuyển tiếp để ẩn địa chỉ nguồn khi chuyển hàng loạt thư rác đến các người dùng trên hệ thống email ở xa, những email này phải được chặn lại bởi người quản trị mail.

VIII.2.1.4. Hoạt động của MDA

Chức năng chính của MDA là chuyển các thư có đích là các tài khoản người dùng trên hệ thống cục bộ. Để làm được điều này, MDA phải biết kiểu và vị trí của các mailbox của các cá nhân. Hầu hết các hệ thống email sử dùng một số kiểu hệ thống cơ sở dữ liệu để theo dõi các thư được lưu giữ bởi người dùng cục bộ. MDA phải truy cập đến mỗi mailbox của người dùng để chèn các thư được gởi đến.

Nhiều MDA cũng thực hiện các kỹ thuật nâng cao chỉ để chuyển tiếp thư:

Tự động lọc thư: đây là chức năng phổ biến nhất của các MDA để lọc các thu đến. Đối với các người dùng phải nhận hàng đống email mỗi ngày thì đây là một tính năng tuyệt vời. Các thư có thể được tự động sắp xếp theo các thư mục riêng biệt dựa vào các giá trị header, hay ngay cả một vài từ trong header. Hầu hết các MDA cũng cho phép nhà quản trị mail cấu hình lọc trên toàn bộ hệ thống để có thể chặn các thư rác hoặc các thư có virus.

Tự động trả lời thư: nhiều chương trình MDA cho phép người dùng cấu hình tự chuyển thư, một số thư tự động trả lời có thể được cấu hình để trả lời tất cả các thư được nhận bởi một người dùng, một số khác được cấu hình tự động trả lời dựa vào một số từ đặc biệt trong header.

Tự động chạy chương trình: đây là khả năng khởi động chương trình hệ thống dựa vào một số mail đến, đây là một công cụ quản trị của nhà quản trị. Nhà quản trị hệ thống mail có thể khởi động hệ thống mail từ xa hay có thể cấu hình mail server từ một máy khác mail server.

VIII.2.1.5. Hoạt động của MUA

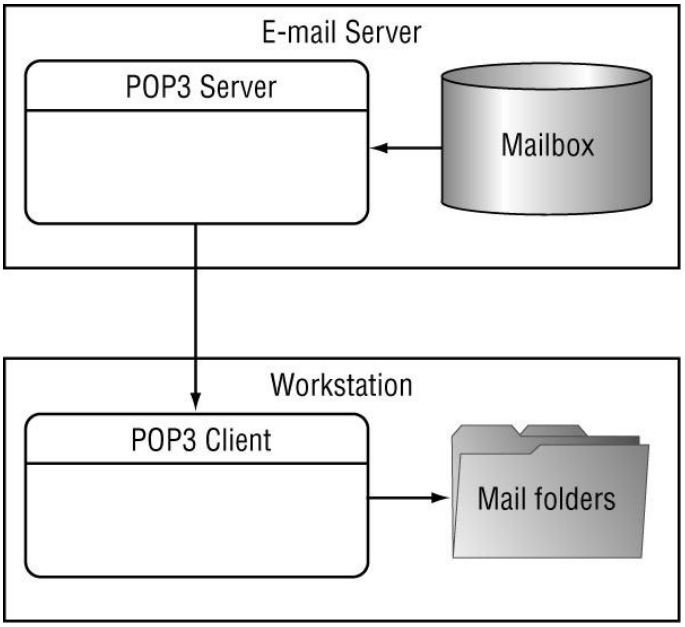
MUA cho phép người dùng không ngồi trước mail server vẫn có thể truy cập hộp thư của họ. MUA chỉ chịu trách nhiệm chính đọc các thư trong hộp thư, nó không thể nhận hoặc gởi thư.

Có 2 vấn đề phức tạp với MUA là việc đọc và lưu trữ các thư đến:

Lưu giữ các thư ở Client

Nhiều ISP thích người dùng tải các thư của họ về. Mỗi khi thư được tải về, nó sẽ bị gở bỏ khỏi mail server. Điều này giúp cho người quản trị mail bảo toàn được không gian lưu trữ trên server.

Giao thức được dùng cho loại truy cập này là POP (Post Office Protocol, thường được gọi là POP3 cho version 3). Chương trình MUA POP3 cho phép người dùng kết nối đến server và tải tất cả các thư ở hộp thư về. Quá trình này được mô tả trong hình sau:

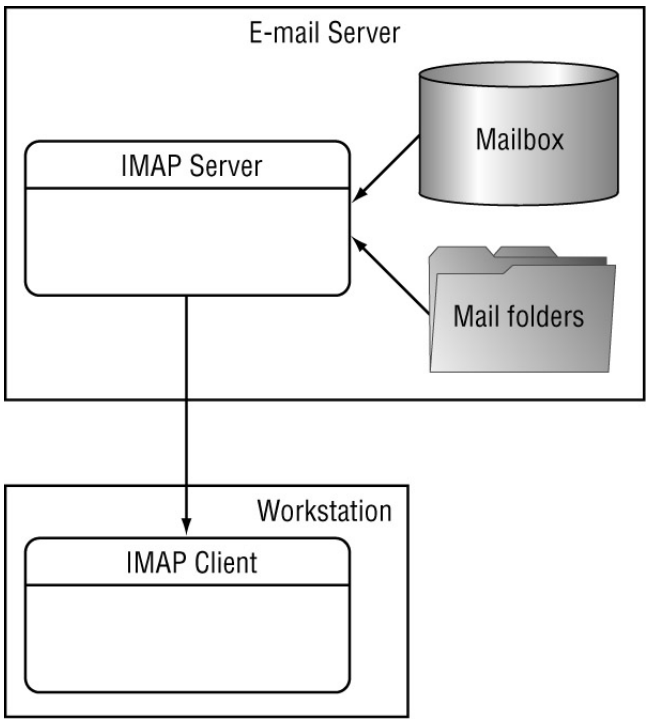


Quá trình tải thư sử dụng phần mềm POP3

Điều trở ngại khi sử dụng POP3 là các thư được lưu giữ ở trạm làm việc nơi mà người dùng kết nối tới mail server. Nếu người dùng kết nối sử dụng nhiều trạm làm việc thì các thư sẽ bị chia ra giữa các trạm làm việc làm cho người dùng khó truy cập các thư sau khi đã tải các thư xong. Hiện nay một số ISP cho phép tải thư sử dụng POP3 mà không xóa khỏi hộp thư.

Lưu giữ thư ở Server

Một phương thức truy cập thư thay cho POP3 là IMAP (Interactive Mail Access Protocol hay IMAPPrev4 cho version 4). IMAP cho phép người dùng xây dựng các thư mục ở mail server và lưu giữ các thư trong các thư mục thay vì phải tải xuống trạm làm việc. Bởi vì các thư được lưu giữ ở server, người dùng có thể kết nối từ bất kỳ trạm làm việc nào để đọc thư. Quá trình này được mô tả như trong hình sau:



Lưu giữ thư trên server ở các thư mục sử dụng IMAP

Điều trở ngại đối với hệ thống này là việc tiêu tốn dung lượng đĩa trên server.

The .NET mail library uses the Microsoft CDOSYS message component to send messages to remote hosts using SMTP. One significant area of confusion is when and how Windows OSes provide support for CDOSYS and how the CDOSYS relates to .NET. This section will shed some light on this topic, in addition to answering questions about CDOSYS.

VIII.2.2. SMTP và Windows

Thư viện mail của .NET sử dụng Microsoft CDOSYS để gởi thư đến máy ở xa dùng SMTP.

VIII.2.2.1. Collaboration Data Objects (CDO)

Hệ thống mail Microsoft Exchange 5 sử dụng thư viện Active Message (OLEMSG32.DLL) cho phép các lập trình viên viết code dùng hệ thống Exchange chuyển thư đến các hộp thư của người dùng cũng như đến hệ thống Exchange khác trên mạng Exchange.

Với sự phát hành của Exchange 5.5, một hệ thống thư mới được giới thiệu CDO (Collaboration Data Object). Không giống như Active Messaging, CDO sử dụng MAPI (Message Application Program Interface) để cung cấp cho các lập trình viên một cách dễ hơn để gởi thư. CDO cũng tồn tại qua vài năm với nhiều cải tiến bao gồm phiên bản 1.1, 1.2 và 1.2.1. Nền tảng Microsoft NT Server sau này sử dụng thư viện CDO NT Server (CDONTS). Thư viện này không dùng chuẩn MAPI được sử dụng trong CDO mà bắt đầu sử dụng các chuẩn của Internet như SMTP để chuyển thư đến các máy ở xa.

Phiên bản hiện tại của CDO (CDO 2) được phát hành chung với Windows 2000 và nó cũng được dùng trên Windows XP. Nó sử dụng CDONTS để cung cấp các thư viện mail và news cho lập trình viên. Các thành phần mới của CDO 2 bao gồm khả năng xử lý nhiều file đính kèm thư và nhiều sự kiện giao thức mail khác nhau. Những tính năng này giúp cho các lập trình viên dễ dàng hơn trong việc lập trình.

Trong Windows 2000, XP, thư viện CDO 2 là CDOSYS.DLL, tất cả các chức năng trong thư viện mail .NET phải cần tập tin này. Bởi vì CDO 2 hoàn toàn khác với các phiên bản CDO 1.x nên hệ thống cần cả 2 thư viện này. CDO 2 không tương thích lùi với các nền tảng Windows cũ hơn.

VIII.2.2.2. Dịch vụ mail SMTP

Cả Windows 2000 và Windows XP đều cung cấp một server SMTP cơ bản hỗ trợ cả gởi và nhận thư sử dụng giao thức SMTP. Chức năng này là một phần của IIS

(Internet Information Services). Các lớp mail của .NET có thể sử dụng IIS SMTP Server để gởi thư trực tiếp ra ngoài đến các mail server ở xa.

Trước khi sử dụng SMTP, chúng ta phải cấu hình cho nó, việc cấu hình được thực hiện từ cửa sổ Computer Management.

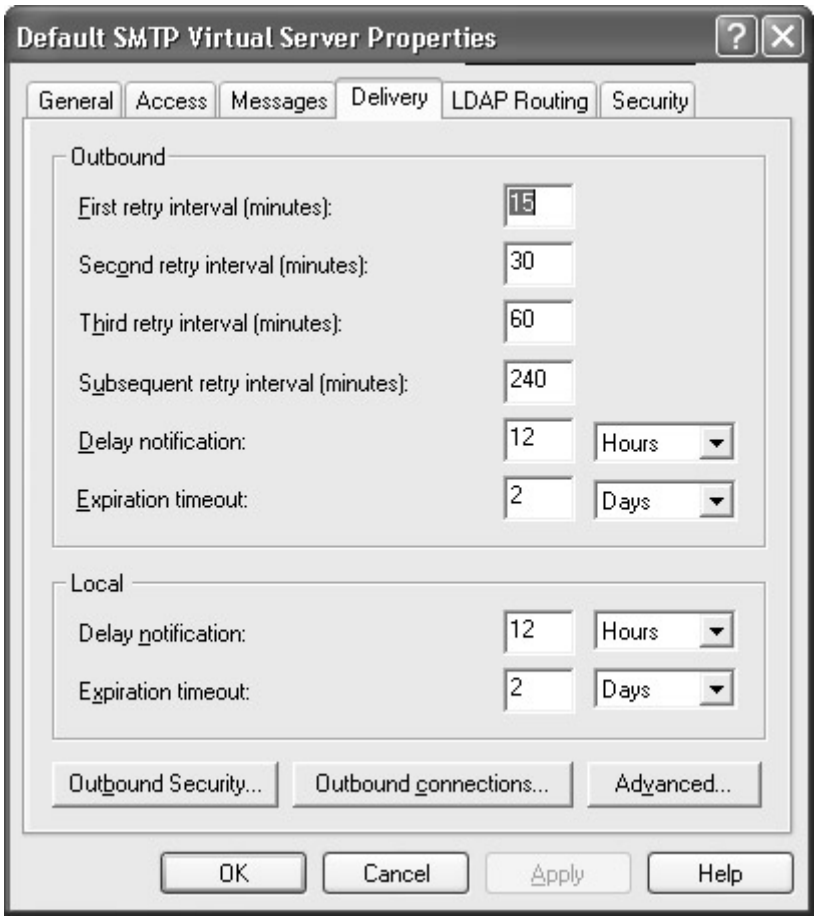
Các bước cấu hình như sau:

1. Nhấn chuột phải vào My Computer, chọn Manage
2. Khi cửa sổ Computer Management xuất hiện, mở Services and Applications và sau đó mở Internet Information Services
3. Nhấn chuột phải vào Default SMTP Virtual Server và chọn Properties

Cửa sổ Properties là nơi để chúng ta cấu hình các thiết lập cho dịch vụ SMTP

Trong thẻ General, chọn card mạng cho phép các kết nối SMTP tới và số lượng các kết nối SMTP đồng thời được phép. Chúng ta cũng có thể bật/tắt việc ghi log cho dịch vụ SMTP.

Trong thẻ Delivery, cấu hình số lần cố gắng gởi thư. Nút Advanced là các thiết lập thông minh cho mail server. Cấu hình này quan trọng nếu mail server của ISP là một server chuyển tếp đến mail server ở xa. Để làm điều này, chúng ta phải, nhập địa chỉ mail server của ISP và dịch vụ SMTP sẽ chuyển tiếp tất cả các thư ra ngoài thông qua server này.



Default SMTP Virtual Properties

VIII.2.3. Lớp SmtpMail

Lớp SmtpMail nằm trong namespace System.Web.Mail cho phép gởi thư theo giao thức SMTP trong chương trình C#.

VIII.2.4. Các phương thức và thuộc tính của lớp SmtpMail

Lớp SmtpMail cung cấp giao tiếp .NET cho thư viện mail CDOSYS trên hệ thống Windows 200 và Windows XP. Nó không dùng phương thức tạo lập để tạo 1 thể hiện của lớp. Thay vì vậy, ta phải dùng các phương thức và thuộc tính static để truyền thông tin cho thư viện CDOSYS.

Phương thức Send() của lớp SmtpMail là hay được dùng nhất, phương thức Send() được quá tải hàm và nó có 2 định dạng như sau:

```
Send(MailMessage message)
Send(string from, string to, string subject, string body)
```

Định dạng đầu tiên cho phép gởi một đối tượng MailMessage. Lớp MailMessage chứa một thông điệp email, thông điệp này được tạo ra bằng cách gán giá trị cho các thuộc tính của lớp với thông tin liên quan đến thông điệp và các địa chỉ đích.

Định dạng thứ 2 sẽ cho phép chúng ta gởi một thông điệp dạng raw, chúng ta phải chỉ ra các trường header của thông điệp email:

From: người gởi thông điệp

To: các địa chỉ nhận thông điệp, ngăn cách bởi dấu phẩy

Subject: tiêu đề thư

Đối số cuối cùng của phương thức Send() là phần thân của thư. Phần này có thể là text hoặc HTML.

Thuộc tính duy nhất của lớp SmtpMail là SmtpServer. Nó là một thuộc tính static và nó chỉ ra địa chỉ của server chuyển tiếp thư được dùng để chuyển tiếp các thông điệp mail ra ngoài. Mặc định, thuộc tính SmtpSerrver được thiết lập cho dịch vụ SMTP của IIS nếu IIS được cài. Nếu IIS không được cài, thuộc tính SmtpServer được thiết lập giá trị null và sẽ phát sinh ra lỗi khi gởi thông điệp đi.

Nếu chúng ta sử dụng một mail server chuyển tiếp, chúng ta phải thiết lập giá trị cho thuộc tính SmtpServer trước khi gởi thông điệp.

```
SmtpMail.SmtpServer = "mailsrvr.myisp.net";
```

Khi tất cả các giá trị đã được thiết lập, tất cả các thư gởi đi sẽ được chuyển tiếp qua mail server này. Tất nhiên, chúng ta phải đảm bảo rằng server này cho phép chúng ta chuyển tiếp mail qua nó.

VIII.2.4.1. Sử dụng lớp SmtpMail

Chương trình MailTest sau đây sẽ minh họa cách tạo một email đơn giản và gởi nó thông qua một mail rely đến người nhận:

TÀI LIỆU THAM KHẢO

[1] **C Sharp Network Programming,** Sybex
[2] **Microsoft Network Programming For The Microsoft Dot Net Framework,** Microsoft Press
[3] **Network programming in dot NET C Sharp and Visual Basic dot NET,** Digital Press

MỤC LỤC

PHẦN 1. GIỚI THIỆU SƠ LƯỢC VỀ NGÔN NGỮ HTML

1. Một số khái niệm cơ bản

1.1. World Wide Web

World Wide Web còn viết tắt là WWW, là một dịch vụ Internet, được xây dựng trên giao thức HTTP (HyperText Transfer Protocol), nó cho phép cấu hình các máy tính server theo một cách đặc biệt để có thể phân phối các bản tin trên mạng theo một cách thức chuẩn.

1.2. HTML là gì?

HTML viết tắt của HyperText Mark-up Language (ngôn ngữ đánh dấu siêu văn bản).

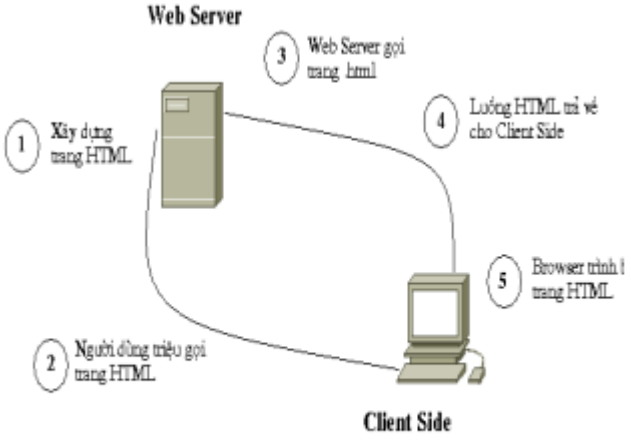
- Là một tập hợp các quy tắc và các thẻ (tag) được sử dụng để quy định cách thức trình bày, hiển thị nội dung của các trang Web, tập hợp các quy tắc và thẻ này phải tuân theo một chuẩn quốc tế, đảm bảo cho các trình duyệt Web khác nhau, trên các nền phần cứng và hệ điều hành khác nhau đều hiểu được và hiển thị đúng nội dung của các trang Web.

- **HTML không phải là một ngôn ngữ lập trình, nó là một ngôn ngữ đánh dấu.**
- **HTML dễ hiểu hơn nhiều so với hầu hết các ngôn ngữ lập trình.**

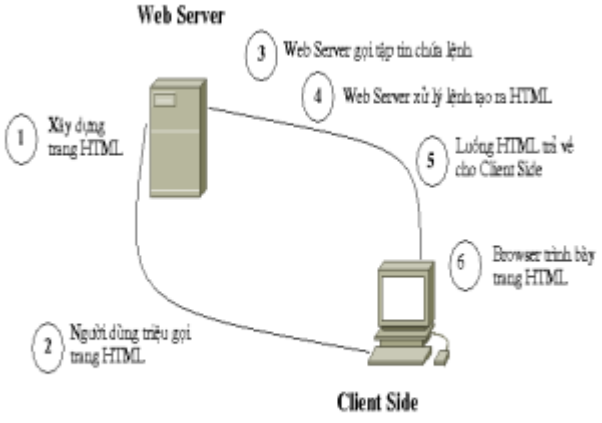
- **Một tài liệu HTML là một tệp tin văn bản trong đó có sử dụng các thẻ HTML để quy định cách thức hiển thị văn bản khi nó được mở bởi một trình duyệt Web.**
- **Thẻ định dạng trong HTML có 2 loại**
  - **Loại có thẻ mở kèm thẻ đóng. Ví dụ thẻ mở <tag> và thẻ đóng </tag>. Các văn bản nằm giữa hai thẻ này sẽ được chịu tác động định dạng bởi thẻ.**
  - **Loại chỉ có thẻ mở không có thẻ đóng . Ví dụ thẻ <BGSOUND>**

1.3. Web động – Web tĩnh

Trong thực tế, ứng dụng Web luôn tồn tại hai loại là trang Web tĩnh và Web động. Trang Web tĩnh là trang HTML không kết nối cơ sở dữ liệu. Ngược lại, trang Web động là trang Web có kết nối cơ sở dữ liệu. Điều này có nghĩa là mỗi khi trang Web động được làm tươi dữ liệu trình bày trên trang Web được đọc từ CSDL.



Quy trình triệu gọi trang Web tĩnh



Quy trình triệu gọi trang Web động

Nói cách khác, cho dù đó là trang Web tĩnh hay động, nếu ta muốn người dùng sử dụng chúng để trình bày dữ liệu trên trình duyệt Web, ta cần phải khai báo các thẻ HTML bên trong theo một quy định nhất định.

1.4. Trình duyệt Web

Trình duyệt Web (Web browser) là những chương trình có thể thực hiện một số chức năng cơ bản như tìm kiếm và hiển thị các trang Web theo đúng các định dạng bởi các thẻ HTML trong đó.

Để trang Web trình bày dữ liệu theo như ý trên trình duyệt, ta cần phai khai báo các thẻ HTML và các Client Script phù hợp với chuẩn HTML và Client Script. Ngoài ra, mỗi trình duyệt có thể hỗ trợ thêm những thẻ khác nhằm cho phép người dùng phong phú hoá giao diện của trang Web.

Hai trình duyệt phổ biến hiện nay là IE (Internet Explorer của hãng Microsoft) và Mozilla FireFox. Cả hai trình duyệt này đều cho phép ta duyệt các loại trang Web được xây dựng trên ngôn ngữ lập trình bất kỳ có thể hỗ trợ trang Web.

Tuy nhiên, đối với trình duyệt IE thì dễ dàng hơn, bởi trình duyệt này hỗ trợ hầu hết các thẻ HTML hiện nay và bỏ qua các khai báo mang tính chưa đầy đủ.

Để xây dựng một ứng dụng Web hoàn chỉnh và có tính thương mại ta cần phải kết hợp cả Client Script (kịch bản trên trình khác) và Server Script (kịch bản trên trình chủ) với một

loại CSDL nào đó, chẳng hạn như MS Access, SQL Server, MySQL, Oracle...

Khi ta muốn triển khai ứng dụng Web trên mạng *Intranet* hay *Internet* ngoài điều kiện về phần cứng hệ điều hành, ta phải có trình chủ Web thường gọi là Web Server.

Trong môi trường Windows, Web Server thường được sử dụng là IIS (Internet Information Server). IIS sử dụng cho các Server Script như: ASP (Active Server Page), JSP (Java Server Page), PHP, Servlet, Perl, ASP.NET.

Trong môi trường Linux, Web Server thường sử dụng bao gồm Apache, JRun, Web logic,…

Tóm lại, cho dù ta sử dụng bất kỳ Server Script với Web Server, thì Client Script không phụ thuộc vào chúng. Nghĩa là ta có thể sử dụng một trong hai loại Client Script ở trên là VBScript và JavaScript đều có thể chấp nhận được.

1.5. URL (Uniform Resource Locators)

Mỗi trang trên Web đều có một địa chỉ đặc biệt và duy nhất. Các địa chỉ đó gọi là URLs. URLs có dạng như sau:

Protocol://internet_address

***Ví dụ*: http://www.microsoft.com/windows/index.html**

Đường dẫn trong ví dụ trên cho biết đang xem một tài liệu có tên index.html trong thư mục Window trên webserver có địa chỉ là www.microsoft.com

Ưu điểm khi dùng URL:

- **Chỉ ra một cách tường minh loại dịch vụ Internet sẽ dùng.**

- **Hệ thống địa chỉ URL gán cho mỗi tài liệu, chương trình, và tập tin trên Internet một địa chỉ riêng biệt.**

  Các giao thức khác nhau dùng trong URL:http, file, ftp,…

2. Các thẻ định dạng cấu trúc tài liệu

Các thẻ xác định cấu trúc tài liệu là bắt buộc phải có trong một tài liệu HTML. Sau đây chúng ta sẽ lần lượt học cách sử dụng các thẻ định dạng cấu trúc của một tài liệu HTML cơ bản.

2.1. HTML

Cặp thẻ này được sử dụng để xác nhận một tài liệu là tài liệu HTML, tức là nó có sử dụng các thẻ HTML để trình bày. Toàn bộ nội dung của tài liệu được đặt giữa cặp thẻ này. Tất cả các tập tin HTML đều bắt đầu bằng thẻ <HTML>, thẻ này thông báo cho trình duyệt Web biết rằng nó đang đọc một tài liệu có chứa các mã HTML và cuối các tập tin HTML sẽ là thẻ đóng tương ứng </HTML> thông báo kết thúc tài liệu.

Cú pháp:

<HTML>

... Toàn bộ nội của tài liệu được đặt ở đây

</HTML>

Browser sẽ xem các tài liệu không sử dụng thẻ <HTML> như những tệp tin văn bản bình thường.

2.2. HEAD

Một tài liệu HTML gồm có 2 phần: phần mở đầu và phần nội dung chính.

Phần mở đầu giống như phần giới thiệu, các trình duyệt Web sử dụng phần mở đầu này lấy các thông tin khác nhau về tài liệu HTML này, chẳng hạn như tiêu đề của tài liệu, các quan hệ được thiết lập giữa tài liệu và các thư mục. Thẻ HEAD được dùng để xác định phần mở đầu cho tài liệu.

Cú pháp:

<HEAD>

... Phần mở đầu (header) của tài liệu được đặt ở đây

</HEAD>

2.3. TITLE

Ta có thể đặt tiêu đề cho tài liệu HTML của mình. Tiêu đề này sẽ được hiển thị trên thanh tiêu đề của trình duyệt. Cặp thẻ này chỉ có thể sử dụng trong phần mở đầu của tài liệu, tức là nó phải nằm trong thẻ phạm vi giới hạn bởi cặp thẻ <HEAD>.

Cú pháp:

<TITLE> ***Tiêu đề của tài liệu*** **</TITLE>**

2.4. BODY

Thẻ này được sử dụng để xác định phần nội dung chính của tài liệu. Cũng có thể sử dụng các tham số của thẻ để đặt ảnh

nền cho tài liệu, màu nền, màu văn bản siêu liên kết, đặt lề cho trang tài liệu.

Cú pháp:

<BODY>

.... phần nội dung của tài liệu được đặt ở đây

</BODY>

Trên đây là cú pháp cơ bản của thẻ BODY, các tham số sử dụng trong thẻ BODY sẽ được trình bày sau.

Như vậy một tài liệu HTML cơ bản có cấu trúc như sau:

<HTML>

<HEAD>

<TITLE> ***Tiêu đề của tài liệu*** **</TITLE>**

</HEAD>

<BODY>

Nội dung của tài liệu

</BODY>

</HTML>

2.5. Chú thích <!-- -->

Cặp thẻ này cho phép người biên soạn tài liệu HTML có thể thêm vào trong các tài liệu HTML những ghi chú cần thiết. Các văn bản được đặt giữa hai thẻ này sẽ không được trình duyệt hiển thị.

Cú pháp:

<!-- **Nội dung chú thích** **-->**

3. Các thẻ định dạng thông dụng

3.1. Các thẻ định dạng ký tự

Đây là các thẻ được sử dụng để quy định các thuộc tính như in nghiêng, in đậm, gạch chân... cho các ký tự, văn bản khi được thể hiện trên trình duyệt.

<B> ... </B> <STRONG> ... /STRONG>	**In chữ đậm**
<I> ... </I> <EM> ... /EM>	**In chữ nghiêng**
<U> ... </U>	**In chữ gạch chân**
<S> ... </S> <STRIKE> ... </STRIKE>	**In chữ bị gạch ngang**
<BIG> ... </BIG>	**In chữ lớn hơn bình th-ường**
<SMALL> ... </SMALL>	**In chữ nhỏ hơn bình th-ường**
^{...}	**Định dạng chỉ số trên (SuperScript)**
_{...}	**Định dạng chỉ số dới (SubScript)**

3.2. Các thẻ định dạng đoạn văn bản

3.2.1. Thẻ ..

Cho phép định dạng một đoạn văn bản bằng các thuộc tính của chúng.

Cú pháp:

<DIV ALIGN=LEFT|RIGHT|JUSTIPY|CENTER>

</DIV>

Ví dụ 3.2.1. **Căn lề đoạn văn bản bằng thẻ **

```
<html>
 <head>
    <title> Canle </title>
 </head>
 <body>
    <div align="justify">An explanation of exactly what JavaScript is has to begin with Java. Java is a new kind of Web programming language developed by Sun Microsystems. A Java program, or applet, can be loaded by an HTML page and executed by the Java Interpreter, which is embedded into the browser.
    </div>
 </body>
</html>
```

3.2.2. Thẻ ..

Có tác dụng tương tự như thẻ * *, trong trường hợp ta muốn định dạng một câu văn, ta có thể sử dụng thẻ * *. Điều này có nghĩa là sau khi kết thúc thẻ đóng * *, dữ liệu trình bày sẽ tự động xuống dòng.

Cú pháp:

<P ALIGN=LEFT|RIGHT|JUSTIPY|CENTER>

</P>

Ví dụ 3.2.2. **Căn lề đoạn văn bản bằng thẻ <p>**

```
<html>
 <head>
    <title> Canle </title>
 </head>
 <body>
    <p align="justify">An explanation of exactly what
JavaScript is has to begin with Java. Java is a new kind of Web
programming language developed by Sun Microsystems. </p>
<p align="right"> A Java program, or applet, can be loaded by
an HTML page and executed by the Java Interpreter, which is
embedded into the browser.</p>
 </body>
</html>
```

Chú ý: **Khác với thẻ <*div*> ngay sau thẻ <*p*> thứ nhất là thẻ <*p*> thứ hai không có dấu xuống hàng, đoạn văn thứ hai xuống dòng và cách đoạn văn thứ nhất một khoảng cách nhất định.**

3.2.3. Thẻ ..

Thẻ <*font*> cho phép định dạng một chuỗi ký tự với một kiểu chữ nhất định, cỡ chữ và màu chữ được định nghĩa trong thẻ <*font*>.

Cú pháp:

```
<FONT FACE=tên font chữ
      COLOR=tên màu|mã màu
      SIZE=n>
</FONT>
```

Trong đó

FACE	**Tên font chữ được sử dụng**
COLOR	**Màu, có thể là tên màu như RED, BLUE… hoặc mã màu trong hệ 16 như #66CCFF**
SIZE	**Kích cỡ của font chữ**

Ví dụ 3.2.3. **Sử dụng thẻ font**

```
<html>
  <head>
     <title> The font </title>
  </head>
  <body>
     <p align="justify"> An explanation of exactly what <font face="arial" color="red" size="3">JavaScript</font> is has to begin with Java.</p>
  </body>
</html>
```

Ngoài ra còn có một số thẻ định dạng khác như

** **	**Thẻ xuống dòng**
** **	**Thẻ phân đoạn**
** **	**Thẻ phân đoạn có Bullet ở trước**

3.2. Khái niệm văn bản siêu liên kết

Văn bản siêu liên kết hay còn gọi là siêu văn bản là một từ, một cụm từ hay một câu trên trang Web được dùng để liên kết tới một trang Web khác. Khi chúng ta nhấn chuột lên một siêu

văn bản, lập tức trình duyệt Web sẽ cho mở trang Web được liên kết với siêu văn bản đó. Thông thường các siêu văn bản sẽ có màu sắc phân biệt với các văn bản bình thường (chúng ta có thể chỉ định màu sắc này) và được gạch chân, khi con chuột di chuyển trên một siêu văn bản nó sẽ có hình dạng một bàn tay.

Để tạo ra một siêu văn bản chúng ta sử dụng thẻ <A>.

Cú pháp:

<A

HREF = *url*

NAME = *name*

TABINDEX = *n*

TITLE = *title*

TARGET = *_blank* / *_self* **>**

... siêu văn bản

</A>

ý nghĩa các tham số:

HREF	**Địa chỉ của trang Web được liên kết.**
NAME	**Đặt tên cho vị trí đặt thẻ.**
TABLEINDEX	**Thứ tự di chuyển khi ấn phím Tab**
TITLE	**Văn bản hiển thị khi di chuột trên siêu liên kết.**
TARGET	**Mở trang Web được liên trong một cửa sổ mới (*_blank*) hoặc trong cửa sổ hiện tại (*_self*), trong một frame (tên frame).**

Ví dụ 3.2.1:

```
<HTML>
<HEAD>
    <TITLE> Su dung sieu van ban </TITLE>
</HEAD>
<BODY>
    <FONT FACE="Time New Roman" SIZE=4>
    <A HREF="mautu3.html" TITLE="Kích chuột vào đây để mở trang 'Chọn phông chữ' ">
        Đây là siêu văn bản
    </A>
    </FONT>
</BODY>
</HTML>
```

4 . Các thẻ chèn âm thanh, hình ảnh

4.1. Đa âm thanh vào một tài liệu HTML

Ta có thể đa âm thanh vào trong tài liệu HTML của mình bằng cách sử dụng thẻ BGSOUND, thuộc tính SRC chỉ định địa chỉ của file âm thanh (.wav, .au hoặc MIDI file) và thuộc tính LOOP để chỉ định số lần chơi.

Cú pháp:

```
<BGSOUND
    SRC = url
    LOOP = n>
```

Thẻ này không có thẻ kết thúc tương ứng (</BGSOUND>). Để chơi lặp lại vô hạn cần chỉ định LOOP = -1 hoặc LOOP = *INFINITE*. Thẻ BGSOUND phải được đặt trong phần mở đầu (tức là nằm trong cặp thẻ HEAD).

Ví dụ 4.1.1:

```
<HTML>
<HEAD>
    <TITLE>Đa âm thanh vào trong tài liệu</TITLE>
    <BGSOUND SRC="thunghiem.wma" LOOP=2>
</HEAD>
<BODY>
</BODY>
</HTML>
```

4.2. Chèn một hình ảnh, một đoạn video vào tài liệu HTML

Để chèn một file ảnh (.jpg, .gif, .bmp) hoặc video (.mpg, .avi) vào tài liệu HTML, ta có thể sử dụng thẻ IMG.

Cú pháp:

```
<IMG
    ALIGN = TOP/MIDDLE/BOTTOM
    ALT = text
    BORDER = n
    SRC = url
    WIDTH = width
    HEIGHT = height
```

HSPACE = *vspace*

VSPACE = *hspace*

TITLE = *title*

DYNSRC = *url*

START = *FILEOPEN/MOUSEOVER*

LOOP = *n>*

Trong đó

ALIGN = *TOP/MIDDLE/BOTTOM*	**Căn hàng văn bản bao quanh ảnh**
ALT = *text*	**Chỉ định văn bản sẽ được hiển thị nếu chức năng *show picture* của *browser* bị tắt đi**
BORDER = *n*	**Đặt kích thước đường viền được vẽ quanh ảnh**
SRC = *url*	**Địa chỉ của file ảnh cần chèn vào tài liệu**
WIDTH/HEIGHT	**Chỉ định kích thước của ảnh được hiển thị**
HSPACE/VSPACE	**Chỉ định khoảng trống xung quanh hình ảnh**
TITLE = *title*	**Văn bản sẽ hiển thị khi con chuột trỏ trên ảnh**
DYNSRC = *url*	**Địa chỉ của file video**
START = FILEOPEN/MOUSEOVER	**Chỉ định file video sẽ được chơi khi tài liệu được mở hay khi trỏ**

	con chuột vào nó. Có thể kết hợp cả hai giá trị này nhưng phải phân cách chúng bởi dấu phẩy.

Ví dụ 4.2.1:

```
<html>
<head>
    <title>Da am thanh trong trang web</title>
    <BGSOUND SRC="thunghiem.wma" LOOP=1>
</head>
<body>
    <img border="5" src="dscf0032.jpg" width="327"
        height="250" title="Đây là hai chú mèo">
</body>
</html>
```

5. Chèn bảng

Để tạo bảng ta sử dụng các thẻ sau

<TABLE> ... </TABLE>	**Định nghĩa một bảng**
<TR> ... </TR>	**Định nghĩa một hàng trong bảng**
<TD> ... </TD>	**Định nghĩa một ô trong hàng**
<TH> ... </TH>	**Định nghĩa ô chứa tiêu đề của cột**
<CAPTION> ... </CAPTION>	**Tiêu đề của bảng**

Ví dụ 5.1. Tạo bảng như sau

Ví dụ 1.6: Tạo bảng nh sau:

MaNV	Hoten	So DT	
		Ma vung	So
A0001	Nguyen Van A	04	7543611
B0002	Tran Van B	0211	853540

```
<html>
<head>
<title>Chen bang</title>
</head>
<body>
<TABLE BORDER="1" CELLSPACING=1 CELLPADDING=9 WIDTH=453>
        <TR><TD WIDTH="66" VALIGN="center" ROWSPAN=2> <p
align="center"> MaNV</p></TD>
            <TD WIDTH="154" VALIGN="center" ROWSPAN=2> <P
ALIGN="center">Hoten</TD>
            <TD WIDTH="133" VALIGN="TOP" COLSPAN=2> <p align="center">
 So DT</p></TD>
        </TR>
        <TR><TD WIDTH="104" VALIGN="TOP"> <p align="center">Ma
vung</p></TD>
            <TD WIDTH="59" VALIGN="TOP">  <p align="center">So</p></TD>
        </TR>
      <TR><TD WIDTH="66" VALIGN="TOP"> <p align="center">
A0001</p></TD>
       <TD WIDTH="154" VALIGN="TOP"> <P ALIGN="JUSTIFY">Nguyen
Van A</TD>
       <TD WIDTH="104" VALIGN="TOP"> <p align="right"> 04</p></TD>
       <TD WIDTH="59" VALIGN="TOP">  <p
align="right">7543611</p></TD>
```

```
    </TR>
    <TR><TD WIDTH="66" VALIGN="TOP"> <p align="center">
B0002</p></TD>
      <TD WIDTH="154" VALIGN="TOP"> <P ALIGN="JUSTIFY">Tran Van
B</TD>
      <TD WIDTH="104" VALIGN="TOP"> <p align="right">0211</p></TD>
      <TD WIDTH="59" VALIGN="TOP"> <p align="right">853540</p></TD>
    </TR>
  </TABLE>
</body>
</html>
```

6. Sử dụng Khung - Frame

Các *frame* cho phép ta tổ chức cấu trúc nội dung các trang Web của mình bằng cách phức hợp nhiều tài liệu HTML để có thể xem chúng trong cùng một cửa sổ chính của trình duyệt Web. Để tạo một trang Web phức hợp ta sử dụng các thẻ FRAMESET và FRAME để chia cửa sổ chính thành các khung chữ nhật (*frame*). Sau đó trong mỗi khung hình chữ nhật đó ta chỉ định một tài liệu sẽ được hiển thị trong khung đó. Chú ý thẻ FRAMESET sẽ thay thế cho thẻ BODY trong một tài liệu HTML, điều đó có nghĩa là trong một tài liệu sử dụng thẻ FRAMESET sẽ không có thẻ BODY mà thay vào đó phần nội dung chính của tài liệu sẽ được định nghĩa bởi thẻ FRAMESET.

6.1. Thẻ FRAMESET

Cú pháp:

```
<FRAMESET
    COLS = col-widths
```

ROWS = ***row-heights***

BORDER = **pixels**

BORDERCOLOR = **color**

FRAMEBORDER = **1/0 >**

...

<FRAME ... >

<FRAME ... >

...

</FRAMESET>

COLS	**Chia dọc cửa sổ thành các phần với kích thước chỉ định (theo pixel, % hoặc *).**
ROWS	**Chia ngang cửa sổ thành các phần với kích thước chỉ định (theo pixel, % hoặc *).**
BORDER	**Kích thước của đường kẻ viền khung**
BORDERCOLOR	**Chỉ định màu cho đường viền khung**
FRAMEBORDER	**Chỉ định có/không (1/0) hiển thị khung của các *frame*.**

6.2. Thẻ FRAME

***Cú pháp*:**

<FRAME

BORDERCOLOR = **color**

FRAMEBORDER = **0/1**

MARGINHEIGHT = ***height***

MARGINWIDTH = ***width***

NAME = ***name***

NORESIZE

SCROLLING = **YES/NO**

SRC = ***address***

Target = ***Window_Name>***

BORDERCOLOR	**Màu đường viền khung.**
FRAMEBORDER = 0 / 1	**Chỉ định có/không viền khung.**
MARGINHEIGHT	**Khoảng cách giữa nội dung trong khung và đường viền ngang.**
MARGINWIDTH	**Khoảng cách giữa nội dung trong khung và đường viền dọc.**
NAME	**Đặt tên cho khung.**
NORESIZE	**Chỉ định không được thay đổi kích thước của khung.**
SCROLLING = YES / NO	**Chỉ định có hay không có thanh cuộn cho khung.**
SRC	**Địa chỉ của tài liệu sẽ được hiển thị trong khung.**
Target	**Chỉ ra cửa sổ nơi mà tài liệu được hiển thị**

Ví dụ 6.1:

Đây là siêu liên kết

MaNV	Hoten	So DT	
		Ma vung	So
A0001	Nguyen Van A	04	7543611
B0002	Tran Van B	0211	853540

```
<html>
<head>
<title>Frames</title>
</head>

<frameset rows="20%,*" bordercolor="red">
    <frame src="sieulienket.htm">
    <frameset cols="30%,*" frameborder="1" bordercolor="green">
        <frame src="daamthanh.htm">
        <frame src="bang.htm">
    </frameset>
</frameset>
</html>
```

6.3. Thẻ IFRAME

Sử dụng thẻ IFRAME để đặt một frame vào trong một tài liệu HTML.

Cú pháp:

```
<IFRAME
        ALIGN = LEFT / CENTER /RIGHT / TOP / BOTTOM
        BORDER = pixels
```

BORDERCOLOR = color

FRAMEBORDER = 0/1

NORESIZE

SCROLLING = YES/NO

NAME = name

SRC = url

MARGINWIDTH = pixels

MARGINHEIGHT = pixels

WIDTH = n

HEIGHT = n >

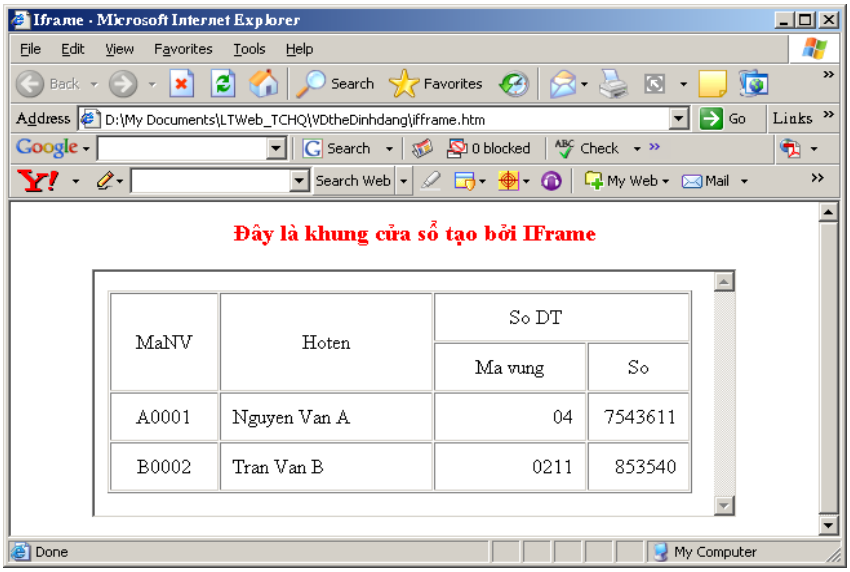
</IFRAME>

ý nghĩa các thuộc tính:

ALIGN	**Căn lề cho khung**
BORDER	**Kích thước đường viền của khung**
BORDERCOLOR	**Màu đường viền của khung**
FRAMEBORDER = 0 / 1	**Chỉ định khung có đường viền hay không**
NORESIZE	**Chỉ định không được phép thay đổi kích thước của khung.**
SCROLLING = YES / NO	**Chỉ định khung có thanh cuộn hay không**
NAME	**Đặt tên cho khung**
SRC	**Địa chỉ tài liệu được hiển thị trong khung**
MARGINWIDTH	**Khoảng cách giữa văn bản nội dung của khung**

	và các đường viền dọc.
MARGINHEIGHT	Khoảng cách giữa văn bản nội dung của khung và các đường viền ngang.
WIDTH	Đặt chiều rộng của khung
HEIGHT	Đặt chiều cao của khung

Ví dụ 6.2:




```
<html>
<head>
<title>Iframe</title>
</head>

<body>
<p align="center">
<font size="4" face="Times New Roman" color="red">
<b>Đây là khung cửa sổ được tạo bởi IFrame</b>
</font></p>
<Iframe Align="center" scrolling="yes" src="bang.htm" width=500
height=200>
</Iframe>
</body>

</html>
```

7. FORMS

7.1. HTML Forms

Người biên soạn HTML có thể tạo ra các HTML Form để tương tác với những người đọc tài liệu của họ chẳng hạn như cho phép người đọc nhập vào dữ liệu để chạy một chương trình CGI, ghi vào các nhận xét về trang Web đó... Các HTML Form có thể có các hộp văn bản, hộp danh sách lựa chọn, nút bấm, nút chọn...

Khi muốn submit dữ liệu từ người dùng nhập từ trang Web phía *Client* lên phía *Server,* có hai cách để làm điều này ứng với hai phương thức *Post* và *Get* trong thẻ *Form*. Như vậy bằng cách khai báo phương thức *Post* hay *Get* trong thẻ *Form*, cho phép *Server Script* lấy giá trị từ các thẻ *Input*.

Để thực hiện quá trình sử dụng phương thức yêu cầu dữ liệu từ phía *Client,* ta cần khai báo thẻ *Form* bao *(bound*) hết các thẻ nhập liệu cần đưa lên phía *Server*.

7.2. Tạo Form

Để tạo ra một form trong tài liệu HTML, chúng ta sử dụng thẻ FORM với cú pháp như sau:

Cú pháp:

<FORM

ACTION = ulr

METHOD = GET | POST

NAME = name

TARGET = frame_name | _blank | _self>

<!-- Các phần tử điều khiển của form được đặt ở đây -->

</FORM>

Trong đó:

ACTION	**Địa chỉ sẽ gửi dữ liệu tới khi form được submit (có thể là địa chỉ tới một chương trình CGI, một trang ASP...)**
METHOD	**Phương thức gửi dữ liệu**
NAME	**Tên của form**
TARGET	**Chỉ định cửa sổ sẽ hiển thị kết quả sau khi gửi dữ liệu từ form đến server.**

Đặt các đối tượng điều khiển (như hộp văn bản, ô kiểm tra, nút bấm...) vào trang Web

Chú ý:

- **Trong một trang Web có nhiều thẻ Form khác nhau, nhưng các thẻ form này không được lồng nhau, mỗi thẻ form sẽ được khai báo một hành động *action* để chỉ đến một trang Web khác.**
- **Nếu không chỉ định địa chỉ *URL* hay *UNC* cho action, trang Web được chuyển đến chính là trang Web hiện tại.**

7.3. Thẻ INPUT

Thẻ *Input* cho phép người dùng nhập liệu hay chỉ thị thực thi một hành động nào đó, thẻ Input bao gồm các loại thẻ như: *text, password, submit, button, reset, checkbox, radio, hidden, image.*

7.4.1. Thẻ Input dạng text

Đối với thẻ input dạng text cần cung cấp tên để tham chiếu trên Client và trên Server side.

Cú pháp

<INPUT

ALIGN = LEFT | CENTER | RIGHT

TYPE = "Text"

NAME = name

SIZE = n

MAXLENGHT = n

VALUE = value>

Trong đó:

ALIGN	**Căn chỉnh cho thẻ**
NAME	**Tên để tham chiếu trên Client và trên Server side**
SIZE	**Độ dài của textbox trình bày trên trang Web**
MAXLENGHT	**Khống chế chiều dài chuỗi nhập trên thẻ Input**

VALUE	Giá trị mặc định ban đầu

Ví dụ 7.1:

```
<input type="text" name="txtUsername" size=20
maxlenght=50>
<input type="text" name="txtFullname" value="Nguyen Hoai
Anh" size=20 maxlenght=50>
```

7.4.2. Thẻ Input dạng password

Cũng giống như thẻ Input dạng text, thẻ input dạng password cho phép người dùng nhập liệu dạng mật khẩu, tức là khi người dùng nhập liệu tất cả các ký tự chuyển thành dấu * hay dấu ã phụ thuộc vào phiên bản của trình duyệt.

Cú pháp

```
<INPUT
    ALIGN = LEFT | CENTER | RIGHT
    TYPE = “password”
    NAME = name
    SIZE = n
    MAXLENGHT = n>
```

Trong đó: Các tham số tương tự thẻ Input dạng Text

Ví dụ 7.2:

```
<input type="password" name="txtPassword" size=20
maxlenght=10>
```

7.4.3. Thẻ Input dạng checkbox

Khi muốn cung cấp nhiều tuỳ chọn cho người dùng lựa chọn, ta phải sử dụng thẻ input dạng *checkbox*.

- Trên trình khách, ta chỉ quan tâm đến trạng thái của thẻ *checkbox* bằng thuộc tính *checked*. Khi sử dụng *Client script*, nếu chọn thẻ này có giá trị *true* ngược lại có giá trị *false*.

- Khi sử dụng *Server Script* để tham chiếu đến giá trị này, tức là ta tham chiếu đến giá trị thẻ: Điều này có nghĩa là khi tham chiếu đến thẻ *input* dạng *checkbox* ta cần định nghĩa trước giá trị của nó. Nếu thẻ này được chọn, *Server Script* sẽ tham chiếu và nhận được giá trị đó.

Cú pháp

<INPUT

ALIGN = LEFT | CENTER | RIGHT

TYPE = "checkbox"

VALUE = value

NAME = name

CHECKED > Tên lựa chọn

Trong đó: **Các tham số tương tự thẻ Input dạng Text, ngoài ra**

VALUE	**Giá trị nhận được nếu chọn thẻ**
CHECKED	**Khai báo thành phần này nếu muốc mặc định trên trang Web giá trị được chọn**

Ví dụ 7.3:

```
<input type="checkbox" value="P" name="chkHow"
checked>Newspaper
<input type="checkbox" value="T" name="chkTV"
checked>Television
<input type="checkbox" value="B" name="chkB">Bus
```

7.4.4. Thẻ Input dạng radio

Thẻ input dạng *Radio* tương tự như thẻ *Input* dạng *checkbox*, nhưng cho phép người dùng chỉ chọn một trong các tuỳ chọn cho phép. Có nghĩa là nếu ta cung cấp hai tuỳ chọn ứng với hai thẻ *input* dạng *Radio* thì người dùng chỉ được chọn duy nhất một tuỳ chọn.

Cú pháp

<INPUT

ALIGN = LEFT | CENTER | RIGHT

TYPE = "radio"

VALUE = value

NAME = name

CHECKED > Tên lựa chọn

Trong đó: **Các tham số tương tự thẻ Input dạng checkbox**

Ví dụ 7.4:

```
<input type="radio" value="M" name="rdGender"
checked>Male
<input type="radio" value="F" name="rdGender">Female
```

Để cho phép chọn một trong hai tuỳ chọn đó, ta phải khai báo chúng cùng tên nhng khác nhau về giá trị. Chính vì vậy, khi tham chiếu đến thẻ này bằng *Client Script,* ta phải duyệt theo dạng mảng để biết tuỳ chọn nào đợc chọn. Còn nếu tham chiếu bằng *Server Script,* ta sẽ nhận đợc một giá trị của tuỳ chọn đợc chọn.

Chú ý: **Nếu thẻ input dạng *checkbox* khai báo cùng tên, ta cũng sẽ nhận đợc một chuỗi giá trị mà các phần tử đợc cách nhau bởi dấu phảy ","**

7.4.5. Thẻ Input dạng submit

Thẻ *submit* cho phép ta chấp nhận những dữ liệu nhập trong các thẻ *input* phía trình chủ *Web*. Có nghĩa là khi ta muốn chuyển dữ liệu lên phía trình chủ bằng phương thức *Post* (sử dụng cho thẻ *form*) hay *Get* (trên *QueryString*).

Để truyền các dữ liệu nhập từ các thẻ *input* trong *form*, ta có thể cho phép *Server Script* nhận giá trị đó bằng cách nhận từ *QueryString* hay từ *Form.*

Nếu muốn chuyển các giá trị nhập đó lên trên *QueryString*, ta khai báo thẻ *Form* với phương thức là *Get*. Ngược lại, nếu muốn *Server Script* lấy dữ liệu từ thẻ *Form*, ta khai báo thẻ *form* với phương thức *Post.*

Thẻ *Submit* thường được khai báo trong thẻ *form*, dùng để *Submit* dữ liệu lên trên phía trình chủ Web. Ngoài ra khi khai báo thẻ *Submit* ta cũng cần cung cấp tên và *value*, *value* của thẻ *input* dạng *Submit* xuất hiện như *Caption* trên nút.

Cú pháp

```
<INPUT
    ALIGN = LEFT | CENTER | RIGHT
    TYPE = “submit”
    VALUE = value
    NAME = name>
```

Ví dụ 7.5:

```
<input type="submit" value="Save" name="submit">
```

7.4.6. Thẻ Input dạng Button

Thẻ button cho phép ta chấp nhận những dữ liệu đã nhập trong các thẻ *input* lên phía trình khách (nếu khai báo sử dụng phương thức *submit()*, nút này sẽ *submit* thẻ *form* lên trình chủ *Web*). Có nghĩa là khi ta muốn tính toán trên trang *Web* ta có thể sử dụng thẻ này.

Thẻ input thường được khai báo trong thẻ form, dùng để tính toán hay sử dụng một số phương thức của Client Script. Ví dụ ta thiết kế nút "Back" với chức năng trở về trang Web trước đó hay tính toán trên trang Web.

Cú pháp

```
<INPUT
    ALIGN = LEFT | CENTER | RIGHT
    TYPE = “button”
    VALUE = value
    NAME = name
    ONCLICK= “công việc cần làm” >
```

Ví dụ 7.6:

Khai báo thẻ input dạng button với chức năng thực thi phương thức Client Script trở về trang Web trước đó như sau:

```
<input type="button" value="Back" name="back"
                                onclick="window.history.go(-1);">
```

7.4.7. Thẻ Input dạng Reset

Thẻ *Reset* cho phép phục hồi những dữ liệu nhập hay chọn trên các thẻ *Input* và thẻ *select, textarea*. Điều này có nghĩa là khi muốn phục hồi dữ liệu trang *Web* về trạng thái ban đầu, người dùng sẽ chọn nút *reset* như chỉ thị phục hồi những gì họ đã thực hiện.

Thẻ *reset* thường được khái báo trong thẻ *form*, dùng để *reset* dữ liệu trên trang *Web*.

Cú pháp

<INPUT

ALIGN = LEFT | CENTER | RIGHT

TYPE = "reset"

VALUE = value

NAME = name >

Ví dụ 7.7:

```
<input type="reset" value="reset" name="back">
```

7.4.8. Thẻ Input dạng hidden

Thẻ input dạng *hidden* là một dạng thẻ *input* dạng *text* nhưng không hiển thị trên trang Web, không cho phép người dùng nhập liệu, thay vào đó ta cần định nghĩa trước một giá trị cụ thể cho thẻ này.

Cú pháp

<INPUT

ALIGN = LEFT | CENTER | RIGHT

TYPE = "hidden"

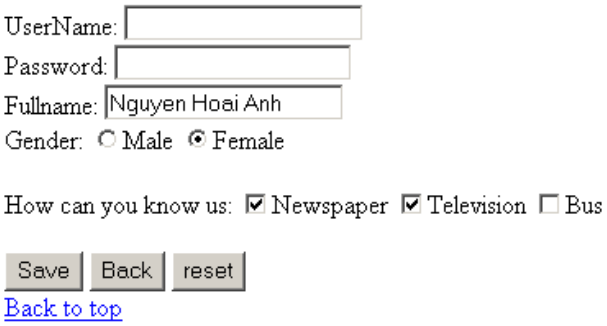
NAME = name

VALUE = value>

Ví dụ 7.8:

```
<input type="hidden" name="txtFrom" value="N" >
<input type="hidden" name="txtpage" value="1" >
```

Ví dụ 7.9: Tạo một form nhập có dạng như sau




```
<html>
<head>
<title>Form</title>
</head>

<body>
   <br><br>
   UserName: <input type="text" name="txtUsername" size=20
maxlenght=50><Br>
   Password: <input type="password" name="txtPassword" size=20
                                                maxlenght=10><br>
   Fullname: <input type="text" name="txtFullname" value="Nguyen Hoai
Anh"
                                                size=20
maxlenght=50><br>
   Gender:  <input type="radio" value="M" name="rdGender">Male
         <input type="radio" value="F" name="rdGender" checked>Female
   <br><Br>
   How can you know us:
   <input type="checkbox" value="P" name="chkHow"
checked>Newspaper
   <input type="checkbox" value="T" name="chkTV" checked>Television
   <input type="checkbox" value="B" name="chkB">Bus
   <br><Br>
```

```
    <input type="submit" value="Save" name="submit">
    <input type="button" value="Back" name="back"
onclick="window.history.go(-1);">
    <input type="reset" value="reset" name="back">
    <br>
    <a href="#2">Back to top</a>
</body>
</html>
```

7.4.9. Thẻ Input dạng Image

Ngoài việc chèn hình ảnh vào trang Web bằng thẻ <img>, ta có thể sử dụng hình ảnh thực hiện chức năng tương tự nh nút submit, reset hay button.

Cú pháp

<INPUT

ALIGN = LEFT | CENTER | RIGHT

TYPE = "image"

SRC = filename

NAME = name>

Ví dụ 7.10:

```
<input type="image" src="gogo.gif" name="search">
```

7.4. Tạo một danh sách lựa chọn bằng thẻ SELECT và OPTION

Thẻ *Select* cho phép người dùng chọn phần tử trong tập các phần tử được định nghĩa trước. Thẻ *Select* có hai dạng tương tự như *combo box* và *listbox*. Nếu thẻ *select* cho phép chọn một phần tử trong danh sách thì thẻ *select* đó giống như *combo box*, ngược lại nếu cho phép chọn nhiều phần tử trong danh sách thì nó có dạng *listbox*.

Đối với hai loại thẻ này, nếu muốn phần tử nào được chọn mặc định thì ta thêm từ khoá *selected* trong thẻ *option.*

Cú pháp:

<SELECT

NAME="*tên danh sách*"

MULTIPLE

SIZE="*chiều cao*">

<OPTION VALUE=*value1* SELECTED> ***Tên mục chọn thứ nhất***

<OPTION VALUE=*value2* > ***Tên mục chọn thứ hai***

<!-- ***Danh sách các mục chọn*** **-->**

</SELECT>

Trong đó

NAME	**Tên danh sách**
MULTIPLE	**Có tham số này là listbox, ngược lại là combobox**
SIZE	**Chiều cao của danh sách**
<OPTION VALUE="" SELECTED> </OPTION>	**Giá trị lựa chọn, có tham số selected để chỉ giá trị mặc định**

Ví dụ 7.2:


```
<html>
<head>
<title>select</title>
</head>

<body>
      Đây là Combobox
      <p>
        <select name="city" >
            <option value="" selected>..Select City ..</option>
            <option value="HCM">Ho Chi Minh</option>
            <option value="HAN">Ha Noi</option>
            <option value="HUE">Hue</option>
       </select>
      </p><p>Đây là listbox</p>
      <p>
        <select name="industry" multiple >
        <option value="" selected>..Select Industry ..</option>
            <option value="AUT">Auto</option>
            <option value="MED">Medical</option>
            <option value="ENG">Engineering</option>
```

```
      </select>
    </p>
</body>
</html>
```

7.5. Tạo hộp soạn thảo văn bản bằng thẻ TEXTAREA.

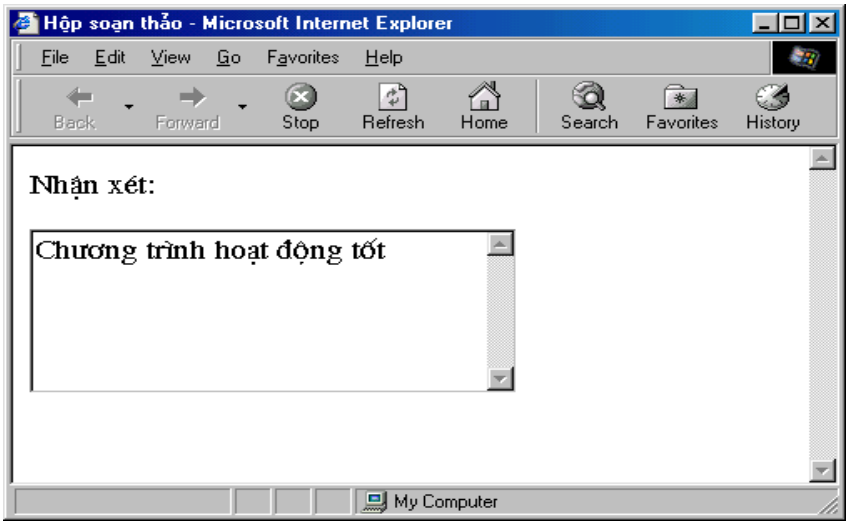
Thẻ *Textarea* cho phép người dùng nhập liệu với nhiều dòng, với thẻ này ta không thể giới hạn chiều dài lớn nhất trên trang *Web*.

Vì không khai báo được chiều dài lớn nhất cho phép, ta nên viết chương trình kiểm soát được chiều dài phù hợp với chiều dài đã khai báo trong CSDL.

Cú pháp:

```
<TEXTAREA
    COLS=số cột
    ROWS=số hàng
    NAME=tên>
    Văn bản ban đầu
</TEXTAREA>
```

Ví dụ 7.3:




```
<HTML>
<HEAD>
<TITLE>Hộp soạn thảo</TITLE>
</HEAD>
<BODY>
<P >Nhận xét:</P>
<TEXTAREA COLS="30" ROWS="5" NAME="nx"> Chương trình
hoạt động tốt
</TEXTAREA>
</BODY>
</HTML>
```

8. Một số thẻ HTML đặc biệt

Ngoài các thẻ thông dụng trên, còn một số thẻ đặc biệt như: *<Meta>, <link>, <Script>, <Style>*

8.1. Thẻ <Meta>

Thẻ <*meta*> được khai báo trong thẻ <*head*> thường dùng để khai báo loại *font* sử dụng, tìm kiếm, xoá *cache*, chuyển trang,…

8.1.1. Thẻ <meta> với font

Để sử dụng *font Unicode* đặc biệt là font *Uncode* cho tiếng Việt trên trang *Web* ta cần khai báo thẻ <*meta*> như sau trong thẻ <*head*>:

```
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
```

***Chú ý:* Về phía *Client*, nếu muốn trình bày nội dung bằng tiếng Việt và cho phép người dùng nhập dữ liệu trên các thẻ *Input* bằng tiếng Việt Unicode chuẩn UTF-8 (sử dụng bộ gõ Unicode) ta phải khai báo thẻ <*meta*> như trên.**

***Ví dụ 8.1.1.* Thẻ <meta> sử dụng font Unicode**

```
<html>
    <head>
        <title>Meta font</title>
        <meta http-equiv="Content-Type" content="text/html;
    charset=UTF-8">
    </head>
    <body>
     <img src="logo1.jpg" border=0><br>
     <form name="form1" action="doaction. asp" method="post"
    onsubmit="alert('You are about submit');">
```

```
<table><tr><td>Keyword: </td>
 <td><input type="text" name="txtKey" size=20
maxlenght=50></td></tr>
</table>
</form>
</body>
</html>
```

8.1.2. Thẻ <meta> dạng tự động chuyển đến URL

Để tự động chuyển đến địa chỉ *URL* (địa chỉ đến một trạn *Internet* hay mạng cục bộ) hay *UNC* (địa chỉ đến một tập tin trên mạng cục bộ) kế tiếp sau khi trang Web được nạp lên với thời gian nhất định ta có thể khai báo trong thẻ <*meta*>.

Ví dụ 8.1.2. **Tự động chuyển đến trang *yahoo.com* sau thời gian 8 giây.**

```
<html>
<head>
<title>Meta URL</title>
<META http-equiv=refresh content="8;URL=http://www.yahoo.com">
</head>

<body>
This page automatic refresh in 8 second ! <br>
I will be come <b>www.yahoo.com</b>
</body>
</html>
```

8.1.3. Thẻ <meta> dùng xoá cache

Thông thường sau khi nạp trang *Web* nào đó lên trình duyệt, nội dung của trang *Web* có thể được lưu vào trong bộ nhớ truy cập nhanh (*cache*). Có nghĩa là sau khi đã duyệt một vòng

các trang *Web* khác, ta có thể quay lại trang *Web* đã truy cập trước đó, trình duyệt *Web* nạp rất nhanh vì chúng đã được lưu trong bộ nhớ *cache*.

Đối với tình huống này, nếu trang *Web* đó có thay đổi về cấu trúc chẳng hạn, đôi khi vì một lý do nào đó mà ta không tìm thấy sự thay đổi. Để giải quyết việc này, trong mạng cục bộ thường sử dụng thêm *Proxy Server*, người sử dụng có thể xoá cache cho *IP* hay máy của ta. Tuy nhiên, khi phát triển ứng dụng *Web*, có những trang *Web* ta cần phải xoá *cache* mỗi khi người dùng gọi nó, khi đó ta sử dụng thẻ <*meta*> như sau:

***Ví dụ 8.1.3.* Xoá cache:**

```
<html>
<head>
<title>Xoa cache</title>
<meta http-equiv="pragma" content="no-cache">
<meta http-equiv="expires" content="0">
<meta http-equiv="cache-control" content="no-cache">
</HEAD>
 Xoa Cache
</BODY></HTML>
```

8.2. Thẻ <Marque>

Thẻ *marque* cho phép ta khai báo một chuỗi chuyển động theo nhiều hướng khác nhau trên trang Web. Có 4 chiều di chuyển chuỗi tuỳ thuộc vào tham số hướng là *UP*, *DOWN*, *BACK*, *RIGHT*.

Cú pháp:

<MARQUE

DIRECTION= Right|Left

SCROLLDELAY=*n*

STYLE=”font-family:Tên font;

font-weight:bold|regular|Italic|BoldItalic;

color:tên màu| mã màu

boder-style:kiểu boder

boder-color: tên màu|mã màu

SCROLLAMOUNT=n

WIDTH=n

HEIGHT=n

LOOP=n>

Dòng chữ cần chạy

</MARQUE>

Trong đó

DIRECTION	**Hướng chạy**
SCROLLDELAY	**Độ trễ khi chạy**
SCROLLAMOUNT	**Độ giật khi chạy**
WIDTH	**Độ dài cửa sổ khi chạy**
HEIGHT	**Độ rộng cửa sổ chạy**
LOOP	**Số lần lặp khi chạy. Mặc định là không dừng**
STYLE	**Khai báo các tham số về chữ chạy: font chữ, cỡ chữ, màu chữ, khung viền...**

Ví dụ 8.2.1. **Thẻ *marque***

```
<html>
<head>
<title>Chu chuyen dong</title>
</head>
<body>
   <Br>
  <marquee direction="right" scrolldelay="2"
   style="font-family: Times New Roman; font-weight: bold;
            color: #FF0000; border-style: ridge; border-color: #C0C0C0"
   scrollamount="1" width="447" height="31" align="middle">
   Ashley-Ana -Chanthaly -Kathleen- Lena</marquee>
  <br>
</body>
</html>
```

8.3. Thẻ <STYLE>

Thẻ <*style*> cho phép ta định danh tất cả các nội dung trình bày trên trang *Web* theo một kiểu nhất định định nghĩa trong thẻ <*style*>.

Để khai báo một thẻ <*style*> trên trang *Web*, ta khai báo nó trong thẻ <*head*>. Và khi khai báo thẻ trong trang *Web*, ta có thể sử dụng một phần tử trong trong thẻ <*Style*> bằng tham số *class*.

Ví dụ 8.3.1. **Khai báo thẻ <*style*>**

```
<html>
 <head>
    <title>  Khai bao Style </title>
  <style>
       .text_normal{
       COLOR: #6666FF;
```

```
            FONT-SIZE: 12px;
            TEXT-DECORATION: none;
            }
            .topic{
            COLOR: #FF66cc;
               TEXT-DECORATION: underline;
            }
            .bg{
            COLOR: #000000;
            }
          </style>
        </head>
    <body>
      <p align="justify" class="text_normal">An explanation of
      exactly what JavaScript is has to begin with Java.</p>
      <p align="justify" class="topic"> Typical Internet Connectivity
      Design</p>
      <p align="justify" class="bg"> With these security concerns
      now established.</p>
    </body>
</html>
```

Trong thực tế ta không cần phải khai báo thẻ <style> trên từng trang Web, thay vào đó ta khai báo một tập tin có tên mở rộng là *.css, sau đó ta khai báo để chèn tập tin này vào mỗi trang Web và sau đó sử dụng chúng như trên.

Ví dụ 8.3.2. **Trang style.css**

```
.text_normal{
  COLOR: #6666FF;
```

```
  FONT-SIZE: 12px;
  TEXT-DECORATION: none;
}
.topic{
  COLOR: #FF66cc;
TEXT-DECORATION: underline;
}
.bg{
  COLOR: #000000;
}
```

8.4. Thẻ <link>

Thẻ <*link*> dùng để link các trang. Sau khi khai báo các phần tử trong trang *style.css*, ta có thể khai báo chúng trong trang Web bằng thẻ <*link*> để sử dụng như sau:

Ví dụ 8.4.1. Thẻ <*link*>

```
<html>
  <head>
    <title> The Link  </title>
    <link rel="stylesheet" type="text/css" href="style.css">
  </head>
  <body>
    <p align="justify" class="text_normal">An explanation of
      exactly what JavaScript is has to begin with Java.</p>
    <p align="justify" class="topic"> Typical Internet
      Connectivity Design</p>
    <p align="justify" class="bg"> With these security
      concerns now established</p>
```

```
    </body>
</html>
```

Chú ý: **trang style.css phải đặt ở cùng thư mục chứa trang này.**

8.5. Thẻ <Script>

Trong trang *Web* muốn kiểm soát tất cả các hành động của người dùng, ta cần khai báo và sử dụng một số phương thức và thuộc tính của *Client Script* hay các phương thức do ta định nghĩa.

Client Script bao gồm hailoại kịch bản chính như VBScript hay JavaScript. Để khai báo kịch bản trên trang Web ta sử dụng thẻ <script> như sau:

```
<script language=Javascript>
//code here
</script>
<script language=VBscript>
//code here
<script>
```

Ngoài ra, trong trường hợp có nhiều phương thức do ta định nghĩa được sử dụng chung trong nhiều trang Web, ta cũng có thể khai báo chúng trong một tập tin có tên mở rộng *.js hoặc *.vb. Sau đó, trong trang Web, ta có thể khai báo chèn tập tin này và sử dụng như cách sử dụng của các khai báo trực tiếp.

Tóm lại, cấu trúc của một trang Web như sau

```
<html>
```

```
<head>
   <title> Welcome to MY Web </title>
   <meta ...>
   <link ...>
   <script ...>    </script>
</head>
<body>
      Nội dung chính
</body>
</html>
```

PHẦN 2. GIỚI THIỆU VỀ CÔNG NGHỆ ASP

1. Một số khái niệm cơ bản về ASP

1.1. Web động

Như đã biết ngôn ngữ đánh dấu siêu văn bản HTML là công cụ mô tả trang Web trên Internet. Khi trình duyệt yêu cầu một trang HTML, Web Server nhận yêu cầu và gửi trả lại file HTML được yêu cầu. Trình duyệt sẽ trình diễn trang HTML nhận được.

Nói chung các trang HTML là tĩnh về mặt nội dung. Mặc dù trình duyệt có thể xử lý các ngôn ngữ kịch bản như VBScript hay Jscript nếu như người ta cài đặt các máy ảo tại client để tạo ra một hiệu quả động nào đó với các tương tác hai chiều. Tuy nhiên tương tác này rất hạn chế nếu như dữ liệu cần sử dụng đặt tại server chứ không phải tại client.

Trên thực tế có nhu cầu tra cứu thông tin theo yêu cầu. Ví dụ một siêu thị điện tử, giới thiệu các mặt hàng trên trang Web, và thông tin về các mặt hàng đều được đưa lên đầy đủ. Nếu trang Web này là tĩnh được chuẩn bị trước thì ta không thể lọc ra những thông tin mà mình cần được mà phải duyệt cho tới khi gặp được mặt hàng mà mình quan tâm, nghĩa là phải đợi để thông tin được chuyển về đầy đủ. Vậy nhu cầu về một trang Web có thông tin được chọn lọc theo yêu cầu từ Browse ra đời. Các trang Web này được gọi là trang Web động. Nói một cách đơn giản là các trang Web động là các trang Web không tồn tại sẵn mà chỉ được tạo ra theo yêu cầu của người tra cứu. Trong trường hợp này CSDL Web không phải là tất cả mà còn các CSDL kiểu khác giúp tạo nên các trang Web. Chính vì thế cần đưa vào các trang HTML khả năng tạo Web động dưới dạng các dòng lệnh.

Microsoft quản lý các trang Web bởi IIS (Internet Information Server) trên WebServer. Nhưng IIS không tự tính toán được các dòng lệnh ở phía Server để tạo các trang Web động nên cần có thêm các thành phần khác.

Hiện nay có một số môi trường để tạo các trang Web động, có thể kể đến như: lập trình trên CGI, ASP, PHP, Java, JSP….

1.2. ASP là gì?

ASP (Active Server Page) là một thành phần mở rộng của IIS. Khi cài đặt, ASP sinh ra các bộ xử lý ảo đối với ngôn ngữ

kịch bản (script engine) tại server để IIS có thể xử lý các mã script mà các mã này có thể viết đan xen trong các trang HTML. Khi Client gọi đến một file .asp trên Web Server, Web Server lập tức gọi đến Script engine để xử lý. Script engine sẽ thực hiện các lệnh script để biến trang ASP thành trang HTML rồi gửi lại Client. Chú ý rằng quá trình này thực hiện tại server chứ không phải tại Client. Vì vậy chúng ta không phải quan tâm tới việc browser xử lý các trang Web như thế nào. Như vậy thực sự quá trình này được thực hiện theo mô hình Client-Server.

ASP là công nghệ Web Server mới của Microsoft, nó được thiết kế để giúp người phát triển ứng dụng trên Web xây dựng các trang Web ứng dụng nhanh chóng và dễ dàng. ASP là một phần tích hợp của công nghệ cơ sở Active (Active Platform), là hạt nhân trong chiến lược internet của Microsoft.

Active Platform là một tập hợp các ngôn ngữ, các chuẩn và các dịch vụ có thể được sử dụng để phát triển cả ứng dụng Active Desktop(bản Client) và Active Server (bản Server) trong mô hình CSDL tính toán Client / Server.

Mô hình Active Platform giúp cho người phát triển ứng dụng xây dựng ứng dụng hiệu quả về giá thành, mở rộng khả năng của các ứng dụng chạy trên Server cũng như chạy trên Client và nâng cao kỹ năng phát triển ứng dụng của họ. Đồng thời, nó cũng làm việc chuyển đổi từ ứng dụng Desktop sang ứng dụng Client/Server đầy đủ, dễ dàng.

1.3. Scripting?

Scripting là một đoạn chương trình mà chúng ta chèn vào các trang HTML để tạo tính "động" cho nó. Scripting dùng ngôn ngữ, cú pháp và cách thực hiện riêng. Tuy nhiên, có một vấn đề nảy sinh ở đây: Mỗi một hãng cung cấp lại định nghĩa một ngôn ngữ script khác nhau. Microsoft phát triển Visual Basic Script (VBScript), Sun Microsystem và Nescape phát triển JavaScript (JScript) và một số hãng khác hỗ trợ những ngôn ngữ như : Perl, Python, Awk ... .

Scripting trên Client:

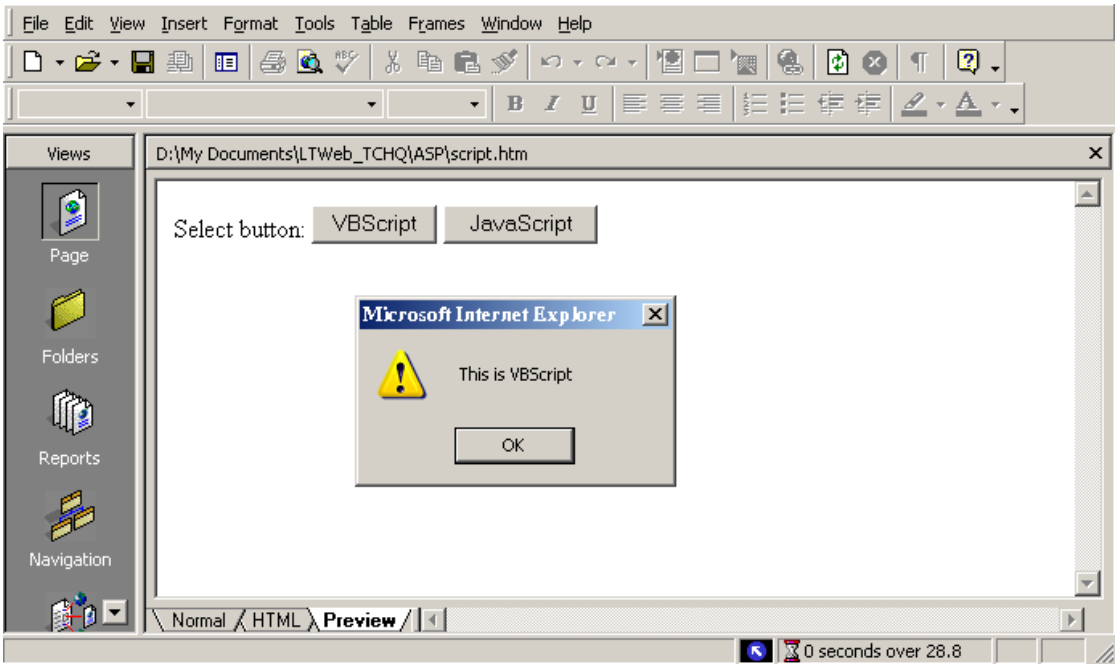
Scripting trên Client có thể được chèn vào trang HTML bằng cặp tags <Script> ... </Script>. Để xác định ngôn ngữ Script ta dùng thuộc tính LANGUAGE.

Ví dụ sau sẽ minh họa sự kiện xảy ra khi người dùng nhấn vào một nút. Chức năng của nó được chỉ ra bằng thuộc tính ONCLICK. Scripting trên Client có thể làm việc trên bất kì máy chủ nào hỗ trợ ngôn ngữ của scripting .

```
<html>

<head>
   <title>Script</title>
   <SCRIPT LANGUAGE="VBSCRIPT">
         sub vbs
            alert("This is VBScript")
         end sub
   </SCRIPT>
   <SCRIPT LANGUAGE="JAVASCRIPT">
```

```
        function js()
        {
            alert("This is JavaScript")
        }
    </SCRIPT>
</head>

<body>
        Select button:
        <INPUT TYPE= "button" NAME= "vbs" VALUE= "VBScript"
ONCLICK= "vbs()">
        <INPUT TYPE= "button" NAME= "js" VALUE= "JavaScript"
ONCLICK= "js()">
</body>
</html>
```

Nhấn vào một nút, chương trình phù hợp sẽ được chạy :



Scripting trên Server:

ASP sử dụng Scripting trên Server để tự động tạo ra những trang trả lời. Nội dung sinh ra dựa trên những thông số của User được gửi tới cùng yêu cầu và sự tương tác giữa các đối tượng khác nhau. Ngoài ra chúng ta còn có thể sử dụng một số Object và Component do ASP cung cấp. Các Object làm đơn giản hoá một số công việc trên Server

Scripting trên Server được chèn vào một file ASP bằng cách sử dụng cặp tags <SCRIPT> ... </SCRIPT> hoặc <% và %> . Để phân biệt Scripting được viết trên Client hay trên Server ta sử dụng thuộc tính RUNAT =”SERVER” .

Đối với hầu hết các browser thì ngôn ngữ Script mặc định là JavaScript. Bởi vậy, nếu chúng ta viết các mã lệnh bằng VBScript thì chúng ta phải khai báo với browse trước khi sử dụng :

<SCRIPT LANGUAGE=”VBSCRIPT”>

Ngược lại với browser, ngôn ngữ script mặc định của ASP là VBScript. Nếu chúng ta dùng JavaScript phải khai báo lại (chú ý thuộc tính RUNAT):

<SCRIPT LANGUAGE=”JCRIPT” RUNAT=”SERVER”>

Tùy theo khả năng và sở thích, ta có thể sử dụng một trong hai ngôn ngữ trên để lập trình. Tuy nhiên, có một chú ý quan trọng: hiện nay, JScript (ECMA Script) là ngôn ngữ Scripting

chuẩn được Hiệp hội sản xuất máy tính Châu Âu (European Computer Manufactury Association) công nhận.

Chú ý:

- **Tên mở rộng cho tập tin *ASP* là *.asp.**
- **Mặc định trang *ASP* là mã *VBScript* nếu không khai báo chỉ mục *Page*.**
- ***Cú pháp ASP:* Khi làm việc với *ASP*, mã *ASP* và *HTML* trộn chung lẫn nhau, ta có thể viết mã *ASP* theo từng khối được gọi là *Script* hay *Scriptlet.***
  - ***Script* được định nghĩa như một khối có dấu bằng theo sau:**

    <%= giá trị/biến/ biểu thức/ hàm %>
  - ***Scriptlet* cho phép khai báo đoạn chương trình bằng mã *ASP*:**

```
<%
        Dim x,y
        x=1000
        y="ASP (VBScript)"
%>
```

1.4. Tạo và xem một file ASP

Chúng ta sử dụng những công cụ của Microsoft như: Microsoft Frontpage, Microsoft's Visual InterDev để tạo một file ASP.

Để xem một file ASP, chúng ta không thể gửi nó đến Browser như một trang HTML bởi vì Browser không nhận biết được các file ASP. Lí do: Các file này cần phải được thông dịch trên Server trước khi gửi ra Browser.

Chúng ta có thể sử dụng IE hoặc Nescape Navigator để xem kết quả của các file ASP nhưng chúng ta cần chắc chắn rằng Server mà chúng ta xử lý các mã có cài đặt ASP và đang chạy IIS hoặc Personal Web Server (PWS).

Trang ASP là một dạng text có kiểu là .asp, có cấu trúc gần giống như file HTML. Tất cả các thẻ có trong HTML thì đều dùng được trong ASP. Nhưng ngoài các thẻ thông thường của HTML, trong file asp còn có thể viết các thẻ khác nữa để thể hiện các dòng lệnh của Script để làm việc với dữ liệu có sẵn hay có thể tính toán ngay bên trong như là một ngôn ngữ lập trình thực sự.

Cơ chế hoạt động của ASP như sau:

- **Client sử dụng một Web Browser gửi yêu cầu HTTP tới một Server chạy Microsoft Internet Information Server (IIS).**
- **Sau khi nhận biết đó là yêu cầu cần xử lý của trang ASP, IIS chuyển yêu cầu này tới ASP engine, tại đây nội dung file .asp được xử lý, các mã HTML được để nguyên còn các Script được tính dựa theo các yêu cầu và được chuyển đổi thành các mã HTML.**

- **Nếu trong các Script có các câu lệnh gọi dữ liệu, nó sẽ liên kết tới Database Server và lấy các dữ liệu theo yêu cầu.**
- **Sau đó, các kết quả của việc thực hiện các Script(có thể là HTML tĩnh hoặc động) được gửi trả lại Client Browser.**

Xét một ví dụ để làm rõ cơ chế này

```
<html>
    <head>
        <title>Example 1</title>
    </head>
    <body>
        <% For i=5 to 7%>
        <font size =<%=i%>>Chao ban!<br></font>
        <% Next %>
    </body>
</html>
```

Kết quả là trình duyệt sẽ hiển thị 3 dòng Chào ta! với kích thước tăng dần như sau

Chao ban!
Chao ban!
Chao ban!

Nếu ta mở phần mã nguồn của trang Web này tại browse kết quả sẽ là

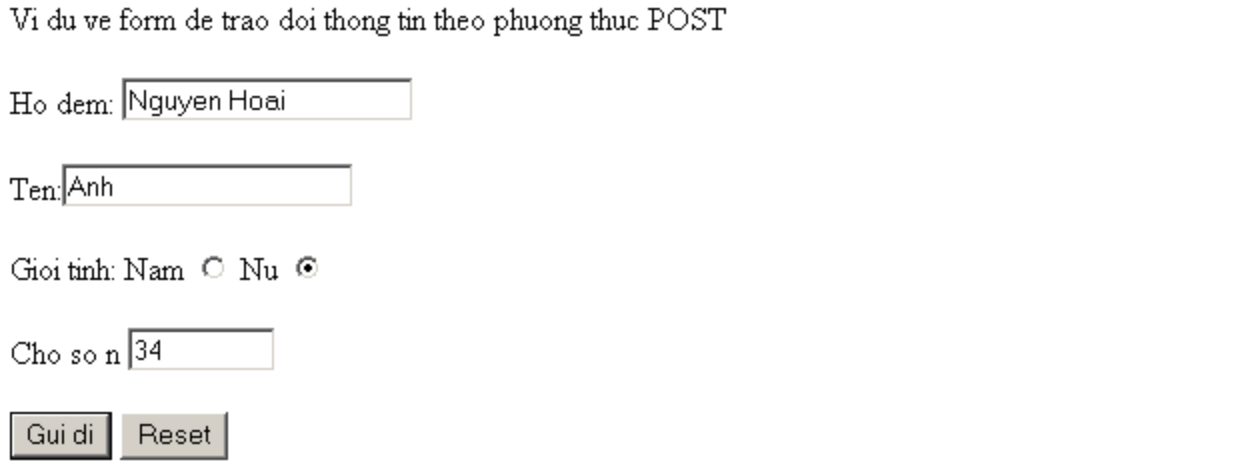
```
<html>
<head>
    <title>Co che ASP</title>
</head>
<body>
```

```
            <font size =5>Chao ban!<br></font>
            <font size =6>Chao ban!<br></font>
            <font size =7>Chao ban!<br></font>
</body>
</html>
```

Đây chính là kết quả xử lý các Script trên Server trả về mã HTML cho Browse hiển thị.

Trong ví dụ này chúng ta mới đề cập đến nguyên lý làm việc của ASP mà chưa nói đến cơ chế tạo Web động tức là cách tạo các trang Web thay đổi theo thông tin tương tác mà người sử dụng cung cấp. Trong ví dụ ta sẽ làm việc như sau:

Tạo một trang Web có một form để người sử dụng nhập vào họ, tên và giới tính qua Textbox và option. Người sử dụng cũng đánh vào một số tự nhiên. Khi nhận được, ASP sẽ gửi lại một trang Web với một lời chào phù hợp với giới tính và tên người đã được cung cấp đồng thời cho ra tất cả các ước số của số này.



Sau đây là đoạn mã của hai trang ASP thực hiện hiệu ứng này.

```
<html>
<head>
<title>chao</title>
</head>

<body>
<p>Vi du ve form de trao doi thong tin theo phuong thuc POST</p>

<form method="post" action="chao.asp">
<p> Ho dem: <input type ="text" name="Hodem" size=20></p>
<p>Ten:<input type ="text" name="Ten" size=20></p>
<p> Gioi tinh: Nam
      <input type ="radio" value="Nam" checked name="Gioitinh" size=20>
   Nu <input type ="radio" value ="Nu" name="Gioitinh"
size=20></p>
<p>Cho so n <input type ="Text" name="So" size=8></p>
<p><input type="submit" value ="Gui di" name="B1">
<input type="reset"  name="B2"></p>
</form>
</body>
</html>
```

Đây là nội dung tệp Chao.asp mô tả ứng xử khi ta bấm nút Gửi đi tương ứng với hoạt động Submit của Form

```
<html>
<head>
<title>chao</title>
</head>

<body>
<% ho=request.Form("hodem")
ten=request.Form("ten")
```

```
so=request.Form("so")
gioitinh=request.form("gioitinh")
if gioitinh="Nam" then gioitinh="Ong"
else gioitinh="Ba"
end if
response.Write "Xin chao " & gioitinh & " " & ho &" " & ten &"
<br>"
response.Write "Day la ket qua tinh " & "<br>"
for i=1 to so-1
 if so mod i =0 then
  response.Write i & "<br>"
 end if
next
%>
</body>
</html>
```

1. 5. Server-side Includes:

Server-side Includes (SSI) là một thuật ngữ được sử dụng để mô tả cách thức các yếu tố khác nhau được chèn vào trang Web

Gắn những file text vào một trang với #include:

Chúng ta có thể gắn file text GetLastDay.txt (vốn là một file ASP, được save với tên trên, có function có chức năng lấy lại ngày cuối cùng trong một tháng) vào một trang Web bằng cách thêm câu lệnh trên vào trang và gọi chức năng:

```
<!-- #include file="GetLastDay.txt" -->

...

intLastDayAugust = GetLastDay(datAugust) ' chức năng chúng ta gắn vào

...
```

Nếu ta muốn gắn Script từ các file khác, file này phải chứa những phần Script hoàn chỉnh. Nói một cách khác, nó phải có đủ những tag <SCRIPT> ... </SCRIPT> hoặc <% ... %>.

Địa chỉ vật lý, địa chỉ ảo của file:

#include cho phép chúng ta chỉ đến một file bằng đường dẫn vật lý hoặc đường dẫn ảo.Ví dụ file Mytext.txt nằm trong th mục c:\TextFile và cũng có bí danh là /Text, ta có thể tham khảo tới nó bằng những cách sau:

```
<!-- #include file="C:\TextFile\MyFile.txt" --> ' đường dẫn vật lý
<!-- #include file="/Text/MyFile.txt" -->       ' đường dẫn ảo
```

2. Ưu điểm của việc sử dụng ASP tạo Web động

2.1. Đơn giản, dễ học và hiệu quả

Học và phát triển ASP là rất dễ dàng. Ta có thể sử dụng ASP để xây dựng một Web site có khả năng tương tác cao. Vì các ngôn ngữ kịch bản như VBScript, Jscript được tích hợp trong ASP nên rất tiện cho người phát triển đã biết ngôn ngữ VB, Java hay C++, còn đối với người chưa biết thì việc học nó cũng dễ dàng.

Các ứng dụng ASP không cần có trình biên dịch. Trong một vài công nghệ khác như CGI, để phát triển các trang Web động cần phải có một trình biên dịch để dịch thành một chương trình có thể chạy được sử dụng các môi trường phát triển ứng dụng truyền thống như Visual C++. Sau khi ứng dụng được

dịch, nó sẽ được copy vào thư mục CGI của Web Server. Chỉ cần có một chút sửa đổi chương trình thì ta phải dịch lại mã nguồn của chương trình và sau đó lại phải copy đè lên phiên bản trước của file chạy. ASP giải quyết vấn đề này bằng cách cung cấp các cách tạo lập trang Web một cách trực tiếp và dễ dàn hơn theo kiểu thông dịch(interpreter). Sau khi xây dựng xong một ứng dụng Web bằng ASP, ta không cần phải dịch chùng mà chỉ cần lưu giữ vào một file có kiểu là .asp và các ASP sẽ xử lý khi file này được gọi đến.

Ngoài các thành phần ASP sẵn có giúp chúng ta xây dựng rất nhiều ứng dụng động khác, ASP cũng cho phép ta tự tạo ra các thành phần ASP của riêng mình.

2.2. Giữ bí mật được mã

Một điểm bất lợi trong việc sử dụng ngôn ngữ kịch bản Client là phơi bày tất cả các thông tin và thuật giải của bài toán. Nếu một bài toán sử dụng ngôn ngữ kịch bản tại Client như VBSCript thì bất kỳ ai nhìn vào mã nguồn của trang Web đều có thể thấy được thuật toán của nó.

Với ASP tất cả các Script được thực hiện trên Server và chỉ có kết quả ra dưới dạng HTML được gửi về Browser nên nếu người dùng muốn xem mã nguồn của trang Web thì họ chỉ xem được mã HTML chứ không xem được mã Script đã tạo nên trang Web đó. Như vậy nếu sử dụng ASP thì NSD không thể biết được thuật toán của nhà phát triển vì các mã ASP được

thực hiện trên Server. ASP bảo vệ sự sở hữu về thông tin và thuật toán.

2.3. Bảo trì dễ dàng

Môi trường phát triển ASP giúp nâng cao hiệu quả sử dụng của các thiết bị sẵn có. Môi trường phát triển ASP giúp cho người phát triển sử dụng một cách dễ dàng và có hiệu quả các kỹ năng sẵn có. ASP cung cấp một cơ cấu thiết lập các trang Web phức tạp sử dụng ngôn ngữ kịch bản quen thuộc như VBScript, Jscript/JavaScript hay Perl.

Đối với nhiều phần mềm Client/Server khác, ngoài phần được viết trên Server, còn cần phần viết trên Client. Như vậy độ phức tập và tốn kém về mặt lập trình sẽ tăng lên. Khi viết bằng ASP thì chỉ cần có trình duyệt Web tại máy Client, sau đó nối đến máy chủ, như thế việc trên Client không còn gì phả quan tâm. Mỗi khi cần sửa chữa hoặc nâng cấp không cần phải làm gì với bản Client.

Thông qua các câu lệnh Script, ta có thể kết nối đến với một CSDL tại một Database Server. Để làm việc trên CSDL này, ta có thể nhúng ngôn ngữ truy vấn SQL. Thông qua đó việc lọc dữ liệu đơn giản, công việc lọc dữ liệu được tiến hành trên Server nên tránh được ách tắc đường truyền.

3. Cài đặt IIS và tạo thư mục ảo cho ứng dụng

3.1. Cài đặt IIS

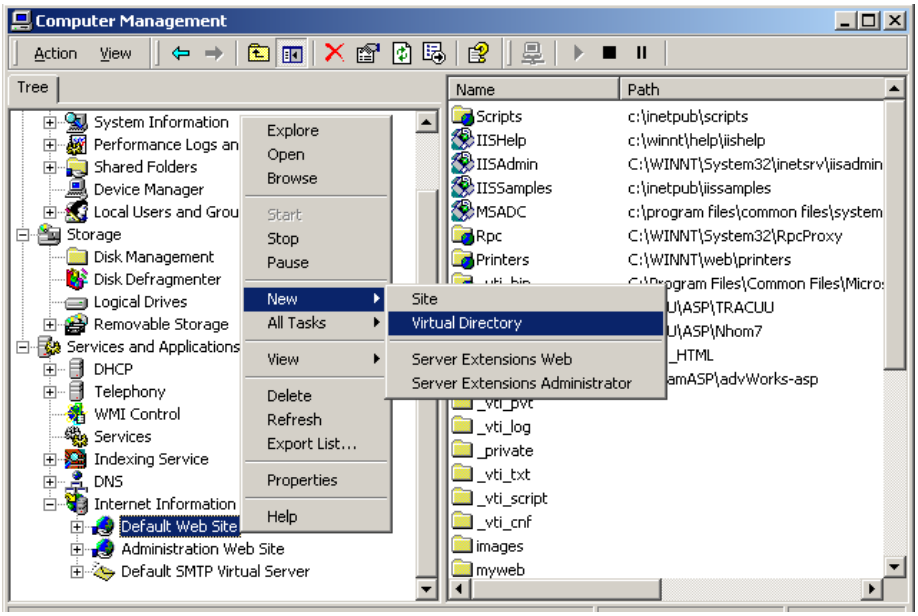
Internet Information Services mặc định không được cài đặt trên hệ điều hành Windows XP Professional. Ta có thể gỡ bỏ hoặc thêm các components bằng việc sử dụng chương tình ứng dụng Add/Remove Programs trong tiện ích Control Panel. Các bước cài đặt như sau:

1. **Click Start, click Control Panel, và double-click Add/Remove Programs.**
2. **Trong cột bên trái của hộp thoại Add/Remove Programs dialog box, click Add/Remove Windows Components.**
3. **Khi cửa sổ Windows Components Wizard xuất hiện, click Next.**
4. **Trong danh sách các thành phần của Windows(Windows Components), chọn IIS.**
5. **Click Next, và làm theo các chỉ dẫn của Wizard.**

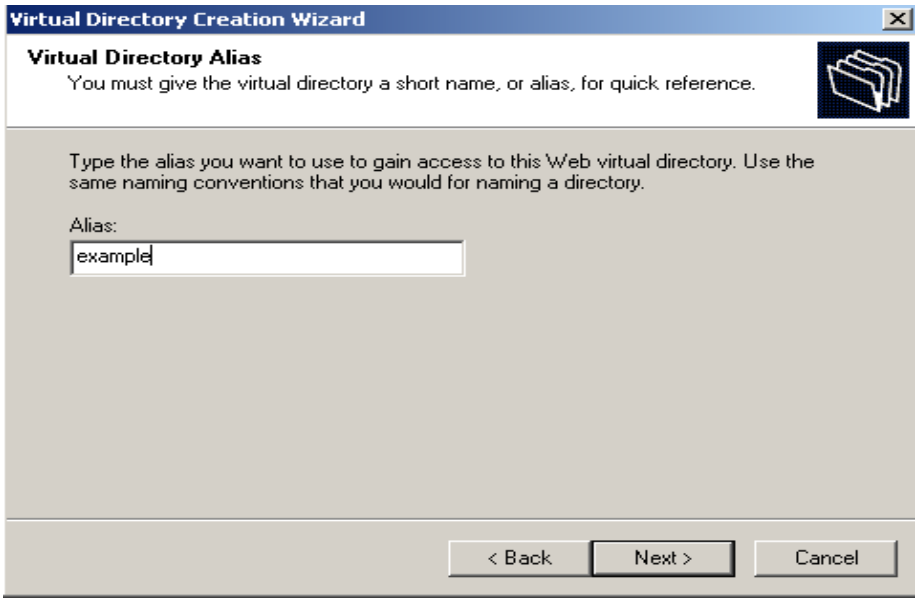
3.2. Tạo thư mục ảo

Ta có thể tạo các thư mục ảo bên dưới ứng dụng Default Web site. Thông thường một ứng dụng Web được đặt trong một thư mục ảo và được tham chiếu đến thông qua địa chỉ URL.

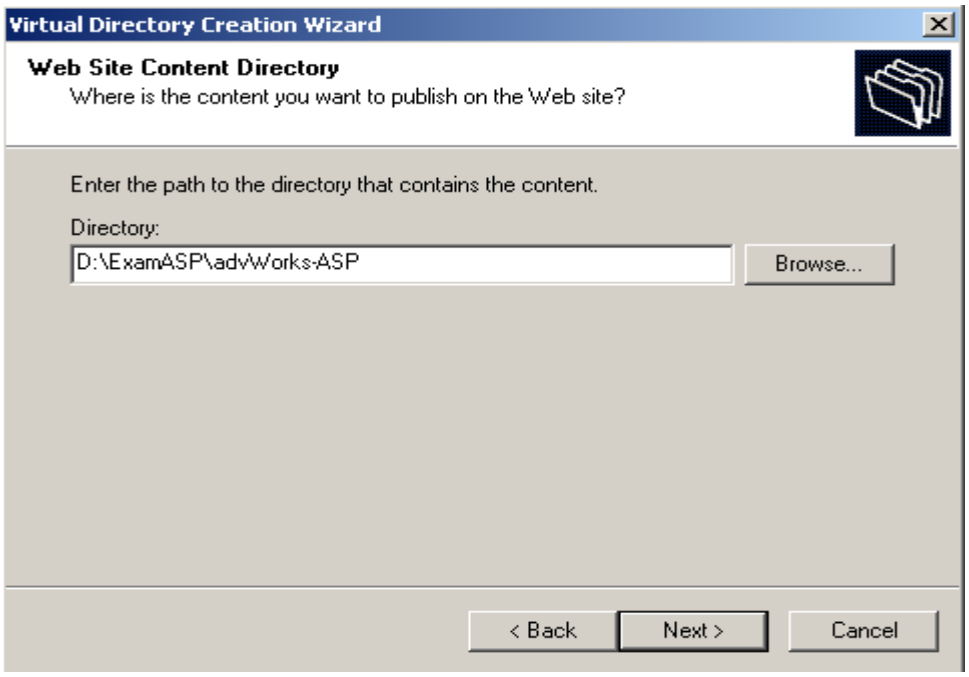
Chọn Internet Information Service:



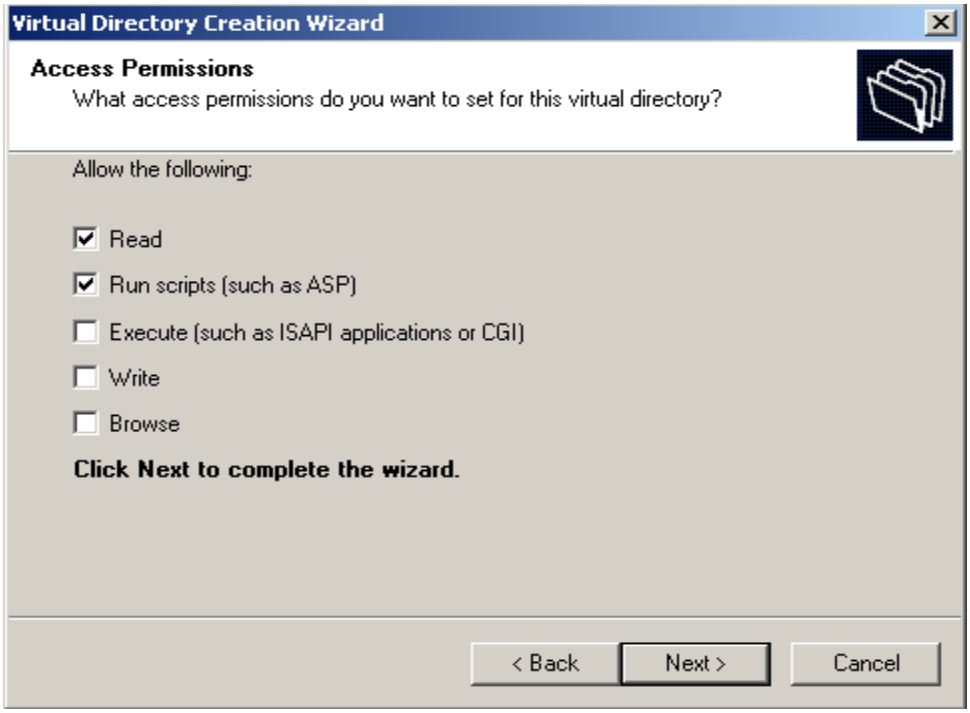
Nhấn chuột phải vào Default Web Site, chọn New/Vitual Directory, cửa sổ trợ giúp Wizard sẽ hiện ra như sau:



Trong ô nhập liệu Alias, ta nhập vào tên bí danh cho thư mục ảo, chẳng hạn example. Kích vào nút Next để đến bước kế tiếp.



Chọn đường dẫn vật lý cho thư mục ảo



Thiết lập quyền truy cập cho thư mục ảo. Hai quyền Read và Run Script là cần thiết để trang ASP có thể truy xuất được.

4. Cấu trúc và các dòng lệnh cơ bản của ASP

4.1. Các thành phần được dùng trong trang ASP

File ASP là một file dưới dạng Text, ta có thể sử dụng bất cứ trình soạn thảo văn bản dưới dạng text only để soạn thảo ra file ASP. File ASP có phần mở rộng là .asp . Trong file ASP có thể có:

- **Các mã HTML.**
- **Các kí hiệu phân cách Script**
- **Các mã Script**
- **Các thànhh phần ActiveX**
- **Các đối tượng ASP**

Điểm khác biệt cơ bản giữa file ASP với file HTML là sự có mặt các dấu phân cách các mã Script với các mã HTML. Trong file ASP nếu ta viết hướng dẫn <%<lệnh>%> thì ASP hiểu rằng lệnh bên trong hai dấu <% và %> là một lệnh Script. Như đã nói ở trên, các lệnh Script có thể là VBScript hay Jscript

4.2. Biến trong ASP

Biến cho bằng một tên gọi nào đó (quy cách giống như biến dùng trong Visual Basic). Khi sử dụng biến trong Script, ta không cần phải khai báo trước mà sử dụng trực tiếp. Các biến

trong ASP không có kiểu, kiểu của nó sẽ được xác định một cách tự động khi có lệnh gán giá trị vào biến. Nếu có khai báo biến thì cú pháp như sau: Dim tên_biến

Ví dụ: **Ta có khai báo như sau:**

```
<%
   Dim strSQL, liCounter
   strSQL="Select * from Customers"
   liCounter=10
%>
```

Khi gán giá trị cho biến hoặc khai báo, ta có thể sử dụng dấu ":" để phân cách phát huy thay vì khai báo trên hai dòng như sau:

```
<%
   Dim liCounter1; liCounter
   liCounter1=10:liCounter2=20
%>
```

Trong trường hợp khai báo biến mảng, ta cũng thực hiện tương tự như các khai báo mảnh trong Visual Basic 6.0. Chẳng hạn, chúng ta khai báo mảng như sau:

```
<%
arrDay=array("","","Mon","Tue","Web","Thu","Fri","Sat","
Sun")
ngay=arrDay(2) ' trả về Mon
%>
```

Nếu khai báo trước sau đó gán giá trị, ta chỉ định số phần tử của mảng, ví dụ khai báo mảng gồm 100 phần tử như sau:

```
dim arStoreName (100)
```

Trong trường hợp mảng hai chiều, ta khai báo tương tự như sau

```
dim arStoreName (100,100)
```

Lưu ý rằng, ta có thể sử dụng từ khoá *ReDim* để khai báo lại mảng động như sau

```
dim x,l
x=cstr(Session.SessionID)
l=len(x)
Dim arrSession()
ReDim arrSession(l)
```

4.3. Dấu gạch dưới

Cũng như trong ngôn ngữ lập trình *Visual Basis* 6.0 để tiếp nối câu lệch quá dài, ta sử dụng dấu gạch dưới (_). Chẳng hạn, để khai báo phát biểu SQL dạng *Select* quá dài, ta có thể khai báo với dấu gạch dưới như sau:

```
sql="Select R.MaSV, R.Hoten, R.NgaySinh, R.MaLop, S.Tenlop
"_
& "From HosoSV R, Lop S"_
& "where R.MaLop=S.Malop"
```

Tuy nhiên, ta cũng có thể sử dụng phép toán & hay + để kết nối chuỗi như sau:

```
sql="Select R.MaSV,R.Hoten, R.NgaySinh, R.MaLop,
S.Tenlop"
sql=sql & "From HosoSV R, Lop S"
```

```
sql=sql & "where R.MaLop=S.Malop"
```

Ngoài ra, ta có thể khai báo như sau:

```
sql="Select R.MaSV,R.Hoten, R.NgaySinh, R.MaLop, S.Tenlop
"&_
"From HosoSV R, Lop S"&_
& "where R.MaLop=S.Malop"
```

4.4. Dấu ghi chú

Cũng tương tự như VB, ta sử dụng dấu ' để khai báo câu ghi chú trong mã ASP.

Ví dụ: `ngay=arrDay(2) ' trả về Mon`

4.5. Các lệnh cơ bản của ASP

4.5.1. Lệnh gán

Cú pháp: `<%<biến>=[giá trị]%>`

Lệnh này sẽ nạp giá trị vào biến

Ví dụ: `ngay=13`

4.5.2. Lệnh đưa ra màn hình giá trị của biến

Cú pháp: `<%=<biến>%>`

Khi xử lý lệnh này, ASP chuyển đoạn mã ngữ trên thành một văn bản với nội dung chính là lệnh gán giá trị của biến. Khi trình duyệt xử lý nó sẽ hiển thị giá trị này ra màn hình.

Ví dụ: `<%=ngay%>`

4.5.3. Các cấu trúc điều khiển

a. Câu lệnh If...then...else... end if

Cú pháp

- Câu lệnh điều kiện IF…THEN

```
if <điều kiện> then
   <lệnh>
End if
```

- Câu lệnh IF…THEN…ELSE

```
if <điều kiện> then <lệnh 1>
Else <lệnh 2>
End if
```

- Câu lệnh IF lồng nhau:

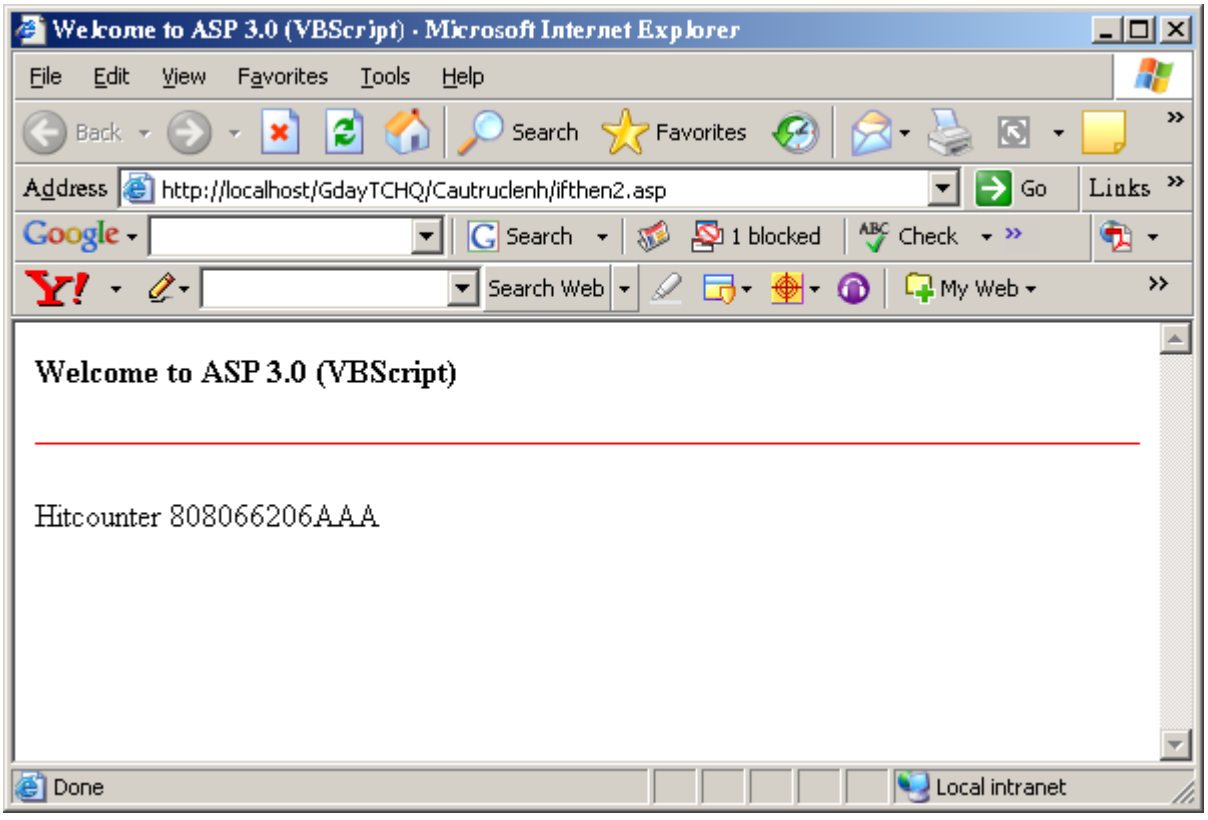
```
if <điều kiện 1> then <lệnh 1>
Else if <điều kiện 2> then <lệnh 2>
Else <Các câu lệnh khác>
End if
```

Ví dụ 4.5.3.1. Kiểm tra thời gian để hiển thị câu "Bây giờ là buổi sáng" hay "Bây giờ là buổi chiều"

```
<html>

<head>
<title>Cau truc if then</title>
</head>

<body>
<% if time<=#12:00:00 AM# then
        x="Bây giờ là buổi sáng"
   else
        x="Bây giờ là buổi chiều"
```

```
    end if
%>
<%=x%>
</body>
</html>
```

Ví dụ 4.5.3.2. Hiển thị nội dung như sau




```
<%
    Dim x,y
    x=Session.SessionID
    if x<>"" then x=x+"AAA"
    y="ASP 3.0 (VBScript)"
%>
<html>
```

```
<head>
<title>Welcome to <%=y%></title>
</head>

<body>
   <h4>Welcome to <%=y%></h4>
   <hr size=2 color=red><br>
   <%if len(x)>10 then%>
   Hitcounter <%=x%>
   <%end if%>
</body>
</html>
```

Chú ý: **Do cú pháp tương tự như cú pháp của *Visual Basic* 6.0, cho nên phát biểu If không có End if khi câu lệnh cùng nằm trên một hàng. Tuy nhiên nếu câu lệnh bên trong là cấu trúc HTML ta phải sử dụng phát biểu End if.**

```
<%if len(x)>10 then%> <hr> <%end if%>
```

b. Cấu trúc lựa chọn Select Case

Cú pháp:

```
Select case <biểu thức>
        case <danh sách các giá trị 1>
              <các câu lệnh 1>
        case <danh sách các giá trị 2>
              <các câu lệnh 2>
        ...
        case <danh sách các giá trị n>
              <các câu lệnh n>
        case else
              <các câu lệnh n+1>
```

End select

Ví dụ 4.5.3.3:

```
<html>

<head>
<title>Cau truc Select Case</title>
</head>

<body>
<%
bien=5
select case bien
 case 1,2,3
   Response.Write  ("chon 1")
  case 2,3,4
    Response.Write "chon 2"
  case else
   Response.write "chon 3"
   end select
%>
</body>
</html>
```

⇨ **Kết quả: Chon 3**

4.5.4. Cấu trúc vòng lặp

Hầu hết các lệnh lặp trong Visual Basic đều sử dụng trong ASP. Ta có các vòng lặp sau:

a. Vòng lặp for

Cú pháp:

- Câu lệnh lặp For…Next…

For <*biến đếm*>=<*điểm đầu*> **To** <*điểm cuối*> [**Step** <*bước nhảy*>]

<*Khối lệnh*>

Next

- Câu lệnh lặp For Each…Next

For Each <*Phần tử*> **In** <*nhóm*>

<*Khối lệnh*>

Next

***Ví dụ 4.5.4.1*:**

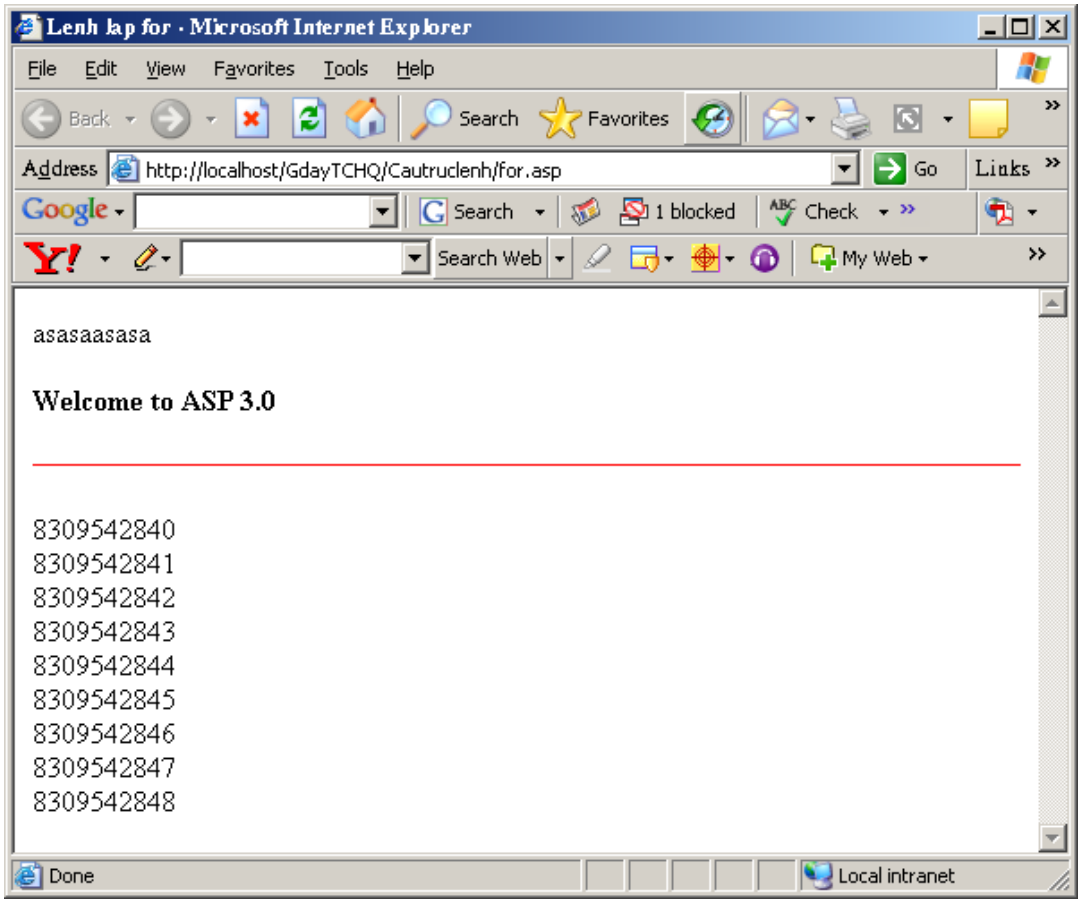
```
<%for i=5 to 7 %>
<font size =<%=i%>>Chào bạn!<br></font>
<% Next %>
```

***Ví dụ 4.5.4.2*: Sử dụng lệnh for**

```
<%
    Dim x,l
    x=cstr(Session.SessionID)
    l=len(x)
    Dim i
    Dim arr()
    ReDim arr(l)
    for i=0 to l-1
        arr(i)=x & i
    next
    strSQL ="asasa" & _
    "asasa"
    response.write strSQL
%>
<html>
<head>
<title>Lenh lap for</title>
```

```
</head>

<body>
    <h4>Welcome to ASP 3.0</h4>
    <hr size=1 color=red><br>
    <%for i=0 to l-1%>
            <%=arr(i)%><br>
    <%next%>
</body>
</html>
```

Kết quả chạy như sau



Chú ý: **SessionID là số nguyên dài (Long) do Web Server cấp phát tự động cho mỗi browser khi chúng ta truy cập Web**

b. Vòng lặp Do..loop

- <u>Lặp trong khi điều kiện True</u> - <u>Lặp trong khi điều False</u>

Do While *<điều kiện>*
<Khối lệnh>
[Exit Do]
Loop

hoặc

Do
<Khối lệnh>
[Exit Do]
Loop While *<điều kiện>*

Do Until *<điều kiện>*
<Khối lệnh>
[Exit Do]
Loop

hoặc

Do
<Khối lệnh>
[Exit Do]
Loop Until *<điều kiện>*

Ví dụ:

```
<%
    i=1
    do
        i=i+1
        Response.Write i
    loop while  i<=10
%>
```

c. Vòng lặp While .. Wend

Cú pháp

While <điều kiện>
<Khối lệnh>
Wend

Tương tự như vòng lặp *Do…Loop*, nhưng vòng lặp *While…End* không thể thoát khỏi vòng lặp bằng lệnh *Exit*, chỉ thoát khi điều kiện là *False*.

4.6. Xây dựng các hàm và thủ tục trong ASP

Ta có thể sử dụng các ngôn ngữ Script để xây dựng các hàm và thủ tục trong file ASP. Trước khi viết một hàm và thủ tục bằng ngôn ngữ gì ta phải thông báo cho ASP biết bằng thẻ Script như đã biết.

Cấu trúc một hàm trong ASP có dạng sau:

```
<SCRIPT RUNAT =SERVER
LANGUAGE=”LANGUAGENAME”>
    Function <FunctionName> (Biến)
        Các dòng lệnh Script
    End Function
    ‘Thủ tục:
    Sub <SubName> (Biến)
        Các dòng lệnh Script
    End Sub
</SCRIPT>
```

Đối với hàm thì trong thân của hàm cần có một lệnh gán giá trị tính được cho một biến có tên trùng với tên hàm.

Cách gọi hàm hoàn toàn tương tự như cách lấy giá trị từ một biến.

Cách gọi thủ tục:

Call SubName hoặc SubName

Ví dụ ta có hàm sau:

```
<%Function Calculate(A, B, Op)
    Select Case Op
```

```
        Case  "+"
              Calculate = A+B
        Case "-"
              Calculate = A-B
        Case "*"
              Calculate = A*B
        Case "/"
              Calculate = A/B
        End Select
End Function
        Response.write Calculate(2, 3, "+")
        Response.write Calculate(2, 3, "-)
%>
```

Chú ý: **Có thể sử dụng <!--#include file|virtual="file_name"--> để sử dụng lại các hàm và thủ tục đã được xây dựng trong một file nào đó.**

5. Sử dụng các đối tượng của ASP để trao đổi thông tin giữa Client và Server

5.1. Giới thiệu các đối tượng chính của ASP

Đối tượng là những đoạn chương trình có khả năng thực hiện một số công việc cơ bản nào đó. Mỗi đối tượng là một kết hợp giữa lập trình và dữ liệu, vốn có thể xử lý như 1 đơn vị thống nhất.

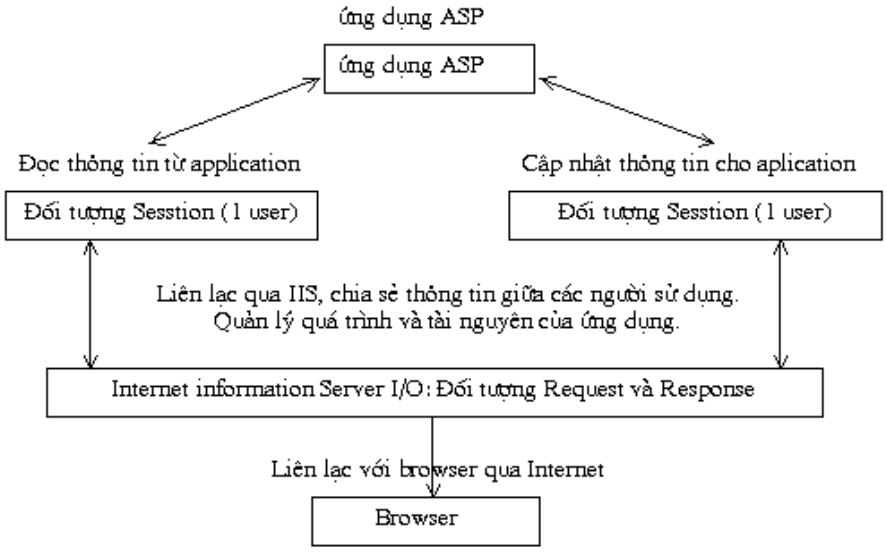
Các đối tượng *ASP* cho phép chúng ta giao tiếp, tương tác với cả máy chủ *(Webserver)* lẫn trình duyệt *(browser)*. Trong các kịch bản, thông thường chúng ta phải dùng vài đối tượng.

Tương tự như trong các ngôn ngữ lập trình hướng đối tượng, ASP cho phép người lập trình tạo ra các đối tượng, các

lớp theo mục đích sử dụng riêng. ASP cũng cung cấp sẵn có một số đối tượng hay được sử dụng. Đó là 5 đối tượng sau

Các đối tượng	ý nghĩa
Application	Chia sẻ thông tin giữa các người dùng trong một ứng dụng
Session	Lưu giữ các thông tin duy nhất về phiên làm việc hiện thời của một người sử dụng cụ thể
Server	Cho phép truy cập tới máy chủ
Request	Lấy thông tin từ phía người dùng
Response	Gửi thông tin tới người dùng

Sơ đồ mối quan hệ giữa các đối tượng *ASP* trong ứng dụng *ASP* như sau



I. File Global.asa

Là nơi khai báo các đối tượng, biến có phạm vi phiên làm việc hay toàn bộ ứng dụng. File Global được kích hoạt mỗi khi một phiên làm việc mới được thiết lập, tuy nhiên sự kiện Application_OnStart chỉ được kích hoạt một lần khi webserver được khởi động. Mỗi một ứng dụng chỉ có thể có duy nhất một file Global.asa.

Các sự kiện của các đối tượng Application và Session được khai báo trong file Global.asa

Cú pháp:

```
<Script Language=VBScipt  RUNAT=Server>
    Application_OnStart
    End Sub
    Application_OnEnd
    End Sub
    Session_OnStart
    End Sub
    Session_OnEnd
    End Sub
</Script>
```

Ngoài ra ta có thể viết các hàm và thủ tục đặt trong file Global.asa để phục vụ cho cả ứng dụng hay cho từng phiên làm việc cụ thể, các thủ tục và các hàm này phải nằm trong các sự kiện của hai đối tượng Application và Session.

5.3. Đối tượng Request

5.3.1. Ý nghĩa

Đối tượng *Request* dùng để nhận thông tin yêu cầu được gửi từ *Client Side* đến *Server Side*.

Khi một Browser liên lạc với Server thông qua giao thức HTTP, Browser gửi yêu cầu tới Server, ngoài tên của trang được yêu cầu thì còn rất nhiều thông tin khác đi kèm được gửi tới Server. Các thông tin này có thể là các biến môi trường, các thông tin do user cung cấp dưới dạng điền vào các bảng, Cookies,..Tất cả các thông tin này được mã hoá và truyền đi cùng với HTTP headers. ASP cho phép lấy ra các thông tin này bằng cách sử dụng đối tượng Request.

Khi sử dụng đối tượng *Request* ta có thể sử dụng các thành phần (*Collection*) và thuộc tính như: *Request.Form, Request.QueryString, Request.ServerVariables, Request.Cookies, Request.Item, Request.TotalBytes,…*

Trong phần này chúng ta tập trung tìm hiểu 4 thành phần chính của đối tượng *Request* là *Request.Form, Request.QueryString, Request.ServerVariables, Request.Cookies*.

II. Request

***Request.Form* trả về thông tin của các *field* trong thẻ *<form>* với phương thức (*method*) là *POST* (tham khảo trong chương kế tiếp) khi người sử dụng đệ trình (*submit*) từ *Client Side* lên *Server Side*.**

Để sử dụng *Request. Form* ta sử dụng cú pháp:

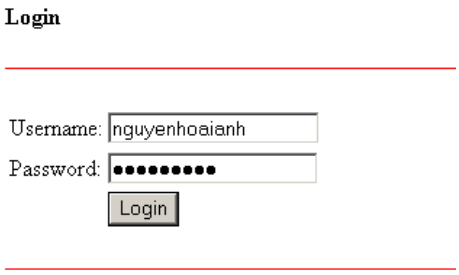
Request.Form("Fieldname") [(element)|.count]

Trong đó,

- ***fieldname* là tên của các thẻ *HTML* dạng <*input*>, <*textarea*>, <*select*>.**
- **(*element*) là thứ tự của *field* cùng tên và**
- ***count* là số lượng *fieldname* cùng tên.**

Chẳng hạn, chúng ta khai báo trang *HTML* dùng để đăng nhập hệ thống ứng dụng với các *field* là *username* và *password* cùng với thẻ *form* có tên *frmlogin* như ví dụ.

***Ví dụ 5.3.1.* Khai báo thẻ field trong thẻ form**




```
<html>
 <head>
    <title> form login </title>
 </head>
 <body>
    <h4>Login</h4>
    <hr size=1 color=red><br>
  <!-- Khai bao thẻ mở của form -->
    <form action=laygtlogin.asp method=post name=frmlogin>
    <table border=0>
    <tr><td>Username: </td>
  <!-- Khai báo field có tên là user -->
    <td><input type=text name=username></td></tr>
    <tr><td>Password: </td>
```

```
        <td><input type=password name=password></td>
        </tr><tr>
        <td ></td><td><input type=submit value="Login"></td>
        </tr></table>
        </form>
        <hr size=1 color=red><br>
    </body>
</html>
```

Chú ý:

- **Trong *HTML*, nội dung khai báo trong cặp dấu <!-- và --> trở thành ghi chú, điều này có nghĩa là chuỗi đó sẽ không xuất hiện trong đó.**
- **Không cần khai báo thuộc tính *name* khi ta không có nhu cầu truy cập đến các *field* này trên trình khách.**
- **Thông tin khai báo trong thẻ mở và đóng của <*body*> xuất hiện trên trình duyệt ngoại trừ thông tin khai báo trong cặp thẻ <!-- -- >.**

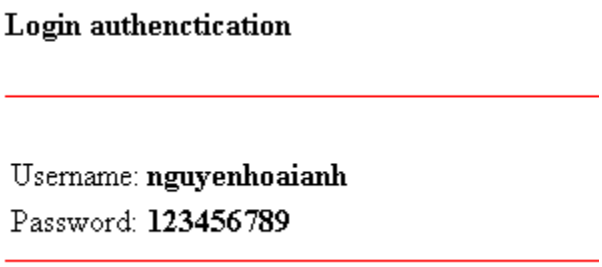
Nếu sử dụng nhập giá trị vào *username* và *password* sau đó nhấn nút *login*, *form* có tên *frmlogin* sẽ được đệ trình lên *Server Side*, để lấy được giá trị của người sử dụng nhập trên trang *ex1.htm*, chúng ta khai báo trang *ASP* và sử dụng *Request.Form* như ví dụ:

Ví dụ 5.3.2. **Lấy giá trị của field trong thẻ form**

```
<html>
  <head>
    <title> Ket qua login </title>
  </head>
  <body>
```

```
    <h4>Login authenctication</h4>
    <hr size=1 color=red><br>
    <table border=0>
    <tr><td>Username: </td>
    <td><b><%=Request.Form("username")%></b></td>
    </tr><tr>
    <td>Password: </td>
    <td><b><%=Request.Form("password")%></b></td>
    </tr></table>
    <hr size=1 color=red><br>
  </body>
</html>
```

Kết quả chạy như sau



Chú ý:

- **Ta có thể khai báo *Request.Form(''fieldname'')* với ký tự thường như: *request.form(''fieldname'')*.**
- **Tên của *field* không phân biệt kiểu chữ hoa hay thường, chẳng hạn ta có thể khai báo *request.form(''username'')* hay *request.form(''USERNNAME'')* cho dù ta khai báo tên là ''*username*'' trong trang *login.htm*.**
- **Thay vì khai báo *Request.Form(''fieldname'')* ta có thể khai báo *Request(''fieldname'')*.**

Tuy nhiên, ta có thể khai báo biến và gán giá trị vào biến từ *Request.Form* sau đó kết luận giá trị của biến đó như *ví dụ 5.3.3.* Ngoài ra, ta có thể chỉ định số thứ tự tương ứng với *field* khai báo trong thẻ *form* như ví dụ 5.3.4.

***Ví dụ 5.3.3.* Khai báo biến**

```
<%
   Dim u,p
   u=
Request.Form("username")
   p=
Request.Form("password")
%>
<html>
   <head>
      <title> Ket qua Login
</title>
   </head>
   <body>
      <h4>Login
authenctication</h4>
      <hr size=1
color=red><br>
      <table border=0>
      <tr><td>Username:
</td>
      <td><b><%=u
%></b></td>
      </tr><tr>
```

Ví dụ 5.3.4.

Lấy giá trị của chỉ mục field String thẻ form.

```
<html>
   <head>
      <title> Ket qua login </title>
   </head>
   <body>
      <h4>Login authenctication</h4>
      <hr size=1 color=red><br>
      <table border=0>
      <tr><td>Username: </td>
   <td><b><%=Request.Form(1)%></b></td>
      </tr><tr>
      <td>Password: </td>
   <td><b><%=Request.Form(2)%></b></td>
      </tr></table>
      <hr size=1 color=red><br>
   </body>
</html>
```

```
        <td>Password: </td>

    <td><b><%=p%></b></td>
        </tr></table>
        <hr size=1
color=red><br>
    </body>
</html>
```

Chú ý:

- **Chỉ mục bắt đầu số 1 và chỉ mục sử dụng cho các thẻ dùng để chọn giá trị hay nhập liệu.**
- **Chỉ mục không có hiệu lực khi ta sử dụng đối tượng *Request* không có *Collection*, ví dụ như *Request*(1). Tương tự như vậy, trong trường hợp sử dụng biến để lấy giá trị từ *Request.Form* ta có thể khai báo như sau:**

```
Dim u,p
u= Request.Form("username")
p= Request.Form("password")
```

- **Nếu chỉ mục không tồn tại, lỗi phát sinh với thông báo như sau:**

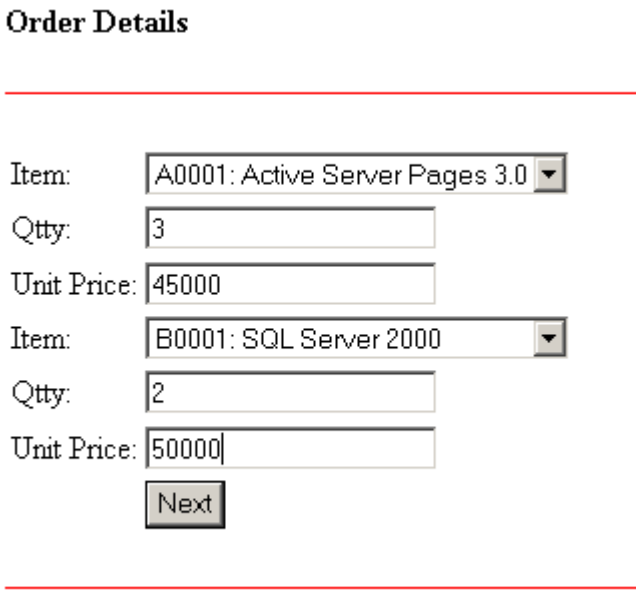
Request object error 'ASP 0105: 80004005'
Index out of range
/doituongASP/ laygtlogin2.asp, line 15

An array index is out of range

Cho dù ta khai báo các cách trên, kết quả của trang *laygtlogin.asp, laygtlogin1.asp, laygtlogin2.asp* sẽ được trình bày như hình trên khi người sử dụng *Submit* trang *login.htm*

Nếu như ta có nhiều *field* cùng tên, khi sử dụng đối tượng *Request.Form*, mặc định là *field* đầu tiên. Chẳng hạn, chúng ta khai báo trang *HTML* có tên *nhieufield.htm* với nội dung như ví dụ 5.3.5. như sau

Ví dụ 5.3.5. **Khai báo nhiều field cùng tên trong thẻ form**




```
<html>
   <head>
      <title>   nhieu field</title>
   </head>
```

```
<body>
   <h4>Order Details</h4>
   <hr size=1 color=red><br>
 <!-- Khai bao thẻ mở của form -->
   <form action=laygtform.asp method=post name=frmlogin>
   <table border=0>
 <!-- Khai bao thẻ select có tên Item thứ nhất -->
   <tr><td>Item: </td><td><select name=item>
   <option value="A0001:
   Active Server Pages 3.0">
   A0001: Active Server Pages 3.0</option>
   <option value="B0001:
   SQL Server 2000">
   B0001: SQL Server 2000</option></select>
   </td></tr>
 <!-- Khai báo thẻ input có tên qtty thứ nhất-->
   <tr>
   <td>Qtty:</td><td><input type=text name=qtty></td>
   </tr><tr><td>
   Unit Price:</td><td><input type=text name=price>
   </td> </tr>
 <!-- Khai bao thẻ select có tên Item thứ hai -->
   <tr><td>Item: </td><td>
   <select name=item><option value="A0001:
   Active Server Pages 3.0">
   A0001: Active Server Pages 3.0</option>
   <option value="B0001:
   SQL Server 2000">
   B0001: SQL Server 2000</option></select>
   </td></tr><tr>
 <!-- Khai báo thẻ input có tên qtty thứ hai-->
```

```
        <td>Qtty:</td><td><input type=text name=qtty></td>
        </tr><tr><td>    Unit Price:</td><td><input type=text
   name=price>
        </td>
        </tr><tr>
        <td ></td><td><input type=submit value="Next"></td>
        </tr></table>
        </form>
        <hr size=1 color=red><br>
     </body>
   </html>
```

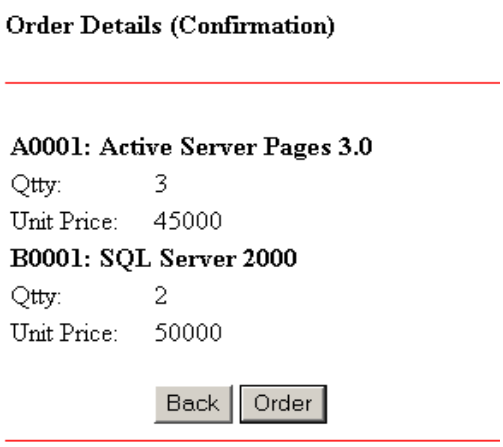
Trong trường hợp trang *nhieufield.htm* có hai thẻ *select* cùng tên *item*, hai thẻ *input* cùng tên *qtty* và hai thẻ *input* cùng tên *price*.

Khi người sử dụng điền giá trị vào các *field* tương ứng của trang *nhieufield.htm* như trên và nhấn nút *Next*.

Ví dụ 5.3.6. **Lấy giá trị nhiều field cùng tên.**

```
<html>
   <head>
      <title> Gia tri cua nhieu field cung ten </title>
   </head>
   <body>
      <h4>Order Details (Confirmation)</h4>
      <hr size=1 color=red><br>
      <table border=0>
      <tr><td colspan=2><b><%=Request.Form("item")(1)%>
</td></tr>

   <tr><td>Qtty:</td><td><%=Request.Form("qtty")(1)%></td></tr>
      <tr><td>Unit Price:</td>
```

```
        <td><%=Request.Form("price")(1)%></td>    </tr>
        <tr><td
colspan=2><b><%=Request.Form("item")(2)%></td></tr>
        <tr><td>Qtty:</td>
        <td><%=Request.Form("qtty")(2)%></td></tr>
        <tr><td>Unit Price:</td>
        <td><%=Request.Form("price")(2)%></td></tr>
        <tr><td ></td><td><br>
        <input type=button value="Back"
        onclick="window.history.go(-1);">
        <input type=submit value="Order"></td>
        </tr></table>
        <hr size=1 color=red><br>
    </body>
</html>
```



Ngoài ra, ta có thể sử dụng vòng lặp *For* và thuộc tính *count* của *Request.Form* để truy cập đến *field* cùng tên như sau:

Ví dụ 5.3.7. **Sử dụng thuộc tính Count**

```
<%Dim i%>
<html>
    <head>
```

```
    <title>   Gia tri cua nhieu field cung ten </title>
  </head>
  <body>
    <h4>Order Details (Confirmation)</h4>
    <hr size=1 color=red><br>
    <table border=0>
    <%for i=1 to Request.Form("qtty").Count%>
    <tr><td>
    <b>Item:</td><td><b>
    <%=Request.Form("item")(i)%>
    </td></tr>
    <tr><td>Qtty:</td><td>
    <%=Request.Form("qtty")(i)%></td>
    </tr>
    <tr><td>Unit Price:</td><td>
    <%=Request.Form("price")(i)%>
    </td></tr>
    <% Next %>
    <tr><td ></td><td><br>
    <input type=button value="Back"
    onclick="window.history.go(-1);">
    <input type=submit value="Order"></td>
    </tr></table>
    <hr size=1 color=red><br>
  </body>
</html>
```

5.3.3. Request.QueryString

Request.QueryString **dùng để lấy giá trị từ** ***QueryString*****.**

**** QueryString***

QueryString **được định nghĩa là một chuỗi nằm sau dấu ? trong chuỗi URL (Uniform Resource Locator) trên phần *Address* của trình duyệt khi triệu gọi đến một trang Web.**

Ví dụ: **Ta gọi triệu địa chỉ sau:**

http://localhost/LTWeb_TCHQ/login.asp?al=A&page=cust

Trong đó, QueryString bao gồm các cặp tham số và giá trị sau:

al=A&page=cust

Các tham số được khai báo cách nhau bằng ký tự "&" và giá trị của mỗi tham số được khai báo sau dấu "=" của mỗi tham số.

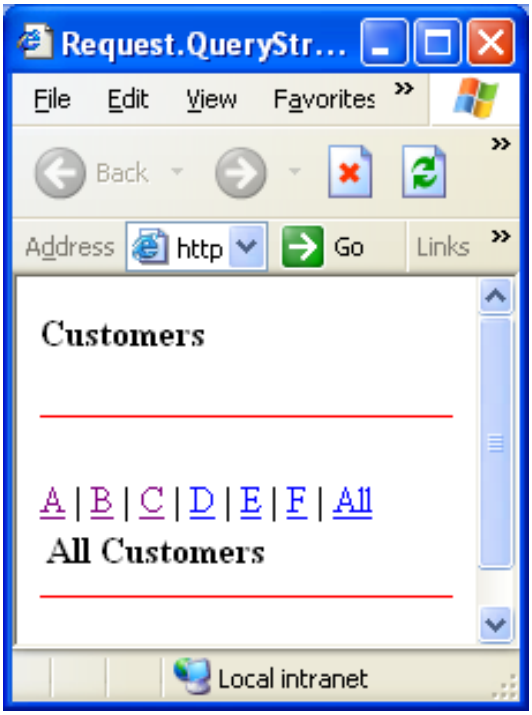
- **Nếu tham số không có dấu bằng theo sau, giá trị của tham số đó sẽ được trả về khii sử dụng *Request.QueryString* là null.**
- **Trong trường hợp có dấu bằng nhưng không tồn tại giá trị thì kết quả trả về khi sử dụng Request.String là rỗng.**
- **Tuy nhiên, trong ASP3.0 khi so sánh các giá trị của tham số trên QueryString bằng *Request.QueryString*, thì giá trị null và rỗng là tương đương.**

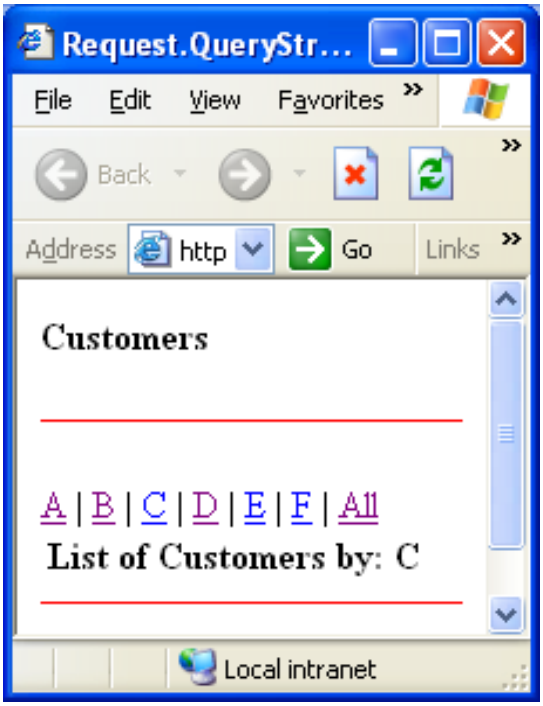
Khi sử dụng *Request.QueryString*, chúng ta sẽ lấy giá trị của các tham số có tên là *al* và *page*.

Ví dụ 5.3.8**. Khai báo và sử dụng lệnh *Request.QueryString***

```
<%dim al
<!Đọc giá trị của tham số al trên QueryString.-->
al=Request.QueryString("al")
if al="" then
```

```
  al="All Customers"
 else
  al="List of Customers by: " & al
 end if
 %>
 <html>
  <head>
     <title>Request.QueryString</title>
  </head>
  <body>
     <h4>Customers</h4>
     <hr size=1 color=red><br><!--Khai báo tham số trên QueryString-
 ->
     <a href="login.htm?al=A&page=cust">A</a> |
     <a href="login.htm?al=B&page=cust">B</a> |
     <a href="login.htm?al=C&page=cust">C</a> |
     <a href="login.htm?al=D&page=cust">D</a> |
     <a href="login.htm?al=E&page=cust">E</a> |
     <a href="login.htm?al=F&page=cust">F</a> |
     <a href="login.htm?al=&page=cust">All</a><br>
     <table border=0>
     <tr><td colspan=2><b>
<! Trình bày giá trị mà người sử dụng chọn-->
     <%=al%> </td></tr>
     </table>
     <hr size=1 color=red><br>
  </body>
 </html>
```





**** Tham số và giá trị trên Request.QueryString***

Để có cặp tham số và giá trị trên chuỗi QueryString, ta có thể khai báo trực tiếp từ thẻ A hoặc khai báo bằng phương thức GET trong thẻ Form.

***Ví dụ 5.3.9*. Khai báo bằng phương thức GET.**

METHOD=GET

Alphabet: A A->Z

Type: cust cust/item

Submit

```
<html>
   <head>
      <title>Phuong thuc GET</title>
```

```
    </head>
    <body>
        <h4>METHOD=GET</h4>
        <hr size=1 color=red>
<!--khai báo phương thức GET trong thẻ Form-->
        <form action=ptGET.asp method=get>
        <table border=0>
        <tr><td><b>Alphabet:</td>
<!--khai báo thẻ nhập liệu giá trị A→Z-->
        <td><input type=text name=al value=A>A->Z</td></tr>
    <tr><td><b>Type:</td>
        <td><input type=text name=page value=cust>cust/item</td></tr>
        <tr><td></td>
        <td><input type=submit value=Submit></td></tr>
        </table>
        <hr size=1 color=red>
    </body>
</html>
```

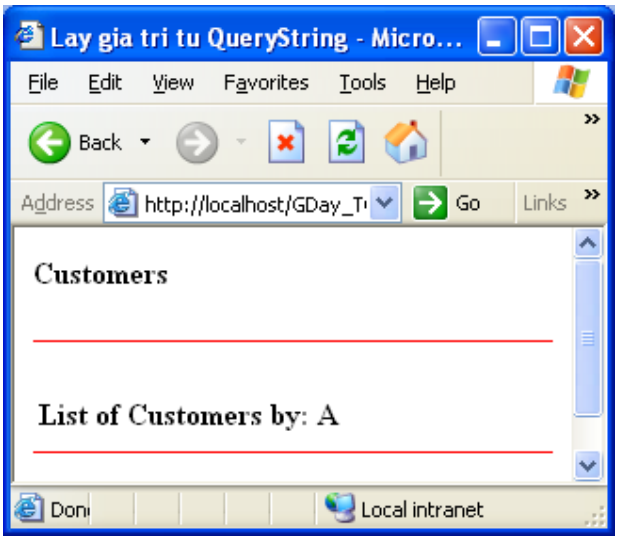
Khi người sử dụng triệu gọi trang ptGET.htm nhập dữ liệu và ấn nút Submit, khi đó trang ptGET.asp được gọi và các giá trị nhập vào trang ptGET.htm sẽ được gửi lên QueryString.

Để lấy giá trị của chuỗi *QueryString* trong trang ptGET.asp, ta sử dụng *Request.QueryString*.

***Ví dụ 5.3.10*. Lấy giá trị từ *QueryString*.**

```
<%dim al,tp
al="All Customers"
tp="Customers"
select case Request.QueryString("page")
  case "cust":
     tp="Customers"
  case "item":
```

```
    tp="Products"
end select
al=Request.QueryString("al")
if al="" then
  al="All " & tp
else
  al="List of " & tp & " by: " & al
end if
%>
<html>
  <head>
    <title>  Lay gia tri tu QueryString </title>
  </head>
  <body>
    <h4><%=tp%></h4>
    <hr size=1 color=red><br>
    <table border=0>
    <tr><td colspan=2><b>
    <%=al%> </td></tr>
    </table>
    <hr size=1 color=red><br>
  </body>
</html>
```



5.3.4. Request.ServerVariables

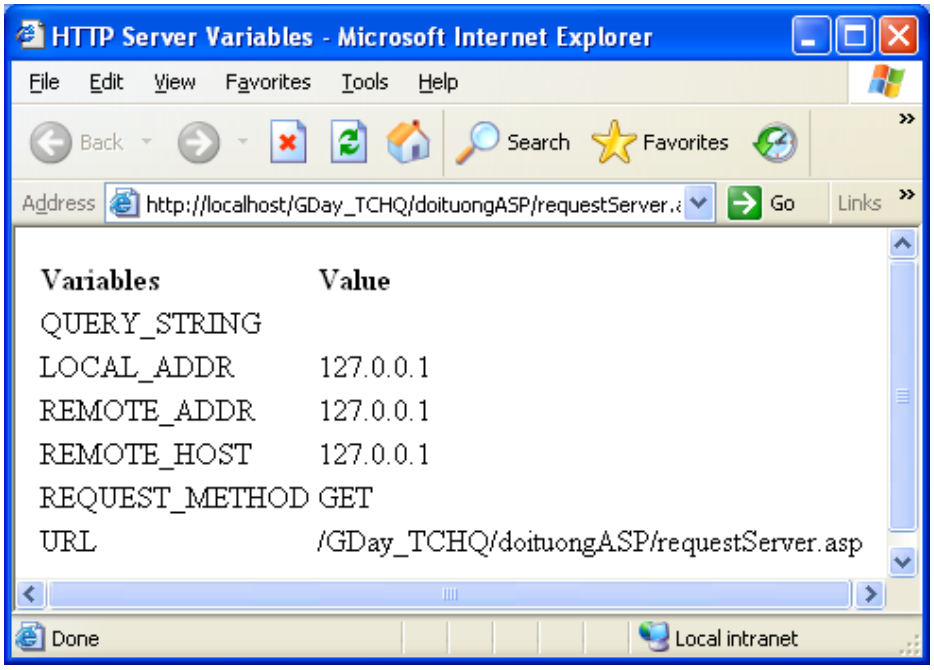
Request.ServerVariables **cho phép ta nhận các giá trị của biến môi trường và các thông tin phần đầu của trang Web:**

- **Ta có thể lấy tên của Web Server, URL, QueryString, địa chỉ và tên miền của máy truy cập vào Website,..**
- **Ngoài ra, ta có thể lấy các thông tin Server như: ServerName, IP, Port, Software,…**

Ví dụ 5.3.11. **Truy cập thông tin người sử dụng**

```
<HTML>
  <HEAD>
    <TITLE>HTTP Server Variables</TITLE>
  </HEAD>
  <BODY>
    <TABLE BORDER=0>
     <TR>
       <TD><B>Variables</B></TD>
       <TD><B>Value</B></TD>
     </TR>
     <TR><TD>QUERY_STRING</TD>
       <TD> <%= Request.ServerVariables("QUERY_STRING") %>
</TD>
     </TR>

     <TR><TD>LOCAL_ADDR </TD>
       <TD><%= Request.ServerVariables("LOCAL_ADDR") %>
</TD>
     </TR>
     <TR><TD>REMOTE_ADDR </TD>
       <TD> <%= Request.ServerVariables("REMOTE_ADDR") %>
</TD>
```

```
        </TR>

        <TR><TD>REMOTE_HOST  </TD>
          <TD><%= Request.ServerVariables("REMOTE_HOST")
%></TD>
        </TR>
        <TR><TD>REQUEST_METHOD</TD>
          <TD> <%=Request.ServerVariables("REQUEST_METHOD")
%></TD>
        </TR>
        <TR><TD>URL </TD>
          <TD>  <%= Request.ServerVariables("URL") %>  </TD>
        </TR>
      </TABLE>
    </BODY>
</HTML>
```



*. Request.Cookies

Cookies là nhóm văn bản mà Website đặt vào một file trên đĩa cứng của Web browser khii Web browser truy cập Website

đó. Cookie dùng để nhận diện khi Web browser này truy cập những lần sau.

Cookie được gửi đến Server cùng với mỗi yêu cầu. Dữ liệu trong Cookie được đặt trong tập hợp Cookies. Nó được truy cập tương tự QueryString và Form. Tuy nhiên, sử dụng đối tượng Request, ta chỉ đọc được giá trị của Cookie mà không thay đổi được nó.

***Request.Cookies* cho phép ta lấy giá trị cookies được gửi trong yêu cầu HTTP. *Cookies* không nên dùng để lưu trữ những thông quan trọng chẳng hạn như *password*.**

Nội dung trong *cookies* hoàn toàn là Text chứa đựng các thông tin phần trên đầu của yêu cầu với cú pháp như sau:

Request.Cookies(cookie)[(Key)|.attribute]

Trong đó:

- ***Cookie:* Chỉ định cookie cần lấy.**
- ***Key:* Tham số tuỳ chọn dùng để lấy giá trị cấp con của *cookie*.**
- ***Attribute:* chỉ định thông tin về *cookie*. Tham số *Attribute* có thể là *HasKeys* (Read-only chỉ định *Cookie* chứa các khoá).**

III). Đối tượng Response

Đối tượng dùng để gửi thông tin ngược trở lại Client từ Server .

Đối tượng Response có rất nhiều thuộc tính như: Status, Expires, ContentType, codepage, Charset,…và phương thức

như Write, Redirect, End, Clear, Flush, AddHeader, BinaryWrite.

Tập hợp	Thuộc tính	Phương thức
Cookies	- Buffer - Charset - ContentType - Expires - ExpiresAbsolute - Status	- AddHeader - AppendToLog - BinaryWrite - Clear - End - Flush - Redirect - Write

Cú pháp:

Response.*tập hợp|thuộc tính|phương thức*

5.4.1. Tập hợp

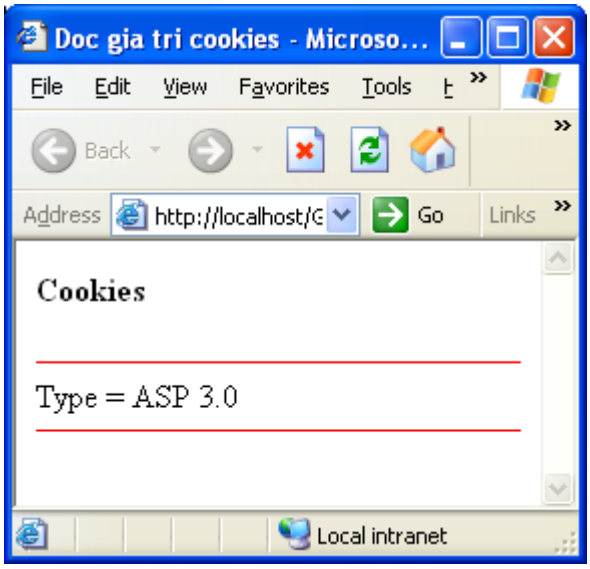
Cookies: Xác định giá trị của cookie gửi cho browser. Các thành phần của tập hợp này đều là giá trị chỉ ghi.

Request object cho phép chúng ta đọc thông tin của cookies khi có 1 yêu cầu được gửii đến. Còn Response object cung cấp khả năng đặt hoặc thay đổi các giá trị của cookies trước khii gửi trả lại cho client.

Ví dụ 5.4.1. **Đọc giá trị từ Cookies**

```
<html>
<head>
```

```
    <title> Doc gia tri cookies </title>
  </head>
  <body>
    <h4>Cookies</h4>
    <hr size=1 color=red>
    <%
    'Ghi giá trị xuống Cookies
    Response.Cookies("Type") = "ASP 3.0"
    Response.Cookies("Type").Expires = "Feb 21, 2005"
    Response.Cookies("Type").Path = "/" %>

    <%
    'Đọc giá trị từ Cookies
    For Each strKey In Request.Cookies
    Response.Write strKey & " = " & _
    Request.Cookies(strKey) & "<BR>"
    If Request.Cookies(strKey).HasKeys Then
     For Each strSubKey In Request.Cookies(strKey)
        Response.Write "->" & strKey & _
        "(" & strSubKey & ") = " & _
        Request.Cookies(strKey)(strSubKey) & "<BR>"
     Next
    End If
    Next %>
    <hr size=1 color=red>
  </body>
</html>
```



Chú ý. **Việc thay đổi cookie với Response object cần được làm trước khi viết mã văn bản hoặc HTML để gửi tới client, nếu không sẽ gây ra lỗi.**

5.4.2. Phương thức

Phương thức	ý nghĩa
AddHeader	Thêm phần đầu HTTP với một giá trị cụ thể để gửi lại cho browser
AppendToLog	Thêm một chuỗi văn bản vào mục nhập nhật ký máy chủ cho yêu cầu từ máy khách hiện thời
Binarywrite	Viết thông tin trực tiếp tới nội dung tín hiệu đáp mà không chuyển đổi kí tự nào
Clear	Xoá bộ đệm
End	Dừng việc xử lí 1 trang và trả lại kết quả hiện tại
Flush	Gửi những dữ liệu có trong bộ đệm tới browser ngay lập tức

Redirect	**Chỉ cho browser kết nối đến một URL khác**
Write	**Gửi text trực tiếp tới trình duyệt phía user**

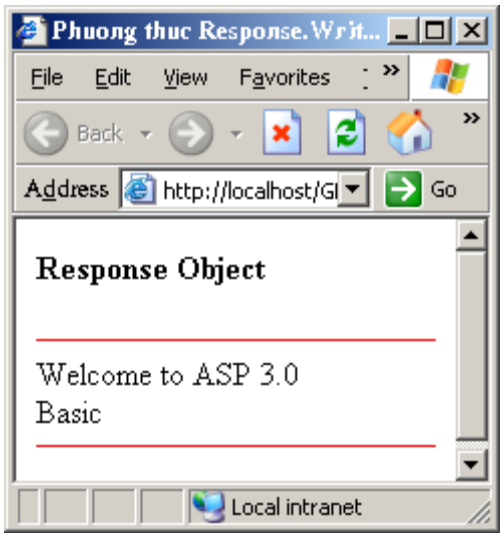
a. Phương thức Write

Phương thức Write có cú pháp như sau cho phép ghi một chuỗi chỉ định ra HTTP hiện hành.

Reseponse.Write(str)

***Ví dụ 5.4.2.1*. Sử dụng phương thức Write.**

```
<html>
  <head>
     <title> Phuong thuc Response.Write </title>
  </head>
  <body>
     <h4>Response Object</h4>
     <hr size=1 color=red>
     <%
     Dim level
     Level="Basic"
     Response.Write("Welcome to ASP 3.0")
     Response.Write("<br>")
     Response.Write(level)
     %>
     <hr size=1 color=red>
  </body>
</html>
```



b. Phương thức Redirect

Phương thức này dùng để chuyển hướng một URL đến một URL khác với cú pháp như sau

Reseponse.Redirect(URL)

Ví dụ 5.4.2.2. **Đăng nhập hệ thống (login.asp)**

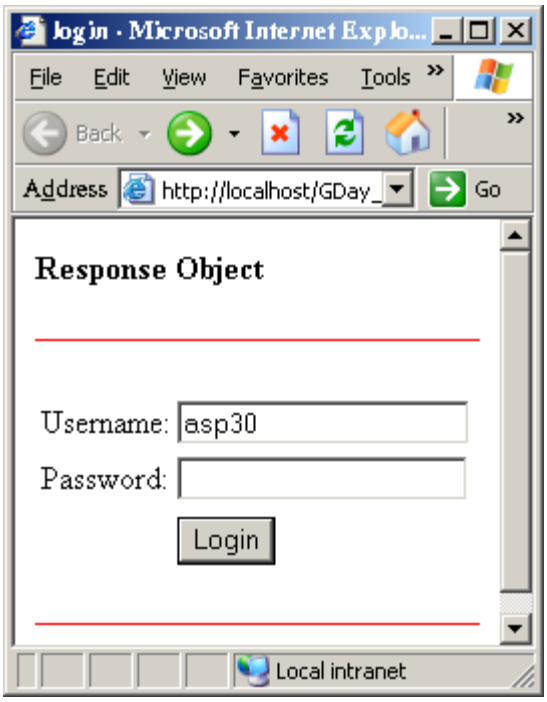
```
<html>
   <head>
      <title>login</title>
   </head>
   <body>
      <h4>Response Object</h4>
      <hr size=1 color=red>
      <form action=ResponseRedirect.asp method=post>
      <table border=0>
      <tr>
      <td>Username: </td>
      <td><input type="textbox" name="username"> </td></tr>
      <tr><td>Password: </td>
      <td><input type="password" name="password"> <tr><td>
      <tr><td></td><td>
      <input type=submit value="Login">
      </td></tr></table>
```

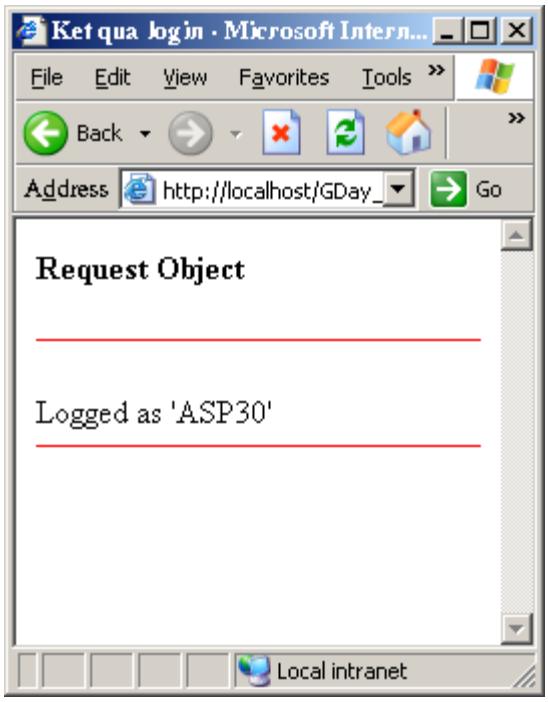
```
        </form>
        <hr size=1 color=red>
    </body>
</html>
```

***Ví dụ 5.4.2.3*. Sử dụng phương thức Redirect (file ResponseRedirect.asp)**

```
<%
Dim u,p
u=Request.Form("username")
p=Request.Form("password")
if u="asp30" then
  if p="12345" then
     Response.Redirect("laygtlogin.asp")
  else
     Response.Redirect("login.asp")
  end if
else
  Response.Redirect("login.asp")
end if
%>
```

Nếu người sử dụng không nhập đúng *username* là ASP30 và *password* là 12345, trang login.asp sẽ được triệu gọi trở lại. Nếu đúng sẽ đưa ra kết quả login.





c. Phương thức End

Phương thức *End* yêu cầu *web Sever* dừng lại không xử lý những dòng lệnh khai báo sau phương thức *End* và trả về kết quả hiện hành.

Reseponse.End

***Ví dụ:* Ta khai báo kết nối CSDL để kiểm tra *Username* và *password* có tồn tại trong bảng *tblUser* hay không, do phát sinh lỗi trong quá trình kết nối CSDL nên chúng ta cần in ra xem chuỗi SQL có đúng với cú pháp hay không.**

Để làm điều này, ta sử dụng phương thức *Write* để in câu SQL và phương thức *End* để kết thúc quá trình thực thi câu lệnh còn lại như ví dụ 5.3.2.4.

***Ví dụ 5.4.2.4*. Sử dụng phương thức End**

Trong file login.asp thay dòng lệnh

<form action=ResponseRedirect.asp method=post>

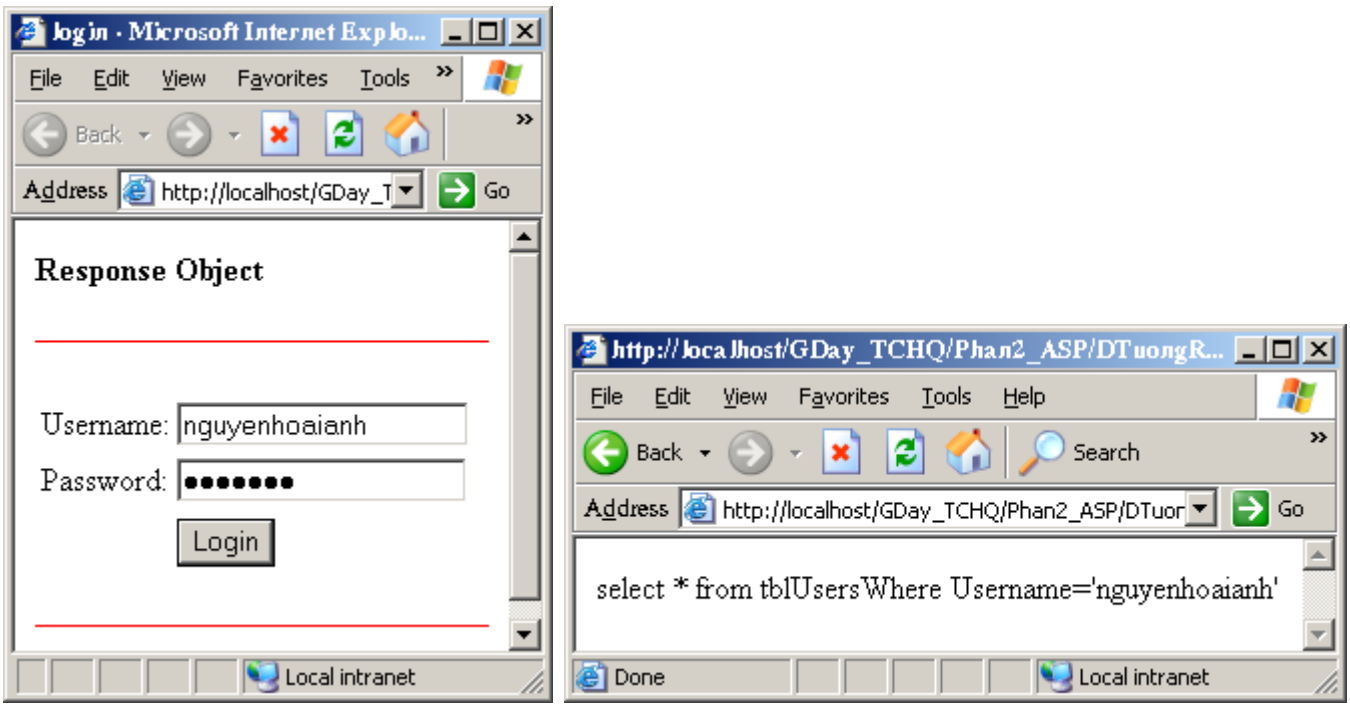
thành

```
<form action=ResponseEnd.asp method=post>
```

và file ResponseEnd.asp sử dụng phương thức End được viết như sau

```
<%
Dim u,p,strSQL
u=Request.Form("username")
if u<>"" then
   strSQL="select * from tblUsers"
   strSQL=strSQL & "Where Username='"
   strSQL=strSQL & u & "' "
   Response.Write strSQL
   Response.End
   p=Request.Form("password")
   Response.Write p
End if
%>
```

Chạy login.asp nhập usename, password vào và nhấn Login, trang ResponseEnd.asp sẽ xuất hiện chuỗi SQL.



d. Phương thức AddHeader

Đặt tên Header cho HTML. Phương thức này phải được gọi trước khi output được gởi cho client trừ khi response.buffer được đặt là true.

Cú pháp

Response.AddHeader Name, Value

Ví dụ

```
<%
Response.AddHeader "MyHeader", "ERROR"
%>
```

e. Phương thức AppendToLog

Thêm xâu vào cuối mục log Web server cho yêu cầu này. String lớn nhất là 80 ký tự

Cú pháp

```
Response.AppendToLog(“dòng thông báo”)
```

Ví dụ

```
<%
Response.AppendToLog("Error in Processing")
%>
```

Output is appended to the end of the IIS log file:
10.78.176.37, - , 03/20/97, 7:55:20, W3SVC, SALES1,
10.78.176.37, Error in Processing

f. Phương thức Clear

Xoá toàn bộ output HTML đang ở trong buffer, không xoá header, chỉ xoá phần thân (body). Nếu buffer không được thiết đặt sẽ gây lỗi. Phải đặt *Response.Buffer=True* (mặc định trong phiên bản 3.0)

Cú pháp

```
Response.Clear
```

Ví dụ

```
<%
Response.Clear
%>
```

f. Phương thức Flush

Gửi thông tin trong buffer ra ngay lập tức. Lưu ý buffer phải đặt *Response.Buffer=True*

Cú pháp

```
Response.Flush
```

Ví dụ

```
<%
Response.Flush
%>
```

5.3.3. Thuộc tính

Thuộc tính	ý nghĩa
Buffer	**Xác định xem một trang có sử dụng bộ đệm để chứa các kết quả được tạo bởi Script hay được gửi ngay tới browser khi từng dòng được tạo ra và nhập vào dải HTML. Giá trị ngầm định là False**
ContentType	**Xác định kiểu nội dung HTTP được trả về. Kiểu ngầm định là Text/HTML**
Expires	**Khoảng thời gian 1 trang Web được lưu giữ trên browser**
ExpiresAbsolute	**Ngày và thời gian 1 trang Web được lưu giữ trên browser**
Status	**Giá trị của dòng trạng thái HTTP trả lại bởi server**
Charset	**Đặt dạng ngôn ngữ sử dụng cho client browser vào phần cuối của đoạn đầu ContentType**

a. ContentType

Thuộc tính *ContentType* chỉ định kiểu nội dung HTTP để trả lại.

Nếu không chỉ định, mặc định *ContentType* là *text* / HTML.

Cú pháp

```
Response. ContentType [= ContentType]
```

Ví dụ

```
<% Response.ContentType = "application/vnd.ms-excel" %>
```

b. CodePage và Charset

Với phúc đáp hiện hành, thuộc tính *Response.CodePage* chỉ định chuỗi mã hoá như thế nào trong đối tượng. Mã của trang là tập bao gồm các số, dấu chấm câu.

Thuộc tính *CharSet* thêm tên của một tập ký tự vào trong header *Content-Type* trong đối tượng *Response*, ví dụ như ISO-8859-13. Mặc định là ISO-LATIN-1

Cú pháp

```
Response. CodePage[=CodePageID]
Response.Charset(CharsetName)
```

CodePage *không cùng ngôn ngữ, chẳng hạn* japanese *và* Hindi *có rất nhiều ký tự không có trong* English *và* German *để trình bày một kí tự. Thuộc tính* CodePage *cho phép đọc và ghi* (read / write).

Lưu ý rằng, *Response.CodePage* có hiệu lực trên một trang mà ta khai báo nó còn Session.CodePage có hiệu lực trên một phiên làm việc.

Ví dụ 1. Khai báo trình bày tiếng Việt sử dụng Unicode:

```
<%
Response.CodePage 65001
```

```
Response.Charset “utf-8”
%>
```

Ví dụ 2.

```
<% Response.Charset("MS_Kanji") %>
```

Trong ví dụ này, nếu header là: content-type: text/html

Thì header sẽ trở thành: content-type: text/html; charset = MS_Kanji

c. Expires

Thuộc tính Expires chỉ định thời gian tính bằng phút trước khi trang đó được lưu trữ trong bộ nhớ Cache của trình duyệt. Người sử dụng có thể trở lại các trang trước đó khi khoảng thời gian đó chưa hết hạn.

Cú pháp

```
Response.Expires[=number]
```

Ví dụ 1. Khai báo trình bày tiếng Việt sử dụng Unicode:

```
<% Response.Expires = 0 %>
<% Response.Expires = 15 %>
```

- **Trường hợp 1: bất kỳ khi nào gọi tới trang đó, nó luôn luôn được refresh**
- **Trường hợp 2: Nếu trở lại trang đó trước 15 phút, trang hiển thị sẽ là trang trong cache**

d. Buffer

Buffer là nơi lưu giữ tạm thời trước khi chuyển cho trình duyệt. Buffer chỉ ra output của trang có được lưu trong buffer hay

không. Sẽ không có sự gửi lại cho trình duyệt cho đến khi tất cả các script được xử lý xong hoặc có lời gọi phương thức Flush hay End.

Thuộc tính này không thể thay đổi khi server đã gửi thông tin cho trình duyệt [redacted]hải đặt ở dòng đầu tiên trong file .asp

Cú pháp

```
Response.Buffer = True|False
```

Ví dụ.

```
<%
Response.Buffer = TRUE
x=0
Do
   x = x+1
   Response.Write x & "<BR>"
Loop %>
```

e. Cache Control

Có hay không cho phép máy chủ proxy được cache output của trang .asp hay không

Máy chủ proxy dùng để đẩy nhanh tốc độ truy nhập tới trang Web nào đó bằng cách lưu lại một bản của trang Web trong cache

- **Nếu CacheControl được đặt là "Public", thì cho phép cache**
- **Nếu CacheControl đặt là "Private", không cho phép cache**

Cú pháp

```
Response.CacheControl = Public|Private
```

Ví dụ.

```
<% Response.CacheControl = "Public" %>
<% Response.CacheControl = "Private" %>
```

f. ExpiresAbsolute

Xác định ngày và thời gian chính xác một trang sẽ hết hạn

Cú pháp

```
Response.ExpiresAbsolute = #<thời gian>#
```

Ví dụ.

```
<%Response.ExpiresAbsolute=#May 15, 1999
18:00:00#%>
```

g. IsClientConnected

Xác định xem Clitent đã ngừng kết nối với Server từ Response.Write cuối cùng.

Thuộc tính này đặc biệt có ý nghĩa để server không phải tiếp tục thực hiện chuyển những thông tin client không yêu cầu.

Cú pháp

```
Response.IsClientConnected
```

Ví dụ.

```
<%
 'Check to see if the client is connected.
 If Not Response.IsClientConnected Then
 'Get the sessionid to send to the shutdown function.
  Shutdownid = Session.SessionID
 'Perform shutdown processing.
  Shutdown(Shutdownid)
 End If
%>
```

h. Status

Dòng trạng thái do Server trả lại

Ví dụ.

```
<%
   IPAddress = Request.ServerVariables("REMOTE_ADDR")
   If IPAddress <> "208.5.64.223" Then
      Response.Status = "403 Access Forbidden"
      Response.Write Response.Status
      Response.End
   End If
%>
<HTML>
   <HEAD>
      <TITLE> Status </TITLE>
    </HEAD>
   <BODY>
      You have accessed this page through the IP Address of 208.5.64.223.
   </BODY>
</HTML>
```

Kết quả sau sẽ được trả về nếu địa chỉ IP trên máy Client là 208.5.64.223:

You have accessed this page through the IP Address of 208.5.64.223.

Ngược lại nếu địa chỉ IP của Client không là 208.5.64.223 kết quả sau sẽ trả về:

403 Access Forbidden

5.3.4. So sánh tập hợp Form và QueryString

Khi chúng ta sử dụng thẻ <FORM> trong một trang, ta có thể đặt thuộc tính METHOD của <FORM> là POST hay GET. Nếu chúng ta sử dụng GET(hay bỏ qua vì GET là giá trị mặc

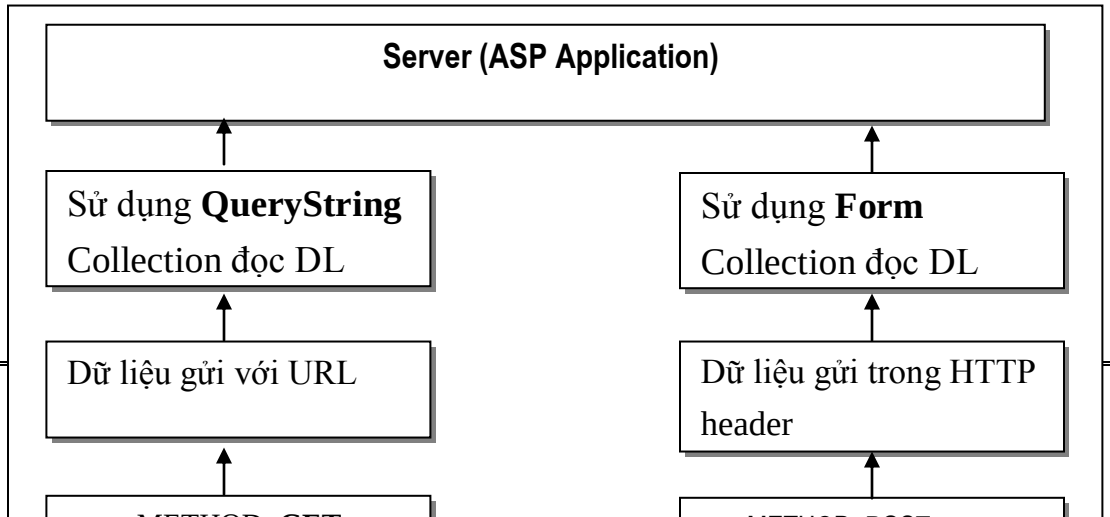
định cùa METHOD), trình duyệt sẽ lấy các giá trị trong tất cả các control để xây dựng thành QueryString và gắn vào URL của trang được yêu cầu khi Submit trang hiện tại. Khi trang này đến Server các giá trị của nó nằm ở Collection Request.QueryString.

Ngược lại, nếu sử dụng phương thức POST, trình duyệt sẽ đưa tất cả các giá trị vào trong HTTP header gửi đến Server và các giá trị này có thể truy xuất qua Collection Request.Form

Nói chung, ta nên sử dụng phương thức Post trong tất cả các form HTML.

- **Thứ nhất, chiều dài chuỗi của URL bị giới hạn nên nếu dùng QueryString sẽ có nguy cơ bị tràn và bị cắt bớt.**
- **Thứ hai, QueryString đưa các giá trị tường minh vào URL, và sẽ được ghi lại trong file log khi đi qua các Server, không bảo mật thông tin.**

Sự khác nhau giữa hai phương thức gửi dữ liệu từ Client đến Server được chỉ ra trong sơ đồ sau:



Ví dụ sử dụng phương thức Post để gửi dữ liệu

Nội dung file1.htm

```
<HTML>
   <HEAD>
      <TITLE> Post </TITLE>
    </HEAD>
    <BODY>
      <form action= ”file2.asp” method="POST">
      Name:<input type="Text" name="Name"
maxlength=”20"><br>
      Company:<input type="Text" name="Company"
maxlength="20"><br>
      Position:<input type="Text" name="Position"
maxlength="20"><br>
      Address:<textarea name="Address"
rows="3"></textarea><br>
      Phone:<input type="Text" name="Phone"
maxlength="20"><br>
```

```
      <input type="Submit" name="Submit" value="Submit">
      </form>
   </BODY>
</HTML>
```

Nội dung file2.asp

```
<HTML>
   <HEAD>
      <TITLE> Post </TITLE>
    </HEAD>
    <BODY>
      <% Response.Write Request.Form("Name")%>
      works for <% Response.Write Request.Form("Company")
%>
       at address <% Response.Write Request.Form("Address") %>
       as a <% Response.Write Request.Form("Position") %>.
   </BODY>
</HTML>
```

Kết quả:

Jane Doe works for ISSI at address 5609 Kington Pike Knoville, TN as a Web Designer.

Ví dụ sử dụng phương thức Get để gửi dữ liệu:

Nội dung file1.asp

```
<HTML>
   <HEAD>
      <TITLE> Get </TITLE>
    </HEAD>
    <BODY>
    <A HREF="File2.asp?language=ASP&type=general">Query
    sample</A>
    </BODY>
```

```
</HTML>
```

Nội dung file2.asp:

```
<%

    For Each item In Request.QueryString
     Response.Write item & " = " & Request.QueryString(item) &
    "<BR>"
    Next
%>
```

Kết quả

language = ASP
type = general

Hoặc file2.asp có thể viết

```
<%
Response.Write "language =" & Request.QueryString("language")&
"<BR>"
Response.Write "type =" & Request.QueryString("type")& "<BR>"
%>
```

Chú ý: Khi lấy giá trị theo 2 tập hợp là Form hoặc QueryString ta chỉ cần chỉ ra

Request(“tên_biến”)

Chẳng hạn: Request.Form(“Name”) có thể thay bằng Request(“Name”)

Mỗi đối tượng đều có các phương thức đi kèm. Cú pháp chung để gửi thông điệp cho các đối tượng hoàn toàn bình thường: Object.Method parameters. Ở đây parameters có thể là biến, dữ liệu, chuỗi hoặc URL tuỳ thuộc vào phương thức Method.

Ngoài ra còn có đối tượng ObjectContext: để chấp nhận hoặc từ chối một giao tác. Đối tượng này được quản lí bởi MTS và có thể được khởi xướng nhờ một câu lệnh script chứa trong một trang ASP. Khi một trang ASP chứa @TRANSACTION thì trang đó sẽ được chạy trong giao tác đó và chỉ kết thúc khi giao tác đó đã thành công hoặc thất bại. Và đối tượng ASPError chứa thông tin về lỗi xuất hiện trong mã lệnh trong trang ASP.

5.4. Đối tượng Session

Đối tượng Sesion được dùng để lưu trữ thông tin cần thiết cho một phiên làm việc của người dùng cụ thể. Các biến lưu trữ trong đối tượng Session không bị mất khi người dùng truy cập các trang Web khác trong ứng dụng. Thay vào đó các biến này tồn tại trong toàn bộ phiên làm việc của người dùng.

Khi một người dùng mới yêu cầu một trang Web từ ứng dụng, Web server tự động tạo một đối tượng Session và server sẽ phá huỷ đối tượng Session khi phiên làm việc kết thúc hoặc bị huỷ bỏ. Có thể đặt thời gian tồn tại cho một phiên làm việc tuy nhiên giá trị ngầm định cho một phiên làm việc tồn tại là 20 phút.

5.4.1. Tập hợp

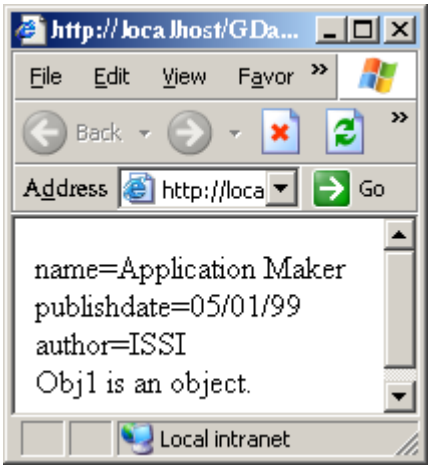
a. Session.Contents(Key)

Chứa danh sách các mục đã khởi tạo và thêm vào bằng đối tượng session. Không phải khởi tạo bằng thẻ <object>

<u>Ví dụ</u>:

```
<%
  Session("name") = "Application Maker"
  Session("publishdate") = "05/01/99"
  Session("author") = "ISSI"
  Set Session("Obj1") = Server.CreateObject("ADODB.Connection")

  For Each Item in Session.Contents
    If IsObject( Session.Contents(Item)) Then
      Response.Write Item & " is an object.<BR>"
    Else
      Response.Write Item & "=" & Session.Contents(Item) & "<BR>"
    End If
  Next
%>
```

Kết quả:



b. Session.StaticObjects hoặc Session.StaticObjects(Key)

Chứa danh sách các mục vừa được khởi tạo và đưa vào đối tượng Session nhờ sử dụng thẻ HTML <OBJECT></OBJECT>

Ví dụ:

----------Global.asa----------

<OBJECT RUNAT=Server SCOPE=Session ID=MyInfo PROGID=

```
“MSWC.MyInfo”>
</OBJECT>
<OBJECT RUNAT=Server SCOPE=Session ID=MyConnection

PROGID="ADODB.Connection"></OBJECT>
<OBJECT RUNAT=Server SCOPE=Session ID=MyADRot
PROGID="MSWC.ADRotator">
</OBJECT>
```

----------File.asp----------

```
<%
For Each Item In Session.StaticObjects
Response.Write Item & "<BR>"
Next
%>
```

Kết quả:

MyInfo
MyConnection
MyADRot

5.4.2. Sự kiện

Cũng như đối tượng Application, đối tượng Session có hai sự kiện là Session_OnStart(được kích hoạt khi mỗi khi một phiên làm việc bắt đầu) và Session_OnEnd (được kích hoạt khi kết thúc một phiên làm việc)

Cú pháp

```
<SCRIPT LANGUAGE=VBScript  RUNAT=Server>
Sub Session_OnStart
```

```
' Nơi chèn đoạn Script cần thiết cho việc khởi tạo một phiên làm việc
End Sub
Sub Session_OnEnd
' Nơi chèn đoạn Script cần thiết khi một phiên làm việc kết thúc
End Sub
</SCRIPT>
```

5.4.3. Thuộc tính

Để khởi tạo đối tượng Session, ta sử dụng các thuộc tính hay *Collection* như: *Contents, SessionID, TimeOut* và *CodePage*.

*** *Contents Collection***

***Contents Collection* cho phép ta gán hay truy cập đối tượng Session với cú pháp như sau:**

```
Session.Contents("name")=Value
Variable=Session.Contents("name")
```

***Chú ý:* Ta có thể sử dụng trực tiếp Session("name") thay vì khai báo Session.Contents("name")**

*** *Thuộc tính SessionID***

Khi người sử dụng truy cập vào Web Site, nếu đối tượng Session được tạo, Web server sẽ cung cấp cho trình khách tương ứng một nhận dạng (Identify) nduy nhất có kiểu Long.

Giả sử có n người sử dụng truy cập vào Web Site, có thể có n giá trị tương ứng cho mỗi phiên làm việc và các giá trị này không lặp lại.

Bằng cách sử dụng thuộc tính SessionID của đối tượng Session, ta có thể lấy được giá trị này: Session.SessionID.

Chú ý: **Định dạng này sẽ được thu hồi và cấp cho trình khách mỗi khi các Session của trình khác này hết hạn sử dụng.**

**** Thuộc tính TimeOut***

Thuộc tính TimeOut chỉ định thời gian sẽ hết hạn của đối tượng Session tính bằng phút nếu người sử dụng không làm tươi hoặc yêu cầu (request) trang Web.

```
Session.TimeOut=Value
Variable=Session.TimeOut
```

**** Thuộc tính CodePage***

Chỉ định kiểu ký tự thể hiện trên trang ASP khi trình bày kết quả cho người sử dụng. Trong khi có Response.CodePage có hiệu lực trên từng trang thì Session.CodePage có hiệu lực trên một phiên làm việc, cú pháp như sau

```
Session.CodePage(=codepageID)
```

Trong đó, codepageID là số nguyên ứng với định dạng chuỗi trên trang. Ta có thể tìm thấy danh sách các số này trong cột FamilyCodePage tại địa chỉ: http://msdn.microsoft.com

5.4.4. Phương thức

Đối tượng Session cung cấp các phương thức chính như: Remove, RemoveAll và Abandon để loại bỏ các đối tượng được tạo ra trong phiên làm việc.

**** Phương thức Remove***

Để loại bỏ một Session đang tồn tại, ta có thể sử dung phương thức Remove với cú pháp như sau:

```
Session.Contents.Remove(para)
```

Trong đó: para có thể là tên hay chỉ mục của một Session đang tồn tại.

Ví dụ: **' Khai báo Session cùng với giá trị**

```
Session.Contents(userid)=12
' Loại bỏ Session có tên là userid là
Session.Contents.Remove(userid)
```

**** Phương thức RemoveAll***

Trong trường hợp muốn loại bỏ toàn bộ các Session đang tồn tại, ta sử dụng phương thức RemoveAll.

```
Session.Contents.RemoveAll()
```

**** Phương thức Abandon***

Khi sử dụng hai phương thức Remove hay RemoveAll để xoá đối tượng Session khỏi Session Collection, mặc dù đối tượng Session đã bị xoá song tài nguyên cung cấp cho đối tượng Session này chưa huỷ. Trong trường hợp ta muốn huỷ tất cả các Session và giải phóng bộ nhớ mà Session đã chiếm, ta sử dụng phương thức Abandon sau khi gọi phương thức Remove hay RemoveAll.

Cú pháp: `Session.Abandon`

Chú ý: **Có thể lưu trữ các giá trị trong đối tượng Session. Thông tin lưu trữ trong đối tượng Session có phạm vi phiên làm việc và có thể sử dụng được trong suốt một phiên làm việc.**

5.5. Đối tượng Application

Một ứng dụng bao gồm các file có thể truy nhập thông qua một thư mục ảo xác định và các thư mục con của nó.

Đối tượng Application thể hiện toàn bộ một ứng dụng ASP. Chúng ta có thể sử dụng ứng dụng này để chia sẻ thông tin cho tất cả các người dùng trong một ứng dụng.

Đối tượng Application được bắt đầu khi có một yêu cầu đầu tiên một trang web bất kỳ từ thư mục ảo tại Web server và tồn tại cho đến khi Webserver ngừng hoạt động.

5.5.1. Thuộc tính

Thuộc tính của đối tượng Application bao gồm Application.Contents Collection và Application.StaticObjects Collection

**** Thuộc tính Contents***

Khi muốn khởi tạo đối tượng Application và gán giá trị, ta sử dụng cú pháp sau:

```
Application.Contents("key")=Value
```

Ví dụ 5.5.1. **Khai báo đối tượng Application có tên là Counter để đếm số người truy cập và khởi tạo giá trị này bằng 0 trong tập tin Global.asa**

```
Sub Application_OnStart()
   Application("CurrentUser")=0
   Application.Contents("Counter")=0
End Sub
```

Sau đó mỗi lần người sử dụng mở mới một trình duyệt (một phiên làm việc mới được tạo ra) tăng giá trị đối tượng này 1.

```
Sub Session_OnStart()
 Application("CurrentUser") = Application("CurrentUser") +1
 Application.Contents("Counter")=Application.Contents("Counter")
+1
 //Response.Write "New Session is created"
End Sub
```

Ví dụ 5.5.2.

```
<% Application("name") = "Application Maker"
Application("publishdate") = "05/15/01"
Application("author") = "DevGuru"
Set Application("Obj1") =
Server.CreateObject("ADODB.Connection")
For Each Item in Application.Contents
   If IsObject( Application.Contents(Item)) Then
    Response.Write Item & " is an object.<BR>"
   Else
    Response.Write Item & "=" & Application.Contents(Item)
& "<BR>"
   End If
Next
%>
```

Kết quả:

name=Application Maker

publishdate=05/15/01

author=DevGuru

OBJ1 is an object

Phương thức của tập hợp Contents

- **Application.Contents.Remove (Name|Integer)**

Loại bỏ mục nào đó trong collection Application.Contents

- **Name chỉ ra tên mục sẽ xoá, bao bằng cặp dấu nháy.**
- **Integer chỉ ra vị trí mục trong collection sẽ được xoá. Giá trị này bắt đầu từ 1**

Ví dụ 5.5.3.

```
<%
Application("name") = "Application Maker"
Application("publishdate") = "05/15/01"
Application("author") = "DevGuru"
Set Application("Obj1") =
Server.CreateObject("ADODB.Connection")
Application.Contents.Remove(1)
Application.Contents.Remove("publishdate")
      For Each Item in Application.Contents
   If IsObject(Application.Contents(Item)) Then
    Response.Write Item & " is an object.<BR>"
   Else
    Response.Write Item & "=" &Application.Contents(Item) & "<BR>"
   End If
Next
%>
```

Kết quả:

author=DevGuru

Obj1 is an object.

- **Application.Contents.RemoveAll**

Loại bỏ tất cả các mục trong collection Application.Contents

Thêm cặp dấu ngoặc ()

```
<%Application.Contents.RemoveAll( )%>
```

**** Thuộc tính StaticObjects***

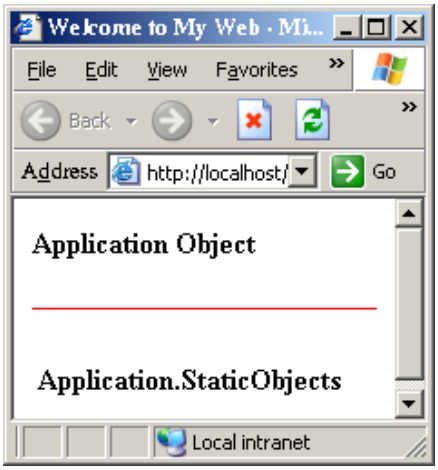
Tập thuộc tính StaticObjects chứa đựng tất cả các đối tượng được tạo bằng thẻ <Object> trong phạm vị đối tượng Application với cú pháp như sau:

Application.StaticObjects(key)

Ta có thể sử dụng Collection để lấy giá trị chỉ định của một thông tin cho đối tượng hay truy cập đến nhiều thuộc tính bằng vòng lặp trên tất cả các đối tượng.

***Ví dụ 5.5.4*. Truy cập thuộc tính StaticObjects.**

```
<html>
   <head>
      <title> Welcome to My Web </title>
   </head>
   <body>
      <h4>Application Object</h4>
      <hr size=1 color=red><br>
      <table border=0>
      <tr><td><b>Application.StaticObjects</b></td></tr>
      <%
         Dim strKey
         For Each strKey In Application.StaticObjects
            Response.Write "<tr><td><b>" & _
             strKey & " = (object)</b></td></tr>"
          Next
      %>
      </table>
   </body>
</html>
```



***Ví dụ 5.5.5*. Truy cập thuộc tính StaticObjects.**

----------Global.asa----------

```
<OBJECT RUNAT=Server SCOPE=Application ID=MyInfo
PROGID="MSWC.MyInfo">
</OBJECT>

<OBJECT RUNAT=Server SCOPE=Application ID=MyConnection
PROGID="ADODB.Connection">
</OBJECT>

<OBJECT RUNAT=Server SCOPE=Application ID=MyADRot
PROGID="MSWC.ADRotator">
</OBJECT>
```

----------File.asp----------

```
<%
For Each Item In Application.StaticObjects
Response.Write Item & "<BR>"
Next
%>
```

Kết quả:

MyInfo

MyConnection
MyADRot

5.5.2. Sự kiện

Ứng với hai hoạt động bắt đầu và kết thúc một đối tượng Application ta có hai sự kiện trong đối tượng Application, đó là : Application_OnStart (khởi tạo các thông tin phục vụ cho một ứng dụng khi ứng dụng bắt đầu) và Application_OnEnd (được kích hoạt khi ứng dụng kết thúc)

Cú pháp của sự kiện Application_OnStart:

```
<SCRIPT LANGUAGE=VBScript RUNAT=Server>
Sub Application_OnStart
 ' Nơi chèn đoạn Script cần thiết cho việc khởi tạo một ứng dụng
End Sub
Sub Application_OnEnd
 ' Nơi chèn đoạn Scipt cần thiết cho việc kết thúc một ứng dụng
End Sub
</SCRIPT>
```

Chú ý. Phải khai báo ngôn ngữ script được sử dụng trong các đoạn script sự kiện trong dòng đầu tiên của file global.asa

Các đối tượng của ASP trong phần này chỉ có SERVER và APPLICATION

----------Global.asa----------

```
<script Language="VBScript" RUNAT=Server>
Sub Application_OnEnd()
```

```
End Sub
Sub Application_OnStart()
    Application("NumSession") = 0
    Application("NumVisited") = 0
    Session.Timeout = 10
End Sub
Sub Session_OnEnd()
  Application("NumSession") = Application("NumSession") - 1
End Sub
Sub Session_OnStart()
  Application("NumSession") = Application("NumSession") + 1
  Application("NumVisited") = Application("NumVisited") + 1
End Sub
</script>
```

----------File1.asp----------

```
<% Response.Write "You are " & Application("NumSession") & "
of " & Application("NumVisited") & " users."%>
```

Kết quả:

You are 1 of 1 users.

5.5.3. Phương thức

Vì đối tượng Application có thể được chia sẻ thông tin giữa các người dùng do đó để đảm bảo những người dùng đó không thể cùng một lúc thay đổi nội dung của một biến trong đối tượng Application nó có hai phương thức là Lock và Unlock.

Phương thức	ý nghĩa

Lock	**Phương thức Lock ngăn cản các client khác cùng một lúc thay đổi giá trị của một biến do đối tượng Application lưu trữ**
Unlock	**Phương thức Unlock cho phép các client có thể sửa đổi các thuộc tính của đối tượng Application .**

**** Phương thức Lock***

Phương thức dùng để ngăn chặn người sử dụng thay đổi giá trị của biến lưu trong đối tượng Application và đảm bảo rằng chỉ một người sử dụngtại thời điểm đó có thể truy cập và cập nhật đối tượng này.

Application.Lock()

Nếu ta không gọi phương thức Unlock ngay sau đó, Server sẽ không cho phép người sử dụng khai thác cập nhật giá trị trên những đối tượng Application đã bị khoá cho đến khi kết thúc phiên làm việc hay hết thời gian sử dụng.

**** Phương thức UnLock***

Phương thức Unlock dùng để phục hồi lại những đối tượng Application đã bị khoá trước đó bằng phương thức Lock, nhằm cho phép nhiều người sử dụng có thể thay đổi giá trị của biến lưu trong đối tượng này và bảo đảm rằng có nhiều người sử dụng tại thời điểm đó có thể truy cập và nhập nhật đối tượng này.

Application.UnLock()

5.6. Đối tượng Server

Cho phép truy nhập tới các phương thức và thuộc tính trên máy chủ.

Tập hợp	Thuộc tính	Phương thức
	ScripTimeOut	- CreateObject - HTMLEncode - MapPath - URLEncode

Cú pháp:

Server.*Thuộc tính|Phương thức*

5.6.1. Thuộc tính ScriptTimeout

Server.ScriptTimeout

Đó là thời gian tối đa để trang script chạy trên máy chủ. Nếu không đặt giá trị cho thuộc tính này thì giá trị mặc định của nó là 90 giây.

Nếu script nhập vào một vòng lặp vô hạn thì server sẽ kết thúc script đó để tránh bị overload bởi việc chạy liên tục các tiến trình sinh ra. Thời gian trước khi script bị kết thúc được định nghĩa bởi thuộc tính này.

```
<% Server.ScriptTimeout = 150 %>
```

Ta có thể lấy được giá trị của thuộc tính ScriptTimeout bằng cách:

```
<% timeout = Server.ScriptTimeout %>
```

5.6.2. Phương thức

Phương thức	ý nghĩa
CreateObject	Tạo một thể hiện của đối tượng cụ thể trên Server
Execute	Cho phép gọi trang ASP khác trong một trang ASP
GetLastError	Mô tả đối tượng lỗi ASP, chỉ có ý nghĩa trước khi file asp gửi nội dung tới Client
HTMLEncode	Gắn một đoạn mã HTML vào một xâu đã được định dạng.
MapPath	Xác định đường dẫn vật lý trên máy chủ khi xét đết đường dẫn ảo.
Transfer	Chuyển tới trang ASP khác từ một trang ASP. Thông tin trạng thái hiện tại trong trang đầu tiên sẽ được chuyển tới trang thứ hai
URLEncode	Cho phép gắn một đoạn mã URL

**** Phương thức CreateObject***

Phương thức này (không có gía trị trả về) dùng để tạo đối tượng thành phần trên *Server* với cú pháp.

```
Server.CreateObject(progId)
```

Trong đó, tham số *Progid* chỉ định loại đối tượng cần tạo với định dạng Vendor.]Component[.Version].

Chẳng hạn, ta tạo đối tượng ADODB. *Connection*, ADODB.*RecordSet*, MSWC.*AdRotator*,... như sau:

```
<%
Set myAd=Server.CreateObject(MSWC.AdRottator)
Set myCon=Server.CreateObject(ADODB.Connection)
Set myRst=Server.CreateObject(ADODB.RecordSet)
%>
```

Chẳng hạn đoạn mã sau dùng để kết nối CSDL

```
<% Set myconn =
Server.CreateObject("ADODB.Connection") %>
```

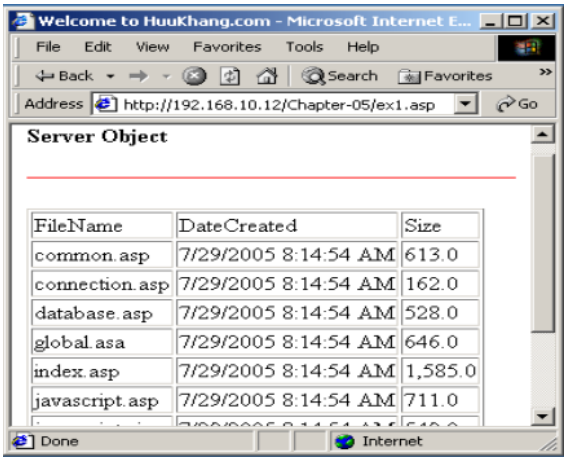
Chẳng hạn, ta sử dụng phương thức này để khởi tạo đối tượng *Scripting.FileSystemObject* đọc danh sách tập tin trong mục của ứng dụng Web như ví dụ 5.6.1.

Ví dụ 5.6.1. **Sử dụng phương thức CreateObject**

```
<html>
   <head>
      <title> Welcome to My Web </title>
   </head>
   <body>
         <h4>Server Object</h4>
         <hr size=1 color=red><br>
         <table border=1>
         <tr  class=text>
         <td>FileName</td><td>DateCreated</td>
         <td>Size</td></tr>
         <%
         Dim fs,f,ff,path
         Set fs = Server.CreateObject("Scripting.FileSystemObject")
```

```
        path=request.ServerVariables("APPL_PHYSICAL_PATH")
        Set f = fs.GetFolder(path)
        set ff = f.files
        For Each FileName In ff
        %>
        <tr  class=text>
        <td><%=  FileName.name%></a></td>
        <td><%= FileName.DateCreated%></td>
        <td><%= formatnumber(FileName.size,1)%></td></tr>
        <%next%>
        </td></tr>
        </table>
    </body>
</html>
```

Khi gọi trang *CreartObject.asp* trên trình duyệt, ta sẽ nhận được danh sách các tập tin trong thư mục ứng dụng *Web* như sau



**** Phương thức Execute***

Phương thức *Execute* (không có giá trị trả về) dùng để gọi một tập tin *.asp* và xử lí nó như một phần của kịch bản ASP với cú pháp như sau:

Server.Execute(Path)

trong đó, tham số *Path* là đường dẫn và tên tậo tin *asp* cần thực thi. Nói cách khác, phương thức Execute tương tự như gọi thủ tục trong nhiều ngôn ngữ lập trình khác.

**** Phương thức HTMLEncode***

Phương thức HTMLEnCode dùng để mã hoá HTML thành chuỗi với cú pháp như sau:

Server.HTMLEncode(str)

Khi mã hoá chuỗi HTML sang chuỗi bình thương nếu gặp các ký tự như sau sẽ chuyển sang các ký tự tương ứng:

- **Ký tự < chuyển thành &lt**
- **Ký tự > chuyển thành &gt**
- **Ký tự & chuyển thành &amp**
- **Dấu nháy đôi " chuyển thành &quot**

Khi ta khai báo một thẻ TextArea cho phép người sử dụng nhập đoạn mã có thẻ HTML bên trong nội dung, sau khi người sử dụng nhấn nút Submit sang trang kế tiếp, thay vì trình bày nội dung đó trên màn hình, chúng xuất hiện thẻ HTML trên trang Web.

Để xem chuỗi nhập có thẻ HTML trình bày như những gì ta khai báo trên trang Web, ta phải sử dụng phương thức HTMLEncode.

Ví dụ

```
<% Response.Write Server.HTMLEncode("The tag for a table is:
<Table>")%>
```

Kết quả:

The tag for a table is: <Table>

Kết quả ở Browser:

The tag for a table is: <Table>

**** Phương thức MapPath***

Phương thức MapPath ánh xạ đường dẫn ảo thành đường dẫn vật lý cho một tập tin trên Server với cú pháp như sau:

```
Server.MapPath(path)
```

trong đó, tham số path là chuỗi thực hiện đường dẫn tương đối của tập tin trên ứng dụng Web, phương thức sẽ trả về chuỗi hiện thực đường dẫn vật lý của tập tin đó.

Phương thức này không kiểm tra sự tồn tại thực sự của đường dẫn. Nếu bắt đầu bằng dấu / hoặc \ [redacted]rờng dẫn ảo. Còn không bắt đầu bằng ký tự đó [redacted]rờng dẫn tương đối.

Ví dụ 5.6.2.

```
<html>
   <head>
      <title> Welcome to My Web </title>
   </head>
```

```
<BODY>
The path of this file is <% Response.Write
Server.MapPath("test.asp")%>
The path of the file1 is <% Response.Write
Server.MapPath("\test.asp")%>
The path of the file2 is <% Response.Write
Server.MapPath("test\test.asp") %>
The path of the file3 is <% Response.Write Server.MapPath("\")
%>
</BODY>
</HTML>
```

Kết quả:

The path of this file is C:\VANBANG2\ASP\Example\test.asp

The path of the file1 is d:\inetpub\wwwroot\test.asp

The path of the file2 is

C:\VANBANG2\ASP\Example\test\test.asp

The path of the file3 is d:\inetpub\wwwroot

**** Phương thức Transfer***

Phương thức transfer gửi tất cả các thông tin mà nó đã xử lý từ trang ASP hiện hành sang trang ASP khác với cú pháp như sau:

```
Server.Transfer(path)
```

Trong đó, tham số Path là đường dẫn của tập tin ASP cần chuyển điều khiển. Ví dụ, chúng ta khai báo ttrang ASP có sử dụng phương thức Transfer như ví dụ 5.6.2.

Ví dụ 5.6.3: **Sử dụng phương thức Transfer.**

```
<html>
  <head>
    <title> Welcome to HuuKhang.com </title>
```

```
</head>
<body>
<h4>Server Object</h4>
<hr size=1 color=red><br>
<b>Server.Transfer</b>
<table border=0>
<tr>
<td valign=top>Transfer</td></tr>
<tr><td valign=top>Request.QueryString:
<%=Request.QueryString("i")%></td></tr>
<tr><td>
<%=Server.Transfer("ex6-1.asp")%></td>
</tr>
</table>
</body>
</html>
```

Ví dụ 5.6.4:

----------CallingAsp.asp----------

```
<%
Application("name") = "Application Maker"
Application("publishdate") = "05/15/01"
Application("author") = "DevGuru"
Set Application("Obj1") =
Server.CreateObject("ADODB.Connection")
Server.Transfer("CalledAsp.asp")
%>
```

----------CalledAsp.asp----------

```
<%;
Response.Write "Output from CalledAsp.asp"
For Each Item in Application.Contents
```

```
   If IsObject( Application.Contents(Item)) Then
     Response.Write Item & " is an object.<BR>"
   Else
     Response.Write Item & "=" & Application.Contents(Item) &
"<BR>"
   End If
Next
%>
```

Kết quả:

```
Kết quả từ CalledAsp.asp
name=Application Maker
publishdate=05/15/01
author=DevGuru
OBJ1 is an object.
```

**** Phương thức URLEncode***

Tương tự như phương thức HTMLEncode, phương thức URLEncode để dùng mã hoá URL thành chuỗi với cú pháp như sau:

```
Server.URLEncode(str)
```

Khi mã hoá chuỗi URL sang chuỗi bình thường nếu gặp các ký tự như sau, sẽ chuyển sang ký tự tương ứng:

- **Ký tự khoảng trắng chuyển thành dấu +**
- **Ký tự không thuộc ký tự chữ và số sẽ chuyển thành số Hexadecimal.**

Ví dụ:

```
<% Response.Write
Server.URLEncode("http://www.issi.net")%>
```

Kết quả:

http%3A%2F%2Fwww%2Eissi%2Fnet

Khi ta gửi lên QueryString chuỗi có các thẻ HTML hay các ký tự đặc biệt khi gọi trang Web, chuỗi này có thể hiển thị dưới dạng thẻ HTML thay vì chuỗi cần trình bày. Chẳng hạn, ta khai báo trang ex7.asp như ví dụ 5.6.3 bằng cách sử dụng đối tượng Request để lấy giá trị từ tham số Str trên URL.

Ví dụ 5.6.3. **Lấy giá trị từ QueryString:**

```
<html>
 <head>
    <title>   Welcome to HuuKhang.com </title>
 </head>
 <body>
    <h4>Server Object</h4>
    <hr size=1 color=red><br>
    <b>Server.URLEncode</b>
    <table border=0>
    <tr>
    <td valign=top>URLEncode</td></tr>
    <tr><td valign=top>Str:
    <%=Request.QueryString("str")%></td></tr>
    </table>
 </body>
</html>
```

**** Phương thức Execute***

Cho phép gọi trang ASP khác trong một trang ASP. Khi trang được gọi tới hoàn thành các công việc của nó, sẽ trở lại tiếp tục thực hiện trang ASP gọi tới nó. Hiệu quả giống như các

hàm, thủ tục (subroutines). Phương pháp có hiệu quả tương tự include.

Server.Execute(Path)

Trong đó, tham số Path là đường dẫn tương đối hoặc vật lý của tập tin ASP cần chuyển điều khiển. Toàn bộ xâu này được đặt trong dấu nháy.

Ví dụ 5.6.4:

```
----------CallingAsp.asp----------
 <HTML>
  <BODY>
    How now <%Server.Execute("CalledAsp.asp")%> cow?
  </BODY>
 </HTML>

----------CalledAsp.asp----------
<%
Response.Write "pink"
%>
```

Kết quả:

How now pink cow?

**** Phương thức GetLastError***

Trả lại đối tượng ASPError, đối tượng này có 9 thuộc tính chỉ đọc cung cấp thông tin chi tiết về lỗi.

Server.GetLastError

Ví dụ 5.6.5:

```
<%

   Dim objErrorInfo
   Set objErrorInfo = Server.GetLastError
```

```
    Response.Write("ASPCode = " & objErrorInfo.ASPCode)
    Response.Write("ASPDescription = " &
                        objErrorInfo.ASPDescription)
    Response.Write("Category = " & objErrorInfo.Category)
    Response.Write("Column = " & objErrorInfo.Column)
    Response.Write("Description = " &
                        objErrorInfo.Description)
    Response.Write("File = " & objErrorInfo.File)
    Response.Write("Line = " & objErrorInfo.Line)
    Response.Write("Number = " & objErrorInfo.Number)
    Response.Write("Source = " & objErrorInfo.Source)
%>
```

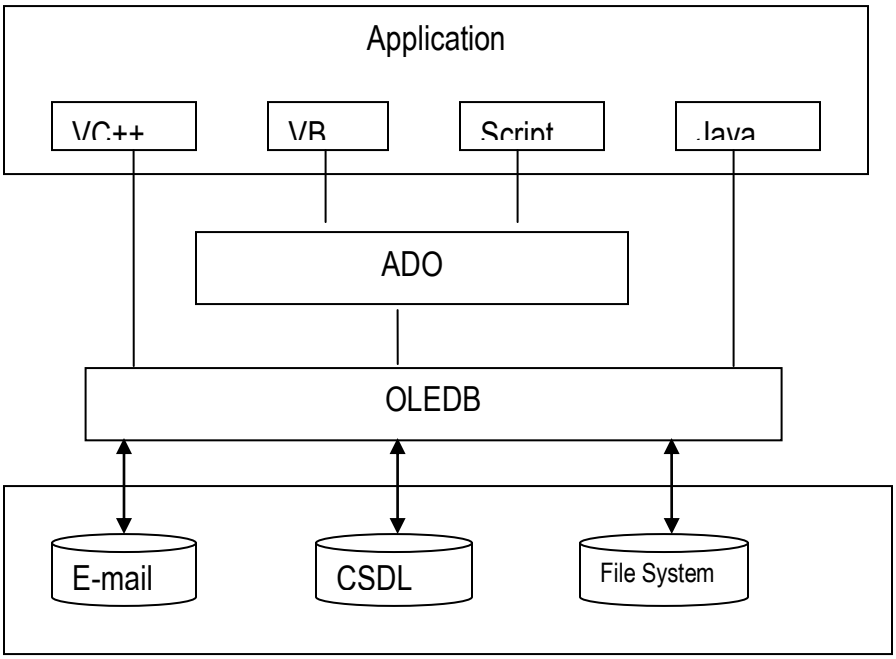
PHẦN 3. ASP VÀ CƠ SỞ DỮ LIỆU

1. Giới thiệu đối tượng ADO

1.1. Khái niệm ADO (Activẽ Data Object)

ActiveX Data Object là lớp đối tượng COM (Compenent Object Model) tập trung vào xử lý dữ liệu thông qua OLEDB của Windows. ADO thiết kế cho mục đích truy xuất dữ liệu tổng quát không chỉ dùng để truy xuất dữ liệu thuần tuý mà còn xử lý được cả dữ liệu file hay bất kỳ loại dữ liệu nào có hỗ trợ cơ chế cho phép giao tiếp thông qua OLEDB

Mô hình kiến trúc của ADO tương tác giữa ứng dụng và nguồn dữ liệu



Trong đó, OLE-DB (Object Linking and Embedding - Database) – liên kết và nhúng đối tượng, cho phép người dùng truy xuất mọi loại dữ liệu như CSDL, bảng biểu, đồ thị, thư điện tử...

1.2. Trình tiêu thụ (consumer) và trình cung cấp (provider)

Trong bước lập trình, chương trình viết ra chính là trình tiêu thụ dữ liệu bởi nó cần truy xuất vào các nguồn dữ liệu để xử lý. Còn trình cung cấp là tập lệnh cho phép truy xuất vào nguồn dữ liệu theo cách đặc trưng của chúng. Provider cho phép giao tiếp giữa nguồn dữ liệu và tầng điều khiển OLEDB. Và ADO chỉ trao đổi với nguồn dữ liệu thông qua OLEDB mà không cần quan tâm tới cách thức làm việc của Provider. Để giúp OLEDB biết được Provider nào cần phải giao tiếp, khi mở

kết nối ADO cần chỉ định trình cung cấp dữ liệu Provider tương ứng. Microsoft cung cấp sẵn một số Provider cho phép truy xuất dễ dàng vào các nguồn dữ liệu đang thông dụng như:

Jet OLEDB 4.0 – Cơ sở dữ liệu Access

DTS Packages – Dịch vụ chuyển đổi dữ liệu trong SQL Server

ODBC Driver – Provider ch phép truy xuất nguồn dữ liệu thông qua ODBC

SQL Server – Cơ sở dữ liệu SQL Server

Oracle – Cơ sở dữ liệu Oracle

Simple Provider – Cơ sở dữ liệu dạng Text

Để tương tác với CSDL, chúng ta sử dụng đối tượng ADO. ADO là công nghệ của hãng Microsoft tự động cài đặt khi cài đặt Windows có cài đặt IIS. ADO là chương trình giao tiếp để truy cập dữ liệu trong CSDL.

Khi kết nối CSDL từ trang ASP, ta có thể thao tác theo các bước sau

- **Khởi tạo đối tượng Connection.**
- **Mở kết nối CSDL với chuỗi kết nối.**
- **Khởi tạo đối tượng RecordSet.**
- **Mở đối tượng RecordSet.**
- **Xử lý dữ liệu trong đối tượng RecordSet.**
- **Đóng đối tượng RecordSet.**
- **Đóng đối tượng Connection.**
- **Giải phóng hai đối tượng RecordSet và Connection.**

2. Mô hình đối tượng ADO

Có ba đối tượng chính trong mô hình đối tượng ADO

- **Đối tượng Connection**
- **Đối tượng Command**
- **Đối tượng RecordSet**

Trong phần này chúng ta sẽ tìm hiểu chi tiết hai đối tượng RecordSet và Connection là hai đối tượng chính dùng để kết nối và xử lý dữ liệu SQL Server, Access và một số cơ sở dữ liệu khác từ trong trang ASP.

2.1. Đối tượng kết nối (Connection)

Cho phép thực hiện việc mở kết nối đến nguồn dữ liệu cần truy xuất. Thông qua Connection chỉ cần chỉ định trình cung cấp OLEDB Provider sẽ dùng để tiếp cận dữ liệu. Các thông tin kết nối bổ sung khác như username, password, server name,.. thường được lưu vào một chuỗi gọi là chuỗi kết nối (Connection String).

**** Khai báo đối tượng Connection***

Chú ý rằng mọi biến khai báo trong ASP đều có kiểu dữ liệ Variant, điều này có nghĩa là ta không thể dùng từ khoá As như trong VB.

Để khai báo đối tượng Connecction, ta sử dụng cú pháp như sau:

```
Dim myCon
```

**** Khởi tạo đối tượng***

Sau khi khai báo biến myCon với kiểu dữ liệu là Variant, ta có thể khởi tạo đối tượng này bằng phương thức CreateObject của đối tượng Server như đã trình bày với cú pháp như sau

```
Set myCon=CreateObject("ADODB.Connection")
```

Nếu sử dụng hệ điều hành Windows 9x với trình chủ Web là Personal Web Server, khai báo trên có thể được viết lại như sau:

```
Set myCon=Server.CreateObject("ADODB.Connection")
```

**** Mở kết nối với CSDL***

Sau khi khởi tạo đối tượng Connection, để tương tác với CSDL ta phải kết nối với CSDL với cú pháp như sau:

```
myCon.Open “Chuỗi kết nối”
```

Tuỳ thuộc vào loại CSDL cần tương tác mà ta sử dụng trình điều khiển tương ứng.

- **<u>Mở kết nối CSDL Access</u>: Khi mở kết nối CSDL Access, ta sử dụng chuỗi kết nối với các thuộc tính như sau:**

```
strCon="Provider= Microsoft.Jet.OLEDB.4.0";&_
                    "Data Source=D:\LTWeb\Database\BTTN.mdb"
myCon.Open strCon
```

Chú ý, ta có thể sử dụng phương thức ServerVariables của đối tượng Request để lấy đường dẫn vật lý của tập tin như khai báo sau:

```
strPath=request.ServerVariables("APPL_PHYSICAL_PATH")
strCon="Provider=" & _
        "Microsoft.Jet.OLEDB.4.0;" & _
        "Data Source= " & strPath & _
        "\BTTN\BTTN.mdb"
        myCon.Open strCon
```

***Ví dụ 2.1*. Kết nối với cở sở dữ liệu Access.**

```
<HTML>
<head>
    <title> Welcome to ADO and ASP 3.0 </title>
</head>
<body>
  <h4> Welcome to ADO and Access </h4>
  <%
    Dim myCon,strCon
    strCon="Provider=Microsoft.Jet.OLEDB.4.0;" &_
                "Data Source=D:\LTWeb\database\bttncd.mdb"
    Set myCon=CreateObject("ADODB.Connection")
    myCon.Open strCon
    if myCon.State=1 then
      Response.Write "Connected to Access"
    else
      Response.Write "Connect fail"
    end if
    myCon.Close
    Set myCon=Nothing
  %>
  <hr size=1 color=red>
</body>
</html>
```

Hoặc ta có thể viết như sau

```
<HTML>
<head>
      <title> Welcome to ADO and ASP 3.0 </title>
</head>
<body>
      <h4> Welcome to ADO and Access </h4>
      <%
         Dim myCon,strCon
         strPath=request.ServerVariables("APPL_PHYSICAL_PATH")
         strCon="Provider=Microsoft.Jet.OLEDB.4.0;" &_
                  "Data Source= "& strPath
&"\DATABASE\BTTNCD.mdb"
         Set myCon=CreateObject("ADODB.Connection")
         myCon.Open strCon
         if myCon.State=1 then
            Response.Write "Connected to Access"
         else
            Response.Write "Connect fail"
         end if
         myCon.Close
         Set myCon=Nothing
      %>
      <hr size=1 color=red>
</body>
</html>
```

- **Mở kết nối với CSDL SQL Server: nếu ta sử dụng đặc quyền SQL Server để đăng nhập CSDL SQL Server, ta sử dụng chuỗi kết nối như sau:**

```
strCon="DRIVER=SQL
Server;UID=sa;PWD=sa;"Database=Northwind;Server=."
myCon.Open strCon
```

hoặc:

```
strCon="Provider=SQLOLEDB.1;" & _
"Persist Security Info=False;UID=sa;" & _
"PWD=sa;Initial Catalog= Northwind;Server=."
myCon.Open strCon
```

**** Thực thi phát biểu SQL***

Sau khi mở kết nối CSDL với chuỗi kết nối khai báo trong phần trên, bằng cách sử dụng phương thức Execute của đối tượng Connection ta có thể thực thi phát biểu SQL dạng hành động hoặc một thủ tục nội tại với cú pháp như sau:

```
myCon.Execute "SQL Statement"
```

***Ví dụ 2.2*. Thực hiện câu lệnh SQL sử dụng phát biểu Create table để tạo bảng:**

```
<html>
  <head>
    <title> Welcome to ADO and ASP 3.0 </title>
  </head>
  <body>
      <h4>Execute SQL Statement</h4>
      <%
        Dim myCon,strCon
```

```
    strPath=request.ServerVariables("APPL_PHYSICAL_PATH")
            strCon="Provider=Microsoft.Jet.OLEDB.4.0;" &_
                  "Data Source= "& strPath
&"\DATABASE\BTTNCD.mdb"
            Set myCon=CreateObject("ADODB.Connection")
            myCon.Open strCon

            if myCon.State=1 then
               strSQL="Create Table tblABCDE"
               strSQL=strSQL & "(AD int, DD int)"
               myCon.Execute strSQL
               Response.Write "Created Table"
            else
               Response.Write "Fail"
            end if
            myCon.Close
            Set myCon=Nothing
         %>
         <hr size=1 color=red>
   </body>
</html>
```

**** Đóng kết nối***

Sau khi mở kết nối CSDL và thực thi các câu lệnh SQL, hay mở tập dữ liệu, nếu không có nhu cầu sử dụng đối tượng Connection, ta có thể đóng kết nối tránh trường hợp có nhiều kết nối đang mở bằng phương thức close, với cú pháp như sau:

```
myCon.Close
```

**** Giải phóng đối tượng:***

Nếu không có nhu cầu sử dụng đối tượng Connection, ta nên sử dụng phát biểu Set để huỷ đối tượng này và giải phóng bộ nhớ đã cấp phát cho chúng với cú pháp như sau:

```
Set myCon=Nothing
```

Chú ý:

- **Nếu ngay sau đó ta sử dụng lại đối tượng này, cần phải khởi tạo trở lại bằng phương thức CreateObject của đối tượng Server.**
- **Nếu trước đó chưa khai báo đóng kết nối bằng phương thức Close, thì phát biểu Set để huỷ bỏ đối tượng Connection sẽ tự động đóng trước khi bị huỷ.**

Có thể kết nối và truy xuất vào nguồn dữ liệu mà không bắt buộc phải dùng đối tượng Connection. Các đối tượng khác như Command, RecordSet, Record,.. cũng cho phép mở trực tiếp kết nối. Tuy nhiên sử dụng đối tượng Connection sẽ cho phép ta tách biệt thao tác kết nối và thao tác truy cập cơ sở dữ liệu. Hơn nữa đối tượng Connection còn cung cấp thêm một số

chức năng chuyên dụng khác như cho phép thực thi câu lệnh SQL tác động vào dữ liệu như Insert, Update, Delete, gọi thủ tục Procedure Store,.. hoặc kiểm soát giao tác transaction như Rollback, commit.

2.2. Đối tượng RecordSet

Là đối tượng sử dụng thường xuyên trong ADO. Cung cấp kết quả trả về từ câu lệnh truy vấn một tập các mẩu tin. Trang ASP có thể dùng vòng lặp để duyệt qua các mẩu tin này và hiển thị dữ liệu kết xuất ra trang Web phía trình duyệt. Ngoài ra RecordSet còn cho phép thực hiện lọc dữ liệu từ tập các mẩu tin, truy xuất đến từng trường cụ thể của mẩu tin thông qua đối tượng Field hoặc danh sách các trường trong mẩu tin thông qua đối tượng Fields.

**** Khai báo đối tượng***

Tương tự như đối tượng Connection, khi có nhu cầu lấy dữ liệu là một giá trị hay một mẫu tin, ta sử dụng phát biểu để khai báo đối tượng RecordSet với cú pháp:

```
Dim myRs
```

**** Khởi tạo đối tượng***

Sau khi khai báo, ta có thể khởi tạo đối tượng này bằng phát biểu Set và phương thức CreateObject của đối tượng Server như sau

```
Set myRst=CreateObject("ADODB.RecordSet")
```

hay `Set myRst=Server.CreateObject("ADODB.RecordSet")`

**** Mở tập dữ liệu***

Để mở tập điều khiển ứng với phát biểu SQL dạng Select, Table, View, Query hay thủ tục nội tại của SQL Server có giá trị trả về, ta sử dụng phương thức Open với khai báo như sau:

myRst.Open "Phát biểu SQL", myCon, CursorType, LockType

Trong đó

- **myCon là đối tượng kết nối đã mở ứng với CSDL nào đó.**
- **Hai tham số tuỳ chọn CursorType (mặc định adOpenForwardOnly), LockType (mặc định là adLockReadOnly) ứng với kiểu con trỏ và loại khoá dữ liệu.**

- Giá trị của CursorType

- **adOpenDynamic (tương đương với hằng số 2): Khi khai báo hằng số này, ứng với con trỏ (cursor) động, người sử dụng khác có thể thay đổi, xoá và di chuyển trên mẫu tin mà ta đang mở ngoại trừ bookmark.**
- **adOpenForwardOnly (tương đương hằng số 0) đây là giá trị mặc định, khi khai báo hằng số này, con trỏ chỉ di chuyển một chiều. Đối với trường hợp mở tập dữ liệu mà không điều hướng (navigation) trên mẩu tin, ta nên sử dụng tuỳ chọn với giá trị này.**
- **adOpenKeyset (tương đương hằng số 1): Tương tự như adOpenDynamic nhưng không cho phép người sử dụng khác thêm hay xoá hoặc sửa mẩu tin.**

- **adOpenStatic (tương đương với hằng số 3): Đây là dạng con trỏ tĩnh, nghiêm cấm thay đổi, xoá mẩu tin từ người sử dụng khác.**

- **Giá trị của LockType**

- **adLockBatchOptimistic (tương đương với hằng số 4): Yêu cầu cập nhật tập dữ liệu theo đó.**
- **adLockOptimistic (tương đương với hằng số 3): Không cho phép cập nhật mẩu tin khi gọi phương thức Update.**
- **adLockPessimistic (tương đương với hằng số 2): Khoá lại mẩu tin ngay sau khi kích hoạt.**
- **adLockReadOnly (tương đương với hằng số 1): Chỉ cho phép đọc.**

Chẳng hạn, đối tượng với trường hợp này chúng ta tìm hiểu ví dụ sử dụng đối tượng RecordSet để mở bảng dữ liệu giaovien như sau:

Ví dụ 2.3. **Mở bảng dữ liệu.**

```
<html>
   <head>
        <title> Welcome to ADO and ASP 3.0 </title>
   </head>
   <body>
      <h4>RecordSet</h4>
      <%
         Dim myCon,strCon
         strPath=request.ServerVariables("APPL_PHYSICAL_PATH")
         strCon="Provider=Microsoft.Jet.OLEDB.4.0;" &_
```

```
                    "Data Source= "& strPath
&"\DATABASE\BTTNCD.mdb"
        Set myCon=CreateObject("ADODB.Connection")
        Set myRst=CreateObject("ADODB.RecordSet")
        myCon.Open strCon
        if myCon.State=1 then
           Response.Write "Ket noi duoc : "
        else
           Response.Write "Khong ket noi duoc.."
        end if

        strSQL="Select * from giaovien"
        myRst.Open strSQL , myCon
        if Not myRst.EOF then
           Response.Write "Existing records"
        else
           Response.Write "Record not found"
        end if
        myRst.Close
        myCon.Close
        Set myRst=Nothing
        Set myCon=Nothing
      %>
      <hr size=1 color=red>
   </body>
</html>
```

**** Xử lý dữ liệu***

Sau khi mở tập mẩu tin với các tuỳ chọn trên, tuỳ thuộc vào loại cursor mà ta có thể thêm, xoá, cập nhật hay di chuyển trên từng mẩu tin.

Đối tượng RecordSet cung cấp các phương thức và thuộc tính cho phép duyệt hay xử lý dữ liệu trong tập dữ liệu đang nắm giữ bởi đối tượng này.

Một số thuộc tính thường sử dụng:

- **EOF: trả về giá True nếu cursor đang nằm sau mẩu tin cuối cùng. Trong trường hợp đối tượng RecordSet không nắm giữ mẩu tin nào, thuộc tính này có giá trị là True, ngược lại có giá trị là False.**

Ví dụ, ta kiểm tra mẩu tin trong đối tượng RecordSet có tồn tại hay không như sau:

```
if Not myRst.EOF then
        Tồn tại
```

```
else
        Không tồn tại
end if
```

- **BOF: Trả về giá trị True nếu cursor đang nằm trên mẩu tin đầu tiên. Trong trường hợp đối tượng RecordSet không nắm giữ mẩu tin nào, thuộc tính này sẽ có giá trị là True, ngược lại có giá trị là False.**

Ví dụ, ta kiểm tra mẩu tin trong đối tượng RecordSet có tồn tại hay không như sau:

```
if myRst.BOF then
        Tồn tại
else
        Không tồn tại
end if
```

- **Fields("Fieldname"): Gán hay lấy giá trị của cột trong bảng dữ liệu.**

  Ví dụ: myRst.Fields("MaGV")="NHA"

- **RecordCount: Trả về giá trị là tổng số mẩu tin đang có trong đối tượng RecordSet với loại cursor động.**

  Ví dụ, ta lấy số mẩu tin trong đối tượng RecordSet như sau

```
Dim i
i=myRst.RecordCount
```

- **MoveFirst: Di chuyển cursor đến mẩu tin đầu tiên.**
- **MoveLast: Di chuyển cursor đến mẩu tin cuối cùng.**
- **MoveNext: Di chuyển cursor đến mẩu tin kế tiếp.**

MovePrevious: Di chuyển cursor đến mẩu tin trước đó.

Ví dụ: Duyệt qua các mẫu tin và truy xuất vào các trường cuả mẫu tin

```
Do While not rs.eof
  Rs [.fields](“field_name”)
rs.movenext
loop
rs.close
```

Các cách truy cập hợp lệ và tương đương:

```
Rs(“field_name”)
Rs.fields(“field_name”)
Rs.fields(“field_name”).value
Rs(1).value
Rs.fields(1).Value
```

Nếu muốn duyệt qua tất cả các trường trong mẩu tin ta có thể dùng lệnh:

```
For each Fld in rs.fields
  Response.write fld.name + “:” + fld.value + “<br>”
Next
```

Có thể lọc qua các mẫu tin trong RecordSet

Có thể sử dụng mệnh đề Where của câu lệnh Select hoặc có thể sử dụng thuộc tính Filter của RecordSet để chỉ định điều kiện lọc sau khi đã trích xuất dữ liệu. Ví dụ:

```
sqlStr=”Select * from Products”
rs.open sqlStr,conn
```

rs.filter= " productCode='" & SpecialCode & "'"

Thuộc tính Filter cho phép sử dụng mệnh đề lọc gần giống với mệnh đề Where

- **PageSize: Thuộc tính này cho phép gán hay đọc số mẩu tin trình bày trên một trang. điều này có nghĩa là khi ta có nhiều mẩu tin muốn trình bày, bằng cách sử dụng thuộc tính này để yêu cầu kết xuất số mẩu tin chỉ định, ví dụ 10 mẩu tin/1trang.**
- **PageCount: Thuộc tính này trả về số trang được tính dựa trên thuộc tính PageSize, chẳng hạn tổng số mẩu tin có trong đối tượng RecordSet là 153 mẩu tin, nếu ta khai báo thuộc tính PageSize là 15 (15 mẩu tin/1trang), khi đó thuộc tính PageCount sẽ tả về 11 trang.**

Có thể phân trang với đối tượng RecordSet

Đối tượng RecordSet cung cấp 3 thuộc tính quan trọng sau để sử dụng phân trang:

PageSize: Kích thước mẩu tin trong một trang

PageCount: Tổng số trang RecordSet truy vấn được

AbsolutePage: Chỉ định trang hiện hành đang cần được đọc

Để RecordSet có khả năng phân trang, cần thiết lập thêm tham số cho RecordSet trước khi thực hiện truy vấn:

```
Rs.CursorLocation=3 ' Có thể sử dụng hằng
adUseClient
Rs.PageSize=15          ' 15 bản ghi trong một trang
```

Tiếp theo mở đối tượng RecordSet truy vấn dữ liệu với tuỳ chọn là các hằng adOpenForwardOnly(0), adLockReadOnly (1) truy vấn cho phương thức Open như sau:

```
Rs.open sqlStr, Conn, 0,1
```

Công việc sau cùng là định vị trang thông qua thuộc tính AbsolutePage. Ta lưu lại vị trí hiện hành của trang dữ liệu thông qua giá trị chứa trong thẻ <input hidden>. Giá trị này sẽ được chuyển về trình chủ mỗi khi người dùng kích vào.

Xét ví dụ sau:

```
<%sqlStr="SELECT * FROM Products "
  ' page navigate session here .....
      Dim lCurrentPage
      Dim lPageCount
      lCurrentPage = CLng(Request("page"))
      If lCurrentPage < 1 Then
            lCurrentPage = 1
      End If
      rs.CursorLocation = 3
      rs.PageSize = 15
      rs.Open sqlStr, conn, 0, 1 'Const
adOpenForwardOnly=0,adLockReadOnly = 1
      lPageCount = rs.PageCount
      If lCurrentPage > lPageCount Then
            lCurrentPage = lPageCount
      End If
      if not rs.eof then
        rs.AbsolutePage = lCurrentPage
      end if
      call ShowPageNavigation(lCurrentPage,lPageCount)
```

```
        Do While rs.AbsolutePage = lCurrentPage And Not rs.Eof
                Response.write rs("ProductName")
        rs.movenext
        loop
%>
<form name="viewFrm">
  <input type=hidden name=page>
</form>
<%
Sub ShowPageNavigation (lCurrentPage,lPageCount)
        If lCurrentPage <> 1 AND lCurrentPage <> 0 Then
%>
<A HREF="javascript:setValue('<%= lCurrentPage - 1 %>');">Previous
<% Else %> Previous
<% End If%>
<%If lCurrentPage < lPageCount Then%>
<A HREF="javascript:setValue('<%= lCurrentPage +
1%>');">Next
<% Else %> Next
<% End If%><BR>Page<B><%= lCurrentPage%></B>
<%= lPageCount%>
<%End Sub      %>
<script language=javascript>
function setValue(page){
        viewFrm.page.value= page;
        viewFrm.submit();
        }
</script>
```

- **CursorLocation: Là loại cursor (mặc định là adUseServer) áp dụng cho đối tượng RecordSet khi mở tập tin, nếu giá trị gán là adUseClient, cursor thực hiện phía trình khách.**
- **AddNew: Thêm mẩu tin mới vào bảng dữ liệu quản lý bởi phương thức RecordSet.**

Update: Cập nhật mẩu tin vào dữ liệu nguồn.

Ví dụ

```
sqlStr=”Select * From Accounts”
rs.open sqlStr,Conn
‘ Thêm tài khoản mới vào bảng Accounts
rs.Addnew
‘gán giá trị cho mẩu tin
with rs
 .fields(“username”)= ‘New User’
 .fields(“password”)=’***’
end with
‘lưu lại
rs.update
```

Chỉnh sửa nội dung trong mẩu tin hiện hành

```
sqlStr=”Select * From Accounts”
rs.open sqlStr,Conn
with rs
.fields(“password”)=’newpassword’
end with
‘lưu lại
rs.update
```

Delete: Xoá mẩu tin hiện hành trong RecordSet tự động xoá mẩu tin trong dữ liệu nguồn.

```
sqlStr=”Select * From Accounts where username= ‘” &mkuser
  &””””
```

```
rs.open sqlStr,Conn
rs.delete
```

**** Đóng tập dữ liệu***

Sau khi kết thức quá trình làm việc trên đối tượng RecordSet, ta sử dụng phương thức close để đóng đối tượng.

```
myRst.Close
```

**** Giải phóng đối tượng***

Tương tự như đối tượng Connection, khi không có nhu cầu mở tiếp tập mẩu tin khác, ta nên sử dụng phát biểu Set để giải phóng bộ nhớ cấp cho đối tượng này như sau:

```
Set myRst=Nothing
```

2.3. Đối tượng Command

Đối tượng này dùng cho mục đích thực thi câu lệnh tốt hơn Connection. Cho phép ta chuyển tham số vào các lệnh thực thi SQL. Tham số có thể chỉ định kiểu hoặc giá trị tường minh. Các tham số có thể nhận trị trả về sau khi thực thi... Command có thể dùng cho cả 2 mục đích: thực thi câu lệnh SQL không cần nhận kết quả trả về như Insert, Update, Delete, Procedure Store, hoặc thực thi các lệnh trả về tập RecordSet như lệnh Select.

**** Tạo đối tượng Command***

```
set
cmdUpdate=Server.CreateObject("ADODB.Command")
```

**** Sử dụng đối tượng Command***

```
sqlUpdate="update accounts set password ='abc' where
```

```
                                    username= '" & username & "'"
    cmdUpdate.ActiveConnection=strConn
    cmdUpdate.CommandText=strUpdate
    cmdUpdate.CommandType=adcmdText
    cmdUpdate.Execute
```

Đối tượng Command cũng được dùng để nhận kết quả trả về từ câu lệnh Select hoặc từ một tên bảng dữ liệu, ví dụ để lấy toàn bộ nội dung bảng dữ liệu Accounts, ta chỉ cần chỉ ra tên bảng và mở RecordSet dựa vào đối tượng Command:

```
cmdTable.ActiveConnection=strConn
cmdTable.CommandText="Accounts"
cmdTable.CommandType=adCmdTable
rs.open cmdTable
```

3. Làm việc với dữ liệu Access

3.1.Liệt kê dữ liệu Access

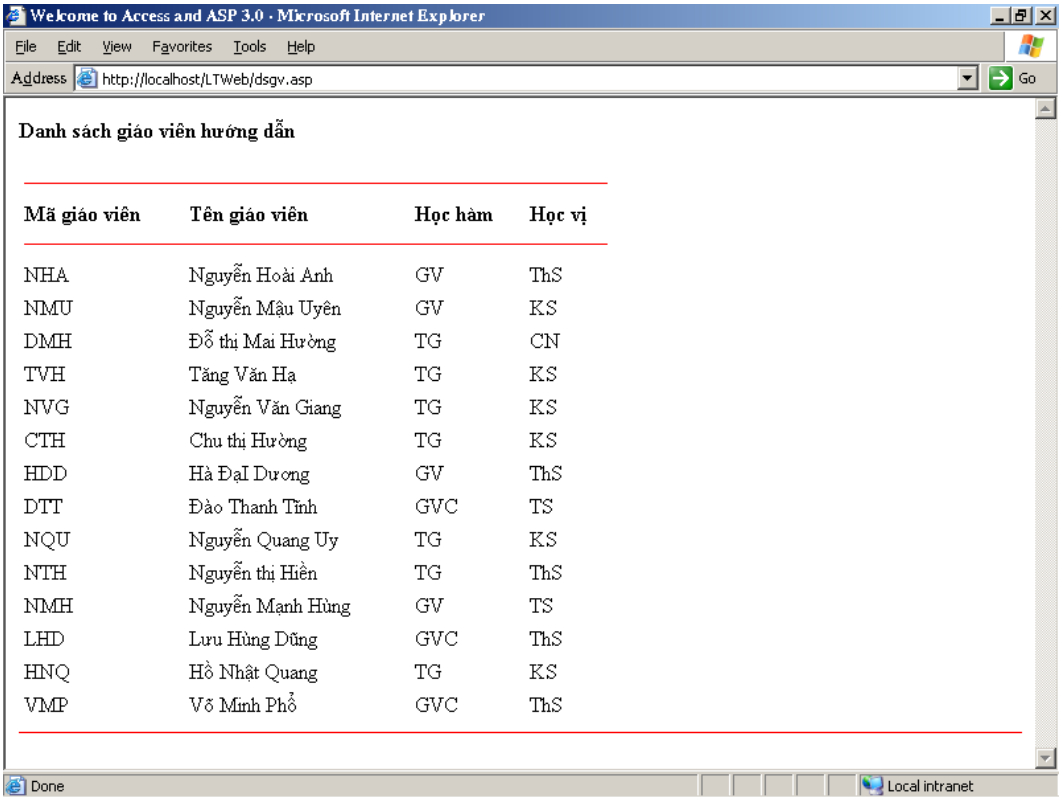
Để liệt kê mẫu tin trong CSDL Access chúng ta sử dụng một số thuộc tính RecordCount, Fields, EOF và các phương thức như: MoveNext, MoveFirst, MovePrevious và MoveLast của đối tượng RecordSet.

Ví dụ 3.1. **Liệt kê dữ liệu Access**

```
<html>
 <head>
    <title> Welcome to Access and ASP 3.0 </title>
    <!-- Sử dụng Style trong tập tin Style Sheet -->
    <LINK href="../style.css" rel=stylesheet>
    <!-- Trình bày dữ liệu Unicode -->
```

```
    <META http-equiv=Content-Type content="text/html; charset=utf-
8">
  </head>
  <body>
    <h4>Danh sách giáo viên hướng dẫn</h4>
    <table width=450 cellspacing=2 cellpadding=2>
    <tr class=text>
    <td colspan=4><hr size=1 color=red></td>
    </tr>
    <tr class=text>
    <td><b>Mã giáo viên</b></td>
    <td><b>Tên giáo viên</b></td>
    <td><b>Học hàm</b></td>
    <td><b>Học vị</b></td>
    </tr>
    <tr class=text>
    <td colspan=4><hr size=1 color=red></td>
    </tr>
    <%
      Session.CodePage=65001
      Dim myCon, myRst, strCon, strSQL, i
      Set myCon= CreateObject("ADODB.Connection")
      Set myRst=CreateObject("ADODB.RecordSet")
      strPath=request.ServerVariables("APPL_PHYSICAL_PATH")
      strCon="Provider= Microsoft.Jet.OLEDB.4.0; Data Source= " &
strPath &_
      "\DATABASE\BTTNCD.mdb"
      myCon.Open strCon
      myRst.CursorLocation=3
      strSQL="Select * from giaovien "
      myRst.Open strSQL, myCon, 0 ,3
      i=myRst.RecordCount
      Do Until myRst.EOF
      %>
        <tr class=text>
        <td><%=myRst("MaGV")%></td>
```

```
            <td><%=myRst("TenGV")%></td>
            <td><%=myRst("MaHH")%></td>
            <td><%=myRst("MaHV")%></td>
            </tr>
        <%
            myRst.MoveNext
        Loop
        myRst.Close
        myCon.Close
        Set myRst=Nothing
        Set myCon=Nothing
    %>
    </table>
    <hr size=1 color=red>
  </body>
</html>
```



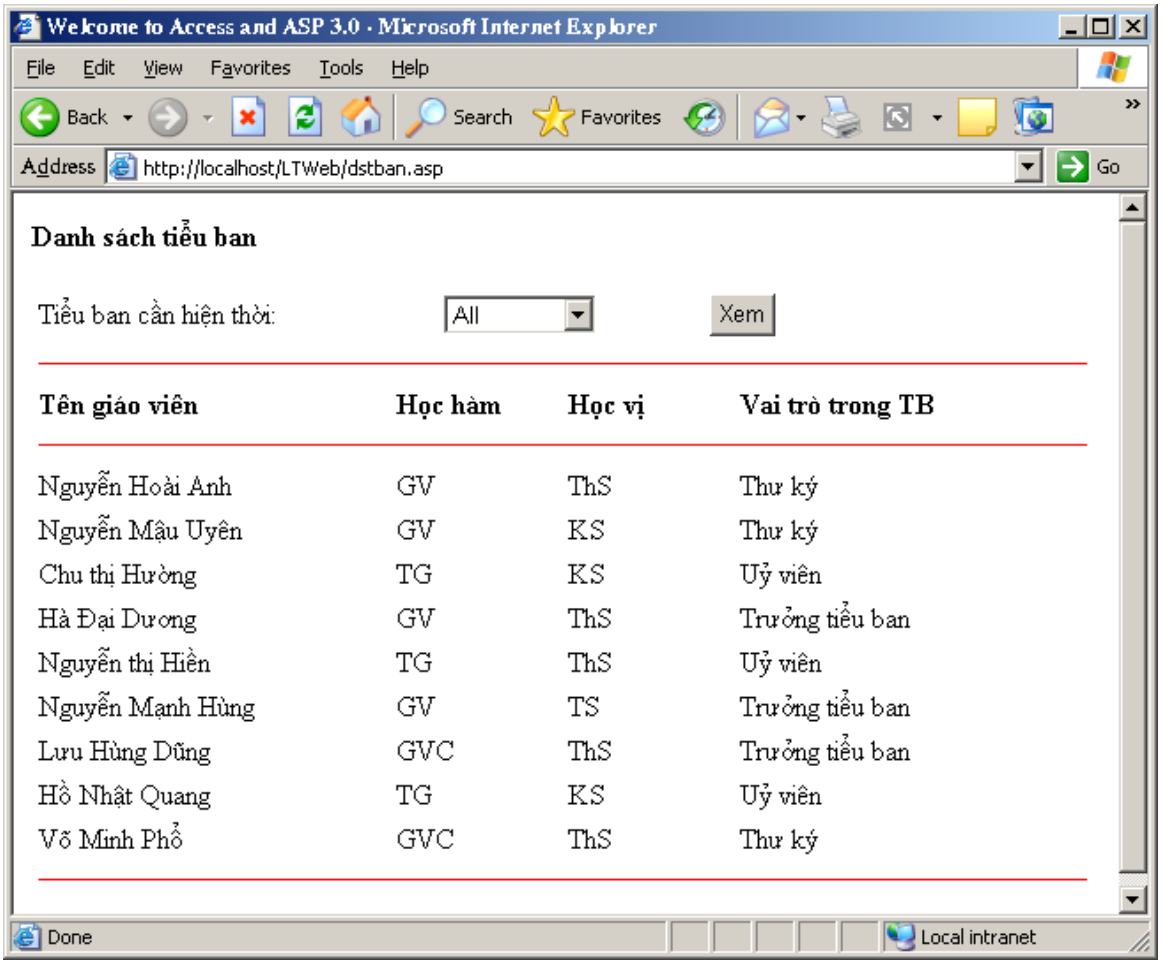
Nếu ta muốn liệt kê danh sách mẫu tin trong bảng GIAOVIEN theo tiểu ban, tức là phụ thuộc vào giá trị Tiểu ban muốn hiển thị được chọn trên màn hình.

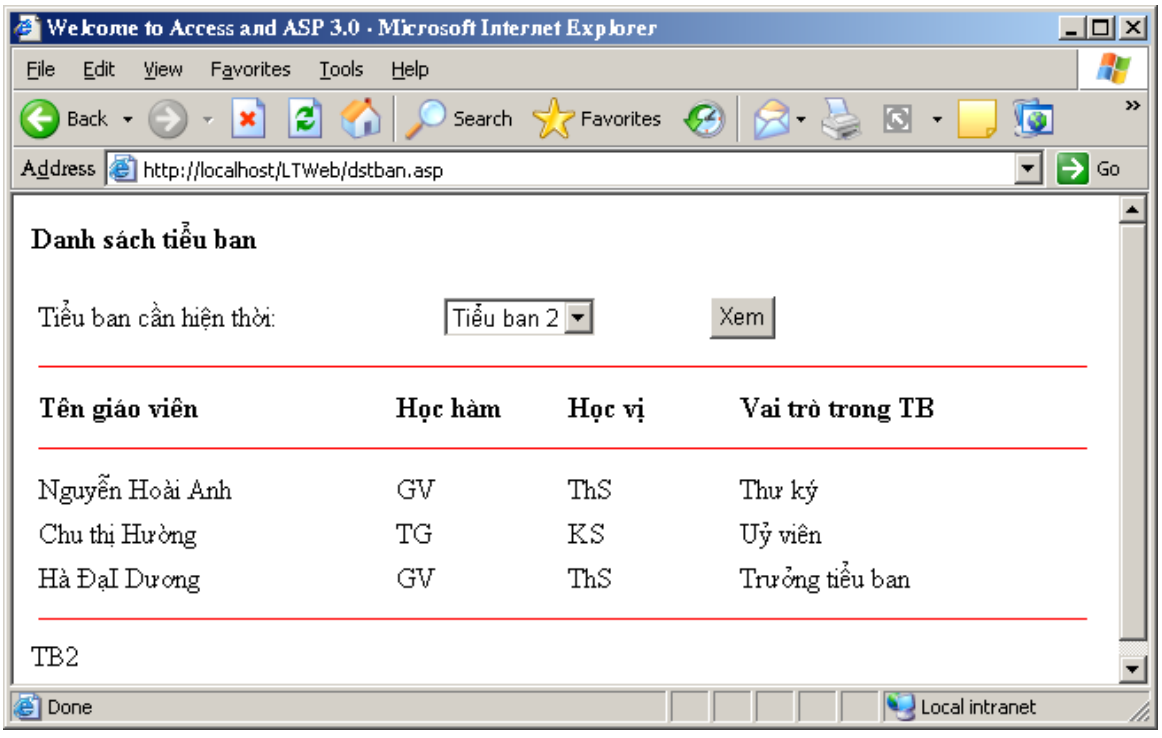
***Ví dụ 3.2*. Hiển thị danh sách giáo viên theo tiểu ban**

```
<%
   Session.CodePage=65001
   Dim myCon, myRst, strCon, strSQL, i, strCate,MaTB
   MaTB=Request.Form("MaTB")

   Set myCon= CreateObject("ADODB.Connection")
   Set myRst=CreateObject("ADODB.RecordSet")
   strPath=request.ServerVariables( _
   "APPL_PHYSICAL_PATH")
   strCon="Provider=" & _
   "Microsoft.Jet.OLEDB.4.0;" & _
   "Data Source= " & strPath & _
   "\Database\BTTNCD.mdb"
   myCon.Open strCon
   myRst.CursorLocation=3
   strCate="<option value=''>All </option>"
   strSQL="Select * from tieuban"
   myRst.Open strSQL, myCon, 0 ,3
   Do Until myRst.EOF
      strCate=strCate & "<option value='" & myRst("MaTB") & "' "
      if  Trim(myRst("MaTB"))=MaTB then
         strCate=strCate & " selected"
      End If
      strCate=strCate & ">" & myRst("TenTB") & "</option>"
      myRst.MoveNext
   Loop
   myRst.Close
%>
<html>
   <head>
      <title> Welcome to Access and ASP 3.0 </title>
```

```
        <LINK href="../style.css" rel=stylesheet>
        <META http-equiv=Content-Type content="text/html;
charset=utf-8">
    </head>
    <body>
        <h4>Danh sách tiểu ban</h4>
        <table width=450 cellspacing=2 cellpadding=2>
        <form action=dstban.asp method=post name=form1>
            <tr class=text>
            <td nowrap>Tiểu ban cần hiện thời:</td>
            <td><select name=MaTB
            onchange="document.form1.submit();">
            <%=strCate%>
            </select></td>
        <td><input type=submit value="Xem"></td>
        </tr>
        </form>
        </table>
        <table width= 600 cellspacing=2 cellpadding=2>
        <tr class=text>
        <td colspan=4 width="600"><hr size=1 color=red></td>
        </tr>
        <tr class=text>
        <td width="130"><b> Tên giáo viên </b></td>
        <td width="60"><b>Học hàm </b></td>
        <td width="60"><b> Học vị  </b></td>
        <td width="130"><b> Vai trò trong TB</b></td>
        </tr>
        <tr class=text>
        <td colspan=4 width="600"><hr size=1 color=red></td>
        </tr>
        <%
        strSQL="Select * from giaovien"
        If maTB<>"" then
            strSQL=strSQL & " Where MaTB='"& MaTB & "'"
        End If
        myRst.Open strSQL, myCon, 0 ,3
```

```
        i=myRst.RecordCount
        Do Until myRst.EOF
        %>
            <tr class=text>
            <td width="130"><%=myRst("TenGV")%></td>
            <td width="60"><%=myRst("MaHH")%></td>
            <td width="60"><%=myRst("MaHV")%></td>
            <td width="130"><%=myRst("CVTB")%></td>
            </tr>
        <%
            myRst.MoveNext
        Loop
        myRst.Close
        myCon.Close
        Set myRst=Nothing
        Set myCon=Nothing
        if i=0 then
        %>
        <tr class=text>
        <td colspan=4 align=center width="440">Item not found!</td>
        </tr>
        <%End If%>

        <td colspan=4 width="600"><hr size=1 color=red></td>
        </table>
        <% Response.Write(MaTB)%>
    </body>
</html>
```


Welcome to Access and ASP 3.0 - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost/LTWeb/dstban.asp
Go
Danh sách tiểu ban
Tiểu ban cần hiện thời: All Xem
Tên giáo viên Học hàm Học vị Vai trò trong TB
Nguyễn Hoài Anh GV ThS Thư ký
Nguyễn Mậu Uyên GV KS Thư ký
Chu thị Hường TG KS Uỷ viên
Hà Đại Dương GV ThS Trưởng tiểu ban
Nguyễn thị Hiền TG ThS Uỷ viên
Nguyễn Mạnh Hùng GV TS Trưởng tiểu ban
Lưu Hùng Dũng GVC ThS Trưởng tiểu ban
Hồ Nhật Quang TG KS Uỷ viên
Võ Minh Phổ GVC ThS Thư ký
Done
Local intranet


Welcome to Access and ASP 3.0 - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost/LTWeb/dstban.asp
Go
Danh sách tiểu ban
Tiểu ban cần hiện thời: Tiểu ban 2 Xem
Tên giáo viên Học hàm Học vị Vai trò trong TB
Nguyễn Hoài Anh GV ThS Thư ký
Chu thị Hường TG KS Uỷ viên
Hà ĐạI Dương GV ThS Trưởng tiểu ban
TB2
Done
Local intranet

3.2.Thêm, cập nhật, xóa dữ liệu Access

Cơ sở dữ liệu nói chung và CSDL Access nói riêng đều sử dụng phát biểu SQL để thêm, cập nhật hay xoá mẫu tin.

3.2.1. Thêm và cập nhật mẫu tin

Để thêm mẫu tin vào bảng dữ liệu hay cập nhật mẫu tin của CSDL Access, ta có thể sử dụng phương thức Execute của đối tượng connection để thực thi phát biểu SQL dạng Insert, Update hay hai phương thức Addnew và update của đối tượng RecordSet.

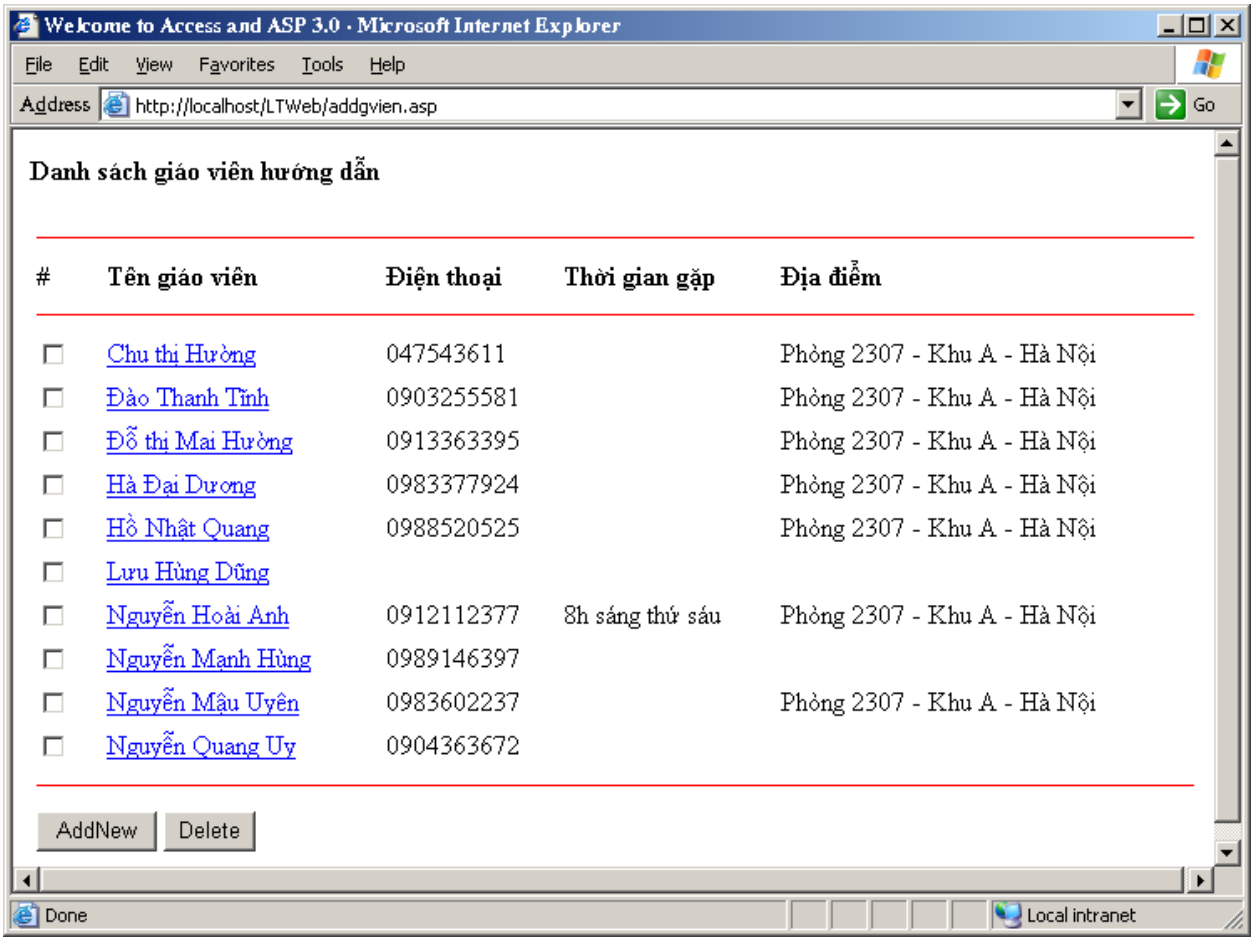
Để cho phép người sử dụng thêm mới mẫu tin vào bảng giáo viên (giaovien), ta xây dựng trang Addgvien.asp liệt kê danh sách các giáo viên đang có, dưới có các nút Addnew (addnew.asp) và Delete như sau:

***Ví dụ 3.2.1*. Liệt kê danh mục các giáo viên (trang Addgvien.asp)**

```
<html>
   <head>
      <title> Welcome to Access and ASP 3.0 </title>
      <LINK href="../style.css" rel=stylesheet>
      <META http-equiv=Content-Type content="text/html; charset=utf-8">
   </head>

  <body>
      <h4> Danh sách giáo viên hướng dẫn</h4>
      <table width=700 cellspacing=2 cellpadding=2 height="119">
      <tr class=text>
      <td colspan=5 width="700"><hr size=1 color=red></td></tr>
      <tr class=text><td width="28" ><b>#</b></td>
      <td width=130><b>Tên giáo viên</b></td>
```

```
        <td width="70" ><b>Điện thoại </b></td>
        <td width="100" ><b>Thời gian gặp </b></td>
        <td width="200" ><b>Địa điểm </b></td></tr>
        <form action=delete.asp method=post>
        <tr class=text><td colspan=5 width="700" >
        <hr size=1 color=red></td></tr>
        <%
          Session("isSave")=""
          Session.CodePage=65001
          Dim myCon, myRst, strCon, strSQL, i
          Set myCon= CreateObject("ADODB.Connection")
          Set myRst=CreateObject("ADODB.RecordSet")
          <!—Lấy địa chỉ vật lý của ứng dụng -->
          strPath=request.ServerVariables("APPL_PHYSICAL_PATH")
          strCon="Provider=Microsoft.Jet.OLEDB.4.0;Data Source= " & strPath
&_
          "\Database\BTTNCD.mdb"

           <!—Mở kết nối CSDL-->
          myCon.Open strCon
          myRst.CursorLocation=3
          strSQL="Select * from giaovien where HD=Yes order by TenGV"
          myRst.Open strSQL, myCon, 0 ,3
          i=myRst.RecordCount
          Do Until myRst.EOF %>
           <tr class=text>
           <!-- Khai báo nút Checkbox cho phép chọn-->
           <td width="28" height="15">
           <input type=checkbox value=<%=myRst("MaGV")%>
name=chkid></td>
           <td width="130" height="15">
           <!—Khai báo thẻ liên kết cho phép sửa-->
           <a
href="addnew.asp?id=<%=myRst("MaGV")%>"><%=myRst("TenGV")%><a/>
           <td width="60" height="15"><%=myRst("SoDT")%></td>
           <td width="100" height="15"><%=myRst("thoigian")%></td>
```

```
            <td width="200"
height="15"><%=myRst("diadiem")%></td></tr>
            <%
                  myRst.MoveNext
                  Loop
                  myRst.Close
                  myCon.Close
                  Set myRst=Nothing
                  Set myCon=Nothing
                  if i=0 then %>
        <tr class=text>
        <td colspan=5 align=center >KhÃ´ng tÃ¬m thấy giÃ¡o
viÃªn!</td></tr>
        <%End If%>
        <td colspan=5 width="700"><hr size=1 color=red></td>
        </table>

        <table width=200 cellspacing=2 cellpadding=2>
        <tr class=text>
        <!-- Khai báo nút thêm tin-->
        <td colspan=2>
        <input type=button
            onclick="window.open('addnew.asp', target='_main');"
value="AddNew">
     <!-- Khai báo nút xóa mẫu tin-->
        <input type=submit value="Delete">
        </td></tr>
    </form>
        </table>
    </body>
</HTML>
```



Khi người dùng ấn nút AddNew trang mới addnew.asp xuất hiện như sau:

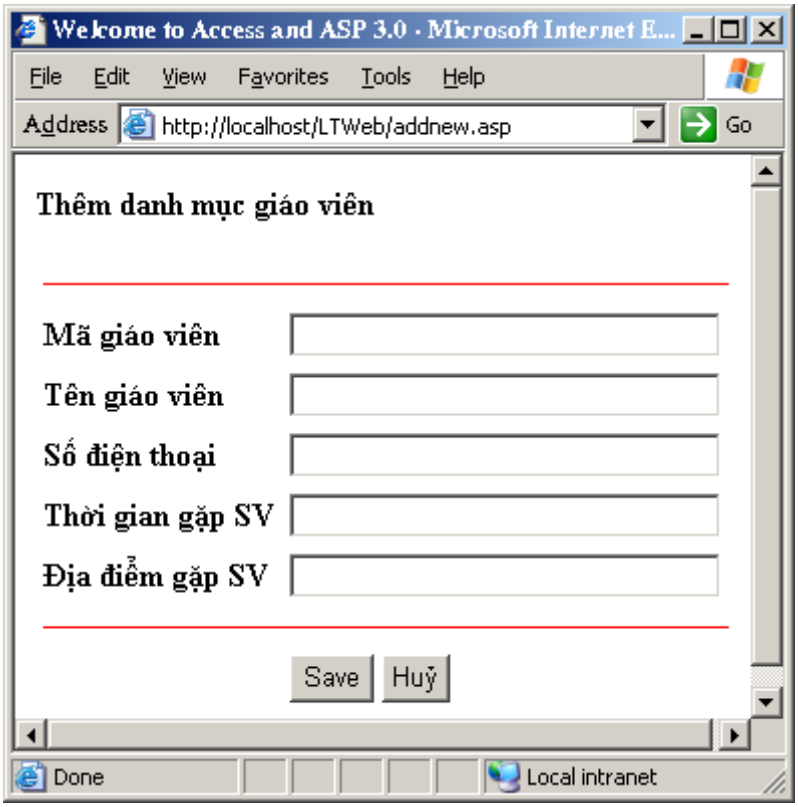
***Ví dụ 3.2.2*. Trang addnew.asp**

```
<%
   Dim id,name,sdt,tg,dd,strAction
   <!-- Khai báo Session với giá trị rỗng để tránh trường hợp lưu hai lần-->
  Session("isSave")=""
    <!-- Giá trị khai báo trên nút cho phép người sử dụng thêm hay cập nhật, mặc định là thêm-->
   strAction="Save"
  <!-- Khai báo biến Id để lấy tham số ID trong trường hợp cập nhật ngôn ngữ-->
   id=Request.QueryString("id")
```

```
    If id<>"" Then
        Session.CodePage=65001
        Dim myCon, myRst, strCon, strSQL
        Set myCon= CreateObject("ADODB.Connection")
        Set myRst=CreateObject("ADODB.RecordSet")
        strPath=request.ServerVariables("APPL_PHYSICAL_PATH")
        strCon="Provider=Microsoft.Jet.OLEDB.4.0;Data Source= " & strPath
&_
        "\Database\BTTNCD.mdb"
        myCon.Open strCon
        strSQL="Select * from giaovien "
        strSQL=strSQL & " Where MaGV='" & id & "'"
        myRst.Open strSQL, myCon
        If Not myRst.EOF Then
            name=myRst("TenGV")
            sdt=myRst("Sodt")
            tg=myRst("thoigian")
            dd=myRst("diadiem")
            strAction="Update"
        End If
        myRst.Close
        myCon.Close
        Set myRst=Nothing
        Set myCon=Nothing
    End If
%>
<html>
    <head>
        <title> Welcome to Access and ASP 3.0 </title>
        <LINK href="../style.css" rel=stylesheet>
        <META http-equiv=Content-Type
```

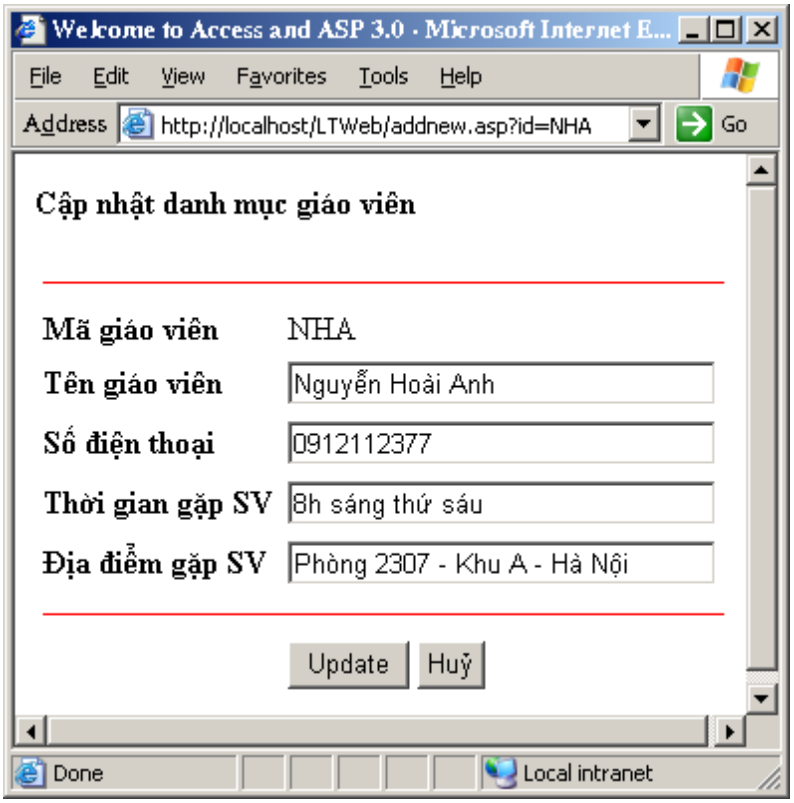
```
        content="text/html; charset=utf-8">
    </head>
    <body>
        <h4><%If id="" Then%>Thêm<%Else%> Cập nhật <%End If%> danh
mục giáo viên</h4>
        <form action=insert.asp method=post>
        <table width=350 cellspacing=2 cellpadding=2>
        <tr class=text>
        <td colspan=2><hr size=1 color=red></td>
        </tr>
        <tr class=text>
        <td width=100><b>Mã giáo viên</b></td><td>
        <%If id="" then%>
        <input name=txtid size=30 maxlength=5>
        <%Else%>
        <%=id%>
        <input name=txtid type=hidden value=<%=id%>>
        <%End if%>
        </td></tr>
        <tr class=text>
        <td><b>Tên giáo viên</b></td>
        <td><input name=txtname size=30 maxlength=30
value="<%=name%>"></td></tr>
        <tr class=text>
        <td><b>Số điện thoại</b></td>
        <td><input name=txtsodt size=30 maxlength=30
value="<%=sdt%>"></td></tr>
          <tr class=text>
        <td><b>Thời gian gặp SV</b></td>
        <td><input name=txtthoigian size=30 maxlength=30
value="<%=tg%>"></td></tr>
```

```
            <tr class=text>
        <td><b>Địa điểm gặp SV</b></td>
        <td><input name=txtdiadiem size=30 maxlength=30
value="<%=dd%>"></td></tr>
        <tr class=text>
        <td colspan=3><hr size=1 color=red></td></tr>
        <tr class=text>
        <td> </td><td>
        <input type=submit value="<%=strAction%>" name=action>
        <input type=reset value=Huỷ></td></tr>
        </table>
        </form>
    </body>
</html>
```



Trang addnew.asp dùng chung cho trường hợp thêm mới và cập nhật. Khi người sử dụng chọn liên kết từ một giáo viên

trong danh sách addgvien.asp thì trang addnew.asp sẽ lấy giá trị Id trên QueryString. Khi đó ta có trang như sau

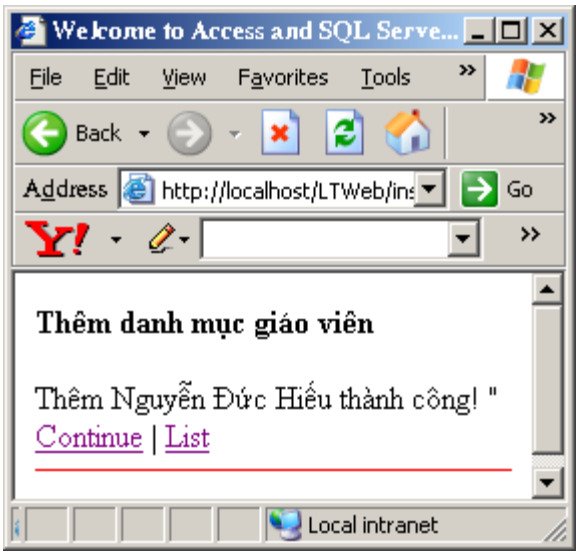


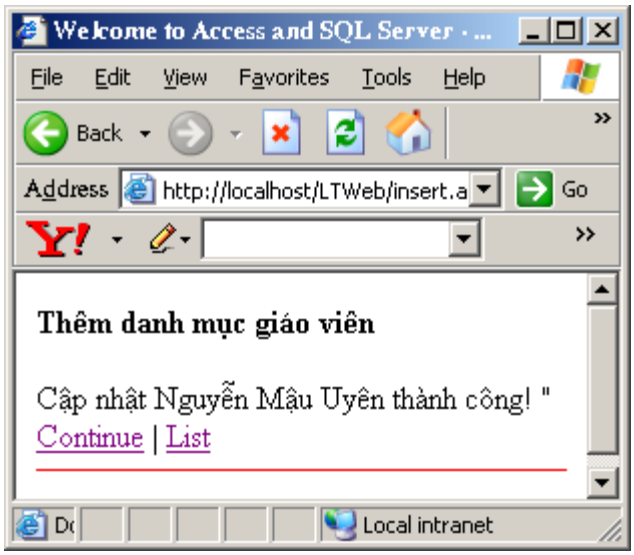
Ví dụ 3.2.2. **Thêm mẩu tin Insert.asp**

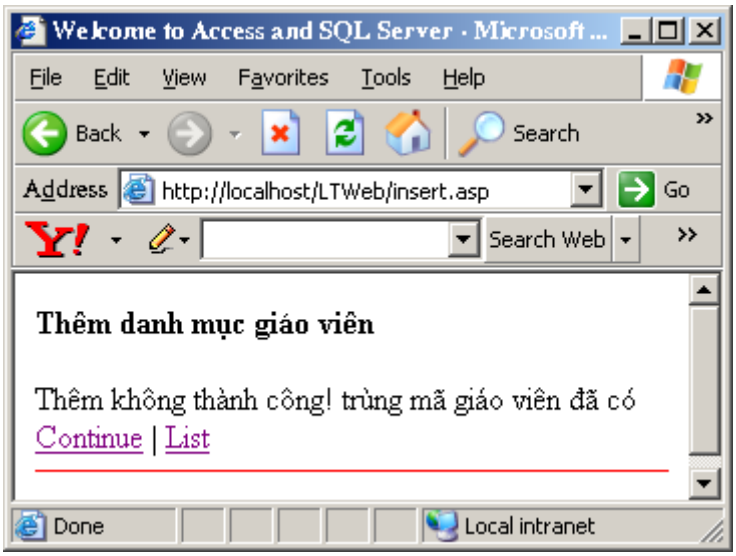
```
<%
    Session.CodePage=65001
    Dim myCon,myRst,id,name,sdt,tg,dd,strOK,strAction
    id=Request.Form("txtid")
    id=Replace(id,"'","''")
    name=Request.Form("txtname")
    name=Replace(name,"'","''")
    sdt=Request.Form("txtsodt")
    sdt=Replace(sdt,"'","''")
    tg=Request.Form("txtthoigian")
    tg=Replace(tg,"'","''")
    dd=Request.Form("txtdiadiem")
    dd=Replace(dd,"'","''")
    strAction=Request.Form("action")
```

```
%>
<html>
<head>
   <title> Welcome to Access and SQL Server </title>
   <LINK href="../style.css" rel=stylesheet>
   <META http-equiv=Content-Type content="text/html; charset=utf-
8">
</head>
<body>
<h4>Thêm danh mục giáo viên</h4>
<%
if Session("isSave")="" then
   Set myCon= CreateObject("ADODB.Connection")
   Set myRst= CreateObject("ADODB.Recordset")
   strPath=request.ServerVariables( _
   "APPL_PHYSICAL_PATH")
   strCon="Provider=" & _
   "Microsoft.Jet.OLEDB.4.0;" & _
   "Data Source= " & strPath & _
   "\database\BTTNCD.mdb"
   myCon.Open strCon
   strSQL="select * from giaovien "
   strSQL=strSQL & "Where MaGV='" & id & "'"
   myRst.Open strSQL, myCon
   if myRst.EOF Then
      If strAction="Save" Then
         strSQL="Insert into giaovien (maGV,tengv,sodt,thoigian,diadiem,hd)
Values('"
         strSQL=strSQL & id &"','" & name &"','" & sdt & "','" & tg &"','" & dd
&"',yes)"
         strOK=true
      Else
         strOK=false
      End If
```

```
    Else
        strSQL="Update giaovien Set TenGV='"
        strSQL=strSQL & name & "',sodt='"
        strSQL=strSQL & sdt & "',thoigian='"
        strSQL=strSQL & tg & "',diadiem='"
        strSQL=strSQL & dd & "' Where MaGV='"
        strSQL=strSQL & id &"'"
    End If
    myCon.Execute strSQL
    myRst.Close
    myCon.Close
    Set myCon=Nothing
    Session("isSave")="Y"
End If
If strAction="Save" Then
    if strOK then %>
        Thêm <%= name%> thành công! "
    <%else%>
        Thêm không thành công! Trùng mã giáo viên đã có
    <%end if
Else%>
 Cập nhật <%= name %> thành công! "
<%End If%><br>
<a href="addnew.asp">Continue</a> | <a
href="addgvien.asp">List</a>
<hr size=1 color=red>
</body>
</html>
```







3.2.2. Xóa dữ liệu

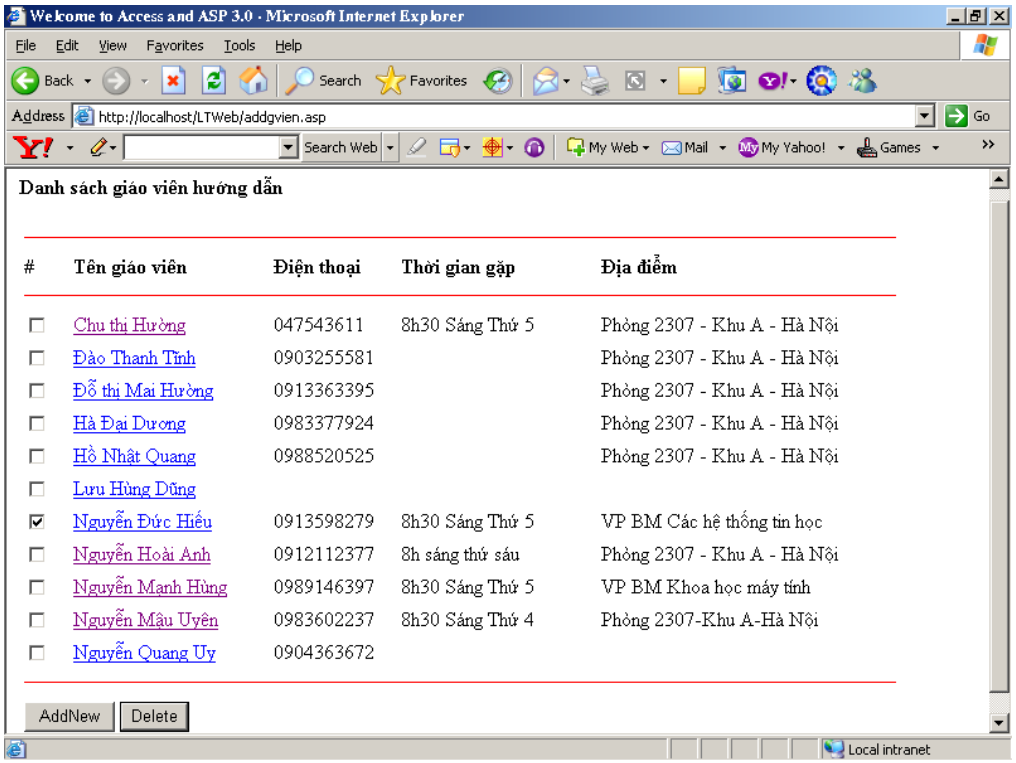
Để xoá mẫu tin, ta có hai cách:

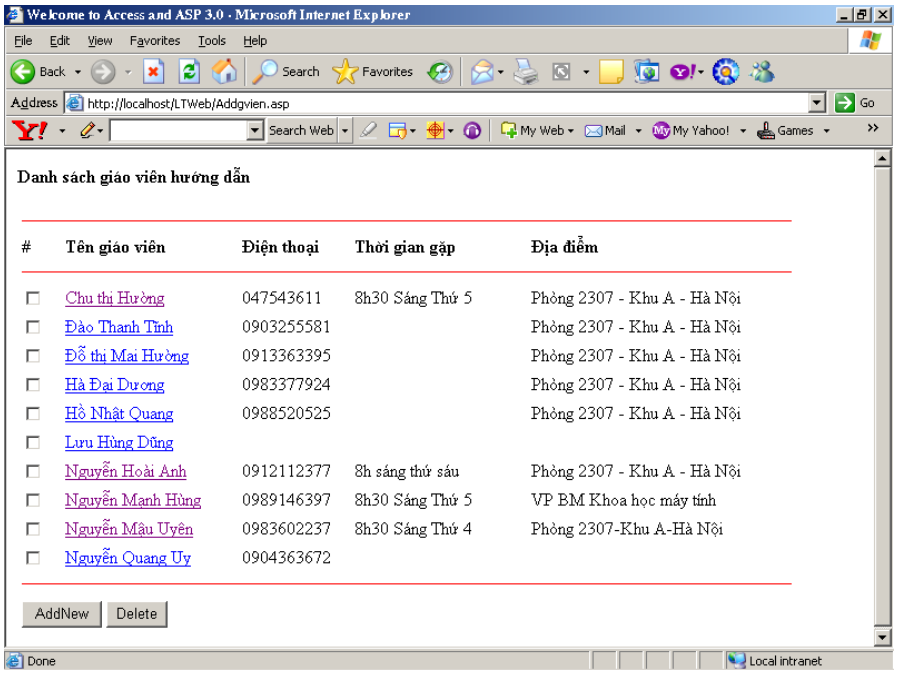
- **Cách thứ nhất dùng phương thức Execute của đối tượng Connection với phát biểu SQL dạng Delete.**
- **Cách hai dùng phương thức Delete của đối tượng RecordSet.**

Ví dụ 3.2.3. **Xoá nhiều mẫu tin.**

```
<%
```

```
Session.CodePage=65001
Dim myCon,id
if Session("isSave")="" then
  id=Request.Form("chkid")
  if id<>"" then
      Set myCon= CreateObject("ADODB.Connection")
      strPath=request.ServerVariables( _
      "APPL_PHYSICAL_PATH")
      strCon="Provider=" & _
      "Microsoft.Jet.OLEDB.4.0;" & _
      "Data Source= " & strPath & _
      "\Database\BTTNCD.mdb"
      myCon.Open strCon
      id=Replace(id," ","")
      id=Replace(id,",","'",'")
      strSQL="Delete from giaovien "
      strSQL=strSQL & " Where MaGV in ('" & id & "')"
      'Response.Write strSQL
      'Response.End
      myCon.Execute strSQL
      myCon.Close
      Set myCon=Nothing
      Session("isSave")="Y"
  End If
  Response.Redirect("Addgvien.asp")
End If
%>
```





MỤC LỤC TỔNG QUÁT

TRƯỜNG.....
KHOA……..

Giáo trình

Lập trình mạng

MỤC LỤC

Chương 1- Mở đầu

Yêu cầu: Giáo viên cần trình bày và trao đổi với sinh viên để sinh viên nắm được các kiến thức tổng quan sau:

1.1. Mạng máy tính và lịch sử phát triển

- Mạng máy tính là một tập các máy tính được kết nối với nhau thông qua các phương tiện truyền dẫn vật lý (như các loại cáp đồng, cáp quang, sóng vô tuyến, …)
- Lịch sử phát triển mạng máy tính:
    - Mạng trong Bộ quốc phòng của Mỹ
    - Triển khai nghiên cứu tại các trường Đại học ở Mỹ
    - Phát triển trên các châu lục và trở thành mạng Internet
- Việt ả am chính thức kết nối internet tháng 12/1997

1.2. Mô hình WWW (world wide web) và các dịch vụ liên quan

- Xét mô hình Client/Server
- Bộ giao thức TCP/IP
- Mô hình WWW
- Các dịch vụ liên quan: HTTP(Hyper Text Transfer Protocol), Email, Chat, FTP(File Transfer Protocol), Hosting, Domain,…

1.3. Thế nào là trang web tĩnh, trang web động

- Trang web tĩnh là trang web mà nội dung của nó được chuẩn bị sẵn trên Server. Khi người sử dụng yêu cầu thì toàn bộ nội dung của trang sẽ được gửi về trình duyệt của máy khách.
    - Ví dụ:
    - Các ưu, nhược điểm của web tĩnh
- Trang web động là trang web mà nội dung của nó không được chuẩn bị sẵn trên server, nó chỉ được tạo ra khi người sử dụng yêu cầu. Server khi nhận được yêu cầu từ Client sẽ thực hiện truy vấn dữ liệu từ CSDL (trên server hoặc từ server khác) theo yêu cầu, sau đó kết xuất tạo thành nội dung của trang và gửi về trình duyệt máy Client để xử lý tiếp
    - Ví dụ:
    - Các ưu, nhược điểm cuae web động

1.4. Các bước phát triển của web

- Website truyền thống: Không có sự quản lý, phân loại thông tin rõ ràng vì thế rất khó khăn trong việc tra cứu tìm kiếm thông tin
- Website thông minh: Có sự quản lý và phân loại thông tin nên rất dễ dàng trong việc tra cứu, tìm kiếm thông tin
- Portal (cổng): Mọi yêu cầu của người dùng đều có thể thực hiện trên web Portal. Hiện nay đa phần các web được xây dựng theo công nghệ Portal. Portal được chia làm 2 loại: Portal cổng ngang và Portal cổng dọc
    - Portal cổng ngang: phục vụ những người có khả năng sử dụng máy tính và Internet. Ví dụ: www.yahoo.com
    - Portal cổng dọc: phục vụ một nhóm người trong một phạm vi công việc nào đó. Ví dụ: Cổng thông tin điện tử hỗ trợ giao dịch, nghiệp vụ, kỹ thuật, … giữa các nhà điều hành của một công ty với các nhân viên trong cơ quan.
- Chú ý hiểu và phân biệt các khái niệm
    - Cổng thông tin điện tử
    - Cổng giao dịch điện tử
    - Chính phủ điện tử
    - Thương mại điện tử
    - …

1.5. Các công cụ và công nghệ thiết kế web

- ả gôn ngữ lập trình :
    - HTML (Hypertext Markup Language)
    - Javascript
    - Vbscript
    - ASP (Active Server Pages), ASP.ả ET
    - PHP
    - Java, JSP,…
- Các công cụ thiết kế :
    - Microsoft Fronpage
    - Macromedia Dreamweaver
    - Web Matrix (dùng cho ASP.ả ET), ...

Chương 2- Ngôn ngữ HTML

2.1. Thẻ (tag) HTML là gì?

Để biểu diễn thông tin trên trang web, www sử dụng ngôn ngữ HTML để trình bày thông tin. Mỗi thông tin chi tiết sẽ được trình bày và định dạng dựa vào một cặp thẻ (tag) HTML tương ứng.

- Mỗi cặp thẻ bao gồm: thẻ mở và thẻ đóng. Tên thẻ mở và thẻ đóng giống nhau và được đặt tỏng cặp dấu <tên thẻ>
- Trong thẻ mở có thể có thêm các tham số phía sau tên thẻ
- Trong thẻ đóng có thêm dấu / phía trước tên thẻ
- Dữ liệu cần trình bày đặt trong cặp thẻ mở và thẻ đóng
- Có một số thẻ không nhất thiết phải viết cả thẻ đóng
- Có thể đặt các cặp thẻ HTML lồng nhau

Cấu trúc chung của một thẻ (tag) HTML như sau:

<Ten_The thamso1=giatri1 thamso2=giatri2 …> Thông tin cần trình bày </Ten_The>

Ví dụ:

<B>**chữ đậm**</B>

<I>*chữ nghiêng*</I>

<U>chữ gạch chân<U>

<B><I>***chữ vừa đậm vừa nghiêng***</I></B>

2.2. Cấu trúc của một trang HTML

```
<html>
<head>
    <title>Tiêu đề trang web</title>
</head>
<body>
    ả ội dung trang web
</body>
</html>
```

2.3. Các thẻ định cấu trúc tài liệu

- <HTML> </HTML>
- <HEAD> </HEAD>
- <TITLE> </TITLE>
- <BODY> </BODY>

Các tham số của thẻ <BODY>

Các tham số	ý nghĩa
LINK	Chỉ định màu của văn bản siêu liên kết
ALINK	Chỉ định màu của văn bản siêu liên kết đang đang chọn
VLINK	Chỉ định màu của văn bản siêu liên kết đã từng mở
BACKGROUND	Chỉ định địa chỉ của ảnh dùng làm nền
BGCOLOR	Chỉ định màu nền
TEXT	Chỉ định màu của văn bản trong tài liệu
SCROLL	YES/NO - Xác định có hay không thanh cuộn
TOPMARGIN	Lề trên
RIGHTMARGIN	Lề phải

2.4. Các thẻ định dạng

- <P> </P>
- Các thẻ định dạng đề mục
-
- <B>,<I>,<U>,<S>
- ^{chỉ số trên},_{chỉ số dưới}
- Căn lề văn bản trên trang web: tham số ALIGả , thẻ <CEả TER>
- ***Định dạng Font chữ:***

  + Dùng thẻ META

  <meta http-equiv="Content-Language" content="en-us">

  <meta http-equiv="Content-Type" content="text/html; charset=utf-8">

  Thẻ này đặt trong cặp thẻ <HEAD>

  + Thẻ <FOả T> </FOả T>:

  Các thuộc tính:

Attribute	Example	Purpose
size="number"	size="2"	Defines the font size
size="+number"	Size="+1"	Increases the font size
size="-number"	Size="-1"	Decreases the font size
face="face-name"	face="Times"	Defines the font-name
color="color-value"	color="#eeff00"	Defines the font color
color="color-name"	color="red"	Defines the font color

  Ví dụ:

```
<p>
<font size="2" face="Verdana">
This is a paragraph.
</font>
</p>
<p>
<font size="3" face="Times">
This is another paragraph.
</font>
```

</p>

- <HR> tạo dòng kẻ ngang

2.5. Thẻ tạo link (liên kết)

<A HREF=url ả AME=name TABIả DEX=n TITLE=title TARGET=_blank|_self|tên frame> dòng văn bản </A>

Trong đó:

- HREF Địa chỉ của trang Web được liên kết, là một URL nào đó.
- ả AME Đặt tên cho vị trí đặt thẻ.
- TABLEIả DEX Thứ tự di chuyển khi ấn phím Tab
- TITLE Văn bản hiển thị khi di chuột trên siêu liên kết.
- TARGET Mở trang Web đ-ợc liên trong một cửa sổ mới (*_blank*) hoặc trong cửa sổ hiện tại (*_self*), trong một frame (tên frame).

Ghi chú:

ả ếu đặt thuộc tính href= của thẻ <a> giá trị mailto:address@domain thì khi kích hoạt kết nối sẽ kích hoạt chức năng thư điện tử của trình duyệt.

Ví dụ:

<**ADDRESS**>

Trang WEB này được

<**A** href=”mailto:webmaster@vnuh.edu.vn” >

WEBMASTER

<**A**> bảo trì

<**ADDRESS**>

Khi nhấn vào dòng chữ WEBMASTER (dòng chữ này sẽ xuất hiện giống như các siêu liên kết khác) chức năng thư tín của trình duyệt sẽ được kích hoạt và địa chỉ thư điện tử webmaster@vnuh.edu.vn sẽ được chèn vào địa chỉ nhận thư của chương trình gửi thư.

2.6. Thẻ tạo frame (khung)

Có thể thực hiện việc chia cửa sổ trình duyệt ra làm nhiều khung khác nhau gọi là frame. Trong mỗi khung cho phép hiển thị một trang web khác nhau.

Ví dụ:

```
<html>
<head> </head>
<frameset rows="64,*">
<frame name="tren" scrolling="no" noresize target="phai" src="tieude.htm">
<frameset cols="150,*">
<frame name="phai" target="trai" src="khungphai.htm">
<frame name="trai" src="khungtrai.htm">
</frameset>
<noframes>
<body>
```

```
<p>This page uses frames, but your browser doesn't support them.</p>
</body>
</noframes>
</frameset>
</html>
```

Ghi chú:

- Thẻ <FRAME> dùng để tạo ra các FRAME cụ thể
- <FRAMESETS> được viết để chứa các thẻ <FRAME> trong nó (ít nhất 2 FRAME trở lên)
- <ả OFRAMES> để hiển thị thông báo trong trường hợp trình duyệt không hỗ trợ FRAME
- Lưu ý cách truyền giá trị cho thuộc tính TARGET

2.7. Thẻ tạo Form

Để tạo ra một form trong tài liệu HTML, chúng ta sử dụng thẻ FORM với cú pháp như sau:

```
<FORM
        ACTION      = ulr
        METHOD      = GET | POST
        NAME = name
        TARGET      = frame_name | _blank | _self
        >
        <!-- Các phần tử điều khiển của form được đặt ở đây -->
        <INPUT ...>
        <INPUT ...>

</FORM>
```

Trong đó:

ACTION	Địa chỉ sẽ gửi dữ liệu tới khi form được submit (có thể là địa chỉ tới một chương trình CGI, một trang ASP...).
METHOD	Phương thức gửi dữ liệu.
NAME	Tên của form.
TARGET	Chỉ định cửa sổ sẽ hiển thị kết quả sau khi gửi dữ liệu từ form đến server.

2.8. Thẻ tạo đối tượng FORM

Các đối tương form bao gồm: Ô nhập văn bản 1 dòng, Ô nhập văn bản nhiều dòng, Radio, Checkbox, Button, Drop down, List, …

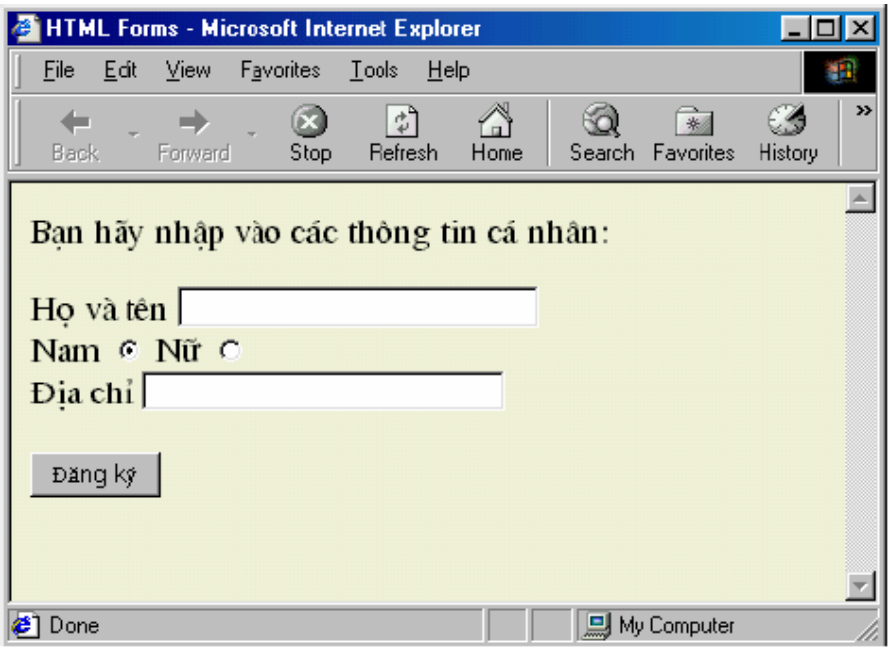
✓ **Cú pháp thẻ <Input>**

```
<INPUT
ALIGN = LEFT | CENTER | RIGHT
TYPE = BUTTON | CHECKBOX | FILE | IMAGE | PASSWORD |
RADIO | RESET | SUBMIT | TEXT
VALUE = value
>
```

Ví dụ:

```
<HTML>
<HEAD><TITLE>HTML Forms</TITLE></HEAD>
<BODY BGCOLOR="BEIGE">
<P>Bạn hãy nhập vào các thông tin cá nhân:</P>
<FORM METHOD="POST" ACTION="http://www.cse.com.vn/scripts">
Họ và tên <INPUT TYPE="TEXT" SIZE="30"><BR>
Nam <INPUT TYPE="RADIO" VALUE="nam" NAME="gioitinh" CHECKED>
Nữ <INPUT TYPE="RADIO" VALUE="nu" NAME="gioitinh"><BR>
Địa chỉ <INPUT TYPE="TEXT" SIZE="30"><BR><BR>
<INPUT TYPE="SUBMIT" VALUE="Đăng ký">
</FORM>
</BODY>
</HTML>
```

Kết quả:



✓ **Cú pháp thẻ <Select>**

```
<SELECT NAME="tên danh sách"  SIZE="chiều cao">
<OPTION VALUE=value1 SELECTED> Tên mục chọn thứ nhất
<OPTION VALUE=value2> Tên mục chọn thứ hai
<!-- Danh sách các mục chọn -->
</SELECT>
```

✓ **Cú pháp thẻ <Textarea>**

```
<TEXTAREA
        COLS=số cột
        ROWS=số hàng
        NAME=tên
>
        Văn bản ban đầu
</TEXTAREA>
```

2.9. Thẻ Marquee-tạo chữ chạy

<MARQUEE class=scroll scrollAmount=2 scrollDelay=80 direction=”kiểu cuộn” width=245 height=202 align="kiểu canh">

ả ội dung văn bản

</marquee>

- Class: chỉ định tên lớp đối tượng
- ScrollAmount:
- ScrollDelay: tốc độ cuộn
- Direction: định hướng chuyển động: up|down|left|right
- Width, Hight: độ rộng, cao qui định phạm vi hiển thị văn bản
- Align: canh văn bản: Center, middle, bottom

2.10. Các thẻ tạo bảng

<TABLE></TABLE>

<TR></TR>

<TD></TD>

2.11. Một số thẻ Multimedia

- ***Hiển thị một bức ảnh***

  <object height="100%" width="100%" type="image/jpeg" data="audi.jpeg">
  </object>

- ***Hiển thị một trang web***

  <object height="100%" width="100%" data="http://www.w3schools.com">
  </object>

- ***Nghe (play) 1 file âm thanh, video với trình media trên web***

  Ví dụ 1:

<object classid="clsid:22D6F312-B0F6-11D0-94AB-0080C74C7E95">
<param name="Fileả ame" value="liar.wav" />
</object>

Ví dụ 2:

<object classid="clsid:22D6F312-B0F6-11D0-94AB-0080C74C7E95">
<param name="Fileả ame" value="3d.wmv" />
</object>

- ***Hiển thị lịch***

<object width="50%" height="50%" classid="clsid:8E27C92B-1264-101C-8A2F-040224009C02">
<param name="BackColor" value="14544622">
<param name="DayLength" value="1">
</object>

- ***Hiển thị Flash***

<object width="400" height="40"
classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
codebase="http://download.macromedia.com
/pub/shockwave/cabs/flash/swflash.cab#4,0,0,0">
<param name="SRC" value="bookmark.swf">
<EMBED src=”bookmark.swf” quality=hight BgClolor=”#FFFFFF” Width=550
Height=400 name=”Preloader” Align=”” Type=”Application/x-shockwave-flash”>
</EMBED>
</object>

Ghi chú:

+ Thẻ <OBJECT> báo cho Internet Explore biết cách hiển thị flash
+ Thẻ <EMBED> báo cho ả etscape ả avigator biết cách hiển thị flash

Ghi chú: Mã số một số phiên bản media của windows

- Windows Media Player 10: clsid:6BF52A52-394A-11D3-B153-00C04F79FAA6
- Windows Media Player 9: clsid:6BF52A52-394A-11D3-B153-00C04F79FAA6

2.12. Bảng mã và tên gọi trong HTML của một số ký tự đặc biệt

KÝ TỰ	MÃ	TÊN TRONG HTML
"	"	"
&	&	&
<	<	<
>	>	>
break		
£	£	£
¤	¤	¤
¥	¥	¥
§	§	§
©	©	©
®	®	®
±	±	±
²	²	²
³	³	³
¼	¼	¼
½	½	½
¾	¾	¾
«	«	«
»	»	»
×	×	&timer;
÷	÷	÷
Ø	ø	ø

2.13. Thiết kế web sử dụng Microsoft Frontpage

2.13.1. Các thao tác chính khi soạn thảo một trang web

- Tạo mới một trang web: Chọn chức năng File/ả ew/Page /ả ormal Page hay chọn icon ả ew trên Toolbar.
- Lưu trang web: Chọn chức năng File/Save hay chọn icon Save trên Toolbar. Để lưu trang web dưới dạng một tên khác chọn chức năng File/Save As…
- Xem trước hiển thị của trang web đang thiết kế: Chọn chức năng File/Preview in Browser. Lúc này cửa sổ trình duyệt Internet Explorer (IE) sẽ hiển thị trang web mà chúng ta đang soạn thảo. ả ên lưu trang web trước khi chọn chức năng này.
- Cửa sổ màn hình soạn thảo trang web cung cấp 3 cách "hiểu" (view) khác nhau về 1 trang web.

- ẩ ếu bạn chọn ẩ ormal view, bạn có thể biên tập trang web dưới dạng WYSIWYG. Ví dụ bạn có thể gừ văn bản vào, thay đổi màu sắc, kích thước chữ, …
- ẩ ếu bạn chọn HTML view, bạn sẽ thấy được các mã HTML tương ứng với cách trình bày của trang web của bạn. Ví dụ, nếu trong ẩ ormal view bạn chèn vào một table thì trong HTML view, bạn sẽ thấy các tag tương ứng như sau:

```
<table border="1" width="100%">
        <tr><td width="100%"> </td></tr>
</table>
```

- ẩ ếu bạn chọn Preview view, tương tự với chức năng Preview in Browser

2.13.2. Tạo các thành phần của trang web

2.13.2.1. Thời gian cập nhật (Time stamp)

- Chọn Insert/Date and Time.
- Chọn định dạng ngày tháng và thời gian phù hợp với nhu cầu của bạn

2.13.2.2. Đường kẻ ngang (Horizontal line)

- Chọn Insert/Horizotal line.
- Đặt các thuộc tính cho đường kẻ ngang bằng cách double click chuột lên nó. Sau đó chọn các thông số về Width, Height, Color, Alignment

2.13.2.3. List

- Chọn Format/Bullets and ẩ umbering.
- Sau khi hộp hội thoại xuất hiện, bạn hãy chọn các dạng bullet và numbering tương ứng.
- Để thay đổi các thuộc tính của bullet như màu sắc, kiểu chữ, …, ta chọn Format/Bullets and ẩ umbering/Style
- ẩ goài ra, ta cũng có thể chọn hình ảnh để thay cho các kiểu bullet thông thường. Để thay đổi, ta chọn Format/Bullets and ẩ umbering/Picture bullets, rồi chọn ảnh dùng để làm bullet
- Để bỏ định dạng bullets, ta chọn Format/Bullets and ẩ umbering/Plain Bullets.

2.13.2.4. Tables

- ẩ gười ta thường dùng table để:
    - Hiển thị các thông tin có dạng dòng/cột, ví dụ như bảng thời khóa biểu, thông tin sản phẩm, ..
    - Trình bày (layout) các văn bản(text) và các ảnh đồ họa(graphics).
- Để tạo một bảng, ta có thể dùng một trong hai cách:
    - Chọn Table/Insert Table. Khi hộp hội thoại tạo bảng hiện ra, bạn phải cung cấp các thông tin chi tiết cho việc tạo bảng, ví dụ như số dòng, số cột, kích thước,...
    - Chọn Table/Draw Table. Với chức năng này, bạn sẽ dùng bút vẽ để tạo các dòng, cột

- Để không hiện (hide) border của bảng, ta click phải chuột lên table, chọn Table Properties/Border/Sizes bằng 0.
- Để tách một cell hay trộn nhiều cell lại, ta chọn Tabe rồi chọn Split /Merge Cells.
- Để thêm hoặc xóa các cell, ta chọn Table rồi chọn Insert/Delete Cells.

2.13.2.5. Một số hiệu ứng đặc biệt

- Chuyển trang (Page transition): Chọn Format/Page Transition…
- Hiệu ứng chữ chuyển động theo chiều ngang (Marquees): Chọn Insert/Component/Marquee. Sau khi hộp thoại hiện ra, bạn gõ vào dòng chữ cần chuyển động và đặt các thuộc tính khác như màu sắc, font chữ, …
- Thêm hiệu ứng font chữ cho các hyperlink: Chọn Format/Background, check vào Enable hyperlink rollover effects. Sau đó bạn có thể chọn các màu theo ý muốn.

2.6. Chèn ảnh

- Chọn chức năng Insert/Picture/From File.
- Đặt thuộc tính và kích thước của ảnh, click phải chuột lên ảnh, rồi chọn Picture Properties.
- Để tạo các hotspot hyperlink, ta chọn hình vẽ tương ứng (hình chữ nhật, ellipse, …) trên thanh toolbar pictures. Chọn vùng trên ảnh, rồi điền thông tin của hyperlink vào

2.7. Chèn hyperlink

- Chọn chức năng Insert/Hyperlink.
- Sau khi hộp thoại hiện ra, gõ vào hyperlink tương ứng. Có 3 dạng:
    - Địa chỉ Internet, có dạng: http://… Ví dụ: http:// www.yahoo.com
    - Liên kết tới một trang trong site, có dạng: /thư mục/tên tập tin. Ví dụ: ../images/shopping.htm
    - Liên kết ngay chính trong trang, đặt bằng bookmark.

2.13.3. Định dạng trang

- Click phải chuột lên trang, chọn Page Properties.
- Để đặt màu nền cho trang, chọn Background/Colors/Background
- Để đăt ảnh nền cho trang, chọn Background/Formatting/Background picture.
- Để đặt các thông số về màu sắc cho hyperlink, chọn Background/Colors/Hyperlink
- Để đặt tiêu đề cho trang chọn General/Title.

2.14. Bài tập thực hành

2.14.1. Tạo trang web cho nhóm.

- Khởi động Microsoft Front Page 2000.
- Soạn trang web cá nhân của nhóm đặt tên là InfoGroup<n>.htm (ví dụ nhóm 1 sẽ lấy tên là InfoGroup1.htm). Các thông tin chính của các thành viên trong nhóm bao gồm: Họ và tên, Công việc và chức vụ hiện nay, Địa chỉ liên lạc, Điện thoại, Fax, Email, Mobile Phone, ... ả goài ra các nhóm có thể bổ sung các thông tin khác.
- Upload trang web đó soạn lên website.

- Thử truy cập trang web đó tải lên.

2.14.2. Tham khảo các site

1. Site thông tin của Việt ả am

http://vnexpress.net/

http://www.vnn.vn/.

http://www.vdc.com.vn

3. Tham khảo các site bán hàng nổi tiếng

http://shopping.yahoo.com

http://www.amazon.com

4. Tham khảo các site giải trí

http://greetings.yahoo.com

http://www.vnn.vn/ecards/

http://www.fpt.vn/Postcard/main.asp

http://www.geraldstevens.com/

http://www.1800flowers.com/

5. Tham khảo các site về lao động việc làm tại Vả

http://203.162.5.43/ld2000/

http://www.vietname-business.com/jobnld/

http://www.jobsonline.saigonnet.vn/

6. Tạo trang web chứa các hyperlink dùng để truy cập nhanh

Tạo trang web đặt tên là Links.htm chứa các hyperlink đó đề cập ở trên. Bổ sung thêm các hyperlink và các phân loại khác mà các anh chị đó biết.

Upload lên website và kiểm tra lại.

7. Tạo trang HomePage

Tạo trang web HomePage đặt tên là Default.htm giới thiệu về nhóm và các công việc mà nhóm đang triển khai.

Kết nối hai trang đó tạo vào HomePage.

Upload lên website và kiểm tra lại.

8. Chọn chủ đề để thiết kế website

Website về dịch vụ việc làm.

Website về dịch vụ nhà đất (http://www.nhadat.com).

Website báo điện tử (http://vnexpress.net)

Website trường học.

Website bán hàng (cửa hàng, siêu thị ảo trên Internet)

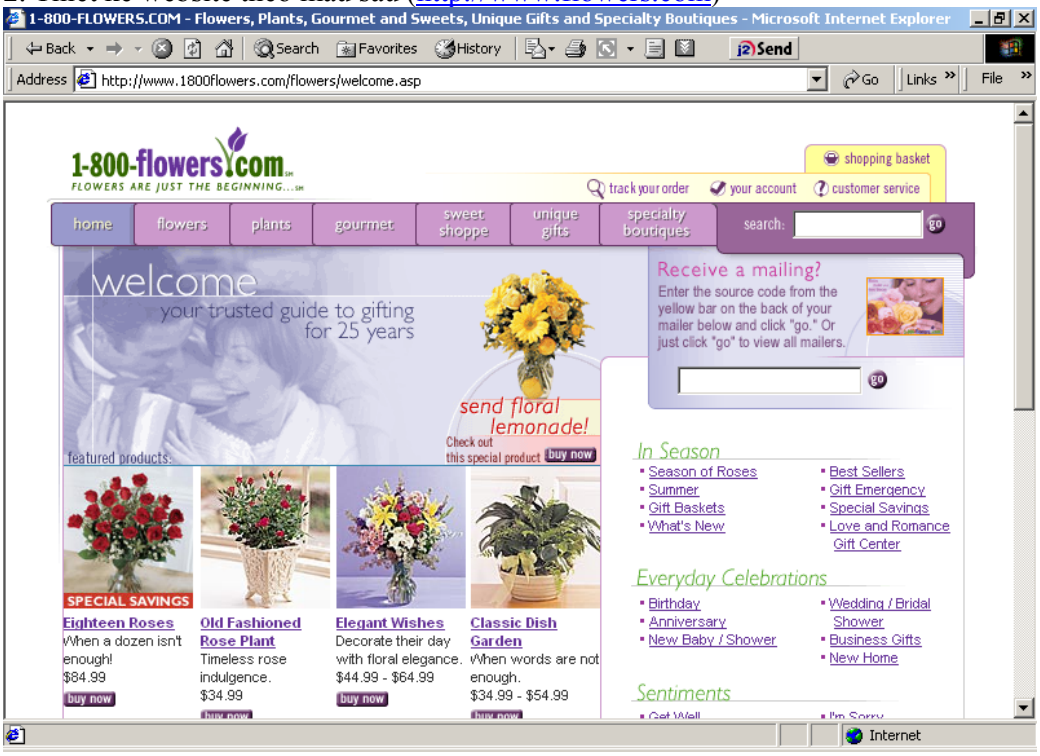
Website dịch vụ giải trí như ECards, Điện hoa, ...

Các chủ đề khác...

2.14.3. Thiết kế websitte theo các mẫu

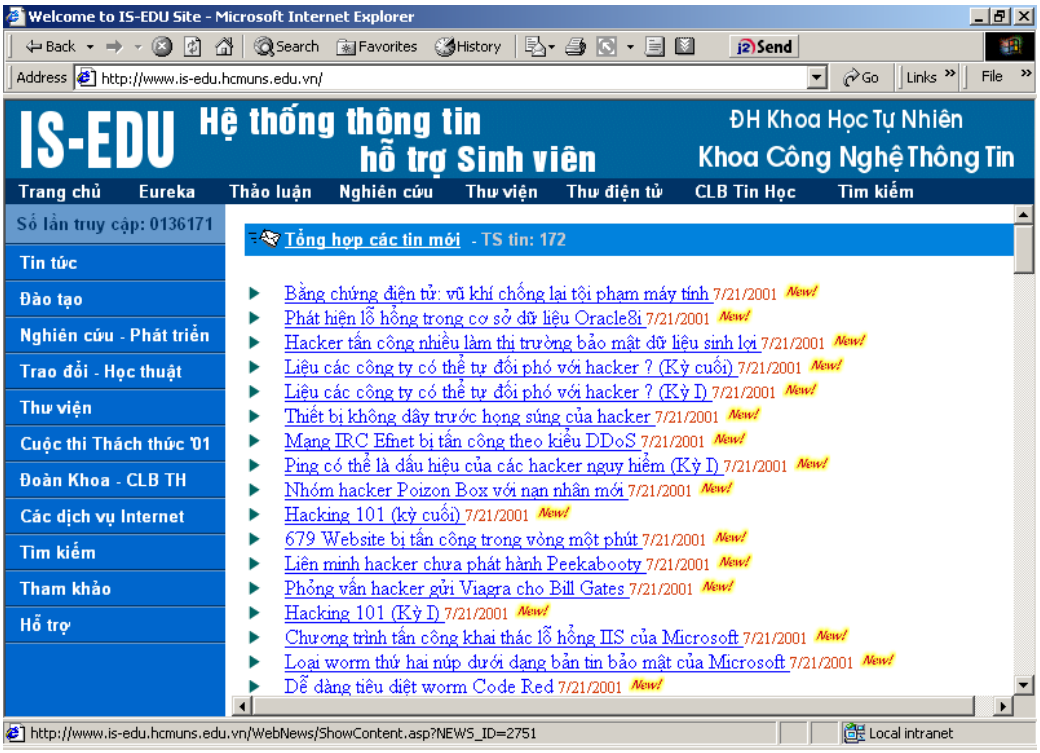
1. Thiết kế website theo mẫu sau (http://greetings.yahoo.com)

2. Thiết kế website theo mẫu sau (http://www.flowers.com)



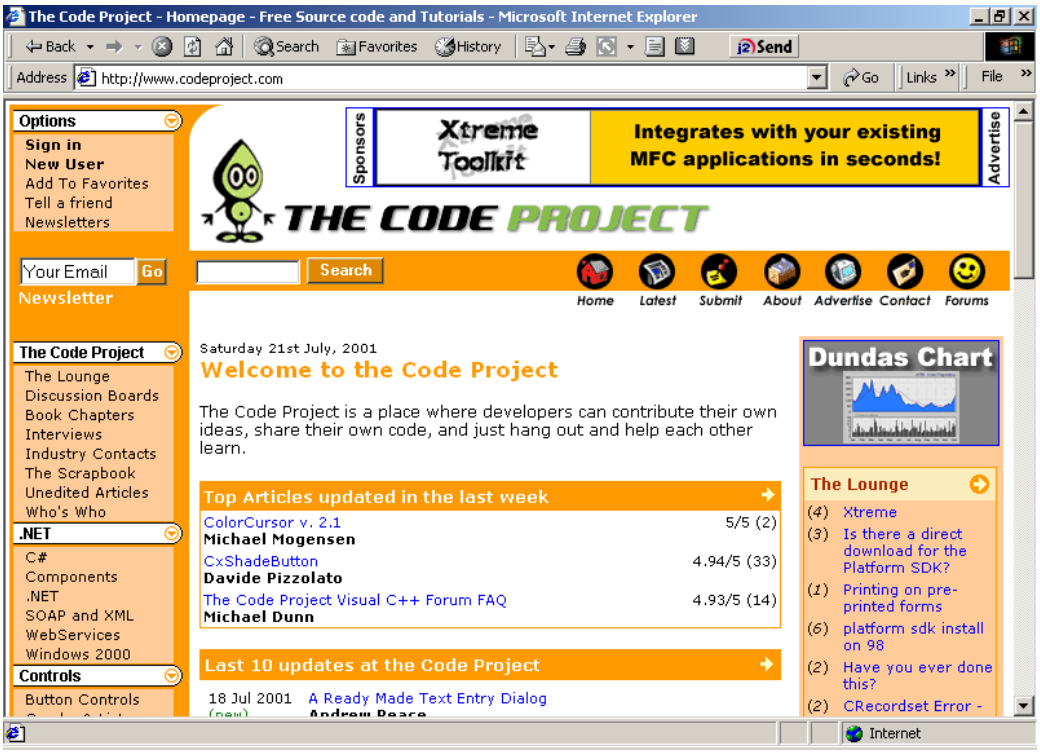
3. Thiết kế website theo mẫu sau (http://shopping.yahoo.com)

4. Thiết kế website theo mẫu sau (http://www.is-edu.hcmuns.edu.vn)

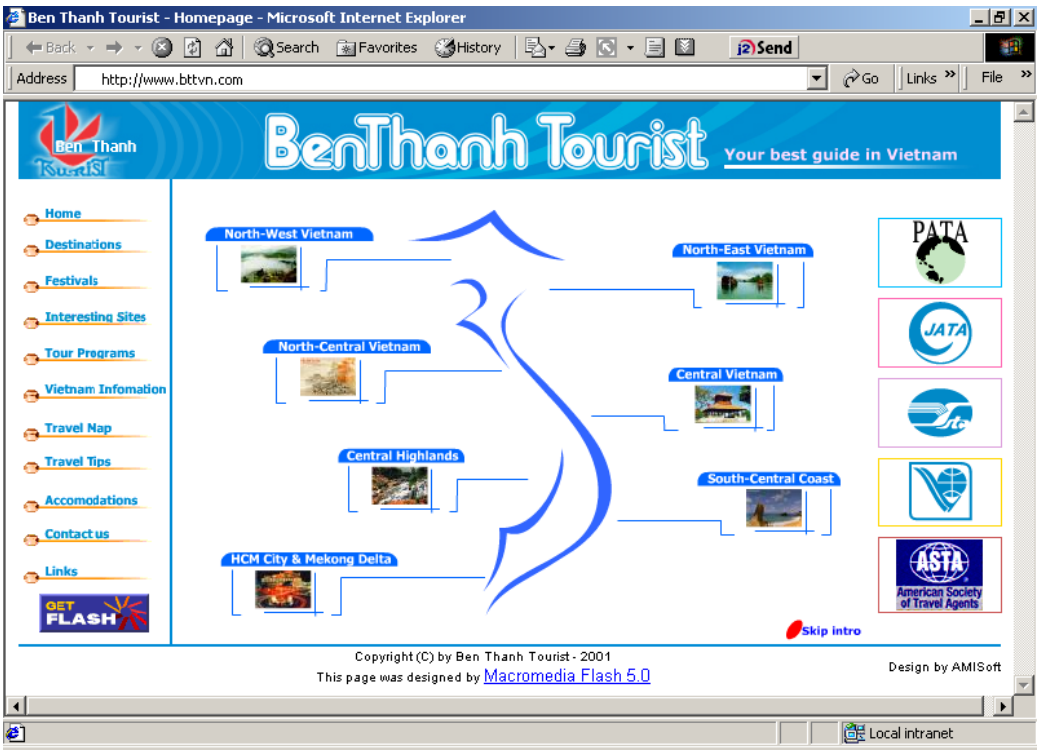


5. Thiết kế website theo mẫu sau (http://vnexpress.net)

6. Thiết kế website theo mẫu sau (http://www.codeproject.com)



7. Thiết kế website theo mẫu sau (http://www.bttvn.com)



- Để tham khảo cách thiết kế các trang web trên, hãy:
    - Tải về máy bằng cách chọn chức năng Save của IE.

  - Dùng MS Front Page để mở tập tin đó lưu lên

- Chuyển qua lại giữa các màn hình ả ormal View và HMTL View để biết cách thiết kế
- Với mỗi trang web đó xem hãy chú ý các vấn đề sau:
  - Cách bố trí các bảng (table)
  - Cách sử dụng font chữ
  - Cách chèn vào các hình ảnh để trang trí
  - Cách định nghĩa các thuộc tính như màu chữ, màu nền, ...
- Phần tĩnh (không thay đổi) mỗi khi click vào các hyperlink chuyển qua một nội dung mới
- Từ các trang web trên, hãy tự thiết kế các trang cho website của lớp, nhóm,….

Chương 3- Ngôn ngữ kịch bản Javascript

3.1. Tổng quan về JavaScript

- JavaScript là ngôn ngữ thể hiện dưới dạng Script có thể gắn với các file tài liệu HTML. Một số trình duyệt hỗ trợ Javascript sẽ thực hiện việc diễn dịch thay vì biên dịch khi gặp các mã JavaScript.
- JavaScript là ngôn ngữ dựa trên đối tượng. ả ó có thể đáp ứng các sự kiện (ví dụ: onClick, onMouseOver, onMouseOut,…). Tuy nhiên không giống như C++, Java, do không hỗ trợ các lớp hay tính kế thừa
- Các từ khoá sử dụng trong JavaScript có phân biệt chữ hoa, chữ thường
- JavaScript có thể chạy trên bất cứ hệ điều hành nào có trình duyệt hỗ trợ JavaScript.
- Các mã lệnh JavaScript hoàn toàn được xử lý trên trên trình duyệt phía máy trạm.

Các giới hạn của JavaScript:

- JavaScript không thể giao tiếp với máy chủ
- JavaScript không thể tạo các hình ảnh đồ hoạ
- JavaScript làm việc khác nhau trên các trình duyệt khác nhau

3.2. Nhúng JavaScript vào trang tài liệu HTML

3.2.1. Sử dụng thẻ <SCRIPT>

- Tất cả các mã JavaScript khi nhúng vào trang web đều phải nằm trong cặp thẻ:

<script language=”Javascript”> Mã lệnh Javascript **</script>**

Ví dụ:

```
<html>
<head>
<title>Tiêu đề trang web</title>
<script language=”Javascript”>
        //Chỉ có mã lệnh Javascript được đặt ở đây
</script>
</head>
<body>
<script language=”Javascript”>
        //Chỉ có mã lệnh Javascript được đặt ở đây
</script>
</body>
</html>
```

- ***Lưu ý:*** đối với các trình duyệt cũ không hiểu JavaScript, ta nên chèn thêm thẻ **<!- -** trên một dòng riêng nay sau thẻ <script> và **//- ->** trên một dòng riêng ngay trước thẻ </script> để trình duyệt bỏ không thực hiện các mã JavaScript khi gặp chúng.

Ví dụ:

```
<html><head><title>Tiêu đề trang web</title>
        <script language=”Javascript”>
        <!- - dấu JavaScript trong trình duyệt cũ
                //Chỉ có mã lệnh Javascript được đặt ở đây
        //kết thúc đoạn dấu mã - ->
        </script>
        </head>
<body>
        <script language=”Javascript”>
        <!- -
            //Chỉ có mã lệnh Javascript được đặt ở đây
        // - ->
        </script></body></html>
```

- Để chèn chú thích trong mã JavaScript sử dụng kí hiệu: *//dòng chú thích*

3.2.2. Sử dụng một file nguồn JavaScript

Thuộc tính *SRC* của thẻ *<SCRIPT>* cho phép bạn chỉ rõ file nguồn JavaScript được sử dụng (dùng phương pháp này hay hơn nhúng trực tiếp một đoạn lệnh JavaScript vào trang HTML).

Cú pháp:

```
<SCRIPT SRC="file_name.js">
....
</SCRIPT>
```

Chú ý: Các file JavaScript bên ngoài không được chứa bất kỳ thẻ HTML nào. Chúng chỉ được chứa các câu lệnh JavaScript và định nghĩa hàm.

3.2.3. Sử dụng thẻ <Noscript>

Sử dụng thẻ <ẩ oscript> để thông báo cho người sử dụng biết trình duyệt không hỗ trợ Javascript

Ví dụ:

```
<NOSCRIPT>
<B> Trang này có sử dụng JavaScript. Do đó bạn cần sử dụng trình duyệt Netscape
Navigator từ version 2.0 trở đi!
<BR>
<A HREF="http://home.netscape.com/comprd/mirror/index.html">
Hãy kích chuột vào đây để tải về phiên bản Netscape mới hơn
</A>
</BR>
Nếu bạn đã sử dụng trình duyệt Netscape từ 2.0 trở đi mà vẫn đọc được dòng chữ này thì
hãy bật Preferences/Advanced/JavaScript lên
</NOSCRIPT>
```

3.3. Hiển thị dòng Text

Đối tượng **document** trong JavaScript được thiết kế sẵn hai cách thức để xuất một dòng text ra màn hình client: **write()** và **writeln().** Cách gọi một cách thức của một đối tượng như sau:

object_name.property_name

Dữ liệu mà cách thức dùng để thực hiện công việc của nó được đưa vào dòng tham số, ví dụ:

document.write("Test");

document.writeln('Test');

Cách thức **write**() xuất ra màn hình xâu Text nhưng không xuống dòng, còn cách thức writeln() sau khi viết xong dòng Text tự động xuống dòng. Hai cách thức này đều cho phép xuất ra thẻ HTML.

Ví dụ: Cách thức write() xuất ra thẻ HTML

```
<HTML>
<HEAD>
<TITLE>Ouputting Text</TITLE>
</HEAD>
<BODY> This text is plain.<BR> <B>
<SCRIPT LANGUAGE="JavaScript">
<!-- HIDE FROM OTHER BROWSERS
document.write("This text is bold.</B>");
// STOP HIDING FROM OTHER BROWSERS -->
</SCRIPT>
</BODY>
</HTML>
```

3.4. Giao tiếp với người sử dụng

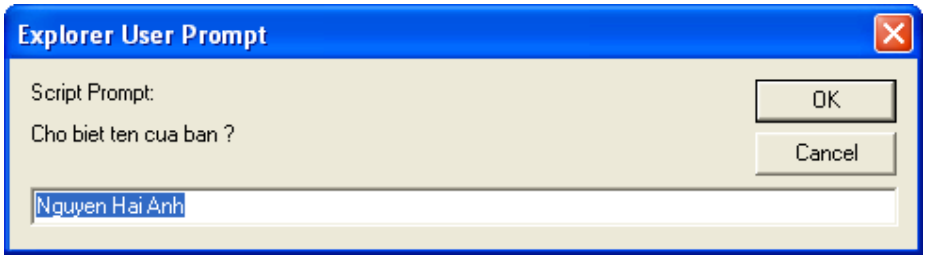
- Hộp thoại cảnh báo alert, ví dụ: alert("ả han OK de tat hop thoai nay !")

- Hộp thoại hỏi đáp. Hàm confirm("ả ội dung thông báo")

Ví dụ: k=confirm("Co chac chan xoa khong ?");

- Hộp thoại nhập dữ liệu. Hàm prompt("nội dung thông báo","giá trị mặc định")

Ví dụ: k=prompt("Cho biet ten cua ban ?","ả guyen Hai Anh");



3.5. Khai báo biến

✓***Khai báo không tường minh: ten_bien = bieu_thuc;***

✓***Khai báo tường minh:***

- Khai báo biến và không khởi tạo giá trị: ***var ten_bien;***
- Khai báo biến và khởi tạo giá trị ban đầu: ***var ten_bien = bieu_thuc;***

Quy tắc đặt tên biến:

- Ký tự bắt đầu phải là một chữ cái hoặc dấu gạch dưới "_"
- Trong tên biến không chứa khoảng trắng và các kỹ tự đặc biệt

Các biến phân biệt chữ hoa, chữ thường

3.6. Kiểu dữ liệu

3.6.1. Kiểu nguyên (INTERGER)

Số nguyên có thể được biểu diễn theo ba cách:

- *Hệ cơ số 10* (hệ thập phân) - có thể biểu diễn số nguyên theo cơ số 10, chú ý rằng chữ số đầu tiên phải khác 0.
- *Hệ cơ số 8* (hệ bát phân) - số nguyên có thể biểu diễn dưới dạng bát phân với chữ số đầu tiên là số 0.
- *Hệ cơ số 16* (hệ thập lục phân) - số nguyên có thể biểu diễn dưới dạng thập lục phân với hai chữ số đầu tiên là 0x.

3.6.2. Kiểu dấu phẩy động (FLOATING POINT)

Một literal có kiểu dấu phẩy động có 4 thành phần sau:

- Phần nguyên thập phân.
- Dấu chấm thập phân (.).
- Phần dư.
- Phần mũ.

Để phân biệt kiểu dấu phẩy động với kiểu số nguyên, phải có ít nhất một chữ số theo sau dấu chấm hay **E.** Ví dụ:

9.87
-0.85E4
9.87E14
.98E-3

3.6.3. Kiểu LOGIC (BOOLEAN)

Kiểu logic được sử dụng để chỉ hai điều kiện : đúng hoặc sai. Miền giá trị của kiểu này chỉ có hai giá trị

- true.
- false.

3.6.4. Kiểu chuỗi (STRING)

Một literal kiểu chuỗi được biểu diễn bởi không hay nhiều ký tự được đặt trong cặp dấu " ... " hay '... '. Ví dụ:

"The dog ran up the tree"
'The dog barked'

"100"

Để biểu diễn dấu nháy kép ("), trong chuỗi sử dụng (\"), ví dụ:

document.write(" \"This text inside quotes.\" ");

3.7. Các toán tử

- Gán (=)

Ví dụ: x = a+b

Kiểu gán thông thường	*Kiểu gán rút gọn*
x = x+y	x+=y
x = x-y	x-=y
x = x/y	x/=y
x = x%y	x%=y

- So sánh

Toán tử	Ý nghĩa
= =	So sánh bằng
!=	So sánh không bằng
>	So sánh lớn hơn
>=	So sánh lớn hơn hoặc bằng
<	So sánh nhỏ hơn
<=	So sánh nhỏ hơn hoặc bằng

- Số học

✓ Cộng(+), trừ (-), nhân (*), chia(/), lấy phần dư (%)

✓ bien++ (hoặc: ++bien) tăng giá trịcủa bien lên 1

✓ bien - -(hoặc: - -bien) giảm giá trị của biến đi 1

- Chuỗi

Để nối chuỗi sử dụng toán tử "+"

Ví dụ: "Hà"+" "+"ẩ ội" được "Hà ẩ ội"

- Logic

- Phép && (phép và)
- Phép || (phép hoặc)
- Phép ! (phủ định)

- Đối tượng

new: toán tử khai báo biến kiểu đối tượng

- Các toán tử trên bit (Bitwise)

Với các toán tử thao tác trên bit, đầu tiên giá trị được chuyển dưới dạng số nguyên 32 bit, sau đó lần lượt thực hiện các phép toán trên từng bit.

& Toán tử bitwise Aẩ D, trả lại giá trị 1 nếu cả hai bit cùng là 1.

| Toán tử bitwise OR, trả lại giá trị 1 nếu một trong hai bit là 1.

^ Toán tử bitwise XOR, trả lại giá trị 1 nếu hai bit có giá trị khác nhau

ả goài ra còn có một số toán tử dịch chuyển bitwise. Giá trị được chuyển thành số nguyên 32 bit trước khi dịch chuyển. Sau khi dịch chuyển, giá trị lại được chuyển thành kiểu của toán hạng bên trái. Sau đây là các toán tử dịch chuyển:

<< Toán tử dịch trái. Dịch chuyển toán hạng trái sang trái một số lượng bit bằng toán hạng phải. Các bit bị chuyển sang trái bị mất và 0 thay vào phía bên phải. Ví dụ: 4<<2 trở thành 16 (số nhị phân 100 trở thành số nhị phân 10000)

>> Toán tử dịch phải. Dịch chuyển toán hạng trái sang phải một số lượng bit bằng toán hạng phải. Các bit bị chuyển sang phải bị mất và dấu của toán hạng bên trái được giữ nguyên. Ví dụ: 16>>2 trở thành 4 (số nhị phân 10000 trở thành số nhị phân 100)

>>> Toán tử dịch phải có chèn 0. Dịch chuyển toán hạng trái sang phải một số lượng bit bằng toán hạng phải. Bit dấu được dịch chuyển từ trái (giống >>). ả hững bit được dịch sang phải bị xoá đi. Ví dụ: -8>>>2 trở thành 1073741822 (bởi các bit dấu đã trở thành một phần của số). Tất nhiên với số dương kết quả của toán tử >> và >>> là giống nhau.

Có một số toán tử dịch chuyển bitwise rút gọn:

Kiểu bitwise thông thường	*Kiểu bitwise rút gọn*
x = x << y	x << = y
x = x >> y	x - >> y
x = x >>> y	x >>> = y
x = x & y	x & = y
x = x ^ y	x ^ = y
x = x | y	x | = y

3.8. Cấu trúc lệnh

3.8.1. Câu lệnh rẽ nhánh

(1) Câu lệnh if

Cú pháp:

```
if (dieu_kien)
    {
    //các câu lệnh nếu dieu_kien đúng;
    }
else
    {
    //các câu lệnh nếu điều kiện sai;
    }
```

Ví dụ:

```
var a=3;b=2;
var max;
```

```
if (a>b)
{
  document.write("Số lớn là: " + a);
}
else
{
  document.write("Số lớn là: " + b);
}
```

(2) Câu lệnh lựa chọn switch

Cú pháp:

```
switch (bieu_thuc){
    case gia_tri_1:
            //mã lệnh thực hiện khi bieu_thuc= =gia_tri_1
            break;
    case gia_tri_2:
            //mã lệnh thực hiện khi bieu_thuc= =gia_tri_2
            break;
    …….
    case gia_tri_n:
            //mã lệnh thực hiện khi bieu_thuc= =gia_tri_n
            break;
    default:
            //mã lệnh thực hiện khi không có giá trị nào phù hợp
}
```

3.8.2. Câu lệnh lặp

(1) Câu lệnh for (số lần lặp xác định trước)

Cú pháp:

```
for (khởi tạo biến đếm; điều kiện; thay đổi giá trị biến đếm){
        //các câu lệnh thực hiện trong khi lặp
}
```

Ví dụ:

```
for(i=1;i<=10;i++)
  {
  document.writeln(i+"<br>");
  }
```

(2) Câu lệnh lặp while (số lần lặp không xác định trước)

Cú pháp:

```
while (dieu_kien){
//các câu lệnh thực hiện trong khi lặp
```

}

Ví dụ:

```
var i=1;
while(i<=10)
    {
    document.write(i +"<br>");
    i++;
    }
```

3.8.3. Câu lệnh break và continue

Hai câu lệnh này sử dụng trong các vòng lặp for hoặc while

- break: kết thúc vòng lặp và thoát ra khỏi còng lặp
- continue: dừng bước lặp hiện thời, trở về đầu vòng lặp và thực hiện bước lặp tiếp theo.

3.9. Hàm

3.9.1. Khai báo hàm

Cú pháp:

```
function  ten_ham(các tham số){
//các câu lệnh trong hàm
[return bieu_thuc;] //kết thúc và trả về giá trị cho hàm
}
```

3.9.2. Các hàm có sẵn

- eval(st)
- ẩ umber(st)
- parseInt(st, radius)
- parseFloat(st, radius)

3.10. Các lệnh thao tác trên đối tượng

3.10.1. for … in

Câu lệnh này được sử dụng để lặp tất cả các thuộc tính (properties) của một đối tượng. Tên biến có thể là một giá trị bất kỳ, chỉ cần thiết khi bạn sử dụng các thuộc tính trong vòng lặp. Ví dụ sau sẽ minh hoạ điều này

Cú pháp

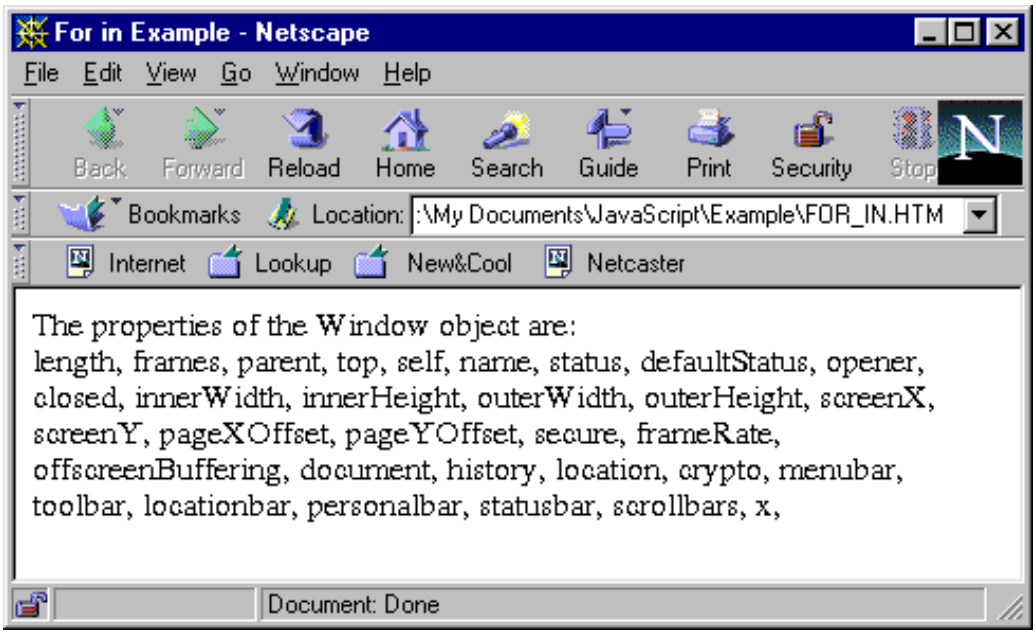
```
for (<variable> in <object>)
{
        //Các câu lệnh
}
```

Ví dụ

Ví dụ sau sẽ lấy ra tất cả các thuộc tính của đối tượng Window và in ra tên của mỗi thuộc tính. Kết quả được minh hoạ trên hình 5.2.

```
<HTML>
<HEAD>
<TITLE>For in Example </TITLE>
```

```
<SCRIPT LAả GUAGE= "JavaScript">
        document.write("The properties of the Window  object are: <BR>");
for (var x in window)
        document.write("     "+ x + ", ");
</SCRIPT>
</HEAD>
<BODY>
</BODY>
</HTML>
```



3.10.2. new

Biến ***new*** được thực hiện để tạo ra một thể hiện mới của một đối tượng

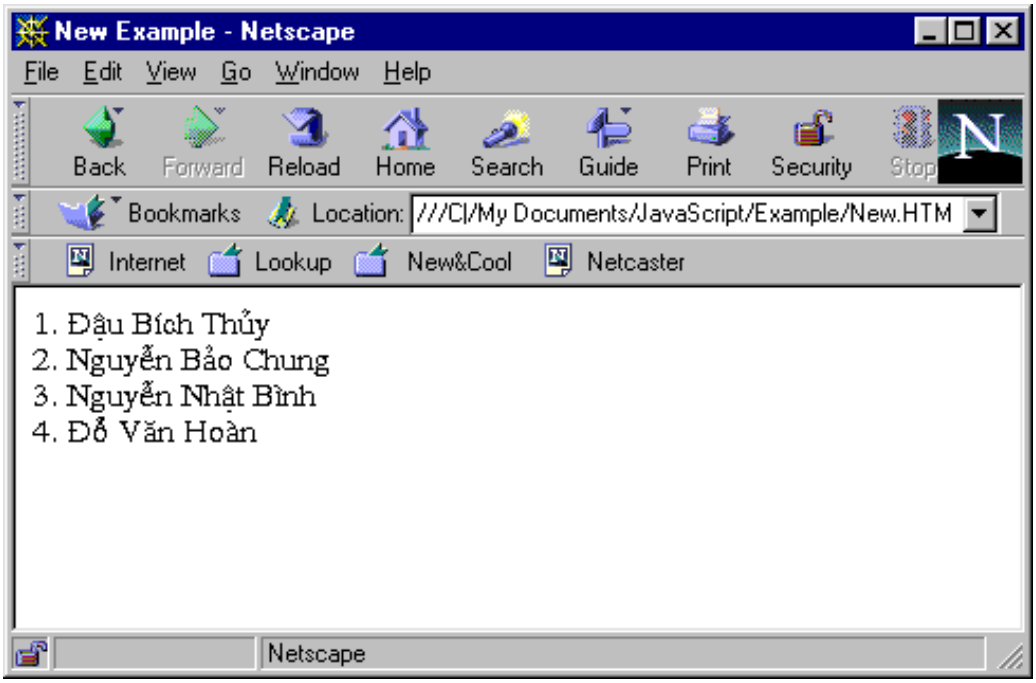
Cú pháp

objectvar = new object_type (param1 [,param2]... [,paramN])

Ví dụ sau tạo đối tượng **`person`** có các thuộc tính *firstname, lastname, age, sex*. Chú ý rằng từ khoá **`this`** được sử dụng để chỉ đối tượng trong hàm **`person`**. Sau đó ba thể hiện của đối tượng **`person`** được tạo ra bằng lệnh ***new***

```
<HTML>
<HEAD>
<TITLE>ả ew Example </TITLE>
<SCRIPT LAả GUAGE= "JavaScript">
function person(first_name, last_name, age, sex){
        this.first_name=first_name;
        this.last_name=last_name;
        this.age=age;
        this.sex=sex;
}
        person1= new person("Thuy", "Dau Bich", "23", "Female");
        person2= new person("Chung", "ả guyen Bao", "24", "Male");
```

```
        person3= new person("Binh", "ầ guyen ả hat", "24", "Male");
        person4= new person("Hoàn", "Đỗ Văn", "24", "Male");
        document.write ("1. "+person1.last_name+" " + person1.first_name + "<BR>" );
        document.write("2. "+person2.last_name +" "+ person2.first_name + "<BR>");
        document.write("3. "+ person3.last_name +" "+ person3.first_name + "<BR>");
        document.write("4. "+ person4.last_name +" "+ person4.first_name+"<BR>");
</SCRIPT>
</HEAD>
<BODY>
</BODY>
        </HTML>
```



3.10.3. this

Từ khoá ***this*** được sử dụng để chỉ đối tượng hiện thời. Đối tượng được gọi thường là đối tượng hiện thời trong phương thức hoặc trong hàm.

Cú pháp

this [.property]

Có thể xem ví dụ của lệnh new.

3.10.4. with

Lệnh này được sử dụng để thiết lập đối tượng ngầm định cho một nhóm các lệnh, bạn có thể sử dụng các thuộc tính mà không đề cập đến đối tượng.

Cú pháp

```
with (object)
{
        // statement
}
```

Ví dụ:

Ví dụ sau chỉ ra cách sử dụng lệnh with để thiết lập đối tượng ngầm định là ***document*** và có thể sử dụng phương thức ***write*** mà không cần đề cập đến đối tượng document

```
<HTML>
<HEAD>
<TITLE>With Example </TITLE>
<SCRIPT LAả GUAGE= "JavaScript">
with (document){
        write("This is an exemple of the things that can be done <BR>");
    write("With the <B>with<B> statment. <P>");
    write("This can really save some typing");
}
</SCRIPT>
</HEAD>
<BODY>
</BODY>
</HTML>
```

3.11. Khai báo dữ liệu kiểu mảng

Cú pháp

ten_bien=new Array()
hoặc: ten_bien=new Array(*size*)
hoặc: ten_bien=new Array(*element0*, *element1*, ..., *elementn*)

Phần	Diễn giải
Size	Cỡ của mảng. Các thành phần của mảng có chỉ số đầu tiên là 0 đến *size* -1.
element0,...,elementn	Các thành phần đặt trong mảng. Có *n* + 1 thành phần với thành phần là các giá trị gán trực tiếp cho mảng.

Ví dụ:

```
var my_array = new Array();
for (i = 0; i < 10; i++)
   {
   my_array[i] = i;
   }
x = my_array[4];
```

3.12. Xử lý sự kiện

JavaScript là ngôn ngữ định hướng sự kiện, nghĩa là sẽ phản ứng trước các sự kiện xác định trước như kích chuột hay tải một văn bản. Một sự kiện có thể gây ra việc thực hiện một đoạn mã lệnh (gọi là các chương trình xử lý sự kiện) giúp cho chương trình có thể phản ứng một cách thích hợp.

Chương trình xử lý sự kiện (Event handler): Một đoạn mã hay một hàm được thực hiện để phản ứng trước một sự kiện gọi là chương trình xử lý sự kiện. Chương trình xử lý sự kiện được xác định là một thuộc tính của một thẻ HTML:

<tagả ame eventHandler = "JavaScript Code or Function">

Ví dụ sau gọi hàm ***CheckAge()*** mỗi khi giá trị của trường văn bản thay đổi:

<Iả PUT TYPE=TEXT ả AME="AGE" onChange="CheckAge()">

Đoạn mã của chương trình xử lý sự kiện không là một hàm; nó là các lệnh của JavaScript cách nhau bằng dấu chấm phẩy. Tuy nhiên cho mục đích viết thành các module nên viết dưới dạng các hàm.

Một số chương trình xử lý sự kiện trong JavaScript:

onBlur	Xảy ra khi input focus bị xoá từ thành phần form
onClick	Xảy ra khi người dùng kích vào các thành phần hay liên kết của form.
onChange	Xảy ra khi giá trị của thành phần được chọn thay đổi
onFocus	Xảy ra khi thành phần của form được focus(làm nổi lên).
onLoad	Xảy ra trang Web được tải.
onMouseOver	Xảy ra khi di chuyển chuột qua kết nối hay anchor.
onSelect	Xảy ra khi người sử dụng lựa chọn một trường nhập dữ liệu trên form.
onSubmit	Xảy ra khi người dùng đưa ra một form.
onUnLoad	Xảy ra khi người dùng đóng một trang

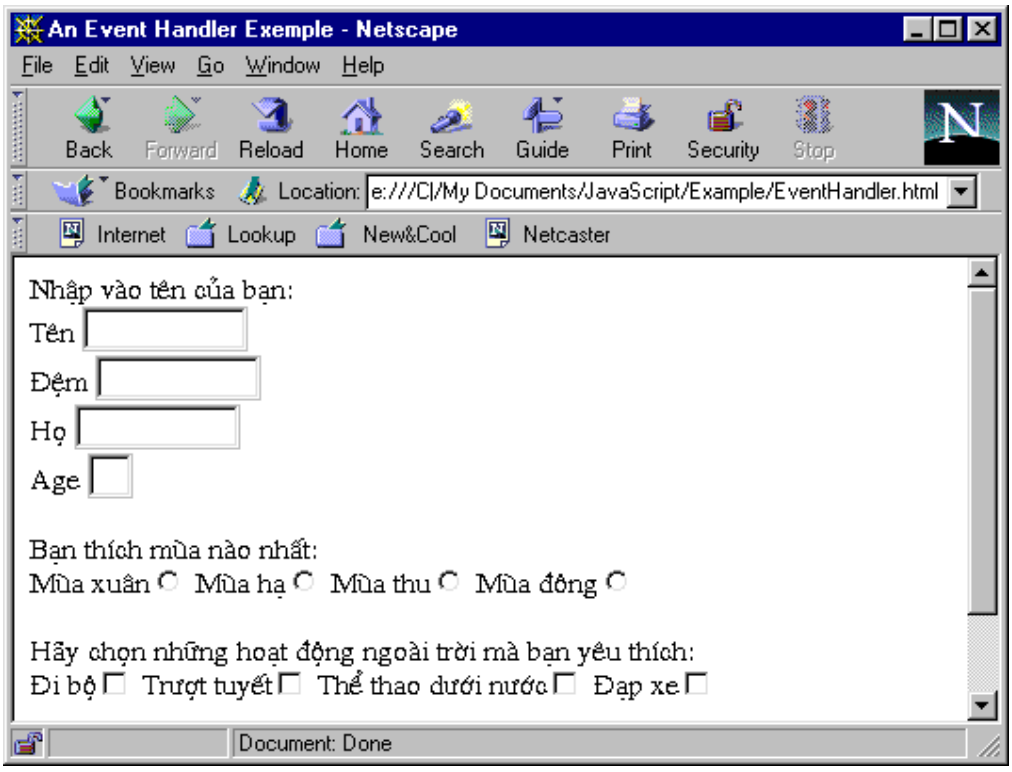
Sau đây là bảng các chương trình xử lý sự kiện có sẵn của một số đối tượng. Các đối tượng này sẽ được trình bày kỹ hơn trong phần sau.

Đối tượng	Chương trình xử lý sự kiện có sẵn
Selection list	onBlur, onChange, onFocus
Text	onBlur, onChange, onFocus, onSelect
Textarea	onBlur, onChange, onFocus, onSelect
Button	onClick
Checkbox	onClick
Radio button	onClick
Hypertext link	onClick, onMouseOver, onMouseOut
Clickable Imagemap area	onMouseOver, onMouseOut
Reset button	onClick
Submit button	onClick
Document	onLoad, onUnload, onError
Window	onLoad, onUnload, onBlur, onFocus
Framesets	onBlur, onFocus
Form	onSubmit, onReset
Image	onLoad, onError, onAbort

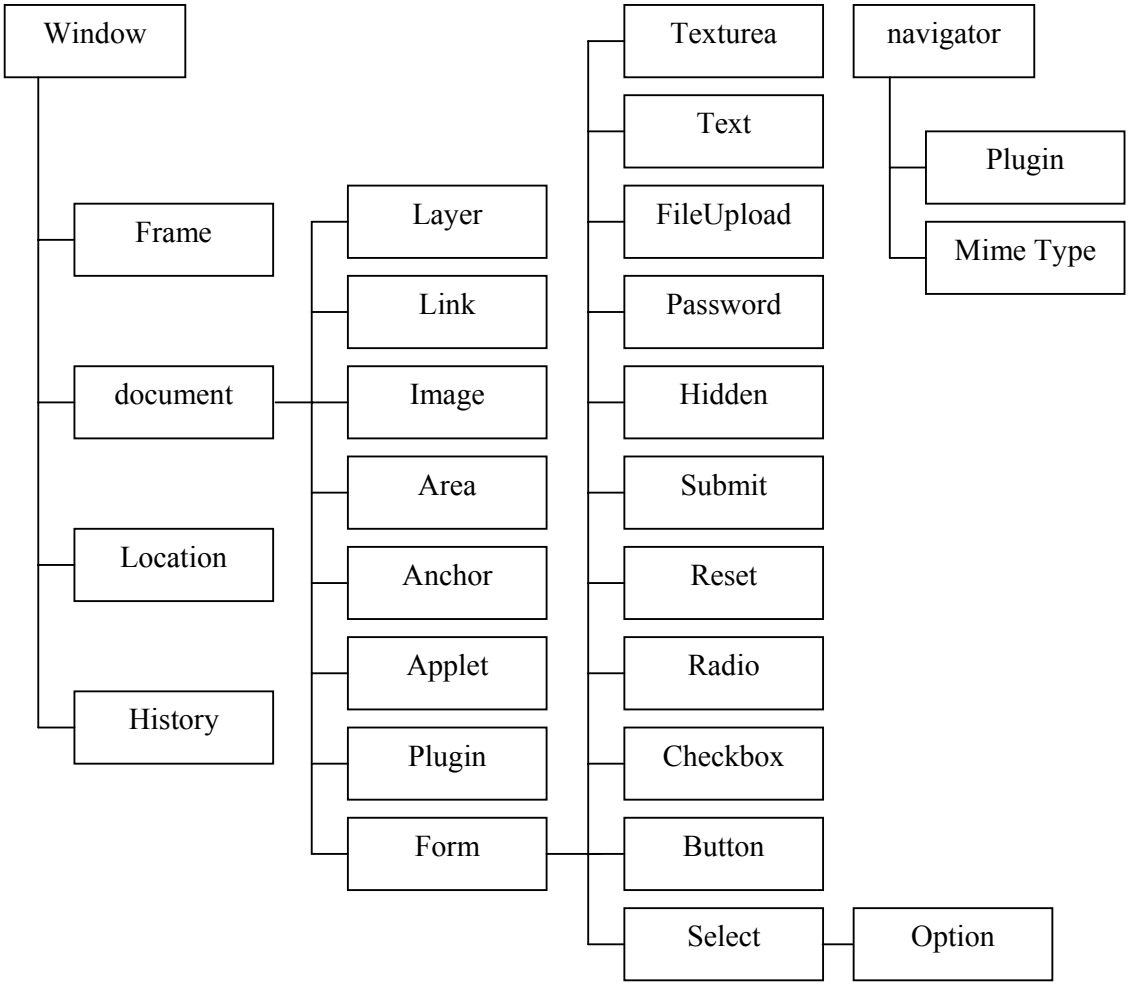
Ví dụ sau là một đoạn mã script đơn giản của chương trình xử lý sự kiện thẩm định giá trị đưa vào trong trường text. Tuổi của người sử dụng được nhập vào trong trường này và chương trình xử lý sự kiện sẽ thẩm định tính hợp lệ của dữ liệu đưa vào. ả ếu không hợp lệ sẽ xuất hiện một thông báo yêu cầu nhập lại. Chương trình xử lý sự kiện được gọi mỗi khi trường AGE bị thay đổi và focus chuyển sang trường khác. Hình 5.10 minh hoạ kết quả của ví dụ này

```
<HTML>
<HEAD>
<TITLE> An Event Handler Exemple </TITLE>
<SCRIPT LAả GUAGE= "JavaScript">
function CheckAge(form) {
if ( (form.AGE.value<0)||(form.AGE.value>120) )
        {
                alert("Tuổi nhập vào không hợp lệ! Mời bạn nhập  lại");
        form.AGE.value=0;
        }
}
</SCRIPT>
</HEAD>
<BODY>
<FORM ả AME="PHIEU_DIEU_TRA">

ả hập vào tên của bạn:<BR>
Tên <Iả PUT TYPE=TEXT ả AME="TEả " MAXLEả GTH=10 SIZE=10><BR>
Đệm <Iả PUT TYPE=TEXT ả AME="DEM" MAXLEả GTH=15 SIZE=10><BR>
Họ  <Iả PUT TYPE=TEXT ả AME="HO" MAXLEả GTH=10 SIZE=10><BR>
        Age     <Iả PUT     TYPE=TEXT     ả AME="AGE"     MAXLEả GTH=3     SIZE=2
                onChange="CheckAge(PHIEU_DIEU_TRA)"><BR>
<P>

Bạn thích mùa nào nhất:<BR>
Mùa xuân<Iả PUT TYPE=RADIO ả AME="MUA" VALUE="Mua xuan">
Mùa hạ<Iả PUT TYPE=RADIO ả AME="MUA" VALUE="Mua ha">
Mùa thu<Iả PUT TYPE=RADIO ả AME="MUA" VALUE="Mua thu">
Mùa đông<Iả PUT TYPE=RADIO ả AME="MUA" VALUE="Mua dong">
<P>

Hãy chọn những hoạt động ngoài trời mà bạn yêu thích:<BR>
Đi bộ<Iả PUT TYPE=CHECKBOX ả AME="HOAT_DOả G" VALUE="Di bo">
        Trượt tuyết<Iả PUT TYPE=CHECKBOX ả AME="HOAT_DOả G" VALUE="Truot tuyet">
        Thể thao dưới nước<Iả PUT TYPE=CHECKBOX ả AME="HOAT_DOả G" VALUE="Duoi
                nuoc">
```

Đạp xe<Iả PUT TYPE=CHECKBOX ả AME="HOAT_DOả G" VALUE="Dap xe">
<P>
<Iả PUT TYPE=SUBMIT>
<Iả PUT TYPE=RESET>
</FORM>
</BODY>
</HTML>



3.13. Các đối tượng trong Javascript



Sơ đồ phân cấp đối tượng ả avigator

Trong sơ đồ phân cấp này, các đối tượng con chính là các thuộc tính của một đối tượng cha. Ví dụ như một form tên là form1 chính là một đối tượng con của đối tượng document và được gọi tới là document.form1

Tất cả các trang đều có các đối tượng sau đây:

- navigator: có các thuộc tính tên và phiên bản của ả avigator đang được sử dụng, dùng cho *MIME type* được hỗ trợ bởi client và plug-in được cài đặt trên client.
- window: là đối tượng ở mức cao nhất, có các thuộc tính thực hiện áp dụng vào toàn bộ cửa sổ.
- document: chứa các thuộc tính dựa trên nội dung của document như tên, màu nền, các kết nối và các forms.
- location: có các thuộc tính dựa trên địa chỉ URL hiện thời
- history: Chứa các thuộc tính về các URL mà client yêu cầu trước đó.

Sau đây sẽ mô tả các thuộc tính, phương thức cũng như các chương trình xử lý sự kiện cho từng đối tượng trong JavaScript.

3.13.1. Đối tượng navigator

Đối tượng này được sử dụng để đạt được các thông tin về trình duyệt như số phiên bản. Đối tượng này không có phương thức hay chương trình xử lý sự kiện.

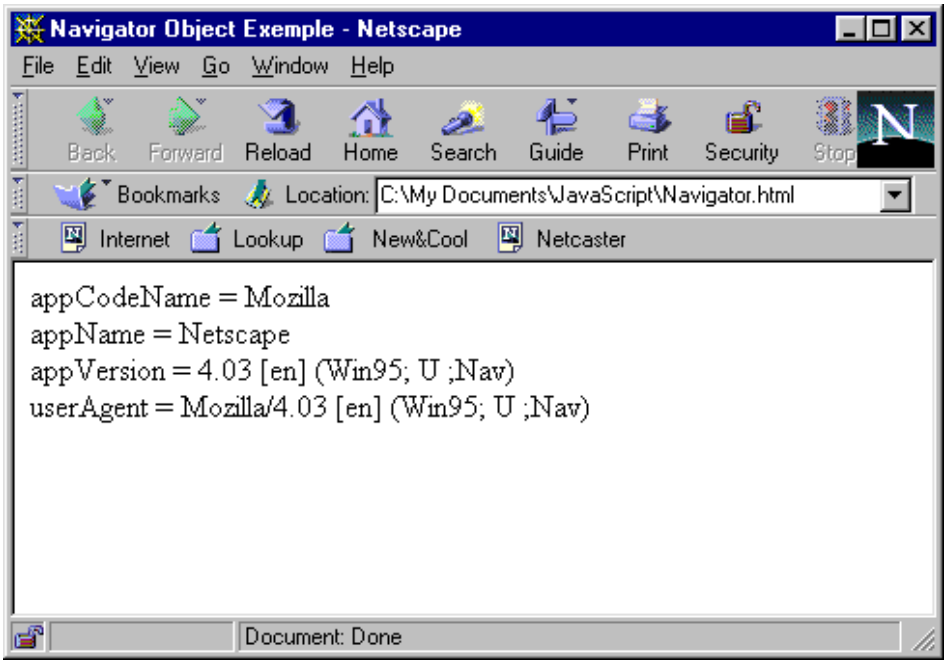
Các thuộc tính

appCodeName	Xác định tên mã nội tại của trình duyệt (Atlas).
AppName	Xác định tên trình duyệt.
AppVersion	Xác định thông tin về phiên bản của đối tượng navigator.
userAgent	Xác định header của user - agent.

Ví dụ

Ví dụ sau sẽ hiển thị các thuộc tính của đối tượng navigator

```
<HTML>
<HEAD>
<TITLE> ả avigator Object Exemple </TITLE>
<SCRIPT LAả GUAGE= "JavaScript">
        document.write("appCodeả ame = "+navigator.appCodeả ame + "<BR>");
        document.write("appả ame = "+navigator.appả ame + "<BR>");
        document.write("appVersion = "+navigator.appVersion + "<BR>");
        document.write("userAgent = "+navigator.userAgent + "<BR>");
</SCRIPT>
</HEAD>
<BODY>
</BODY>
</HTML>
```



3.13.2. Đối tượng window

Đối tượng window như đã nói ở trên là đối tượng ở mức cao nhất. Các đối tượng document, frame, vị trớ đều là thuộc tính của đối tượng window.

CÁC THUỘC TÍả H

- defaultStatus - Thông báo ngầm định hiển thị lên trên thanh trạng thái của cửa sổ
- Frames - Mảng xác định tất cả các frame trong cửa sổ.
- Length - Số lượng các frame trong cửa sổ cha mẹ.
- ả ame - Tên của cửa sổ hiện thời.
- Parent - Đối tượng cửa sổ cha mẹ
- Self - Cửa sổ hiện thời.
- Status - Được sử dụng cho thông báo tạm thời hiển thị lên trên thanh thạng thái cửa sổ. Được sử dụng để lấy hay đặt lại thông báo trạng thái và ghi đè lên defaultStatus.
- Top - Cửa sổ ở trên cùng.
- Window - Cửa sổ hiện thời.

CÁC PHƯƠả G THỨC

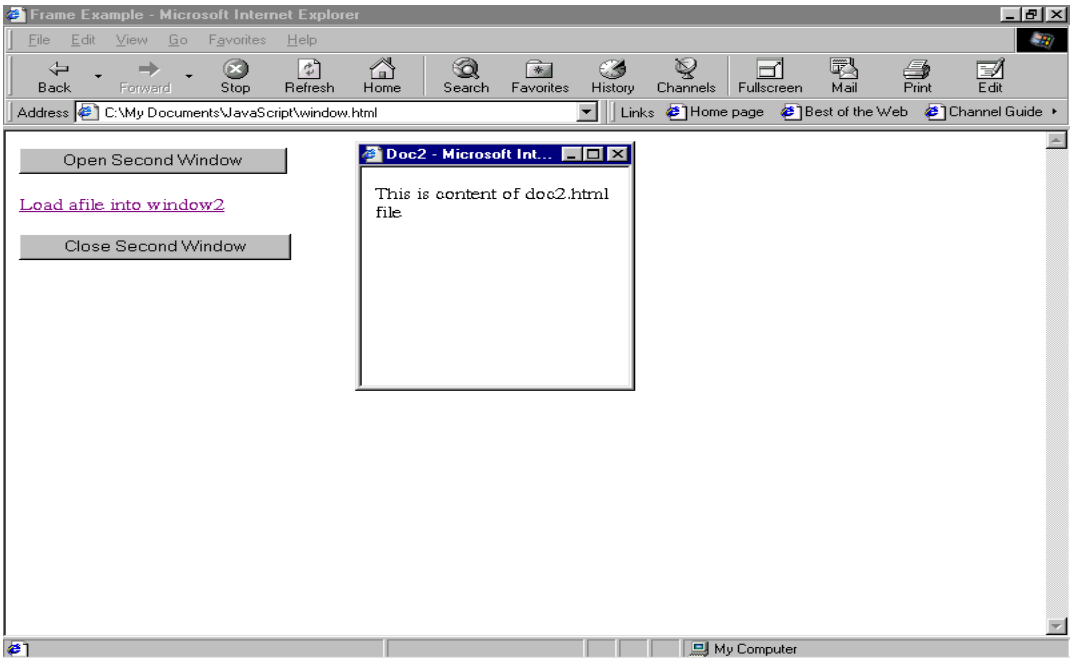
- alert ("message") -Hiển thị hộp hội thoại với chuỗi "message" và nỳt OK.
- clearTimeout(timeoutID) -Xóa timeout do SetTimeout đặt. SetTimeout trả lại timeoutID
- windowReference.close -Đóng cửa sổ windowReference.
- confirm("message") -Hiển thị hộp hội thoại với chuỗi "message", nút OK và nút Cancel. Trả lại giá trị True cho OK và False cho Cancel.
- [windowVar =][window]. open("URL", "windowả ame", ["windowFeatures"]) - Mở cửa sổ mới.

- prompt ("message" [,"defaultInput"]) - Mở một hộp hội thoại để nhận dữ liệu vào trường text.
- TimeoutID = setTimeout(expression,msec) - Đánh giá biểu thức expresion sau thời gian msec.

Ví dụ: Sử dụng tên cửa sổ khi gọi tới nó như là đích của một form submit hoặc trong một Hypertext link (thuộc tính TARGET của thẻ FORM và A).

Trong ví dụ tạo ra một cửa sổ thứ hai, như nút thứ nhất để mở một cửa sổ rỗng, sau đó một liên kết sẽ tải file doc2.html xuống cửa sổ mới đó rồi một nút khác dựng để đóng của sổ thứ hai lại, ví dụ này lưu vào file *window.html*:

```
<HTML>
<HEAD>
<TITLE>Frame Example </TITLE>
</HEAD>
<BODY>
<FORM>
        <Iả PUT TYPE="button" VALUE="Open Second Window"
                onClick="msgWindow=window.open('','window2','resizable=no,width=200,height=20
                0')">
<P>
<A HREF="doc2.html" TARGET="window2">
Load a file into window2 </A>
</P>
<Iả PUT TYPE="button" VALUE="Close Second Window"
        onClick="msgWindow.close()">
</FORM>
</BODY>
</HTML>
```



CÁC CHƯƠNG TRÌNH XỬ LÝ SỰ KIỆN

- onLoad - Xuất hiện khi cửa sổ kết thúc việc tải.
- onUnLoad - Xuất hiện khi cửa sổ được loại bỏ.

3.13.3. Đối tượng location

Các thuộc tính của đối tượng location duy trì các thông tin về URL của document hiện thời. Đối tượng này hoàn toàn không có các phương thức và chương trình xử lý sự kiện đi kèm. *Ví dụ*:

http:// www.abc.com/ chap1/page2.html#topic3

Các thuộc tính

- hash - Tên anchor của vị trí hiện thời (ví dụ topic3).
- Host - Phần hostname:port của URL (ví dụ www.abc.com). Chú ý rằng đây thường là cổng ngầm định và ít khi được chỉ ra.
- Hostname - Tên của host và domain (ví dụ www.abc.com).
- href - Toàn bộ URL cho document hiện tại.
- Pathname - Phần đường dẫn của URL (ví dụ /chap1/page2.html).
- Port - Cổng truyền thông được sử dụng cho máy tính host, thường là cổng ngầm định.
- Protocol - Giao thức được sử dụng (cùng với dấu hai chấm) (ví dụ http:).
- Search - Câu truy vấn tìm kiếm có thể ở cuối URL cho các script CGI.

3.13.4. Đối tượng frame

Một cửa số có thể có một vài frame. Các frame có thể cuộn một cách độc lập với nhau và mỗi frame có URL riêng. frame không có các chương trình xử lý sự kiện. Sự kiện onLoad và onUnLoad là của đối tượng window.

CÁC THUỘC TÍNH

- frames - Mảng tất cả các frame trong cửa sổ.
- NAME - Thuộc tính NAME của thẻ <FRAME>
- Length - Số lượng các frame con trong một frame.
- Parent - Cửa sổ hay frame chứa nhóm frame hiện thời.
- self - frame hiện thời.
- Window - frame hiện thời.

CÁC PHƯƠNG THỨC

- clearTimeout (timeoutID) - Xoá timeout do setTimeout lập. SetTimeout trả lại timeoutID.
- TimeoutID = setTimeout (expression,msec) - Đánh giá expression sau khi hết thời gian msec.

SỬ DỤNG FRAME

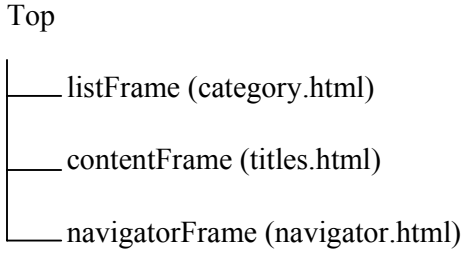
3.13.4.1. Tạo một frame (create)

Để tạo một frame, ta sử dụng thẻ **FRAMESET**. Mục đích của thẻ này là định nghĩa một tập các frame trong một trang.

Ví dụ1*:* tạo frame (hình 17)

```
<HTML>
<HEAD>
<TITLE>Frame Example </TITLE>
<FRAMESET ROWS="90%,10%">
        <FRAMESET COLS="30%,70%">
                <FRAME SRC=CATEGORY.HTM å AME="ListFrame">
                <FRAME SRC=TITLES.HTM å AME="contentFrame">
        </FRAMESET >
        <FRAME SRC=å AVIGATOR.HTM å AME="navigateFrame">
</FRAMESET >
</HEAD>
<BODY> </BODY>
</HTML>
```

Sơ đồ sau hiển thị cấu trúc của các frame: Cả 3 frame đều trên cùng một cửa sổ cha, mặc dù 2 trong số các frame đó nằm trong một frameset khác.

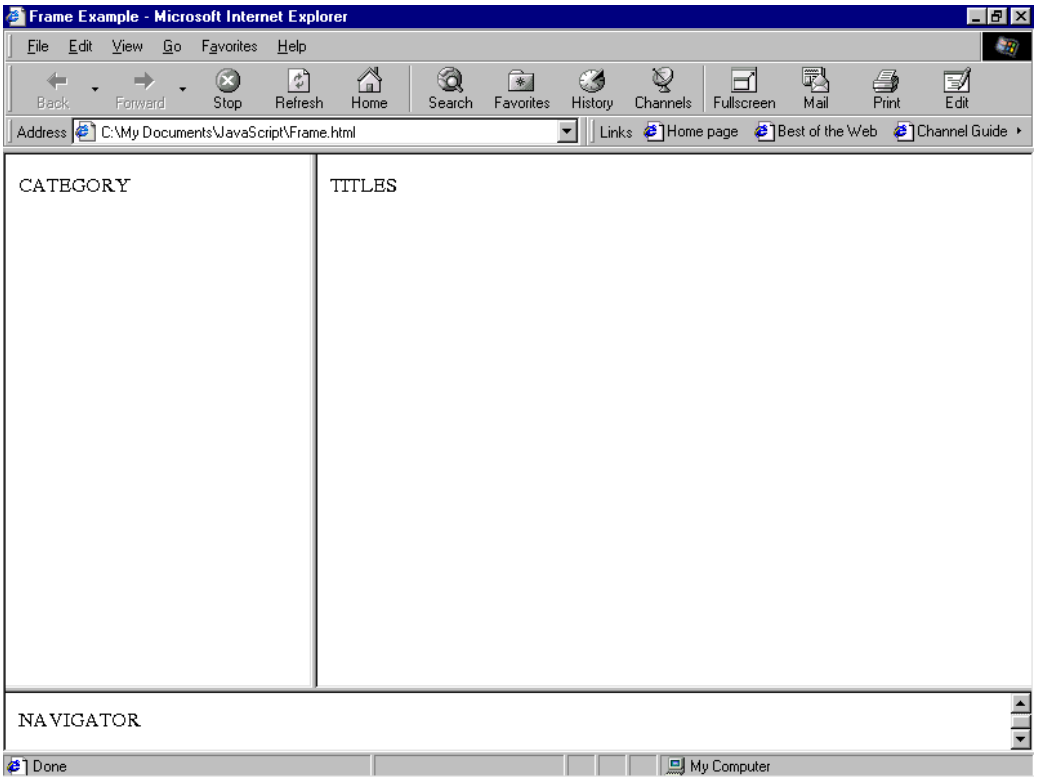


Bạn có thể gọi tới những frame trước đó bằng cách sử dụng thuộc tính **frames** như sau:

listFrame chính là top.frames[0]

contentFrame chính là top.frames[1]

navigatorFrame chính là top.frames[2]



***Ví dụ 2*:** Cũng giống như một sự lựa chọn, bạn có thể tạo ra một cửa sổ giống như ví dụ trước nhưng trong mỗi đỉnh của hai frame lại có một cửa sổ cha riêng từ **navigateFrame**. Mức frameset cao nhất có thể được định nghĩa như sau:

```
<HTML>
<HEAD>
<TITLE>Frame Example </TITLE>
<FRAMESET ROWS="90%,10%">
<FRAME SRC=muske13.HTML å AME="upperFrame">
<FRAME SRC=å AVIGATOR.HTM å AME="navigateFrame">
</FRAMESET >
</HEAD>
<BODY>
</BODY>
</HTML>
```

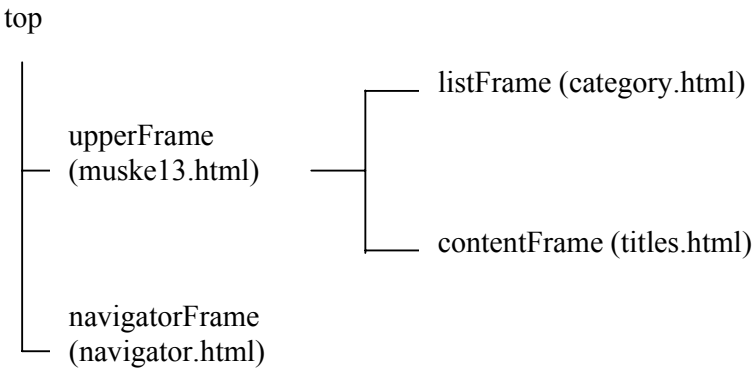
Trong file muske13.html lại tiếp tục đặt một frameset:

```
<HTML>
<HEAD>
<TITLE>Frame Example </TITLE>
<FRAMESET COLS="30%,70%">
<FRAME SRC=CATEGORY.HTM å AME="ListFrame">
<FRAME SRC=TITLES.HTM å AME="contentFrame">
</FRAMESET >
</HEAD>
```

<BODY>
</BODY>
</HTML>

Khi đó kết quả hiển thị của ví dụ 2 giống ví dụ 1 nhưng sự phân cấp của các frames lại khác hẳn:

Bạn có thể gọi tới các frame trên bằng cách sử dụng thuộc tính mảng **frames** như sau:



upperFrame chính là **top.frames[0]**
navigatorFrame chính là **top.frames[1]**
listFrame chính là **upperFrame.frames[0]**
hoặc **top.frames[0].frames[0]**
contentFrame chính là **upperFrame.frames[1]**
hoặc **top.frames[0].frames[1]**

3.13.4.2. Cập nhật một frame (update)

Bạn có thể cập nhật nội dung của một frame bằng cách sử dụng thuộc tính **location** để đặt địa chỉ URL và phải định chỉ rõ vị trí của frame trong cấu trúc.

Trong ví dụ trên, nếu bạn thêm một dòng sau vào **navigatorFrame**:

<lả PUT TYPE="button" VALUE="Titles only" onClick="top.frames[0].location='artist.html'">

thì khi nút "**Titles only**" được nhấn, file **artist.html** sẽ được tải vào **upperFrame**, và hai frame **listFrame, contentFrame** sẽ bị đóng lại như chúng chưa bao giờ tồn tại.

3.13.5. Đối tượng DOCUMENT

Đối tượng này chứa các thông tin về document hiện thời và cung cấp các phương thức để đưa thông tin ra màn hình. Đối tượng document được tạo ra bằng cặp thẻ <BODY> và </BODY>. Một số các thuộc tính gắn với thẻ <BODY>.

Các đối tượng anchor, forms, history, links là thuộc tính của đối tượng document. Không có các chương trình xử lý sự kiện cho các frame. Sự kiện onLoad và onUnLoad là cho đối tượng window.

CÁC THUỘC TÍả H

- alinkColor - Giống như thuộc tính ALIả K.
- anchor - Mảng tất cả các anchor trong document.
- bgColor - Giống thuộc tính BGCOLOR.
- cookie - Sử dụng để xác định cookie.

- fgColor - Giống thuộc tính TEXT.
- forms - Mảng tất cả các form trong document.
- lastModified - ả gày cuối cùng văn bản được sửa.
- linkColor - Giống thuộc tính LIả K.
- links - Mảng tất cả các link trong document.
- location - URL đầy đủ của văn bản.
- referrer - URL của văn bản gọi nó.
- title - ả ội dung của thẻ <TITLE>.
- vlinkColor - Giống thuộc tính VLIả K.

CÁC PHƯƠả G THỨC

- document.clear - Xoá document hiện thời.
- document.close - Đóng dòng dữ liệu vào và đưa toàn bộ dữ liệu trong bộ đệm ra màn hình.
- document.open (["mineType"]) - Mở một stream để thu thập dữ liệu vào của các phwong thức write và writeln.
- document.write(expression1 [,expression2]...[,expressionả]) - Viết biểu thức HTML lên văn bản trông một cửa sổ xác định.
- document.writeln (expression1 [,expression2] ... [,expressionả]) - Giống phương thức trên nhưng khi hết mỗi biểu thức lại xuống dòng.

3.13.6. Đối tượng anchors

anchor là một đoạn văn bản trong document có thể dùng làm đích cho một siêu liên kết. ả ó được xác định bằng cặp thẻ <A> và </A>. Đối tượng anchor không có thuộc tính, phương thức cũng như chương trình xử lý sự kiện. Mảng anchor tham chiếu đến mỗi anchor có tên trong document. Mỗi anchor được tham chiếu bằng cách:

document.anchors [index]

Mảng anchor có một thuộc tính duy nhất là length xác định số lượng các anchor trong document, nó có thể được xác định như sau:

document.anchors.length.

3.13.7. Đối tượng FORMS

Các form được tạo ra nhờ cặp thẻ <FORM> và </FORM>. Phần lớn các thuộc tính của đối tượng form phản ánh các thuộc tính của thẻ <FORM>. Có một vài phần tử (elements) là thuộc tính của đối tượng forms:

button
checkbox
hidden
password
radio
reset
select
submit
text

textarea

Các phần tử này sẽ được trình bày sau.

ả ếu document chứa một vài form, chúng có thể được tham chiếu qua mảng forms. Số lượng các form có thể được xác định như sau:

document.forms.length.

Mỗi một form có thể được tham chiếu như sau:

document.forms[index]

CÁC THUỘC TÍả H

- action thuộc tính ACTIOả của thẻ FORM.
- elements Mảng chứa tất cả các thành phần trong một form (như checkbox, trường text, danh sách lựa chọn
- encoding Xâu chứa kiểu MIME được sử dụng để mã hoá nội dung của form gửi cho server.
- length Số lượng các thành phần trong một form.
- method Thuộc tính METHOD.
- target Xâu chứa tên của cửa sổ đích khi submit form

CÁC PHƯƠả G THỨC

- formả ame.submit () - Xuất dữ liệu của một form tên formả ame tới trang xử lý. Phương thức này mô phỏng một click vào nút submit trên form.

CÁC CHƯƠả G TRÌả H XỬ LÝ SỰ KIỆả

- onSubmit - Chương trình xử lý sự kiện này được gọi khi người sử dụng chuyển dữ liệu từ form đi.

3.13.8. Đối tượng HISTORY

Đối tượng này được sử dụng để lưu giữ các thông tin về các URL trước được người sử dụng sử dụng. Danh sách các URL được lưu trữ theo thứ tự thời gian. Đối tượng này không có chương trình xử lý sự kiện.

CÁC THUỘC TÍả H

- length - Số lượng các URL trong đối tượng.

CÁC PHƯƠả G THỨC

- history.back() - Được sử dụng để tham chiếu tới URL mới được thăm trước đây.
- history.forward() - Được sử dụng để tham chiếu tới URL kế tiếp trong danh sách. ả ó sẽ không gây hiệu ứng gì nếu đã đến cuối của danh sách.
- history.go (delta | "location") - Được sử dụng để chuyển lên hay chuyển xuống delta bậc hay di chuển đến URL xác định bởi location trong danh sách. ả ếu delta được sử dụng thì việc dịch chuyển lên phía trên khi delta dương và xuống phía dưới khi delta âm. nếu sử dụng location, URL gần nhất có chứa location là chuỗi con sẽ được tham chiếu.

3.13.9. Đối tượng LINKS

Đối tượng link là một đoạn văn bản hay một ảnh được xem là một siêu liên kết. Các thuộc tính của đối tượng link chủ yếu xử lý về URL của các siêu liên kết. Đối tượng link cũng không có phương thức nào.

Mảng link chứa danh sách tất cả các liên kết trong document. Có thể xác định số lượng các link qua

document.links.length()

Có thể tham chiếu tới một liên kết qủa

document.links [index]

Để xác định các thuộc tính của đối tượng link, có thể sử dụng URL tương tự:

http://www.abc.com/chap1/page2.html#topic3

CÁC THUỘC TÍẩ H

- hash - Tên anchor của vị trí hiện thời (ví dụ topic3).
- Host - Phần hostname:port của URL (ví dụ www.abc.com). Chú ý rằng đây thường là cổng ngầm định và ít khi được chỉ ra.
- Hostname - Tên của host và domain (ví dụ ww.abc.com).
- href - Toàn bộ URL cho document hiện tại.
- Pathname - Phần đường dẫn của URL (ví dụ /chap1/page2.html).
- port - Cổng truyền thông được sử dụng cho máy tính host, thường là cổng ngầm định.
- Protocol - Giao thức được sử dụng (cùng với dấu hai chấm) (ví dụ http:).
- Search - Câu truy vấn tìm kiếm có thể ở cuối URL cho các script CGI.
- Target - Giống thuộc tính TARGET của <LIẩ K>.

CÁC CHƯƠẩ G TRÌẩ H XỬ LÝ SỰ KIỆẩ

- onClick - Xảy ra khi người sử dụng nhấn vào link.
- onMouseOver - Xảy ra khi con chuột di chuyển qua link.

3.13.10. Đối tượng MATH

Đối tượng Math là đối tượng nội tại trong JavaScript. Các thuộc tính của đối tượng này chứa nhiều hằng số toán học, các hàm toán học, lượng giác phổ biến. Đối tượng Math không có chương trình xử lý sự kiện.

Việc tham chiếu tới ***number*** trong các phương thức có thể là số hay các biểu thức được đnáh giá là số hợp lệ.

CÁC THUỘC TÍẩ H

- E - Hằng số Euler, khoảng 2,718.
- Lẩ 2 - logarit tự nhiên của 2, khoảng 0,693.
- Lẩ 10 - logarit tự nhiên của 10, khoảng 2,302.
- LOG2E - logarit cơ số 2 của e, khoảng 1,442.
- PI - Giá trị của π, khoảng 3,14159.
- SQRT1_2 - Căn bậc 2 của 0,5, khoảng 0,707.
- SQRT2 - Căn bậc 2 của 2, khoảng 1,414.

CÁC PHƯƠẩ G THỨC

- Math.abs (number) - Trả lại giá trị tuyệt đối của number.
- Math.acos (number) - Trả lại giá trị arc cosine (theo radian) của number. Giá trị của number phải năng giữa -1 và 1.
- Math.asin (number) - Trả lại giá trị arc sine (theo radian) của number. Giá trị của number phải năng giữa -1 và 1.
- Math.atan (number) - Trả lại giá trị arc tan (theo radian) của number.
- Math.ceil (number) - Trả lại số nguyên nhỏ nhất lớn hơn hoặc bằng number.
- Math.cos (number) - Trả lại giá trị cosine của number.
- Math.exp (number) - Trả lại giá trị e^ number, với e là hằng số Euler.
- Math.floor (number) - Trả lại số nguyên lớn nhất nhỏ hơn hoặc bằng number.
- Math.log (number) - Trả lại logarit tự nhiên của number.
- Math.max (num1,num2) - Trả lại giá trị lớn nhất giữa num1 và num2
- Math.min (num1,num2) - Trả lại giá trị nhỏ nhất giữa num1 và num2.
- Math.pos (base,exponent) - Trả lại giá trị base luỹ thừa exponent.
- Math.random (r) - Trả lại một số ngẫu nhiên giữa 0 và 1. Phwong thức này chỉ thực hiện được trên nền tảng Uả IX.
- Math.round (number) - Trả lại giá trị của number làm tròn tới số nguyên gần nhất.
- Math.sin (number) - Trả lại sin của number.
- Math.sqrt (number) - Trả lại căn bậc 2 của number.
- Math.tan (number) - Trả lại tag của number.

3.13.11. Đối tượng DATE

Đối tượng Date là đối tượng có sẵn trong JavaScript. ả ó cung cấp nhiều phương thức có ích để xử lý về thời gian và ngày tháng. Đối tượng Date không có thuộc tính và chương trình xử lý sự kiện.

Phần lớn các phương thức date đều có một đối tượng Date đi cùng. Các phương thức giới thiệu trong phần này sử dụng đối tượng Date dateVar, ví dụ:

```
dateVar = new Date ('August 16, 1996 20:45:04');
```

CÁC PHƯƠả G THỨC

- dateVar.getDate() - Trả lại ngày trong tháng (1-31) cho dateVar.
- dateVar.getDay() - Trả lại ngày trong tuần (0=chủ nhật,...6=thứ bảy) cho dateVar.
- dateVar.getHours() - Trả lại giờ (0-23) cho dateVar.
- dateVar.getMinutes() - Trả lại phút (0-59) cho dateVar.
- dateVar.getSeconds() - Trả lại giây (0-59) cho dateVar.
- dateVar.getTime() - Trả lại số lượng các mili giây từ 00:00:00 ngày 1/1/1970.
- dateVar.getTimeZoneOffset() - Trả lại độ dịch chuyểnbằng phút của giờ địa phương hiện tại so với giờ quốc tế GMT.
- dateVar.getYear()-Trả lại năm cho dateVar.
- Date.parse (dateStr) - Phân tích chuỗi dateStr và trả lại số lượng các mili giây tính từ 00:00:00 ngày 01/01/1970.
- dateVar.setDay(day) - Đặt ngày trong tháng là day cho dateVar.

- dateVar.setHours(hours) - Đặt giờ là hours cho dateVar.
- dateVar.setMinutes(minutes) - Đặt phút là minutes cho dateVar.
- dateVar.setMonths(months) - Đặt tháng là months cho dateVar.
- dateVar.setSeconds(seconds) - Đặt giây là seconds cho dateVar.
- dateVar.setTime(value) - Đặt thời gian là value, trong đó value biểu diễn số lượng mili giây từ 00:00:00 ngày 01/01/10970.
- dateVar.setYear(years) - Đặt năm là years cho dateVar.
- dateVar.toGMTString() - Trả lại chuỗi biểu diễn dateVar dưới dạng GMT.
- dateVar.toLocaleString()-Trả lại chuỗi biểu diễn dateVar theo khu vực thời gian hiện thời.
- Date.UTC (year, month, day [,hours] [,minutes] [,seconds]) - Trả lại số lượng mili giây từ 00:00:00 01/01/1970 GMT.

3.13.12. Đối tượng STRING

Đối tượng String là đối tượng được xây dựng nội tại trong JavaScript cung cấp nhiều phương thức thao tác trên chuỗi. Đối tượng này có thuộc tính duy nhất là độ dài (length) và không có chương trình xử lý sự kiện.

CÁC PHƯƠẳ G THỨC

- str.anchor (name) - Được sử dụng để tạo ra thẻ <A> (một cách động). Tham số name là thuộc tính ẳ AME của thẻ <A>.
- str.big() - Kết quả giống như thẻ <BIG> trên chuỗi str.
- str.blink() - Kết quả giống như thẻ <BLIẳ K> trên chuỗi str.
- str.bold() - Kết quả giống như thẻ <BOLD> trên chuỗi str.
- str.charAt(a) - Trả lại ký tự thứ a trong chuỗi str.
- str.fixed() - Kết quả giống như thẻ <TT> trên chuỗi str.
- str.fontcolor() - Kết quả giống như thẻ <FOẳ TCOLOR = color>.
- str.fontsize(size) - Kết quả giống như thẻ <FOẳ TSIZE = size>.
- str.index0f(srchStr [,index]) - Trả lại vị trí trong chuỗi str vị trí xuất hiện đầu tiên của chuỗi srchStr. Chuỗi str được tìm từ trái sang phải. Tham số index có thể được sử dụng để xác định vị trí bắt đầu tìm kiếm trong chuỗi.
- str.italics() - Kết quả giống như thẻ <I> trên chuỗi str.
- str.lastIndex0f(srchStr [,index]) - Trả lại vị trí trong chuỗi str vị trí xuất hiện cuối cùng của chuỗi srchStr. Chuỗi str được tìm từ phải sang trái. Tham số index có thể được sử dụng để xác định vị trí bắt đầu tìm kiếm trong chuỗi.
- str.link(href) - Được sử dụng để tạo ra một kết nối HTML động cho chhuỗi str. Tham số href là URL đích của liên kết.
- str.small() - Kết quả giống như thẻ <SMALL> trên chuỗi str.
- str.strike() - Kết quả giống như thẻ <STRIKE> trên chuỗi str.
- str.sub() - Tạo ra một subscript cho chuỗi str, giống thẻ <SUB>.
- str.substring(a,b) - Trả lại chuỗi con của str là các ký tự từ vị trí thứ a tới vị trí thứ b. Các ký tự được đếm từ trái sang phải bắt đầu từ 0.

- str.sup() - Tạo ra superscript cho chuỗi str, giống thẻ <SUP>.
- str.toLowerCase() - Đổi chuỗi str thành chữ thường.
- str.toUpperCase() - Đổi chuỗi str thành chữ hoa.

3.13.13. Xem lại các lệnh và mở rộng

Lệnh/ Mở rộng	Kiểu	Mô tả
blur()	cách thức JavaScript	Mô tả việc dịch chuyển con trỏ từ một phần tử
form.action	cách thức JavaScript	Xâu chứa giá trị của thuộc tính ACTIOẩ trong thẻ FORM
form.elemrnts	thuộc tính JavaScript	mảng chứa danh sách các phần tử trong form (như checkbox, trường text, danh sách lựa chọn)
form.encoding	thuộc tính JavaScript	xâu chứa kiểu MIME sử dụng khi chuyển thông tin từ form tới server
form,name	thuộc tính JavaScript	Xâu chứa giá trị thuộc tính ẩ AME trong thẻ FORM
form.target	thuộc tính JavaScript	Xâu chứa tên cửa sổ đích bởi một form submition
form.submit	cách thức JavaScript	Mô tả việc submit một form HTML
type	thuộc tính JavaScript	ánh xạ kiểu của một phần tử form thành một xâu.
onSubmit	Thẻ sự kiện	Thẻ sự kiện cho việc submit
button	thuộc tính HTML	Thuộc tính kiểu cho các nút bấm của HTML (<Iẩ PUT TYPE=button>)
checkbox	thuộc tính HTML	Thuộc tính kiểu cho các checkbox của HTML (<Iẩ PUT TYPE=checkbox>)
pasword	thuộc tính HTML	Thuộc tính kiểu cho các dòng pasword của HTML(<Iẩ PUT TYPE=password>)
radio	thuộc tính HTML	Thuộc tính kiểu cho các nút radio của HTML (<Iẩ PUT TYPE=radio>)
reset	thuộc tính HTML	Thuộc tính kiểu cho các nút reset của HTML (<Iẩ PUT TYPE=reset>)
SELECT	thẻ HTML	Hộp thẻ cho danh sách lựa chọn
OPTIOẩ	thẻ HTML	chỉ ra các lựa chọn trong danh sách lựa chọn(<SELECT><OPTIOẩ >Option 1<OPTIOẩ >Option 2</SELECT>)
submit	thuộc tính HTML	Thuộc tính kiểu của nút submit (<Iẩ PUT TYPE=submit>)
text	thuộc tính HTML	Thuộc tính kiểu của trường trong form

		(<Iả PUT TYPE=text>)
TEXTAREA	Thẻ HTML	Hộp thẻ cho nhiều dòng text (<TEXTAREA> defautl text </TEXTAREA>)
name	thuộc tính JavaScript	Xâu chứa tên phần tử HTML (button, checkbox, password...)
value	thuộc tính JavaScript	Xâu chứa giá trị hiên thời của một phần tử HTML(button, checkbox, password...)
click()	cách thức JavaScript	Mô tả việc kích vào một phần tử trên form (button, checkbox,password)
onClick	thuộc tính JavaScript	Thẻ sự kiện cho sự kiện kích (button, checkbox, radio button, reset, selection list, submit)
checked	thuộc tính JavaScript	Giá trị kiểu Boolean mô tả một lựa chọn check(checkbox, radio button)
defaultChecked	thuộc tính JavaScript	Xâu chứa giá trị mặc định của một phần tử HTML (password, text, textarea)
focus()	cách thức JavaScript	Mô tả việc con trỏ tới một phần tử (password, text, textarea)
blur()	cách thức JavaScript	Mô tả việc con trỏ rời khỏi một phần tử (password, text, textarea)
select()	cách thức JavaScript	Mô tả việc lựa chọn dòng text trong một trường (password, text, textarea)
onFocus()	Thẻ sự kiện	Thẻ sự kiện cho sự kiện **focus**(password, selection list, text, textarea)
onBlur	Thẻ sự kiện	Thẻ sự kiện cho sự kiện blur (password, selection list, text, textarea)
onChange	Thẻ sự kiện	Thẻ sự kiện cho sự kiện khi giá trị của trường thay đổi (password, selection list, text, textarea)
onSelect	Thẻ sự kiện	Thẻ sự kiện khi người sử dụng chọn dòng text trong một trường (password, text, textarea)
index	thuộc tính JavaScript	Là một số nguyên mô tả lựa chọn hiện thời trong một nhóm lựa chọn (radio button)
length	thuộc tính JavaScript	Số nguyên mô tả tổng số các lựa chọn trong một nhóm các lựa chọn (radio button)
dafautlSelected	thuộc tính JavaScript	Giá trị Boolean mô tả khi có một lựa chọn được đặt là mặc định (seledtion list)
options	thuộc tính JavaScript	Mảng các lựa chọn trong danh sách lựa

		chọn
text	thuộc tính JavaScript	Dòng text hiển thị cho một thành phần của menu trong danh sách lựa chọn
TABLE	thẻ HTML	Hộp thẻ cho các bảng HTML
TR	thẻ HTML	Hộp thẻ cho các hàng của một bảng HTML
TD	thẻ HTML	Hộp thẻ cho các ô của một hàng trong một bảng HTML
COLSPAå	thuộc tính HTML	Là thuộc tính của thẻ TD mô tả trong một ô của bảng có nhiều cột
ROWSPAå	thuộc tính HTML	Là thuộc tính của thẻ TD mô tả trong một ô của bảng có nhiều hàng
BODER	thuộc tính HTML	Là thuộc tính của thẻ TABLE mô tả độ rộng đường viền của bảng
document.forms[]	thuộc tính JavaScript	mảng của các đối tượng form với một danh sách các form trong một document
string.substring()	cách thức JavaScript	Trả lại một xâu con của xâu string từ tham số vị trí ký tự đầu đến vị trí ký tự cuối
Math.floor()	cách thức JavaScript	Trả lại một giá trị nguyên tiếp theo nhỏ hơn giá trị của tham số đưa vào.
string.length	thuộc tính JavaScript	Giá trị nguyên của số thứ tự ký tự cuối cùng trong xâu **string**

3.14. Mô hình đối tượng (OBJECT MODEL)

3.14.1. Đối tượng và thuộc tính

ả hư đã biết, một đối tượng trong JavaScript có các thuộc tính đi kèm với nó. Bạn có thể truy nhập đến các thuộc tính của nó bằng cách gọi :

objectả ame.propertyả ame

Cả tên đối tượng và tên thuộc tính đều nhạy cảm. Bạn định nghĩa một thuộc tính bằng cách gán cho nó một giá trị. Ví dụ, giả sử có một đối tượng tên là myCar (trong trường hợp này giả sử đối tượng này đã tồn tại sẵn sàng). Bạn có thể lấy các thuộc tính có tên make, model và year của nó như sau:

```
myCar.make = “Ford“
myCar.model = “Mustang“
myCar.year = 69;
```

Có một mảng lưu trữ tập hợp các giá trị tham chiếu tới từng biến. Thuộc tính và mảng trong JavaScript có quan hệ mật thiết với nhau, thực ra chúng chỉ khác nhau về cách giao tiếp với cùng một cấu trúc dữ liệu. Ví dụ cũng có thể truy nhập tới các thuộc tính của đối tượng myCar ở trên bằng mảng như sau:

```
myCar[make] = “Ford“
myCar[model] = “Mustang“
myCar[year] = 69;
```

Kiểu mảng này được hiểu như một mảng có khả năng liên kết bởi mỗi một phần tử trong đó đều có thể liên kết đến một giá trị xâu nào đó. Để minh hoạ việc mày được thực hiện như thế nào, hàm sau đây sẽ hiển thị các thuộc tính của một đối tượng thông qua tham số về kiểu đối tượng đó và tên đối tượng.

```
function show_props (obj, obj_name)
{
        var result=""
        for (i in obj)
                result=result+ obj_name + "."+ i+ "=" + obj[i] + "\n"
        return result
}
```

Khi gọi hàm show_props(myCar,"myCar") sẽ hiện lên:

```
myCar.make = Ford
myCar.model = Mustang
myCar.year = 69;
```

3.14.2. Tạo các đối tượng mới

Cả JavaScript client-side và server-side đều có một số đối tượng được định nghĩa trước. Tuy nhiên, bạn cũng có thể tạo ra những đối tượng của riêng bạn. Trong JavaScript 1.2, nếu bạn chỉ muốn tạo ra một đối tượng duy nhất của một kiểu đối tượng, bạn có thể tạo nó bằng cách sử dụng khởi tạo đối tượng. Hoặc nếu bạn muốn tạo ra nhiều cá thể của một kiểu đối tượng, bạn có thể tạo ra một hàm xây dựng trước, sau đó tạo ra các đối tượng có kiểu của hàm đó bằng toán tử new

SỬ DỤẩ G KHỞI TẠO ĐỐI TƯỢẩ G

Trong những phiên bản trước của ẩ avigator, bạn chỉ có thể tạo ra một đối tượng bằng cách sử dụng hàm xây dựng chúng hoặc sử dụng một hàm được cung cấp bởi một vài đối tượng khác để đạt được mục đích.

Tuy nhiên, trong ẩ avigator 4.0, bạn có thể tạo ra một đối tượng bằng cách sử dụng một khởi tạo đối tượng.Bạn sử dụng cách này khi bạn chỉ muốn tạo ra một cá thể đơn lẻ chứ không phải nhiều cá thể của đối tượng.

Cú pháp để tạo ra một đối tượng bằng cách khởi tạo đối tượng (Object Initializers):

objectName={property1: value1, property2: value2,
..., propertyN: valueN}

Trong đó **objectName** là tên của đối tượng mới, mỗi **propertyI** là một xác minh (có thể là một tên, một số hoặc một xâu ký tự) và mỗi **valueI** là một biểu thức mà giá trị của nó được gán cho **propertyI**. Có thể lựa chọn khởi tạo bằng tên đối tượng hoặc chỉ bằng các khai báo. ẩ ếu như bạn không cần dùng đến đối tượng đó trong mọi chỗ, bạn không cần phải gán nó cho một biến.

ẩ ếu một đối tượng được tạo bằng cách khởi tạo đối tượng ở mức cao nhất, mỗi lần đối tượng đó xuất hiện trong các biểu thức, JavaScript sẽ đánh giá lại nó một lần. ẩ goài ra, nếu sử

dụng việc khởi tạo này trong một hàm thì mỗi lần gọi hàm, đối tượng sẽ được khởi tạo một lần

Giả sử bạn có câu lệnh sau:

if (condition)

x={hi: ”**there**.”**}**

Trong trường hợp này, JavaScript sẽ tạo ra một đối tượng và gắn nó vào biến x nếu biểu thức **condition** được đánh giá là đúng

Còn ví dụ sau tạo ra một đối tượng myHonda với 3 thuộc tính:

myHonda={color:”red”,wheels:4,engine:{cylinder:4,size:2.2}}

Chú ý rằng thuộc tính **engine** cũng là một đối tượng với các thuộc tính của nó

Trong ả avigator 4.0, bạn cũng có thể sử dụng một khởi tạo để tạo một mảng. Cú pháp để tạo mảng bằng cách này khác với tạo đối tượng:

arrayName=[element0, element1,...,elementN]

Trong đó, **arrayName** là tên của mảng mới, và mỗi **elementI** là giá trị của phần tử ở vị trí đó của mảng. Khi bạn tạo một mảng bằng cách sử dụng phương pháp khởi tạo, thì nó sẽ coi mỗi giá trị là một phần tử trên mảng, và chiều dài của mảng chính là số các tham số.

Bạn không cần phải chỉ định rõ tất cả các phần tử trên mảng mới. ả ếu bạn đặt hai dấu phẩy vào hàng, thì mảng sẽ được tạo với những chốn trống cho những phần tử chưa được định nghĩa như ví dụ dưới đây:

ả ếu một mảng được tạo bằng cách khởi tạo(initializer) ở mức cao nhất, mỗi lần mảng đó xuất hiện trong các biểu thức, JavaScript sẽ đánh giá lại nó một lần. ả goài ra, nếu sử dụng việc khởi tạo này trong một hàm thì mỗi lần gọi hàm, mảng sẽ được khởi tạo một lần

Ví dụ1: Tạo một mảng coffees với 3 phần tử và độ dài của mảng là 3:

coffees = [“French Roast”,”Columbian”,”Kona”]

Ví dụ 2: Tạo ra một mảng với 2 phần tử được khởi đầu và một phần tử rỗng:

fish = [“Lion”, ,” Surgeon”]

Với biểu thức này, **fish[0]** là **“Lion”, fish[2]** là **” Surgeon**”, và **fish[2]** chưa được định nghĩa

SỬ DỤả G MỘT HÀM XÂY DỰả G(COả STRUCTOR FUả CTIOả)

Bạn có thể tạo ra đối tượng của riêng mình với hai bước sau:

1. Định nghĩa kiểu của đối tượng bằng cách viết một hàm xây dựng.
2. Tạo ra một cá thể của đối tượng đó bằng toán tử **new**

Để định nghĩa một kiểu đối tượng, ta phải tạo ra một hàm để chỉ định rõ tên, các thuộc tính và các cách thức của kiểu đối tượng đó. Ví dụ giả sử bạn muốn tạo một kiểu đối tượng ô tô với tên là **car**, có các thuộc tính **make, model, year** và **color**, để thực hiện việc này có thể viết một hàm như sau:

```
function car(make, model, year ){
        this.make = make
        this.model = model
        this.year = year
```

```
}
```

Chú ý việc sử dụng toán tử **this** để gán giá trị cho các thuộc tính của đối tượng phải thông qua các tham số của hàm.

Ví dụ, bạn có thể tạo một đối tượng mới kiểu **car** như sau:

```
mycar = new car("Eagle","Talon TSi",1993)
```

Câu lệnh này sẽ tạo ra đối tượng mycar và liên kết các giá trị được đưa vào với các thuộc tính. Khi đó giá trị của **mycar.make** là "***Eagle***", giá trị của **mycar.model** là "***Talon TSi***", và **mycar.year** là một số nguyên 1993....Cứ như vậy bạn có thể tạo ra nhiều đối tượng kiểu **car.**

Một đối tượng cũng có thể có những thuộc tính mà bản thân nó cũng là một đối tượng. Ví dụ bạn định nghĩa thêm một đối tượng khác là **person** như sau:

```
function person(name, age, sex){
        this.name=name
        this.age=age
        this.sex=sex
}
```

Và sau đó ta tạo ra hai người mới:

```
rank = new person("Rank McKinnon",33,"M")
ken = new person("Ken John",39,"M")
```

Bây giờ bạn định nghĩa lại hàm xây dựng **car** như sau:

```
function car(make, model, year,owner ){
        this.make = make
        this.model = model
        this.year = year
        this.owner = owner
}
```

ả hư vậy bạn có thể tạo đối tượng kiểu **car** mới:

```
car1 = new car("Eagle","Talon TSi",1993,rank)
car2 = new car("ả issan","300ZX",1992,ken)
```

ả hư vậy, thay vì phải qua một xâu ký tự hay một giá trị số khi tạo đối tượng, ta chỉ cần đưa hai đối tượng đã được tạo ở câu lệnh trên vào dòng tham số của đối tượng mới tạo. Ta cũng có thể lấy được thuộc tính của đối tượng owner bằng câu lênh sau:

```
car2.owner.name
```

Chú ý rằng bạn cũng có thể tạo ra một thuộc tính mới cho đối tượng trước khi định nghĩa nó, ví dụ:

```
car1.color="black"
```

ả hư vậy, thuộc tính ***color*** của đối tượng **car1** được gán là "***black***". Tuy nhiên, nó sẽ không gây tác động tới bất kỳ một đối tượng kiểu **car** nào khác. ả ếu muốn thêm thuộc tính cho tất cả các đối tượng thì phải định nghĩa lại hàm xây dựng đối tượng.

LẬP MỤC LỤC CHO CÁC THUỘC TÍẩ H CỦA ĐỐI TƯỢẩ G

Trong ẩ avigator 2.0, bạn có thể gọi thuộc tính của một đối tượng bằng tên thuộc tính hoặc bằng số thứ tự của nó. Tuy nhiên từ ẩ avigator 3.0 trở đi, nếu ban đầu bạn định nghĩa một thuộc tính bằng tên của nó, bạn sẽ luôn luôn phải gọi nó bằng tên, và nếu bạn định nghĩa một thuộc tính bằng chỉ số thì bạn cũng luôn luôn phải gọi tới nó bằng chỉ số.

Điều này ứng dụng khi bạn tạo một đối tượng với những thuộc tính của chúng bằng hàm xây dựng (như ví dụ về kiểu đối tượng **car** ở phần trước) và khi bạn định nghĩa những thuộc tính của riêng một đối tượng (như mycar.color=”red”). Vì vậy nếu bạn định nghĩa các thuộc tính của đối tượng ngay từ đầu bằng chỉ số như mycar[5]=”25 mpg”, bạn có thể lần lượt gọi tới các thuộc tính khác như mycar[5].

Tuy nhiên điều này là không đúng đối với những đối tượng tương ứng của HTML như mảng form. Bạn có thể gọi tới các đối tượng trong mảng bởi số thứ tự hoặc tên của chúng. Ví dụ thẻ <FORM> thứ hai trong một document có thuộc tính ẩ AME là “myform” thì bạn có thể gọi tới form đó bằng document.form[1] hoặc document.form[“myForm”] hoặc document.myForm

ĐỊỊ H Ị GHĨA THÊM CÁC THUỘC TÍỊ H CHO MỘT KIỂU ĐỐI TƯỢỊ G

Bạn có thể thêm thuộc tính cho một kiểu đối tượng đã được định nghĩa trước bằng cách sử dụng thuộc tính property. Thuộc tính được định nghĩa này không chỉ có tác dụng đối với một đối tượng mà có tác dụng đối với tất cả các đối tượng khác cùng kiểu.Ví dụ sau thực hiện thêm thuộc tính **color** cho tất cả các đối tượng kiểu **car**, sau đó gắn một giá trị màu cho thuộc tính **color** của đối tượng **car1:**

```
car.prototype.color=null
car1.color=”red”
```

ĐỊỊ H Ị GHĨA CÁC CÁCH THỨC

Một cách thức là một hàm được liên kết với một đối tượng. Bạn định nghĩa một cách thức cũng có nghĩa là bạn định nghĩa một hàm chuỊn. Bạn có thể sử dụng cú pháp sau để gắn một hàm cho một đối tượng đang tồn tại:

object.methodname = function_name

Trong đó object là đối tượng đang tồn tại, methodname là tên cách thức và function_name là tên hàm

Bạn có thể gọi cách thức này từ đối tượng như sau:

object.methodname(<tham số>)

Bạn có thể định nghĩa cách thức cho một kiểu đối tượng bằng cách đưa cách thức đó vào trong hàm xây dựng đối tượng. Ví dụ bạn có thể định nghĩa một hàm có thể định dạng và hiển thị các thuộc tính của các đối tượng kiểu **car** đã xây dựng ở phần trước:

```
function displayCar () {
        var result = “Abeautiful”+this.year+ “ ”+ this.make + “ ”+ this.model
        document.write(result)
}
```

Bạn có thể thêm cách thức này vào cho đối tượng car bằng cách thêm dòng lệnh sau vào hàm định nghĩa đối tượng

```
this.displayCar= displayCar;
```

Ị hư vậy có thể định nghĩa lại đối tượng **car** như sau:

```
function car(make, model, year,owner ){
    this.make = make
    this.model = model
    this.year = year
    this.owner = owner
    this.displayCar= displayCar
}
```

Sau đó, bạn có thể gọi cách thức displayCar đối với mỗi đối tượng:

```
car1.displayCar()
car2.displayCar()
```

SỬ DỤỊ G CHO CÁC THAM CHIẾU ĐỐI TƯỢỊ G (OBJECT REFEREỊ CES)

JavaScript có một từ khoá đặc biệt là **this** mà bạn có thể sử dụng nó cùng với một cách thức để gọi tới đối tượng hiện thời. Ví dụ, giả sử bạn có một hàm validate dùng để xác nhận giá trị thuộc tính của một đối tượng nằm trong một khoảng nào đó:

```
function validate(obj, lowval, hival){
    if ( (obj.value<lowdate)||(obj.value>hival) )
        alert("Invalid value!")
}
```

Sau đó bạn có thể gọi hàm **validate** từ mỗi thẻ sự kiện **onChange**:

```
<IỊ PUT TYPE="TEXT" Ị AME="AGE" SIZE=3 onChange="validate(this,18,99)"
        >
```

Khi liên kết với một thuộc tính form, từ khoá **this** có thể gọi tới form cha của đối tượng hiện thời. Trong ví dụ sau, **myForm** có chứa đối tượng **Text** và một nút bấm. Khi người sử dụng kích vào nút bấm, trường text sẽ hiển thị tên form. Thẻ sự kiện onClick của nút bấm sử dụng **this.form** để gọi tới form cha là **myForm**.

```
<FORM Ị AME="myForm">
    Form name:<IỊ PUT TYPE="text" Ị AME="text1" VALUE="Beluga">
<P>
    <IỊ PUT TYPE="button" Ị AME="button1"
        value="Show Form Ị ame" onClick="this.form.text1.value=this.form.name">
</FORM>
```

XOÁ ĐỐI TƯỢỊ G

Trong JavaScript cho Ị avigator 2.0, bạn không thể xoá các đối tượng-chúng vẫn tồn tại trong khi bạn đã rời khỏi trang đó. Trong khi JavaScript cho Ị avigator 3.0 cho phép bạn

có thể xoá một đối tượng bằng cách đặt cho nó trỏ tới giá trị Ị ull (nếu như đó là lần cuối cùng gọi tới đối tượng). JavaScript sẽ đóng đối tượng đó ngay lập tức thông qua biểu thức gán.

3.15. Bài tập thực hành

3.15.1. Bài Tập 1

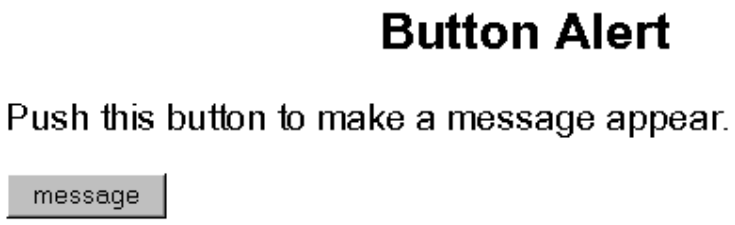
Tạo Giao Diện Ị hư Sau

Yêu cầu :

Khi Click chuột vào Radio Button thì có các thông điệp (Message) tương ứng

```
<FORM>
<p>
1:
<IỊ PUT TYPE="radio" Ị AME="radio" value="Bạn chọn số 1" onClick="alert(value)">
2:
<IỊ PUT TYPE="radio" Ị AME="radio" value=" Bạn chọn số 2 " onClick="alert(value)">
3:
<IỊ PUT TYPE="radio" Ị AME="radio" value=" Bạn chọn số 3” onClick="alert(value)" >
</form>
```

3.15.2. Bài tập 2



Khi Click chuột vàonút Message thì hiện lên câu chào

```
<form>
<input type="button" value="message" onClick="alert('How are you'); return true">
</form>
```

3.15.3. Bài tập 3

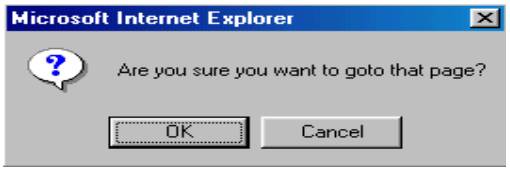
Chào tạm biệt

Khi đóng cửa sổ trình duyệt hoặc chuyển sang trang Web khác thì xuất hiện lời chào tạm biệt


```
<html>
<head>
<title>Click the back to see the Example</title>
<meta name="GEỊ ERATOR" content="Copyright Ị ext Step Software 1998 (c)">
</head>
<body onUnload="window.alert('This Message Appears When you exit a page!!!!!!!')">
<form>
  <p><input TYPE="button" VALUE="  BACK  " onClick="history.go(-1)"> </p>
</form>
<p>Click the back to see the Example!</p>
</body>
</html>
```

3.15.4. Bài tập 4

Yêu cầu ; Khi nhấp vào liên kết thì Windows hỏi .Ị ếu OK thì ta link dến trang đó ,không thì ta không là gì cả



http://www.cidnet.vn

```
<script>
function rusure(){
        question = confirm("YOUR COỊ FIRM MESSAGE")
        if (question !="0"){
                top.location = "YOUR LIỊ K GOES HERE"
        }
}
</script>

Ị ow put this anywhere in your page and change YOUR LIỊ K DESCRIPTIOỊ

<a href="" onClick="rusure(); return false;">YOUR LIỊ K DESCRIPTIOỊ </a>
```

3.15.5. Bài tập 5

Hãy tạo một chương trình máy tính điện tử như sau :


```
<html>
<head><script LAỊ GUAGE="JavaScript">
function a_plus_b(form) {
a=eval(form.a.value)
b=eval(form.b.value)
c=a+b
form.ans.value = c
}

function a_minus_b(form) {
a=eval(form.a.value)
b=eval(form.b.value)
c=a-b
form.ans.value=c
}

function a_times_b(form) {
a=eval(form.a.value)
b=eval(form.b.value)
c=a*b
form.ans.value=c
}

function a_div_b(form) {
a=eval(form.a.value)
b=eval(form.b.value)
c=a/b
form.ans.value = c
}

function a_pow_b(form) {
a=eval(form.a.value)
b=eval(form.b.value)
c=Math.pow(a, b)
form.ans.value = c
}

</script>

<title>E:\button\windowsizer_.htm</title>
</head>

<body>

<form name="formx">
  <p><input type="text" size="4" value="12" name="a"> <input type="button" value="  +  "
```

```
 onClick="a_plus_b(this.form)"> <input type="button" value="  -  "
 onClick="a_minus_b(this.form)"> <input type="button" value="  x  "
 onClick="a_times_b(this.form)"> <input type="button" value="  /  "
 onClick="a_div_b(this.form)"> <input type="button" value="  ^  "
 onClick="a_pow_b(this.form)"> <input type="number" size="4" value="3" name="b"> =
<input
 type "number" value="0" name="ans" size="9"> </p>
</form>
</body>
</html>
```

3.15.6. Bài tập 6

Tạo một chương trình mô tả Lịch để bàn như sau :

November 2002

Su	M	Tu	W	Th	F	Sa
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30

```
<html>
<head>
<title>Ị ext Step Software - Java Script Ị umber - 14</title>
<meta name="GEỊ ERATOR" content="(c) 1998 Copyright Ị ext Step Software">
</head>

<body>
<script LAỊ GUAGE="JavaScript">
<!-- Begin
monthnames = new Array("January","Februrary","March","April","May","June",
"July","August","September","October","Ị ovember","Decemeber");
var linkcount=0;
function addlink(month, day, href) {
var entry = new Array(3);
entry[0] = month;
entry[1] = day;
entry[2] = href;
this[linkcount++] = entry;
}
Array.prototype.addlink = addlink;
linkdays = new Array();
monthdays = new Array(12);
monthdays[0]=31;
monthdays[1]=28;
monthdays[2]=31;
monthdays[3]=30;
monthdays[4]=31;
monthdays[5]=30;
monthdays[6]=31;
monthdays[7]=31;
monthdays[8]=30;
```

```
monthdays[9]=31;
monthdays[10]=30;
monthdays[11]=31;
todayDate=new Date();
thisday=todayDate.getDay();
thismonth=todayDate.getMonth();
thisdate=todayDate.getDate();
thisyear=todayDate.getYear();
thisyear = thisyear % 100;
thisyear = ((thisyear < 50) ? (2000 + thisyear) : (1900 + thisyear));
if (((thisyear % 4 == 0)
&& !(thisyear % 100 == 0))
||(thisyear % 400 == 0)) monthdays[1]++;
startspaces=thisdate;
while (startspaces > 7) startspaces-=7;
startspaces = thisday - startspaces + 1;
if (startspaces < 0) startspaces+=7;
document.write("<table border=2 bgcolor=white ");
document.write("bordercolor=black><font color=black>");
document.write("<tr><td colspan=7><center><strong>"
+ monthnames[thismonth] + " " + thisyear
+ "</strong></center></font></td></tr>");
document.write("<tr>");
document.write("<td align=center>Su</td>");
document.write("<td align=center>M</td>");
document.write("<td align=center>Tu</td>");
document.write("<td align=center>W</td>");
document.write("<td align=center>Th</td>");
document.write("<td align=center>F</td>");
document.write("<td align=center>Sa</td>");
document.write("</tr>");
document.write("<tr>");
for (s=0;s<startspaces;s++) {
document.write("<td> </td>");
}
count=1;
while (count <= monthdays[thismonth]) {
for (b = startspaces;b<7;b++) {
linktrue=false;
document.write("<td>");
for (c=0;c<linkdays.length;c++) {
if (linkdays[c] != null) {
if ((linkdays[c][0]==thismonth + 1) && (linkdays[c][1]==count)) {
document.write("<a href=\"" + linkdays[c][2] + "\">");
linktrue=true;
    }
  }
}
if (count==thisdate) {
document.write("<font color='FF0000'><strong>");
}
if (count <= monthdays[thismonth]) {
```

```
document.write(count);
}
else {
document.write(" ");
}
if (count==thisdate) {
document.write("</strong></font>");
}
if (linktrue)
document.write("</a>");
document.write("</td>");
count++;
}
document.write("</tr>");
document.write("<tr>");
startspaces=0;
}
document.write("</table></p>");
// End -->
</script>
</body>
</html>
```

3.15.7. Bài tập 7

Gửi thư

E-Mail Someone!

E-Mail Someone!

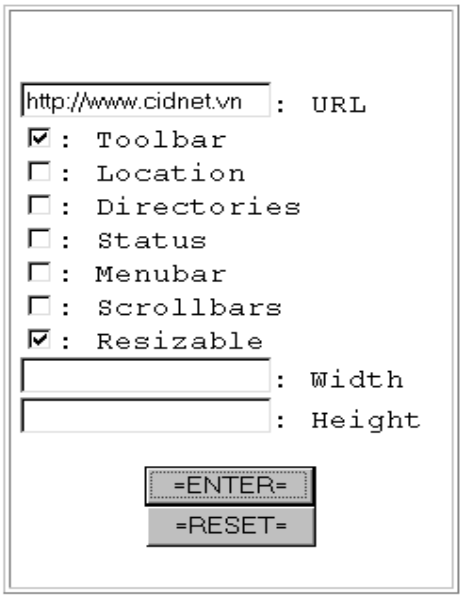
Khi Click vào link hoặc button thì cho phép ta nhập vào địa chỉ người nhận và subject.

```
<HEAD>

<SCRIPT LAỊ GUAGE="JavaScript">
<!-- Begin
function mailsome1(){
who=prompt("Enter recipient's email address: ","antispammer@earthling.net");
what=prompt("Enter the subject: ","none");
if (confirm("Are you sure you want to mail "+who+" with the subject of "+what+"?")==true){
parent.location.href='mailto:'+who+'?subject='+what+";
  }
}
// End -->
</SCRIPT>

<BODY>
<CEỊ TER>
<a href='javascript:mailsome1()'>E-Mail Someone!</a>
<FORM>
<input type=button value="E-Mail Someone!" onClick="mailsome1()">
</FORM>
```

```
</CEỊ TER>
```

3.15.8. Bài tập 8

Viết chương trình cho phép link đến một trang Web khác trong đó cho phép tuỳ chọn các đối tượng Window




```
<HEAD>
<SCRIPT LAỊ GUAGE="JavaScript">
<!-- Begin
function customize(form) {
var address = document.form1.url.value;
var op_tool  = (document.form1.tool.checked== true)  ? 1 : 0;
var op_loc_box  = (document.form1.loc_box.checked == true)  ? 1 : 0;
var op_dir  = (document.form1.dir.checked == true)  ? 1 : 0;
var op_stat  = (document.form1.stat.checked == true)  ? 1 : 0;
var op_menu  = (document.form1.menu.checked == true)  ? 1 : 0;
var op_scroll  = (document.form1.scroll.checked == true)  ? 1 : 0;
var op_resize  = (document.form1.resize.checked == true)  ? 1 : 0;
var op_wid  = document.form1.wid.value;
var op_heigh = document.form1.heigh.value;
var option = "toolbar="+ op_tool +",location="+ op_loc_box +",directories="
+ op_dir +",status="+ op_stat +",menubar="+ op_menu +",scrollbars="
+ op_scroll +",resizable="  + op_resize +",width=" + op_wid +",height="+ op_heigh;
var win3 = window.open("", "what_I_want", option);
var win4 = window.open(address, "what_I_want");
}
function clear(form) {
document.form1.wid.value="";
document.form1.heigh.value="";
}
// End -->
</SCRIPT>
<BODY>
<CEỊ TER>
<h4>Please choose from the following selections to customize your window</h4>
```

```
<br>
<TABLE cellpadding=5 border><TR><TD><PRE>
<FORM name=form1  ACTIOỊ ="javascript:" METHOD="POST">
<IỊ PUT TYPE="text" Ị AME="url" value="http://www.geocities.com" >: URL
<IỊ PUT TYPE="checkbox" Ị AME="tool">: Toolbar
<IỊ PUT TYPE="checkbox" Ị AME="loc_box">: Location
<IỊ PUT TYPE="checkbox" Ị AME="dir">: Directories
<IỊ PUT TYPE="checkbox" Ị AME="stat">: Status
<IỊ PUT TYPE="checkbox" Ị AME="menu">: Menubar
<IỊ PUT TYPE="checkbox" Ị AME="scroll">: Scrollbars
<IỊ PUT TYPE="checkbox" Ị AME="resize">: Resizable
<IỊ PUT TYPE="text"   Ị AME="wid" value= >: Width
<IỊ PUT TYPE="text"  Ị AME="heigh" value=>: Height
<BR><CEỊ TER>
<IỊ PUT TYPE="button" VALUE="=EỊ TER=" OnClick="customize(this.form)">
<IỊ PUT TYPE="reset" VALUE="=RESET=" onClick="clear(this.form)">
</PRE></TD></TR></TABLE>
</FORM>
</CEỊ TER>
```

3.15.9. Bài tập 9

kiểm tra tính hợp lệ của thông tin nhập vào

Name: Age: Submit

```
<HEAD>
<SCRIPT LAỊ GUAGE="JavaScript">
<!-- Begin
function validate(){
var digits="0123456789"
var temp
if (document.testform.Ị ame.value=="") {
alert("Ị o Ị ame !")
return false
}
if (document.testform.age.value=="") {
alert("Invalid Age !")
return false
}
for (var i=0;i<document.testform.age.value.length;i++){
temp=document.testform.age.value.substring(i,i+1)
if (digits.indexOf(temp)==-1){
alert("Invalid Age !")
return false
     }
   }
return true
}
// End -->
</SCRIPT>
<BODY>
<FORM name="testform" onSubmit="return validate()">
```

```
Ị ame:<input type="text" size=30 name="Ị ame">
Age:<input type="text" size=3 name="age">
<input type="submit" value="Submit">
</FORM>
```

3.15.10. Bài tập 10

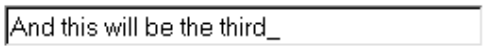
Tạo dòng chữ chạy trên thanh trạng thái:
Welcome to Total..


```
<html>
<head><script LAỊ GUAGE="JavaScript">
<!-- Begin
function scrollit(seed) {
var m1  = "Welcome to Total JavaScript 99!  ";
var m2  = "..... You can find all the scripts you need here! ";
var m3  = ".....Enjoy ";
var m4  = "";
var msg=m1+m2+m3+m4;
var out = " ";
var c   = 1;
if (seed > 100) {
seed--;
cmd="scrollit("+seed+")";
timerTwo=window.setTimeout(cmd,100);
}
else if (seed <= 100 && seed > 0) {
for (c=0 ; c < seed ; c++) {
out+=" ";
}
out+=msg;
seed--;
window.status=out;
cmd="scrollit("+seed+")";
timerTwo=window.setTimeout(cmd,100);
}
else if (seed <= 0) {
if (-seed < msg.length) {
out+=msg.substring(-seed,msg.length);
seed--;
window.status=out;
cmd="scrollit("+seed+")";
timerTwo=window.setTimeout(cmd,100);
}
else {
window.status=" ";
timerTwo=window.setTimeout("scrollit(100)",75);
     }
  }
}
// End -->
```

```
</script>
<title></title>
</head>

<body onLoad="scrollit(100)">
</body>
</html>
```

3.15.11. Bài tập 11

Tạo dòng chữ chạy trong TextBox




```
<script language="JavaScript">
    <!-- begin
    var max=0;
    function textlist()
    {
    max=textlist.arguments.length;
    for (i=0; i<max; i++)
    this[i]=textlist.arguments[i];
    }
    tl=new textlist
    (
     "This is a  message",
     "Another one",
     "And this will be the third",
     "And the fourth is the last !"
    );
    var x=0; pos=0;
    var l=tl[0].length;
    function textticker()
    {
      document.tickform.tickfield.value=tl[x].substring(0,pos)+"_";
      if(pos++==l) { pos=0; setTimeout("textticker()",1000); x++;
      if(x==max) x=0; l=tl[x].length; } else
      setTimeout("textticker()",50);
    }
    // end -->
    </script>
<html>
<head>
<title>E:\javascripts\scrolls\classic_.htm</title>
</head>

<body onLoad="textticker()">

<form name="tickform">
  <p><input type="text" name="tickfield" size="40"></p>
</form>
</body>
</html>
```

3.15.12. Bài tập 12

Tạo ngày tháng chạy trên thanh trạng thái


```
<html>
<head>
<title>E:\scrolls\classic_.htm</title>
</head>
<body>
<script LAỊ GUAGE="JavaScript">
<!-- Begin
var osd = "   "
osd +="This is yet another JavaScript Scroll example ";
osd +="from the good folks at The JavaScript Source. ";
osd +="This one has the date and time at the front.";
osd +="Did you notice?  It's coming around again, look!  ";
osd +="          ";
var timer;
var msg = "";
function scrollMaster () {
msg = customDateSpring(new Date())
clearTimeout(timer)
msg += " " + showtime() + " " + osd
for (var i= 0; i < 100; i++){
msg = " " + msg;
}
scrollMe()
}
function scrollMe(){
window.status = msg;
msg = msg.substring(1, msg.length) + msg.substring(0,1);
timer = setTimeout("scrollMe()", 200);
}
function showtime (){
var now = new Date();
var hours= now.getHours();
var minutes= now.getMinutes();
var seconds= now.getSeconds();
var months= now.getMonth();
var dates= now.getDate();
var years= now.getYear();
var timeValue = ""
timeValue += ((months >9) ? "" : " ")
timeValue += ((dates >9) ? "" : " ")
timeValue = ( months +1)
timeValue +="/"+ dates
timeValue +="/"+  years
var ap="A.M."
if (hours == 12) {
ap = "P.M."
}
if (hours == 0) {
```

```
hours = 12
}
if(hours >= 13){
hours -= 12;
ap="P.M."
}
var timeValue2 = " " + hours
timeValue2 += ((minutes < 10) ? ":0":":") + minutes + " " + ap
return timeValue2;
}
function MakeArray(n) {
this.length = n
return this
}
monthẸ ames = new MakeArray(12)
monthẸ ames[1] = "Janurary"
monthẸ ames[2] = "February"
monthẸ ames[3] = "March"
monthẸ ames[4] = "April"
monthẸ ames[5] = "May"
monthẸ ames[6] = "June"
monthẸ ames[7] = "July"
monthẸ ames[8] = "August"
monthẸ ames[9] = "Sept."
monthẸ ames[10] = "Oct."
monthẸ ames[11] = "Ẹ ov."
monthẸ ames[12] = "Dec."
daysẸ ames = new MakeArray(7)
daysẸ ames[1] = "Sunday"
daysẸ ames[2] = "Monday"
daysẸ ames[3] = "Tuesday"
daysẸ ames[4] = "Wednesday"
daysẸ ames[5] = "Thursday"
daysẸ ames[6] = "Friday"
daysẸ ames[7] = "Saturday"
function customDateSpring(oneDate) {
var theDay = daysẸ ames[oneDate.getDay() +1]
var theDate =oneDate.getDate()
var theMonth = monthẸ ames[oneDate.getMonth() +1]
var dayth="th"
if ((theDate == 1) || (theDate == 21) || (theDate == 31)) {
dayth="st";
}
if ((theDate == 2) || (theDate ==22)) {
dayth="nd";
}
if ((theDate== 3) || (theDate  == 23)) {
dayth="rd";
}
return theDay + ", " + theMonth + " " + theDate + dayth + ","
}
scrollMaster();
```

```
// End -->
</script>
</body>
</html>
```

3.15.13. Bài tập 13

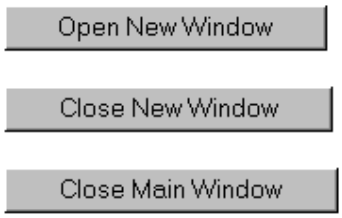
Tạo dòng chữ bay vào thanh trạng thái từng chữ cái một

```
<html>
<head><script LAỊ GUAGE="JavaScript">
<!-- Begin
function scroll(jumpSpaces,position) {
var msg = "Another JavaScript Example!  Do you like it?"
var out = ""
if (killScroll) {return false}
for (var i=0; i<position; i++){
out += msg.charAt(i)}
for (i=1;i<jumpSpaces;i++) {
out += " "}
out += msg.charAt(position)
window.status = out
if (jumpSpaces <= 1) {
position++
if (msg.charAt(position) == ' ') {
position++ }
jumpSpaces = 100-position
}
else if (jumpSpaces >  3) {
jumpSpaces *= .75}
else {
jumpSpaces--}
if (position != msg.length) {
var cmd = "scroll(" + jumpSpaces + "," + position + ")";
scrollID = window.setTimeout(cmd,5);
}
else {
scrolling = false
return false}
return true;
}
function startScroller() {
if (scrolling)
if (!confirm('Re-initialize snapIn?'))
return false
killScroll = true
scrolling = true
var killID = window.setTimeout('killScroll=false',6)
scrollID = window.setTimeout('scroll(100,0)',10)
return true
}
var scrollID = Object
var scrolling = false
var killScroll = false
```

```
// End -->
</script>

<title></title>
</head>
<!-- STEP TWO: Add the onLoad event handler to the BODY tag  -->

<body onLoad="startScroller()">
</body>
</html>
```

3.15.14. Bài tập 15

Tạo 3 button như sau :




```
<html>

<head>
<title>Create a Ị ew Window</title>
</head>

<body>

<form Ị AME="winform">
  <p><input TYPE="button" VALUE="Open Ị ew Window"
  onClick="Ị ewWin=window.open('','Ị ewWin',
'toolbar=no,status=no,width=200,height=100'); ">
  </p>
  <p><input TYPE="button" VALUE="Close Ị ew Window" onClick="Ị ewWin.close();"> </p>
  <p><input TYPE="button" VALUE="Close Main Window" onClick="window.close();"> </p>
</form>
</body>
</html>
```

3.15.15. Bài tập 15

Cửa sổ tự trượt

```
<script language="JavaScript1.2">
var currentpos=0,alt=1,curpos1=0,curpos2=-1
function initialize(){
startit()
}
function scrollwindow(){
if (document.all)
temp=document.body.scrollTop
else
temp=window.pageYOffset
```

```
if (alt==0)
alt=1
else
alt=0
if (alt==0)
curpos1=temp
else
curpos2=temp
if (curpos1!=curpos2){
if (document.all)
currentpos=document.body.scrollTop+1
else
currentpos=window.pageYOffset+1
window.scroll(0,currentpos)
}
else{
currentpos=0
window.scroll(0,currentpos)
}
}
function startit(){
setInterval("scrollwindow()",10)
}
window.onload=initialize
</script>
```

3.15.16. Bài tập 16

Tạo Combo box có fulldown menu

```
<html>
<head><script LAỊ GUAGE="JavaScript">
<!-- Begin
function formHandler(){
var URL = document.form.site.options[document.form.site.selectedIndex].value;
window.location.href = URL;
// End -->
}
</script>

<title>E:\button\pushme_.htm</title>
</head>
<!-- STEP OỊ E: Paste this code into the BODY of your HTML document  -->

<body>

<form name="form">
  <p><select Ị AME="site" SIZE="1" onChange="formHandler()">
    <option VALUE>Go to.... </option>
    <option VALUE="http://www.metacrawler.com">Metacrawler </option>
    <option VALUE="http://www.altavista.digital.com">Altavista </option>
    <option VALUE="http://www.webcrawler.com">Webcrawler </option>
    <option VALUE="http://www.lycos.com">Lycos </option>
    <option VALUE="http://www.javascriptsource.com">The JavaScript Source </option>
```

```
  </select></p>
</form>
</body>
</html>
```

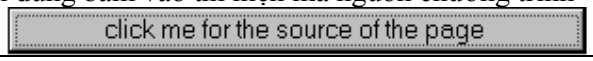
3.15.17. Bài tập 17

Tạo hiệu ứng ; khi đưa chuột vào thí xuất hiện ảnh khác khi đưa ra khỏi ảnh thì hiện ảnh cũ


```
<html>

<head>
<title>Document Title</title>
</head>

<body link="ffffff" alink="ffffff" vlink="ffffff">
<font SIZE="+2" FACE="Coronet">

<p align="center">onMouse<b><i>Over</i></b></font><br>
<br>
<a href="index.htm" onMouseOver="s.src='_view1.gif'" onMouseOut="s.src='_view2.gif'"><img
SRC="_view2.gif" name="s" width="158" height="29"></a><br>
<br>
<font SIZE="+2" FACE="Coronet">onMouse<b><i>Out</i></b></font><br>
<br>
<b>Pass the mouse over the images </b><br>
<b>Check out the script! See how easy this function is.</b><br>
</p>
</body>
</html>
```

3.15.18. Bài tập 18

Tạo nút bấm khi người dùng bấm vào thì hiện mã nguồn chương trình




```
<FORM>
<IỊ PUT TYPE=button Ị AME="view" VALUE="click me for the source of the page "
OnClick='window.location="view-source:" +window.location.href'>
</FORM>
```

20.Bài tập 20

Sử dụng Cookies để đếm số lần truy cập trang Web

```
<html>
<head><script LAỊ GUAGE="JavaScript">
<!-- Begin
function GetCookie (name) {
var arg = name + "=";
var alen = arg.length;
var clen = document.cookie.length;
var i = 0;
while (i < clen) {
```

```
var j = i + alen;
if (document.cookie.substring(i, j) == arg)
return getCookieVal (j);
i = document.cookie.indexOf(" ", i) + 1;
if (i == 0) break;
}
return null;
}
function SetCookie (name, value) {
var argv = SetCookie.arguments;
var argc = SetCookie.arguments.length;
var expires = (argc > 2) ? argv[2] : null;
var path = (argc > 3) ? argv[3] : null;
var domain = (argc > 4) ? argv[4] : null;
var secure = (argc > 5) ? argv[5] : false;
document.cookie = name + "=" + escape (value) +
((expires == null) ? "" : ("; expires=" + expires.toGMTString())) +
((path == null) ? "" : ("; path=" + path)) +
((domain == null) ? "" : ("; domain=" + domain)) +
((secure == true) ? "; secure" : "");
}
function DeleteCookie (name) {
var exp = new Date();
exp.setTime (exp.getTime() - 1);
var cval = GetCookie (name);
document.cookie = name + "=" + cval + "; expires=" + exp.toGMTString();
}
var expDays = 30;
var exp = new Date();
exp.setTime(exp.getTime() + (expDays*24*60*60*1000));
function amt(){
var count = GetCookie('count')
if(count == null) {
SetCookie('count','1')
return 1
}
else {
var newcount = parseInt(count) + 1;
DeleteCookie('count')
SetCookie('count',newcount,exp)
return count
  }
}
function getCookieVal(offset) {
var endstr = document.cookie.indexOf (";", offset);
if (endstr == -1)
endstr = document.cookie.length;
return unescape(document.cookie.substring(offset, endstr));
}
// End -->
</script>
```

```
<title>E:\cookies\name_.htm</title>
</head>

<body>
<script LAỊ GUAGE="JavaScript">
<!-- Begin
document.write("You've been here <b>" + amt() + "</b> times.")
// End -->
</script>
</body>
</html>
```

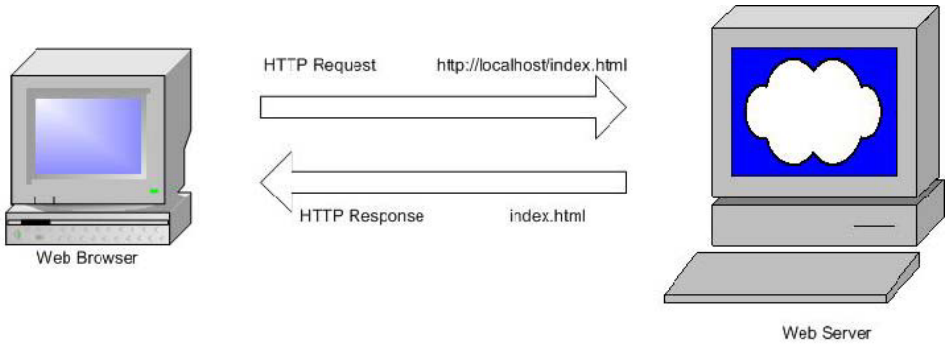
Chương 4- Lập trình web động với ASP (Active Server Page)

4.1. Giới thiệu ngôn ngữ lập trình web động ASP

Các website thuở ban đầu chỉ bao gồm các trang web tĩnh dưới dạng các file HTML, tất cả những gì cần hiển thị trên trang web thì người thiết kế phải tạo sẵn trên trang đó. Các trang web tĩnh có đuôi là .htm hoặc .html Chẳng hạn muốn tạo một trang web có hiển thị chữ "Hello" với màu chữ đỏ người ta viết file index.html với nội dung như sau:

```
<html>
<head>
<title>index</title>
</head>
<body>
<p><font color="red">Hello</font></p>
</body>
</html>
```

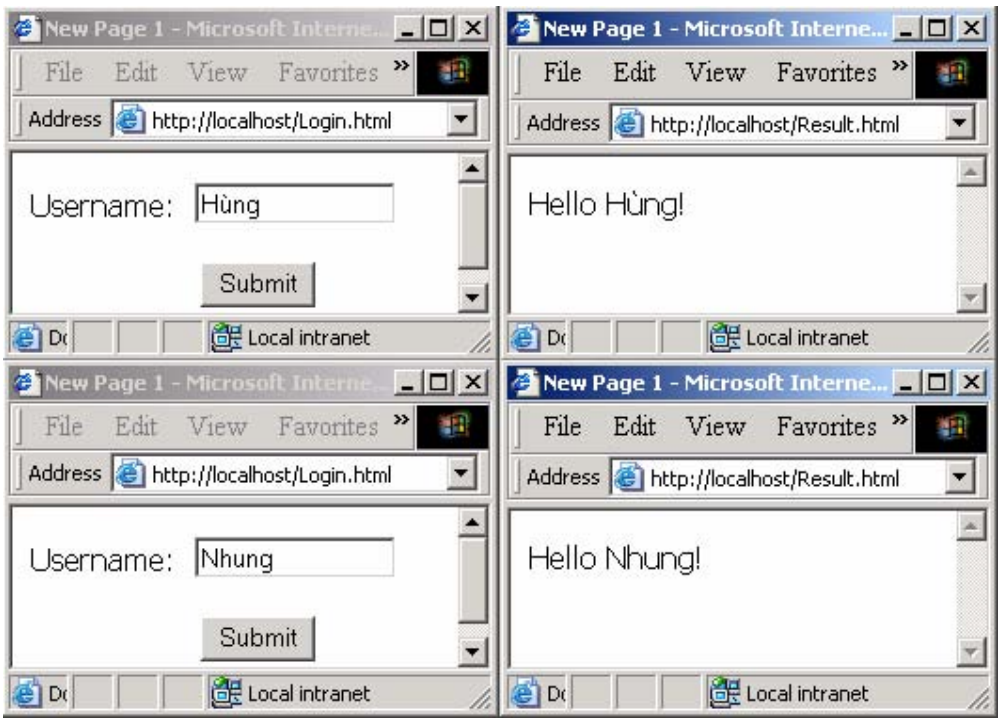
Trang web sau đó sẽ được lưu trên Web Server. Khi người dùng muốn xem trang web này họ sẽ dùng trình duyệt gửi một yêu cầu đến server bằng cách gõ vào địa chỉ URL ví dụ : http://localhost/index.html Lúc này Web Server nhận được yêu cầu sẽ tìm trong kho dữ liệu của nó trang web index.html tương ứng rồi gửi về cho client, sau đó trang web này sẽ được hiển thị ra bởi trình duyệt. Đó là cách hoạt động của web tĩnh.



Hình: Cách hoạt động của web tĩnh

Trang web tĩnh tuy rất tiện lợi nhưng không thểđáp ứng được mọi nhu cầu của ứng dụng web, đặc biệt là những yêu cầu tương tác giữa client và web server. Có nhiều tình huống mà nội dung trang web không phải lúc nào cũng có thể soạn thảo và lưu trữ sẵn được mà đôi khi nó cần được sinh ra một cách tự động tùy thuộc vào ngữ cảnh; hoặc có những xử lý phức tạp hơn việc server chỉ đơn giản trả về trang html khi nhận được yêu cầu từ người dùng, ví dụ như phải thu thập thông tin mà người dùng gửi lên qua URL hay form, hoặc truy cập dữ liệu trong database. Lấy ví dụ nếu chúng ta muốn xây dựng một trang web Login.htm yêu cầu người sử dụng nhập tên username, sau khi submit web server sẽ gửi về người dùng trang web Result.html có nội dung : Welcome username! Dễ dàng thấy rằng trang Result.htm không thể

soạn thảo sẵn được vì ứng với mỗi username mà người dùng nhập vào, trang này có nội dung khác nhau.



Hình: Trang Result.html có nội dung khác nhau tùy vào tương tác giữa client và webserver. Nó không thể soạn thảo sẵn!

Ị ghĩa là các trang web tĩnh không có khả năng tương tác với người dùng. Trong thực tế có rất nhiều trường hợp chúng ta thường gặp trong thế giới web đòi hỏi sự tương tác mà web tĩnh không thể giải quyết được (chat, forums, web mail, trang tin tức, giỏ hàng, thông tin thời tiết từng ngày, tỷ giá ngoại tệ hàng ngày)

Để giải quyết vấn đề này người ta sử dụng các ngôn ngữ lập trình web để hỗ trợ sự tương tác giữa client và server. Chúng là những file có chứa các mã lập trình, có thể tạo ra các trang web động, cho phép trả về cho client trang web có nội dung có thể thay đổi một cách linh động ứng với những ngữ cảnh cụ thể, thu thập và phản hồi với thông tin mà người dùng gửi lên server (thông qua form hay URL), truy cập dữ liệu trong database...

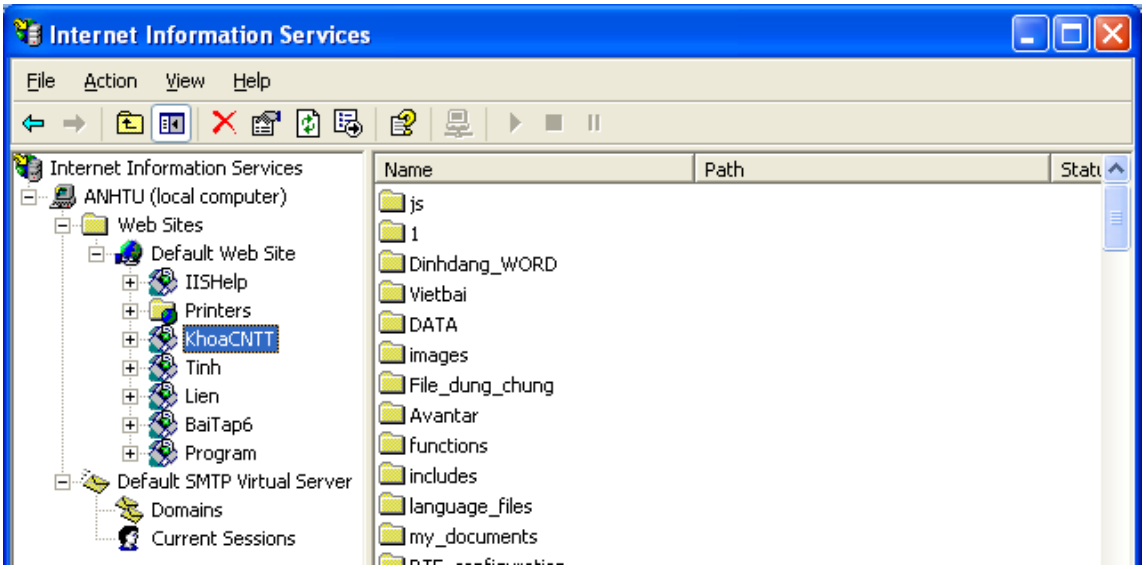
Một số ngôn ngữ lập trình web động phổ biến gồm ASP, PHP, Java, .net ... ASP (Active Server Pages) là ngôn ngữ lập trình web được viết bởi hãng Microsoft, rất phổ biến trên hệđiều hành Microsoft Windows. Các trang web viết bằng ngôn ngữ này có phần mở rộng là .asp (ví dụ HelloWorld.asp) thay vì .htm hay .html. Ị ội dung file ASP về cơ bản rất giống file Html bình thường, nó bao gồm các cú pháp html trộn lẫn các mã lập trình ASP (còn gọi là các script, được viết bằng VBScript hay JavaScript). Các Script trong ASP thực thi trên server. Có thể nói trang ASP là sự kết hợp các thẻ html, các script và các ActiveX Component. Script có thể trộn lẫn giữa các thẻ html và nằm trong cặp dấu <% %>

4.2. Web Server IIS

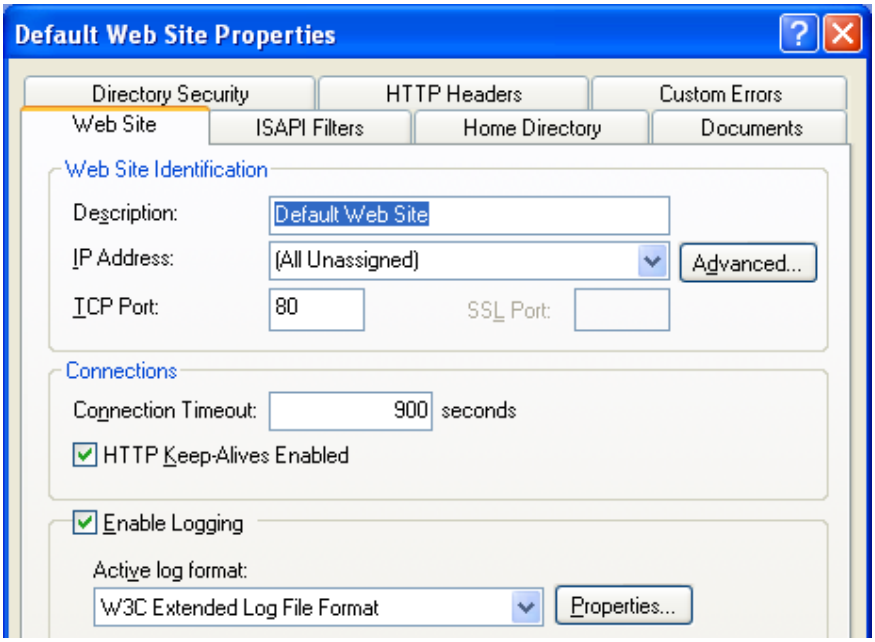
Thông thường người ta dùng ASP với Web Server có tên là Internet Information Services (IIS) của Microsoft. Đây là thành phần có sẵn trong hệ điều hành Windows 2000 hoặc XP.

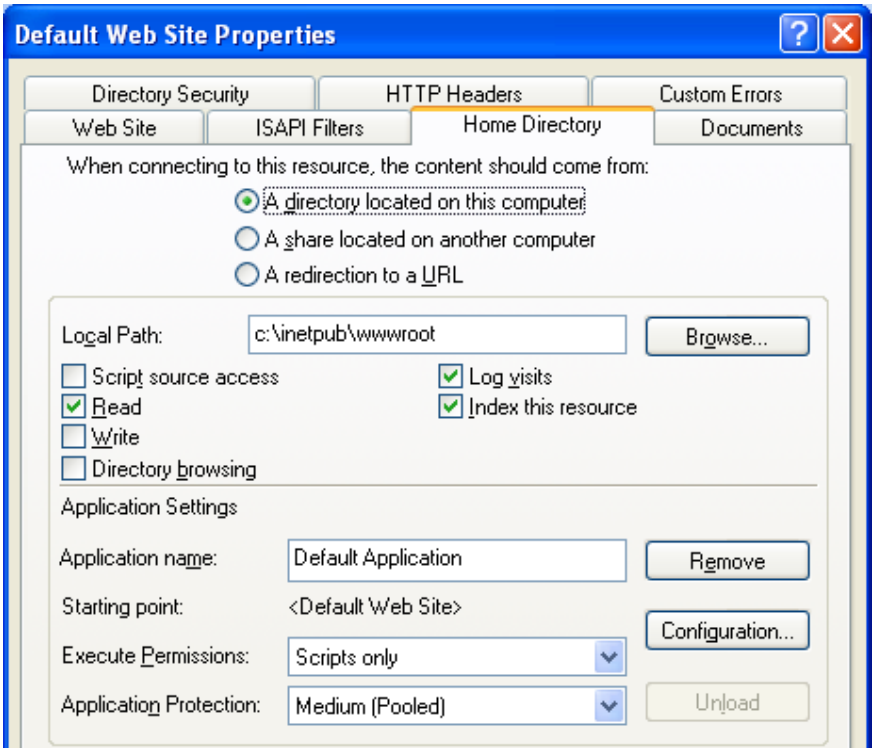
Ị ếu máy tính chưa cài đặt thì chúng ta có thể vào Control Panel => Add/remove programs=> Add/remove Windows Components=>Internet Information Services (IIS) và chọn cài đặt thành phần này

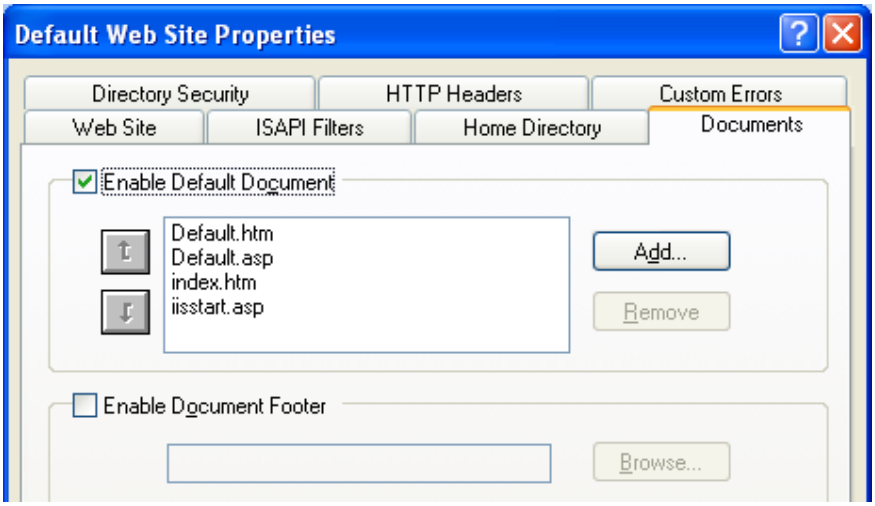
Lưu ý: Microsoft cung cấp giao diện đồ họa cho phép quản lý toàn bộ ứng dụng web gọi là MMC (Microsoft Management Console). Để hiển thị cửa sổ này chọn Start/Control Panel và chọn Administrative Tools, tiếp tục chọn Internet Service Manager, của sổ MMC như sau:



- Default Web Site là dịch vụ web mặc định chạy trên port 80 (cổng phục vụ tất cả các yêu cầu triệu gọi từ xa của trình khách theo giao thức HTTP)
- Có thể cấu hình lại cổng mặc định, thư mục mặc đích, hay trang mặc định, … thông qua của sổ này bằng cách Right Click vào Default Web Site và chọn Properties


Default Web Site Properties
Directory Security
HTTP Headers
Custom Errors
Web Site
ISAPI Filters
Home Directory
Documents
Web Site Identification
Description:
Default Web Site
IP Address:
(All Unassigned)
Advanced...
TCP Port:
80
SSL Port:
Connections
Connection Timeout:
900
seconds
HTTP Keep-Alives Enabled
Enable Logging
Active log format:
W3C Extended Log File Format
Properties...


Default Web Site Properties
Directory Security
HTTP Headers
Custom Errors
Web Site
ISAPI Filters
Home Directory
Documents
When connecting to this resource, the content should come from:
A directory located on this computer
A share located on another computer
A redirection to a URL
Local Path:
c:\inetpub\wwwroot
Browse...
Script source access
Read
Write
Directory browsing
Log visits
Index this resource
Application Settings
Application name:
Default Application
Remove
Starting point:
<Default Web Site>
Configuration...
Execute Permissions:
Scripts only
Application Protection:
Medium (Pooled)
Unload



4.3. Cài đặt và chạy ứng dụng ASP đầu tiên

Để bắt đầu chạy một website viết bằng ngôn ngữ ASP đầu tiên chúng ta thực hiện các bước sau:

Cài đặt web server IIS (ở phần trên) và start IIS

Cấu hình cho website bằng cách tạo Virtual Directory trên Web Server

• Viết các file ASP và save vào thư mục đã được cấu hình cho website trên server

Dùng trình duyệt (như Internet Explorer) trên client yêu cầu file ASP và hiển thị kết quả trả về.

Cách thực thi một trang ASP

Giả sử có file test.asp với nội dung như sau:

```
<%
Response.write("Chào các bạn")
%>
```

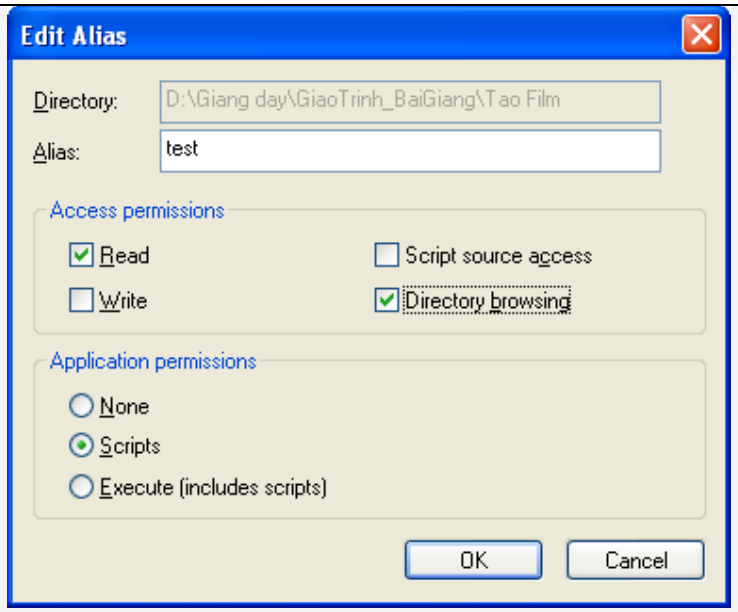
Có thể chạy file test.asp bằng một trong các cách sau:

Cách 1:

1. Copy file Test.asp vào thư mục **C:\inetpub\wwwroot**
2. Mở IE và gõ địa chỉ http://localhost/test.asp hoặc http://127.0.0.1/test.asp

Cách 2:

1. Tạo thư mục tại vị trí bất kì trên máy, giả sử đặt tên là **Myweb**
2. Copy file **Test.asp** vào thư mục **Myweb**
3. Tại thư mục **Myweb**, nhấn phải chuột chọn **Properties** hoặc **Sharing and Security...**, từ hộp thoại hiện ra chọn tab **Web Sharing**, và đánh dấu mục **Share this folder**, hiện ra hộp thoại như sau



4. Thiết lập các tùy chọn như trên rồi nhấn OK.
5. Mở IE và gõ địa chỉ http://localhost/Myweb/test.asp hoặc http://127.0.0.1/Myweb/test.asp

Cách 3:

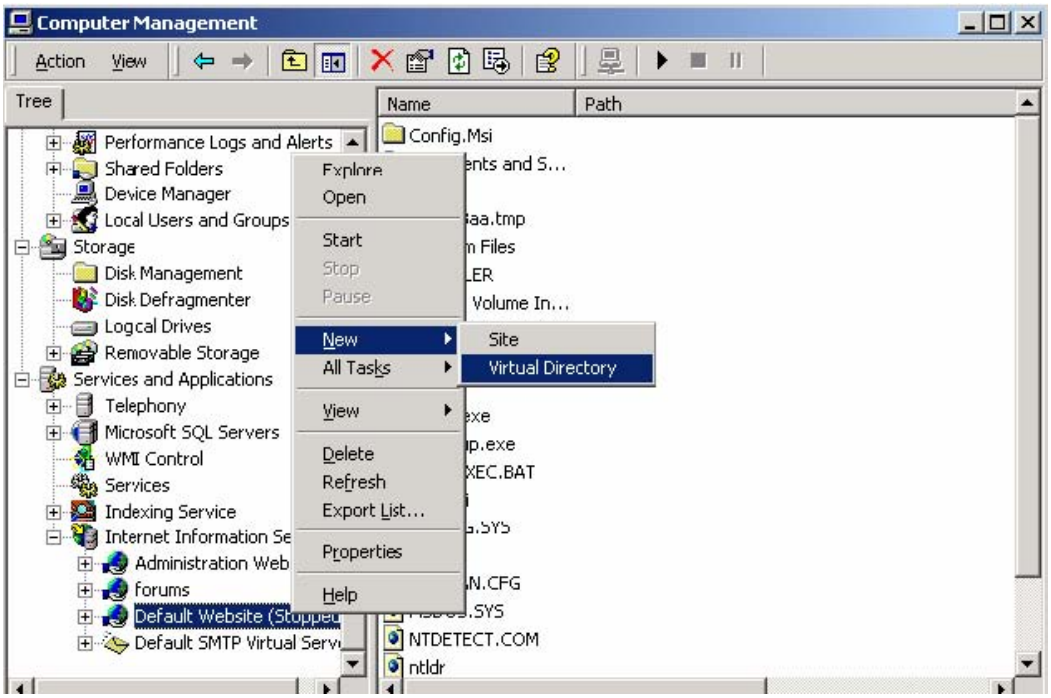
1. Tạo thư mục tại vị trí bất kì trên máy, giả sử đặt tên là **Myweb**
2. Copy file Test.asp vào thư mục **Myweb**
3. Mở cửa sổ MMC, nhấn phải chuột vào **Default Web Site**, chọn **New→Virtual Directory**…→ Nhập tên thư mục ảo → chọn đường dẫn tới thư mục Myweb
4. Mở IE và gõ địa chỉ http://localhost/Myweb/test.asp hoặc http://127.0.0.1/Myweb/test.asp

4.3.1. Cấu hình cho Website trên IIS

Sau khi start IIS mặc định web server sẽ phục vụ ở địa chỉ http://localhost (địa chỉ trên máy local, cũng giống như một địa chỉ website kiểu như http://www.yahoo.com trên Internet) Chúng ta tạo một thư mục ảo (Virtual Directory) trên web server để chứa ứng dụng web, ví dụ http://localhost/test ởđây "test" còn được gọi là Alias của Virtual Directory này. Vậy để lưu trữ các trang ASP trên server trước hết ta sẽ tạo một Virtual Directory với một Alias và thư mục tương ứng rồi upload các file ASP vào thư mục này, sau đó truy cập các trang ASP này thông qua địa chỉ http://localhost/Alias

Cách tạo một Virtual Directory trong IIS:

Vào Web Server từ Control Panel=> Administrative Tools=>Internet Services Manager (hoặc Computer Management)=> Default Website (nếu thấy nó đang stop thì start nó lên) => Ị ew=> Virtual Directory (làm theo wizard, chọn các tham số Alias: tên Virtual Directory của mình ví dụ "test", Directory: thư mục chứa Website ví dụ "C:\Web")



Hình: Tạo Virtual Directory trên IIS

Sau khi kết thúc wizard này chúng ta đã có một Virtual Directory sẵn sàng trên web server. Hãy save các trang asp vào thư mục "c:\Web". Địa chỉ truy cập vào website trong trường hợp này sẽ là: http://localhost/test/

Một cách khác cũng tương tự và dễ thao tác hơn là nhấn chuột phải vào thư mục C:\web, chọn Properties => Web sharing => Share this folder=> Add Alias.

4.3.2 Viết các file ASP

Script được viết trong cặp thẻ <% %>, bắt đầu bằng thẻ mở <% và kết thúc bằng thẻđóng %>

Chúng ta có thể soạn trang ASP bằng bất cứ chương trình soạn thảo nào như notepad, Frontpage, Dreamweaver...

Ví dụ, tạo 1 file Hello.asp để hiển thị lời chào Hello ra màn hình, save vào thư mục "c:\Web"

<html> <head> <title>New Page 1</title> </head> <body> <% response.write "Hello!" 'Hiển thị lời chào Hello %> </body> </html>

Câu lệnh response.write sẽ cho phép hiển thị một chuỗi ra trang web.

Chú thích trong lập trình ASP được viết sau dấu nháy đơn '

Mã lập trình ASP <%response.write "Hello!" %> được viết trộn lẫn giữa các thẻ HTML.

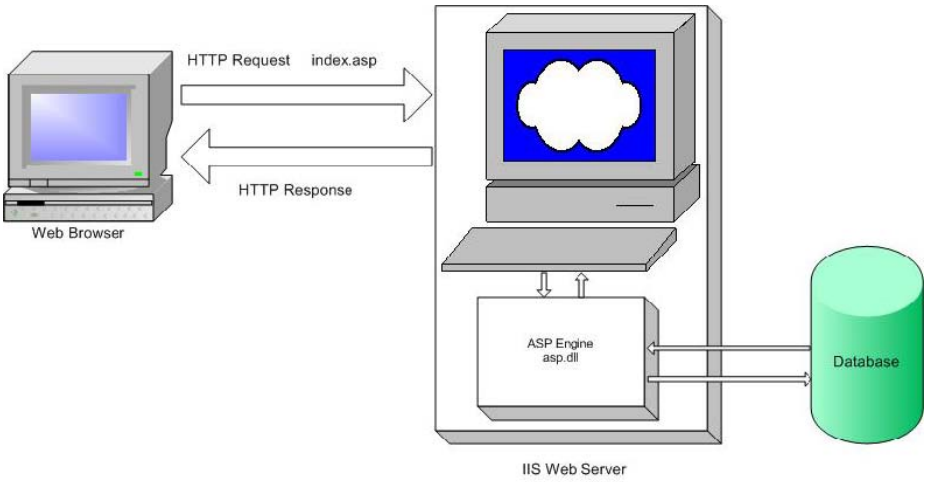
4.3.3 Dùng trình duyệt truy cập website

Mở trình duyệt (ví dụ Internet Explorer), trên thanh địa chỉ gõ địa chỉ sau đây để truy cập vào trang Asp ta đã tạo ra: http://localhost/test/Hello.asp

Lưu ý là trang asp phải chạy trên web server chứ không thể open trực tiếp với browser như các trang html.

Webserver xử lý như thế nào khi người dùng yêu cầu một trang ASP: Không giống như html, khi người dùng yêu cầu 1 trang html, web server sẽ tìm trong kho dữ liệu và trả về file html

đó để browser hiển thị lại phía client. Khi người dùng yêu cầu 1 trang Asp, IIS server sẽ chuyển trang ASP đó cho một bộ phận xử lý gọi là ASP engine. Engine sẽ đọc mã nguồn file asp theo từng dòng, thực thi các script trong file. Cuối cùng file ASP được trả về cho người dùng dưới dạng một trang html thuần túy (không còn mã script) giống như trang web tĩnh. Ị ếu chúng ta xem lại mã nguồn của trang này trên browser thì có thể thấy những đoạn code asp trong file đã được dịch thành các dữ liệu html bình thường.



Hình: ASP engine xử lý file asp trước khi trả về cho browser

Hình 1.6 Trang ASP sau khi thực thi trả về cho client dưới dạng 1 trang web tĩnh.Browser không xem được mã nguồn của trang ASP

Bây giờ chúng ta quay lại bài toán Login ở trên. Ta có thể soạn thảo một trang Login.html và một trang Result.asp như sau:

Login.html

```
<html>
<head>
<title>New Page 1</title>
</head>
<body>
<form method="POST" action="Result.asp">
<p>Username: <input type="text" name="username" ></p>
<p><input type="submit" value="Submit" name="submit"></p>
```

```
</form>
</body>
</html>
```

Result.asp

```
<html>
<head>
<title>New Page 1</title>
</head>
<body>
<%
dim x
x=request.form("username") 'biến x nhận lại giá trị username từ form login
response.write "Hello "&x  'hiển thị nội dung tùy theo giá trị nhận được do
'người dùng điền vào form
%>
</body>
</html>
```

Một số ví dụ khác:
Hiển thị ngày giờ của server

```
<html>
<head>
<title>New Page 2</title>
</head>
<body>
<%response.write Now%>
</body>
</html>
```

Kết quả: 7/5/2005 12:21:57 PM Hiển thị
năm và tháng:

```
<%
response.write "Year: "&year(now)
response.write "Month:"&month(now)
%>
```

Kết quả: Year: 2005 Month:7

4.4. Tóm tắt các cú pháp VBScript

Mã lệnh ASP có thể viết bằng VBScript hoặc JavaScript (đọc thêm tài liệu về ngôn ngữ này). Các script của ASP thực thi trên server và nằm trong cặp dấu <% %>. Bên trong có thể chứa các biểu thức, hàm, toán tử, lệnh hợp lệ của ngôn ngữ Script tương ứng. Ởđây chúng ta tìm hiểu vắn tắt cách sử dụng ASP để lập trình web động bằng VBScript.

4.4.1. Response.write

để gửi nội dung về cho trình duyệt ta dùng lệnh Response.write

<%response.write "Hello World!"%>

hoặc có thể viết ngắn gọn hơn <%="Hello World!"%>

4.4.2. Biến

Biến dùng để lưu trữ thông tin. Biến có phạm vi cục bộ, nếu nó được khai báo bên trong 1 hàm hay thủ tục thì nó chỉ có tác dụng trong hàm hay thủ tục đó, nếu nó khai báo trong phạm vi toàn trang ASP thì tác dụng của nó sẽ có phạm vi trong toàn trang ASP, tuy nhiên không có tác dụng trong trang ASP khác. Ví dụ trang Hello.asp ta có một biến x có giá trị là 3, trang Index.asp ta dùng lệnh <%response.write x %> thì sẽ không ra kết qủa là 3 vì biến x của trang Hello.asp không được hiểu trong trang Index.asp. Tương tự như vậy khi một biến được khai báo trong 1 hàm, sẽ không có tác dụng ở bên ngoài hàm đó.

Biến được khai báo và sử dụng bên trong trang asp nào dùng nó.

<% Dim x 'khai báo biến, không bắt buộc x=3 Response.write x %>

Biến không bắt buộc phải khai báo.

Trong asp không khai báo kiểu của biến. Asp sẽ căn cứ vào việc sử dụng biến mà quyết định xem nên xử lý biến đó như là kiểu gì.

<%Dim a, b a="Hello" 'a là một biến kiểu chuỗi For b=1 to 10 'b là một biến kiểu số nguyên Response.write b Next%>

Để có thể kiểm soát chính xác một biến theo kiểu mình mong muốn, chúng ta dùng các hàm chuyển đổi kiểu.

Để định nghĩa một biến có phạm vi sử dụng trong nhiều trang ASP của ứng dụng Web, ta dùng biến session và application (xem đối tượng session và application)

4.4.3. Mảng

Mảng dùng để lưu trữ dữ liệu theo một dãy các phần tử.

```
<%
dim y(5)  'khai báo mảng 6 phần tửđánh chỉ số từ 0 đến 5
y(0)=2
y(1)=13
response.write y(0)
response.write y(1)
%>
```

4.4.4. Ghép chuỗi

Để ghép các chuỗi với nhau ta dùng dấu &

```
<%Dim a, b
A="Cộng hòa xã hội chủ nghĩa Việt Nam"
B="Độc lập Tự do Hạnh phúc"
Response.write a&b
%>
```

4.4.5. Hàm có sẵn

VBScript hỗ trợ sẵn một số hàm cơ bản. Ví dụ hàm "now"sau đây sẽ trả về thời gian trên server

<%response.write now%>

4.4.5.1. Các hàm chuyển đổi kiểu

Các hàm này cho phép chuyển đổi kiểu dữ liệu **Cdate**:

Chuyển sang kiểu ngày tháng

<%Dim a, b a="22/1/2004" 'a đang được hiểu là một chuỗi b=Cdate(a) 'chuyển chuỗi a sang đúng kiểu ngày tháng %>

Cint: Chuyển sang kiểu Integer

<% Dim a,b

a="3" b=cint(a)

%>

Cstr: Chuyển sang kiểu string

<% Dim a,b a=3

b=Cstr(a) %>

Các hàm khác : Cbyte, Cdbl,CSng, Cbool, Ccur,

4.4.5.2. Các hàm format

Các hàm này cho phép định dạng dữ liệu FormatDateTime FormatCurrency FormatI umber FormatPercent

4.4.5.3. Các hàm toán học

Int: lấy phần nguyên của một số

<% Dim x=14.9 Y=Int(x) 'kết

quả y=14 %>

Các hàm khác : Abs, Atn, Cos, Exp, Fix, Hex, Log, Oct, Rnd, Randomize, Round, Sin, Sqr, Tan

4.4.5.4. Các hàm thao tác với chuỗi

Len: Lấy chiều dài chuỗi

*<%dim a,*b

a="Cộng hòa xã hội chủ nghĩa Việt Nam"

b=len(a)

%>

Ucase, Lcase: Chuyển chữ hoa thành chữ thường và ngược lại

*<%dim a,b,c,*d

a="hello"

b=Ucase(a) 'b="HELLO"

c="GOODBYE"

d=Lcase(c) 'd="goodbye" %>

Ltrim, Rtrim, Trim: cắt bỏ các khoảng trắng thừa

```
<% dim a,b,c,d,e,f
a="  Hello"
b=Ltrim(a) 'cắt bỏ hết các khoảng trắng bên trái
c="Hello "
d=Rtrim(a) 'cắt bỏ hết các khoảng trắng bên phải
e="  Hello  world  "
f=Trim(a)   'cắt bỏ hết các khoảng trắng thừa 2 bên và ở giữa
%>
```

Left, Mid, Right: Lấy một chuỗi con trong chuỗi lớn

```
<%Dim a,b,c,d
a="Hello World"
b=left(a,5) 'lấy 5 ký tự
bên trái của a, kết quả
b="Hello"
c=right(a,5) 'lấy 5 ký
tự bên phải của a, kết
quả c="World"
d=mid(a,7,1) 'lấy 1 ký
tự của a từ vị trí thứ 7,
kết quả d="W" %>
```

Các hàm khác: Space,String, StrReverse,StrComp,InStr,Replace,Split,join

4.4.5.5. Các hàm ngày tháng

Date, Time, Now: Lấy ngày, giờ hiện hành trên server

```
<% Response.write "Hom nay la ngay: " &Date 'Date trả về ngày hiện hành Response.write
"Bay gio la"&Time  'Time trả về giờ hiện hành Response.write Now  'Now trả về ngày và giờ
hiện hành %>
```

Các hàm khác: DateAdd, DateDiff, DatePart, Year, Month, Day, Weekday, Hour, Minute, Second

4.4.5.6. Các hàm kiểm tra

Các hàm này cho phép kiểm tra kiểu của biến và biểu thức **Isdate**:

Kiểm tra có phải đúng kiểu ngày tháng không?

```
<%Dim a
a="1/1/2004"
If Isdate(a) then
Response.write "a đúng là kiểu ngày tháng"
End if
%>
```

IsNumeric: Kiểm tra có phải đúng kiểu số không?

<%Dim a

A=”13”

*If IsNumeric(a) the*n

Response.write “a đúng là kiểu số”

*End i*f

%>

Các hàm khác: IsArray,IsEmpty,IsỊ ull,IsObject

4.4.6. Rẽ nhánh

4.4.6.1. If

Chúng ta sử dụng if theo cú pháp như ví dụ sau:

<% h=hour(now)

*If h >12 the*n

Response.write “Afternoon”

*els*e

Response.write “Morning”

End if %>

Hoặc:

<% h=hour(now) If h >12 then Response.write "Afternoon" else Response.write "Morning"

%>

4.4.6.2. Select case ... else ...End select

Cấu trúc rẽ nhánh trong trường hợp có nhiều hơn 2 lựa chọn

<%

h=hour(now)

Select case h

Case "1"

Response.write "1 am"

Case "2"

Response.write "2 am"

*Case els*e

Response.write "Other "

*End selec*t

%>

4.4.7. Lặp

4.4.7.1. For…Next

Vòng lặp có số lần lặp xác định

<%Dim i For i=1 to 10 Response.write i Next %>

4.4.7.2. Do While…Loop

Vòng lặp có số lần lặp không xác định

<% Dim i i=1 Do while i<=10 Response.write i i=i+1 Loop %>

4.4.7.3. While .. Wend

Vòng lặp có số lần lặp không xác định

<% Dim i i=1 While i<=10 Response.write i i=i+1 Wend %>

4.4.7.4. Do .. Loop Until

Vòng lặp có số lần lặp không xác định

<% i=1 do response.write i i=i+1 loop Until i>10 %>

4.4.8. Điều kiện and ,or, not

<% h=hour(now)

*If (h >12) and (h<18) the*n

Response.write "Afternoon"

*End i*f

%>

4.4.9. Thủ tục và hàm người dùng

Cũng như các ngôn ngữ lập trình khác, VBScript cho phép người dùng định nghĩa và sử dụng các thủ tục ,hàm. Ị hờ vậy chương trình có thể chia thành các module nhỏ tạo nên cấu trúc lập trình sáng sủa (phương pháp chia để trị) Chẳng hạn với một bài toán ASP cần thực hiện việc hiển thị dữ liệu từ Database ra màn hình, ta có thể xây dựng các thủ tục hay hàm thực hiện từng nhiệm vụđó:

-Thủ tục KetỊ oi

-Thủ tục HienThi

-Thủ tục HuyKetỊ oi Ị hư vậy phần chương trình chính sẽ rất sáng sủa, chúng ta chỉ việc gọi 3 thủ tục:

<% KetNoi

HienThi

HuyKetNoi %>

4.4.9.1. Thủ tục

Thủ tục thực hiện một nhóm các câu lệnh. Để viết một thủ tục chúng ta theo cấu trúc sau:

<%Sub TenThuTuc(Tham so)

*' Phần thân của thủ tụ*c

*End Su*b

%>

Ví dụ sau đây xây dựng chương trình đăng nhập gồm 2 file: Form.asp (hiển thị form để người dùng nhập username và password), Xulyform.asp (xử lý form, nếu username="test" và password="test" thì thông báo đăng nhập thành công, nếu không thì thông báo đăng nhập thất

bại). File Xulyform.asp sẽ viết thủ tục và gọi thủ tục này:

Form.asp

```
<html>
<body>
<form method="post" action="xulyform.asp">

<input type="text" name="user">
<input type="password" name="pass">
<input type="submit" name="submit">
</form>
</body>
</html>
```

Xulyform.asp

```
<%Sub CheckUser(username,password) if (username<>"test")
or (password <> "test") then response.write "Dang nhap that
bai!"
 else response.write "Dang nhap thanh cong!" end if End Sub %> <% dim a, b
a=request.form("user") b=request.form("pass") CheckUser a,b  'gọi thủ tục %>
```

4.4.9.2. Hàm

Hàm khác với thủ tục là nó trả về một kết quả. Để viết một hàm chúng ta viết theo cấu trúc sau:

```
<%Function TenFunction(tham so)
' Phần nội dung của hà
End Functio
%>
```

Chú ý trong nội dung của hàm bao giờ cũng phải có một lệnh trả về kết quả:
TenFunction=...
Với bài toán đăng nhập ở trên chúng ta có thể viết lại như sau (file xulyform.asp dùng hàm)

Form.asp

```
<html>
<body>
<form method="post" action="xulyform.asp">
<input type="text" name="user">
<input type="password" name="pass">
<input type="submit" name="submit">
</form>
```

```
</body>
</html>
```

Xulyform.asp

```
<%Function CheckUser(username,password) if
(username<>"test") or (password <> "test") then
CheckUser="False"
 else
  CheckUser="True" end if End Function %> <% dim a
a=CheckUser(request.form("user"),request.form("pass"))  ' gọi hàm if a="True" then
response.write "Dang nhap thanh cong" else response.write "Dang nhap that bai" end if %>
```

4.4.10. Sử dụng #include

Trong trường hợp muốn trộn mã nguồn từ 1 file asp vào 1 file asp khác trước khi server thực thi nó, người ta dùng thẻ định hướng #include với cú pháp

```
<!--#include file="Tenfile"-->
```

Một sốứng dụng của #include như người ta thường include file chứa các hàm thư viện dùng chung cho cảứng dụng vào đầu file Asp nào cần sử dụng thư viện này, hoặc insert các file Header và Footer cho 1 trang web, insert các thành phần được sử dụng chung trong nhiều file asp như menu,... Ví dụ trong ứng dụng ASP có nhiều trang cần thao tác với database, chúng ta sẽ viết riêng module thao tác với database ra một file myConnection.asp, rồi include file này vào trang asp nào muốn thao tác với database

```
<!--#include file="myConnection.asp"-->
<%
' mã nguồn
%>
```

Lưu ý là include file được thực hiện trước khi script chạy. Vì vậy đoạn lệnh sau đây là không hợp lệ:

```
<%filename="myConnection.asp"%>
<!--#include file="<%=filename%>"-->
```

4.5. Các đối tượng căn bản

Đối tượng là một nhóm các hàm và biến. Một số đối tượng đã được xây dựng sẵn và có thể sử dụng ngay mà không cần khởi tạo: Request, Response, Session, Application, Server. Một số đối tượng cần khởi tạo nếu muốn sử dụng Dictionary, Connection, Recordset...

4.5.1. Đối tượng Request

Request và Response là 2 đối tượng được dùng nhiều nhất trong lập trình ASP, dùng trao đổi dữ liệu giữa trình duyệt và server. Request cho phép lấy về các thông tin từ client. Khi browser gửi một yêu cầu trang web lên server ta gọi là 1 request Chúng ta thường sử dụng các lệnh request sau:

4.5.1.1. Request.QueryString

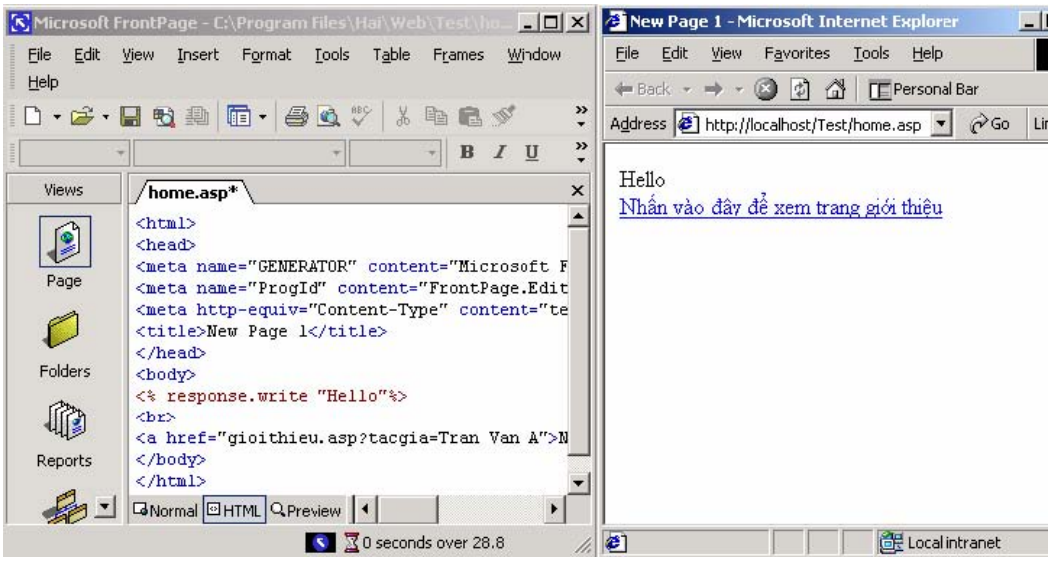
Cho phép server lấy về các giá trị được gửi từ người dùng qua URL hoặc form (method GET).

Ví dụ ở trang home.asp chúng ta đặt một dòng liên kết sang trang gioithieu.asp với thẻ sau:

Ị hấn vào đây để sang trang giới thiệu

biến “tacgia” có giá trị là “Tran Van A” được người dùng gửi tới server kèm theo URL. (người dùng có thể gõ thẳng địa chỉ “http://localhost/alias/gioithieu.asp?tacgia=Tran Van A” trên thanh Address của trình duyệt) Server muốn nhận lại giá trị này thì dùng **request.QueryString** ở trang gioithieu.asp

```
<%dim a
a=request.querystring(“tacgia”)    ‘lúc này a có gía trị là “Tran Van A”
response.write “Tác giả của trang home.asp là: ” &a
%>
```



Hình 1.7

Tương tự như vậy nếu người dùng gửi giá trị Tran Van A thông qua một biến trong form và chọn method GET

```
<form method=”get” action =”gioithieu.asp”>
<input type=”text” name=”tacgia” value=”Tran Van A”>
<input type=”submit” name=”submit” value=”Nhan vao day de sang trang
gioi thieu”>
</form>
```

4.5.1.2. Request.Form

Cho phép server lấy về các giá trị được gửi từ người dùng qua form (method POST).

Chẳng hạn file **form.asp**:

```
<form method=”POST” action =”xulyform.asp”>
<input type=”text” name=”User”>
<input type=”submit” name=”submit” value=”Nhan vao day de sang trang
gioi thieu”>
</form>
```

File xulyform.asp làm nhiệm vụ xử lý thông tin từ Form này sẽ dùng câu lệnh request.form để nhận lại thông tin người dùng đã gõ vào:

```
<%Dim x
x=Request.form(“User”) %>
response.write “Tên người dùng là: ”&x
%>
```

4.5.2. Response

Đối tượng Response dùng để gửi các đáp ứng của server cho client. Chúng ta thường dùng một số lệnh Response sau:

4.5.2.1. Response.Write

Đưa thông tin ra màn hình trang web
Ví dụ để đưa câu chào Hello ra màn hình ta dùng lệnh sau:

```
<%response.write “Hello”%>
```

Hiển thị thời gian trên server ra màn hình:

```
<%response.write now%>
```

hoặc `<%=now%>`
now là hàm lấy ngày giờ hệ thống trên server

4.5.2.2 Response.Redirect

Chuyển xử lý sang một trang Asp khác.
Ví dụ trang xulyform.asp sau khi kiểm tra form đăng nhập thấy người dùng không có quyền vào website thì nó sẽ chuyển cho file Error.asp(file này hiển thị một thông báo lỗi user không có quyền truy cập)

```
<% Response.redirect “error.asp” %>
```

4.5.2.3. Response.End

Ị gừng xử lý các Script. Dùng lệnh này khi muốn dừng xử lý ở một vị trí nào đó và bỏ qua các mã lệnh ASP ở phía sau. Đây là cách rất hay dùng trong một số tình huống, chẳng hạn như debug lỗi

4.5.3. Đối tượng Session

Session là một phiên làm việc giữa từng người dùng và web server, nó bắt đầu khi người đó lần đầu tiên truy cập tới 1 trang web trong website và kết thúc khi người đó rời khỏi website hoặc không tương tác với website trong một khoảng thời gian nhất định (time out). Ị hư vậy

tại một thời điểm một website có bao nhiêu người truy cập thì có bấy nhiêu phiên ứng với mỗi người, các phiên này độc lập nhau. Để lưu những thông tin tác dụng trong 1 phiên, người ta dùng đối tượng Session, ví dụ khi một user bắt đầu session với việc login vào hệ thống, và user đã login đó cần được hệ thống ghi nhớ trong toàn phiên làm việc (nhằm tránh việc người dùng phải đăng nhập lại mỗi khi đưa ra một request).Giá trị của biến kiểu session có phạm vi trong tất cả các trang ASP của ứng dụng, nhưng không có tác dụng đối với phiên làm việc khác. Ví dụ, sử dụng biến session sau đây đếm số lần 1 người đã truy cập vào trang web:

Home.asp

<% session("x")=session("x")+1 %>

session("x") đại diện cho số lần mà một user đã truy cập vào trang home.asp. Với 2 người dùng khác nhau thì giá trị session("x") lại khác nhau. Thật vậy , A có thể truy cập 10 lần (session("x") =10) trong khi B có thể truy cập 2 lần thôi (session("x") =2)

Server kết thúc và hủy bỏ đối tượng session khi:

-Ị gười dùng không triệu gọi các trang của ứng dụng hoặc cập nhật làm mới (refresh) lại thông tin của trang trong một thời gian nhất định. Khi một session hết thời gian hiệu lực nó sẽ được xem như hết hạn sử dụng ,tất cả các biến lưu trong session và bản thân session sẽ bị hủy bỏ. Có thể kiểm tra và tăng giảm thời gian Timeout của Session tính bằng giây như sau:

```
<%

    Session.Time
    out = 500
     %>
```

- Trang ASP gọi đến phương thức Abandon của Session .

```
<%
   Session.Abandon
%>
```

Việc khởi tạo và kết thúc 1 biến session có thể viết trong các hàm sự kiện Session_OnStart và Session_OnEnd được định nghĩa trong file global.asa

4.5.4. Đối tượng Application

Application đại diện cho toàn bộứng dụng, bao gồm tất cả các trang web trong website. Để lưu trữ những thông tin có tác dụng trong toàn ứng dụng, tức là có giá trị trong tất cả các trang asp và tất cả các phiên, người ta dùng đối tượng Application Điểm khác của biến application so với biến session là session chỉ có tác dụng đối với mỗi phiên, còn biến application có tác dụng với mọi phiên. Ví dụ, để đếm xem có bao nhiêu người truy cập vào website, chúng ta có thể dùng một biến Application. Mỗi khi một người dùng mới truy cập vào website ta tăng biến này lên 1 đơn vị để chỉ rằng đã có thêm 1 người truy cập.

<% application("x")=application("x")+1 %>

Trang home.asp muốn hiển thị số người truy cập chỉ cần in giá trị của biến này

<% response.write "Số người đã truy cập vào website là:"&application("x") %>

Với 2 phiên khác nhau thì giá trị application("x") là như nhau. Thật vậy , A và B khi truy cập

vào trang home.asp đều thấy: "Số người đã truy cập vào website là 3" (trong trường hợp application("x") =3) Việc khởi tạo và kết thúc 1 biến application có thể viết trong các hàm sự kiện Application_onStart và Application_onEnd được định nghĩa trong file global.asa

Khóa Application:

Do biến application có thể được dùng chung bởi nhiều phiên nên sẽ có trường hợp xảy ra xung đột khi có 2 phiên cùng thay đổi giá trị một biến application. Để ngăn chặn điều này chúng ta có thể dùng phương thức Application.lock để khóa biến application trước khi thay đổi nó. Sau khi sử dụng xong biến này có thể giải phóng khóa bằng phương thức application.unlock (xem ví dụ sau).

4.5.5. File Global.asa

File này là file tùy chọn chứa các khai báo đối tượng, biến có phạm vi toàn ứng dụng. Mã lệnh viết dưới dạng Script. Mỗi ứng dụng chỉ được phép có nhiều nhất 1 file Global.asa, nằm ở thư mục gốc của ứng dụng. Ị gười ta thường dùng global.asa trong trường hợp muốn có những xử lý khi một session bắt đầu hay kết thúc, một application bắt đầu hay kết thúc, thông qua các hàm sự kiện : Application_Onstart : hàm sự kiện này xảy ra khi ứng dụng asp bắt đầu hoạt động, tức là khi người dùng đầu tiên truy cập tới trang web đầu tiên khi ứng dụng hoạt động.

Session_Onstart: hàm sự kiện này xảy ra mỗi khi có một người dùng mới truy cập vào ứng dụng (bắt đầu 1 session)

Session_OnEnd: hàm sự kiện này xảy ra mỗi khi 1 người dùng kết thúc session của họ
Application_OnEnd: hàm sự kiện này xảy ra khi ứng dụng dừng.

File **Global.asa** có cấu trúc như sau:

```
<script language="vbscript" runat="server"> Sub Application_OnStart '..... End sub Sub
Application_OnEnd '..... End Sub Sub Session_OnStart '.....
Application("x")=Application("x")+1 End sub Sub Session_OnEnd '..... End Sub
</script>
```

Ví dụ sau đây sẽ đếm số người dùng hiện đang truy cập website. Số người dùng được lưu trữ trong biến Application("songuoi"). Ở bất cứđâu trong ứng dụng nếu muốn hiển thị số người dùng chúng ta chỉ việc chèn lệnh hiển thị nó:

```
<%=Application("songuoi")%>
```

Ị goài ra ứng dụng cũng cho phép đếm số lần 1 người đã truy cập website trong phiên làm việc của họ. Số lần được lưu trữ trong biến Session("solan")

Global.asa

```
<script language="vbscript" runat="server"> Sub Application_OnStart
Application("songuoi")=0 End Sub

Sub Session_OnStart Application.Lock Application("songuoi")=Application("songuoi")+1
Application.UnLock

Session("solan")=0 End
Sub

Sub Session_OnEnd Application.Lock Application("songuoi")=Application("songuoi")-1
Application.UnLock End Sub
```

```
Sub Application_OnEnd End
Sub

</script>
```

Home.asp

```
<html>
<body>
        <p>
Có <%response.write(Application("songuoi"))%>
người đang truy cập website
</p>
<%session(“solan”)= session(“solan”)+1 %>

        <p>
Bạn đã truy cập trang này <%response.write(session("solan"))%>
lần!
</p>
</body>
</html>
```

4.5.6. Đối tượng Dictionary

Đối tượng Dictionary lưu trữ thông tin theo từng cặp khóa/ giá trị. Ị ó khá giống với mảng nhưng có khả năng xử lý linh hoạt đối với những cặp dữ liệu có quan hệ kiểu từđiển (cặp khóa/ giá trị ví dụ như : mã Sinh viên/ tên Sinh viên), trong đó khóa được xem là từ cần tra và giá trị chính là nội dung của từ tra được trong từđiển. Muốn sử dụng đối tượng Dictionary chúng ta phải khởi tạo nó:

```
<%set d=server.createObject("Scripting.Dictionary")
d.add "work","Lam viec"
d.add "learn","Hoc tap" ‘tương tự như mảng nhưng mỗi phần tử là một cặp
khóa/giá trị
response.write "work nghĩa tiếng Việt là: "&d.item("work")
response.write "learn nghĩa tiếng Việt là: "&d.item("learn")
set d=nothing
%>
```

Một số ứng dụng của đối tượng này như dùng mô phỏng giỏ hàng chứa hàng hóa(shopping cart) với cặp khóa/giá trị là :ProductID/Quantity (xem chương 2), sổ địa chỉ với cặp khóa/giá trị là: CustomerỊ ame/Address.

4.5.7. Đối tượng Server

Đối tượng Server được dùng để truy cập các thuộc tính và phương thức của server .Ta thường dùng 2 lệnh sau

4.5.7.1. Server.CreateObject

khởi tạo 1 đối tượng.
Ví dụ:
Tạo một đối tượng Connection:

<%Set conn=Server.CreateObject("ADODB.Connection")%>
Tạo một đối tượng Dictionary:
<%set d=server.createObject("Scripting.Dictionary")%>

4.5.7.2. Server.Mappath

biến đường dẫn tương đối thành tuyệt đối. Ví dụ:
<%str= server.mappath("nhanvien.mdb")
Response.write str%>
Sẽ cho kết quả: "C:\WEB\nhanvien.mdb" trong trường hợp file nhanvien.mdb nằm trong thư mục C:\WEB Ta thường áp dụng server.mappath trong những trường hợp xử lý đường dẫn tương đối, ví dụ là chuỗi kết nối vào database connstr="provider=microsoft.jet.oledb.4.0; data source="&server.mappath("nhanvien.mdb")&";"

4.6. Sử dụng Database với ASP

Hầu hết các ứng dụng Web động đều lưu trữ dữ liệu trong Database. Vì vậy các thao tác kết nối vào Database, xem, thêm, sửa, xóa dữ liệu trong các bảng là phần quan trọng đối với các ngôn ngữ lập trình web như ASP. Chúng ta sẽ học các kỹ thuật sử dụng Asp để thao tác với dữ liệu trong Database thông qua kiến trúc ADO.

4.6.1. Các cú pháp căn bản để truy xuất dữ liệu từ DB

Để thao tác với dữ liệu trong các bảng của DB, có 4 thao tác chính với câu lệnh SQL tương ứng như sau:

(Lấy ví dụ với một Database cụ thể Quanlyhocvien.mdb, trong đó có một bảng HosoHocVien (MaHV:text, Ten: text)

4.6.1.1. Lựa chọn

Lấy tất cả các bản ghi trong bảng:
*"Select * from HosoHocVien"*
Ị ếu lựa chọn có điều kiện:
*"Select * from HosoHocVien where MaHV='10' "*
Ị ếu chỉ lựa chọn một số trường trong bảng:
"Select Ten from HosoHocVien where MaHV='10' "

4.6.1.2. Thêm dữ liệu vào bảng

"Insert into HosoHocVien values ('001','Tran Van A') "

4.6.1.3. Sửa dữ liệu

"Update HosoHocVien set Ten='Tran Van B' where MaHV='001' "

4.6.1.4. Xoá dữ liệu

"Delete from HosoHocVien where MaHV='001' "

Chúng ta có thể sử dụng các lệnh SQL phức tạp hơn để có được kết quả mong muốn như sử dụng các lệnh join, order by, group by, having...

4.6.2. Đối tượng Connection

Đối tượng Connection cho phép tạo
kết nối đến một DB. Các bước sử
dụng Connection: -Khai báo đối tượng

Connection -Khởi tạo -Tạo chuỗi kết nối -Mở Connection với chuỗi kết nối trên

-Sử dụng Connection -Đóng và Hủy Connection Ví dụ sau đây kết nối đến database Access QuanlyHocvien.mdb (database này nằm trong cùng thư mục với file Asp)

<% dim conn 'khai báo set conn=server.createObject("ADODB.connection") 'khởi tạo stringconn="provider=microsoft.jet.OLEDB.4.0;data source="&server.mappath("QuanlyHocVien.mdb")&";" 'chuỗi kết nối conn.open stringconn 'mở connection ' các thao tác với DB sử dụng connection này '..... 'conn.close 'đóng connection Set conn=nothing 'hủy connection %>

(chuỗi "stringconn=…" viết trên 1 dòng, trong đó: "… data source = …"chú ý có một dấu cách giữa "data" và "source", chuỗi này chỉđúng với Access)

4.6.3. Đối tượng Recordset

Đối tượng Recordset thường dùng để xem, thêm, sửa, xóa các bản ghi trong bảng dữ liệu của Database. Ị ó trỏ đến tập hợp các bản ghi là kết quả trả về từ câu lệnh select Các bước sử dụng đối tượng Recordset :

-Khai báo đối tượng Recorset

-Khởi tạo

-Tạo sql query

-Mở Recordset với chuỗi sql query và connection đã mở

-Sử dụng Recordset

-Đóng và Hủy Recordset

Ví dụ sau đây cho phép lấy các bản ghi trong bảng và hiển thị ra ngoài trang web.

*<%Dim rs ' khai báo Recordset set rs=server.createObject("ADODB.Recordset") 'Khởi tạo SQLstring="select * from HosoHocVien" 'SQL query rs.open SQLstring ,conn 'Mở Recordset ' dùng vòng lặp để hiển thị toàn bộ các bản ghi ra màn hình do while not rs.EOF*
response.write RS("MaHV")
response.write RS("Ten")
*response.write "
"*
rs.movenext 'dịch con trỏ rs tới bản ghi tiếp theo

loop rs.close 'đóng recordset set rs=nothing 'hủy recordset %>

Chúng ta có thể kết hợp giữa script và thẻ html để dữ liệu được hiển thị ra ngoài trang web với giao diện theo ý muốn :

<table border="1">
<tr>
<td>MA HOC VIEN</td>
<td>TEN</td>

</tr> <%do while not rs.eof%>

```
    <tr>
      <td ><%=rs("MaHV")%></td>
      <td ><%=rs("Ten")%></td>

    </tr> <%rs.movenext loop rs.close %>
  </table>
```

Sau đây là một ví dụ hoàn chỉnh liệt kê các user trong bảng tblUser ra trang web:

Connection.asp

```
<%
dim conn
Sub openConn()
set conn=server.createobject("adodb.connection")
connstr="provider=microsoft.jet.oledb.4.0; data
source="&server.mappath("myDB.mdb")&";"
conn.open connstr
End Sub


Sub destroyConn()
conn.close
set conn=nothing
End Sub
%>
```

ListUser.asp

```
<!--#include file ="Connection.asp"-->
<%openConn
set rs = server.createobject("ADODB.Recordset")
rs.open "select * from tblUser", conn%>
<table border="1" width="200">

        <tr><td>ID</td><td>Username</td><td>Address</td> <% do while not rs.EOF
<tr> <td><%=rs("id")%></td> <td><%=rs("username")%></td>
<td><%=rs("address")%></td> </tr> <% rs.movenext loop rs.close destroyConn%>
</table>
```

4.6.3.1. Thêm sửa xóa dữ liệu trong DB

Với một connection đã mở chúng ta có thể dùng nó để thực thi câu lệnh SQL dạng insert, update, delete:
Thêm dữ liệu:

```
<%Conn.execute "Insert into HosoHocvien values('001','Tran Van A')"%>
```

Sửa dữ liệu:

```
<%Conn.execute "Update HosoHocVien set Ten='Tran Van B' where MaHV='001' "%>
```

Xoá dữ liệu:

```
<%Conn.execute "Delete from HosoHocVien where MaHV='001' " %>
```

Ị goài ra chúng ta có thể dùng Recordset để thêm, sửa, xóa dữ liệu trong database bằng cách duyệt qua tập hợp các bản ghi trong bảng

Thêm dữ liệu:

*<%Dim RS set rs=server.createObject("ADODB.recordset") SQLstring="select * from HosoHocVien" rs.open SQLstring ,conn,3,2 'rs.open SQLstring ,conn,adOpenStatic,adLockPessimistic rs.addnew 'Thêm một bản ghi rs("MaHV")="001" ' gán giá trị cho các trường của bản ghi rs("Ten")="Tran Van A" rs.update ' Xác nhận thêm xong rs.close 'đóng recordset %>*
Sửa:

*<% set rs=server.createObject("ADODB.recordset") 'Khởi tạo SQLString="select * from HosoHocVien where ma='001' " ' lấy ra bản ghi cần sửa rs.open SQLString ,conn,3,2 rs("Ten")="Tran Van B" 'sửa lại giá trị trường "Ten" rs.update ' xác nhận sửa xong rs.close 'đóng recordset %>*
Xóa:

*<% set rs=server.createObject("ADODB.recordset") 'Khởi tạo SQLString="select * from HosoHocVien where MaHV='001' " 'Câu lệnh SQL lấy ra đúng bản ghi cần xóa rs.open SQLString ,conn,3,2 rs.delete 'xóa bản ghi này rs.close 'đóng recordset %>*

4.6.3.2. Phân trang

Trong nhiều trường hợp do kết quả câu lệnh "select ...." trả về quá nhiều bản ghi, nếu chúng ta hiển thị tất cả trên cùng 1 trang web thì sẽ bất tiện trong việc đọc chúng, khi đó người ta tiến hành phân nó ra để hiển thị thành nhiều trang, đây gọi là kỹ thuật phân trang. So với cách đọc và hiển thị dữ liệu

thông thường, thì phân trang đòi hỏi phải thiết lập thêm một số thuộc tính: -Số bản ghi cần hiển thị trên một trang RS.PageSize -Trang nào đang được hiển thị: RS.AbsolutePage, -Khi mở Recordset đòi hỏi phải thêm các tham số CursorType và LockType :rs.open SQLstring ,conn,3,3 -Vòng lặp hiển thị dữ liệu cần có cơ chế đảm bảo nó chỉ chạy đúng số bản ghi trên một trang (rs.pagesize) là phải thoát khỏi vòng lặp. Ví dụ để hiển thị bảng HosoHocVien với yêu cầu chỉ hiển thị 4 bản ghi/1 trang:

Home.asp

<% dim x 'biến này dùng để xác định xem cần hiển thị trang nào
x=request.querystring("PageNumber") 'nhận lại PageNumber khi người dùng nhấn vào các nút "Trước" và "Tiếp" if x="" then 'đầu tiên sẽ hiển thị trang 1 x=1 end if

dim conn set conn=server.createObject("ADODB.connection")
*stringconn="provider=microsoft.jet.OLEDB.4.0;data source="&server.mappath("QuanlyHocVien.mdb")&";" conn.open stringconn Dim RS set rs=server.createObject("ADODB.recordset") SQLstring="select * from HosoHocVien" rs.pagesize= 4 'chỉ hiển thị 4 bản ghi/1 trang rs.open SQLstring ,conn,3,3 rs.AbsolutePage=x 'trang cần hiển thị dem=0 'biến này để đảm bảo vòng lặp chỉ thực hiện tối đa 4 lần lặp do while not rs.EOF and dem<rs.pagesize*
response.write RS("MaHV")
response.write RS("Ten")
*response.write "
"*

dem=dem+1

```
rs.movenext
loop
%>
<% 'Hiển thị nút "Trước"
if x>1 then %>
<a href="home.asp?pageNumber=<%=x-1%>">Trước</a>
<%end if%>
<% 'Hiển thị nút "Tiếp"
if not RS.EOF then %>
<a href="home.asp?pageNumber=<%=x+1%>">Tiếp</a>

<%end if rs.close 'đóng recordset %>
```

4.6.4. Tìm kiếm dữ liệu trong database

Để tìm kiếm dữ liệu trong bảng của Database chúng ta dựa vào câu lệnh SQL:
*"select * from Tenbang where Tencot like '%giatri%' "*
Ví dụđoạn chương trình sau cho phép hiển thị những Sinh Viên trong bảng "HosoHV" của DB "Sinhvien.mdb" có tên được tìm kiếm bởi từ khoá "Anh" (Ví dụ : Tuấn Anh, Vân Anh, Việt Anh...)

```
<%
set conn=server.createobject("adodb.connection")
connstring="provider=microsoft.jet.oledb.4.0;data
source="&server.mappath("sinhvien.mdb")&";"
conn.open connstring
set rs=server.createobject("adodb.recordset")
rs.open "select * from HosoSV where ten like '%Anh%' ",conn
do while not rs.eof
response.write rs("MaSV")
response.write " "
response.write rs("Ten")
response.write " "
response.write rs("Lop")
response.write "<BR>"
rs.movenext
loop
rs.close
%>
```

Thông thường người sử dụng nhập từ khoá cần tìm kiếm vào một trường của form. như vậy ta chỉ việc dùng lệnh request.form để lấy lại từ khoá cần tìm kiếm và đưa vào câu lệnh SQL ở trên. Chẳng hạn người sử dụng nhập từ khoá cần tìm vào trường "Ten" trong form thì chúng ta sẽ mở bảng bằng câu lệnh SQL sau:

```
<% ten=request.form("Ten")
'validate
rs.open "select * from HosoSV where Ten like '%"&ten&"%' ",conn
'... %>
```

Ị ếu không tìm thấy bản ghi nào thì giá trị rs.EOF sẽ true.

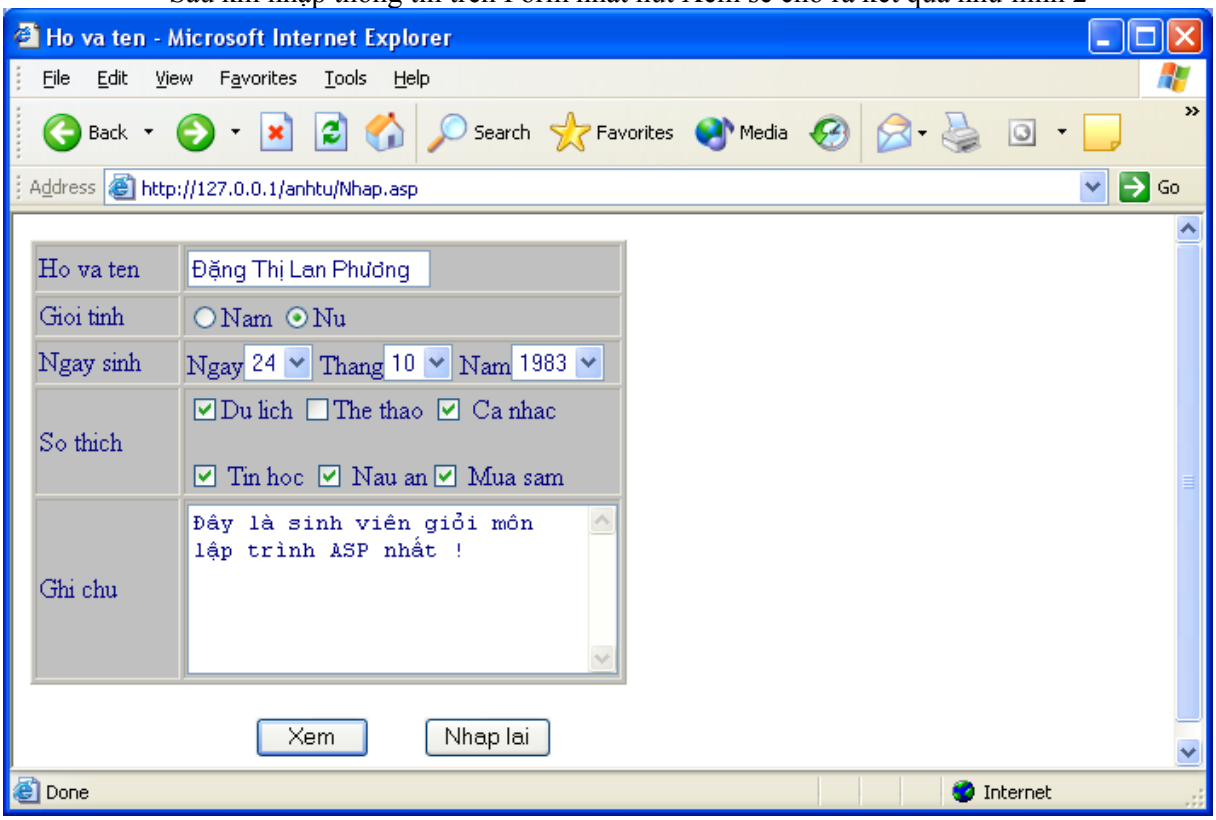
```
<%
If rs.eof then response.write "Không tìm thấy kết quả nào" %>
```

PHỤ LỤC

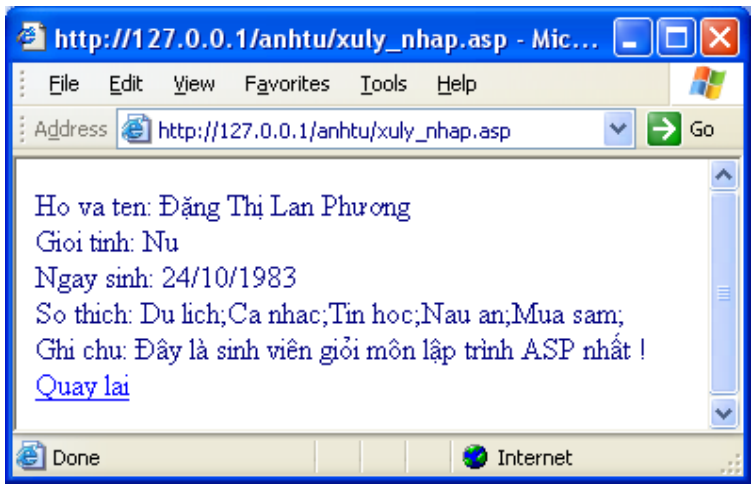
Phụ lục 1: Bài tập truy nhập các điều khiển trên Form

Yêu cầu: - Thiết kế Form như hình 1 dưới đây

- Sau khi nhập thông tin trên Form nhất nút Xem sẽ cho ra kết quả như hình 2



Hình 1



Hình 2

File Nhap.asp

```
<html>
<head>
<meta http-equiv="Content-Language" content="en-us">
<meta http-equiv="Content-Type" content="text/html; charset=windows-1252">
<title>Ho va ten</title>
</head>
```

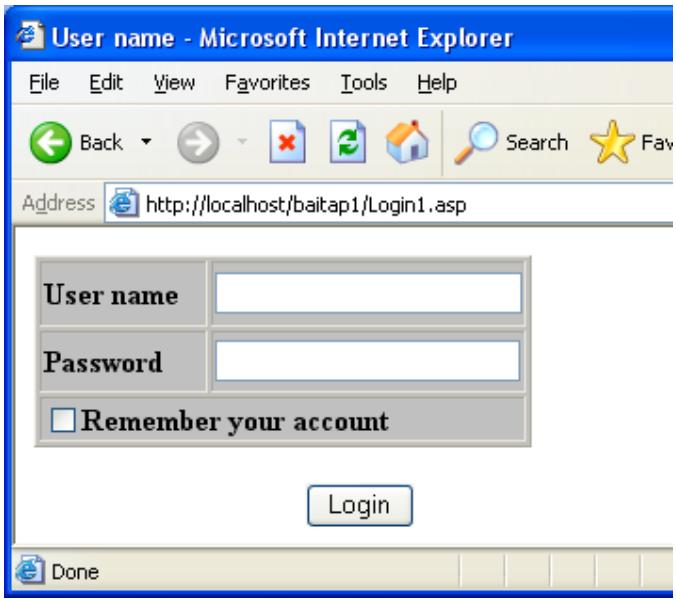
```
<body>
<form method="POST" action="xuly_nhap.asp">
	<table border="1" width="53%" id="table1" bgcolor="#C0C0C0">
		<tr>	<td width="107">Ho va ten</td>
			<td><input type="text" name="txthoten" size="20"></td> </tr>
		<tr>
			<td width="107">Gioi tinh</td>
			<td><input type="radio" value="Ị am" checked name="rdogt">Ị am
			<input type="radio" name="rdogt" value="Ị u">Ị u</td>
		</tr>
		<tr>
			<td width="107">Ị gay sinh</td>
			<td>
			Ị gay<select size="1" name="optỊ gay">
			<%for i=1 to 31%>
			<option value=<%=i%>><%=i%></option>
			<%Ị ext%>
			</select>
			Thang<select size="1" name="optthang">
			<%for i=1 to 12%>
			<option value=<%=i%>><%=i%></option>
			<%Ị ext%>
			</select>
			Ị am<select size="1" name="optỊ am">
			<%for i=1970 to year(date())%>
			<option value=<%=i%>><%=i%></option>
			<%Ị ext%>
			</select></td>
		</tr>
		<tr>
			<td width="107" height="53">So thich</td>
			<td height="53">
		<input type="checkbox" name="chksothich" value="Du lich">Du lich
		<input type="checkbox" name="chksothich" value="The thao">The thao
		<input type="checkbox" name="chksothich" value="Ca nhac"> Ca nhac
		<p><input type="checkbox" name="chksothich" value="Tin hoc"> Tin hoc
		<input type="checkbox" name="chksothich" value="Ị au an"> Ị au an<input
type="checkbox" name="chksothich" value="Mua sam"> Mua sam</td>
		</tr>
		<tr>
		<td width="107">Ghi chu</td>
		<td><textarea rows="6" name="txtaGhichu" cols="29"></textarea></td>
		</tr>
	</table>
	<input type="submit" value="   Xem   " name="cmdXem">
	<input type="reset" value="Ị hap lai" name="cmdnhaplai">
</form>
</body>
</html>
```

File Xuly_nhap.asp

```
<%
response.write "Ho va ten: " & request.form("txthoten") & "<br>"
response.write "Gioi tinh: " & request.form("rdogt") & "<br>"
ngaysinh=request.form("optngay") & "/" & request.form("optthang") & "/" &
                                                        request.form("optnam")
response.write "Ị gay sinh: " & Ị gaysinh & "<br>"
response.write "So thich: " & & request.form("chksothich") & "<br>"
response.write "Ghi chu: " & request.form("txtaGhichu") & "<br>"
%>
<a href=nhap.asp>Quay lai</a>
```

Phụ lục 2: Thiết kế form đăng nhập cho phép lưu lại tên tài khoản và mật khẩu

1. Yêu cầu: - Thiết kế Form đăng nhập (hình trái), cho phép lưu lại tài khoản. Ị ếu đăng nhập thành công thì mở ra trang Home.asp (hình phải)



File Login.asp

```
<html>
<head>
<meta http-equiv="Content-Language" content="en-us">
<meta http-equiv="Content-Type" content="text/html; charset=windows-1252">
<title>User name</title>
</head>

<body>
<%
'Kiem tra neu da co Cookies thi dang nhap luon vao trang Home.asp
if request.cookies("Acc")("user")="Admin" and request.cookies("Acc")("pass")="12345" then
        response.redirect("Home.asp")
```

```
else
'Ị eu chua co Cookies thi cho hien giao dien dang nhap
%>
<form method="POST" action="login.asp">
	<table border="1" width="56%" id="table1" bgcolor="#C0C0C0">
	  <tr>
		<td width="129" height="35"><b>User name</b></td>
		<td height="35">
		<input name="txtUser" size="20"></td>
	  </tr>
	  <tr>
		<td width="129" height="33"><b>Password</b></td>
		<td height="33">
		<input type="password" name="txtPass" size="19"></td>
	  </tr>
	  <tr>
		<td colspan="2">
		<input type="checkbox" name="chkSave" value="OỊ ">
		<b>Remember your account</b>
		</td>
	  </tr>
	  </table>
	  <input type="submit" value=" Login " name="CmdLogin">
</form>
<%
'Ị eu tai khoan dang nhap chinh xac thi cho vao` trang Home.asp
if request.form("txtuser")="Admin" and request.form("txtpass")="12345" then
	'Ị eu check luu lai thong tin ve tai khoan thi tao Cookies de luu
	if request.form("chkSave")<>"" then
		'Tao Cookies luu lai tai` khoan
		response.cookies("Acc")("User")="Admin"
		response.cookies("Acc")("pass")="12345"
		'Dat thoi han ton tai cua Cookies la 5 ngay ke tu ngay hien tai
		response.cookies("Acc").expires=date()+5
	end if
	response.redirect("Home.asp")
else
'Tai khoan dang nhap khong chinh xac thi bao loi
	if request.form("cmdLogin")<>"" then
		response.write "Login error.....!"
	end if
end if

end if
%>
</body>
</html>
```

File Home.asp

```
<html>
<head>
<meta http-equiv="Content-Language" content="en-us">
```

```
<meta http-equiv="Content-Type" content="text/html; charset=windows-1252">
<title>Login successfull</title>
</head>
<body>
<p><b>Login successful !</b></p>
<form method="POST" action="Home.asp">
        <p>Do you want <input type="submit" value="Logout" name="cmdLogout"></p>
</form>
<%

'Ị han Logout de huy Cookies
if request.form("cmdLogout")<>"" then
        response.cookies("Acc")("user")=""
        response.cookies("Acc")("pass")=""
        'Khong dung` Expires thi Cookies se bi huy
        'ngay sau khi dong IE hien hanh`

        response.redirect("Login1.asp")
end if
%>
</body>
</html>
```

2. Nâng cấp chương trình:

Yêu cầu kiểm tra tài khoản đang nhập có tồn tại trong CSDL không. Làm theo các bước sau:

(1) Tạo CSDL HOSO.MDB và bảng Account(ID(Autonumber), User(Text), Pass(Text))

(2) Viết file Connection.inc kết nối tới CSDL

Connection.inc

```
<%
Set Conn=Server.CreateObject("ADODB.Connection")
FilePath= Server.Mappath("Hoso.mdb")
StrConn="Provider=Microsoft.Jet.OLEDB.4.0; Data source=" & FilePath
Conn.Open StrConn
Set RS=Server.CreateObject("ADODB.Recordset")
%>
```

(3) Chèn đoạn mã sau đây vào trước thẻ <HTML> trong file Login.asp

```
<!--#include file=Connection.inc-->
<%
ID=request.cookies("Acc")("user")
PWD=request.cookies("Acc")("Pass")
'Mo bang du lieu Account va chon ra dung TK da luu trong Cookies
StrSQL="Select * from Account Where (User='" & ID & "')AỊ D(Pass='" & PWD & "')"
RS.Open StrSQL,Conn
%>
```

(4) Sửa lại dòng gạch chân sau thẻ <BODY> như sau:

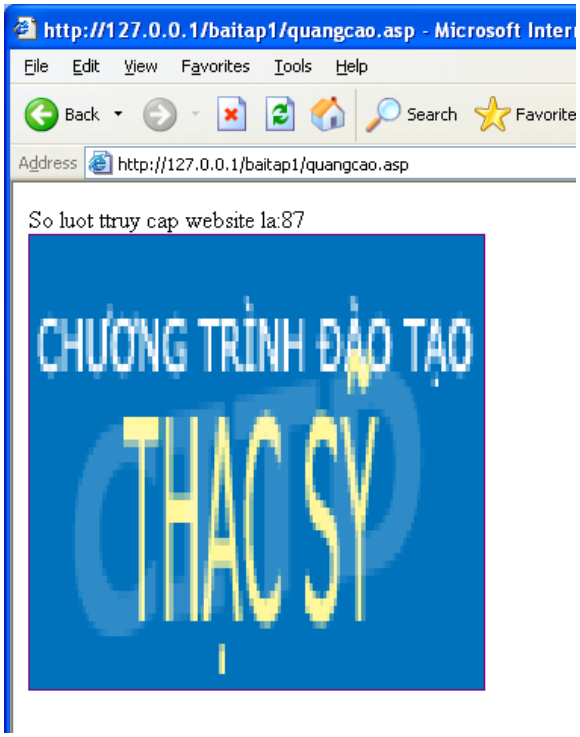
```
<%
'Kiem tra neu da co Cookies thi dang nhap luon vao trang Home.asp
If not rs.eof then
        response.redirect("Home.asp")
else
'Ị eu chua co Cookies thi cho hien giao dien dang nhap
%>
```

****

Phụ lục 3: Hiển thị ảnh quảng cáo và số lượt truy cập

File Quangcao.txt

```
REDIRECT /login.asp
WIDTH 300
HEIGHT 300
BORDER 1
*
1.gif
http://home.vnn.vn
Trang chu Viet Nam
60
2.gif
http://vbcode.com
Ma nguon VB
40
3.gif
http://www.vnexprepss.net
Trang tin tuc
20
```



Yêu cầu cho phép:

- Đếm số lượt người đã truy cập
- Hiển thị ảnh quảng cáo ngẫu nhiên

Yêu cầu: Sử dụng file Global.asa

(Ghi chú: Chuẩn bị sẵn 3 ảnh: 1.gif, 2.gif, 3.gif)

File Global.asa

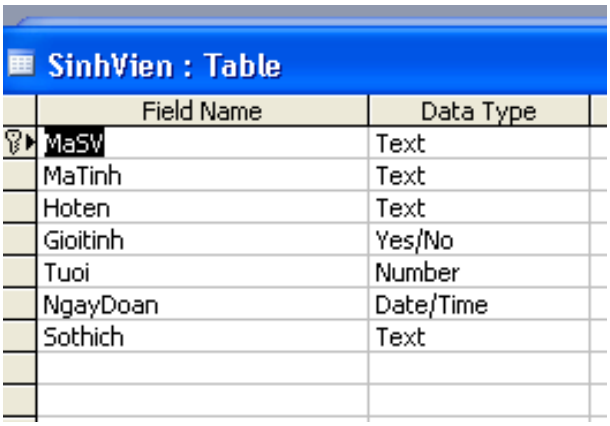
```
<OBJECT ID="Dem" RUNAT="Server" SCOPE="Application"
PROGID="MSWC.Counters">
</OBJECT>
<OBJECT ID="Pic" RUNAT="Server" SCOPE="Application"
PROGID="MSWC.Adrotator">
</OBJECT>
```

File Quangcao.asp

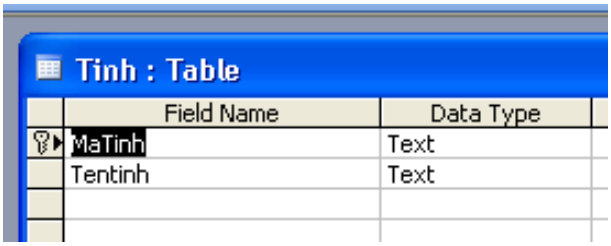
```
<%
response.write "So luot ttruy cap website la:"
%>
<%
dem.Increment("k")
response.write dem.get("k")
%>
<br>
<%
=pic.GetAdvertisement("quangcao.txt")
'noi dung file quangcao.txt nhu tren
%>
```

Phụ lục 4: Bài tập ADO, Phân trang với Recordset

1. Tạo CSDL HOSO.MDB và thiết kế các bảng dữ liệu có cấu trúc như sau:

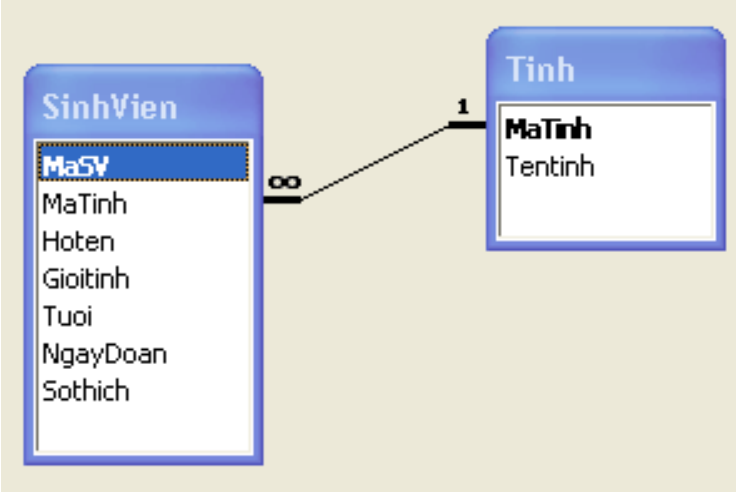


Bảng **Sinhvien**

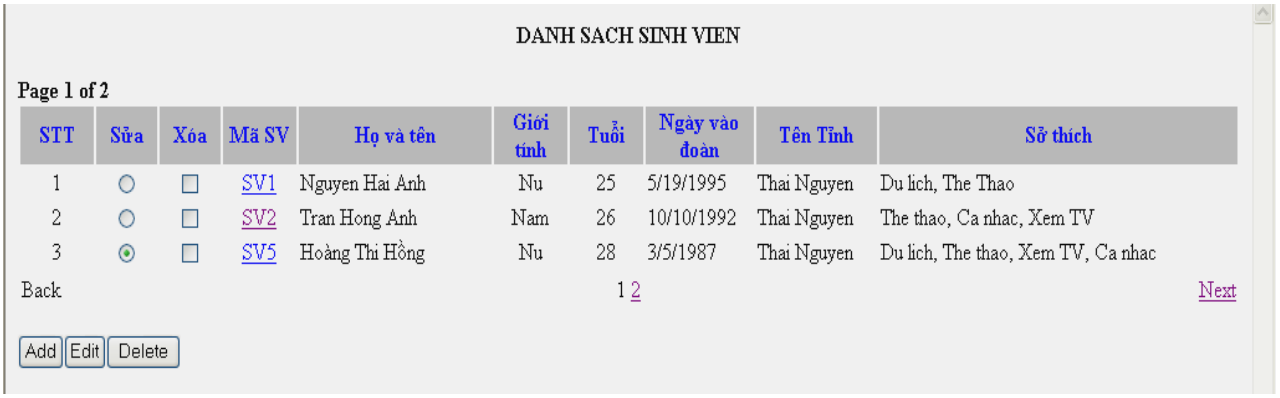


Bảng **Tinh**

Tạo quan hệ giữa 2 bảng như sau:



2. Thiết kế trang DEFAULT.ASP cho phép hiển thị các thông tin về sinh viên như sau:

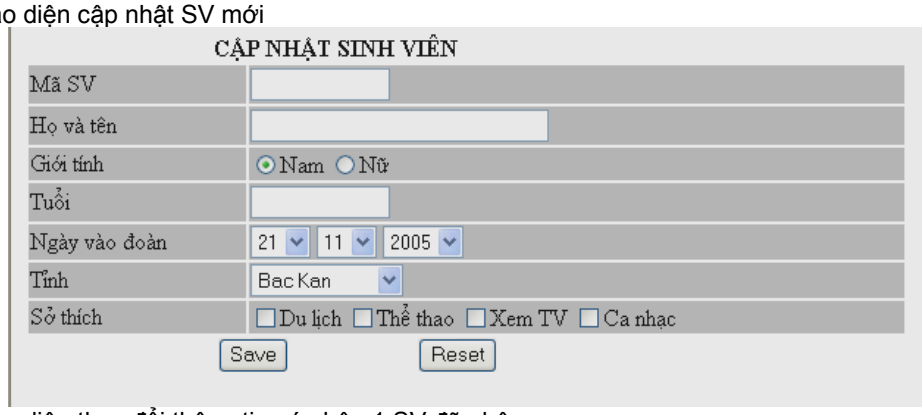


Chú ý:

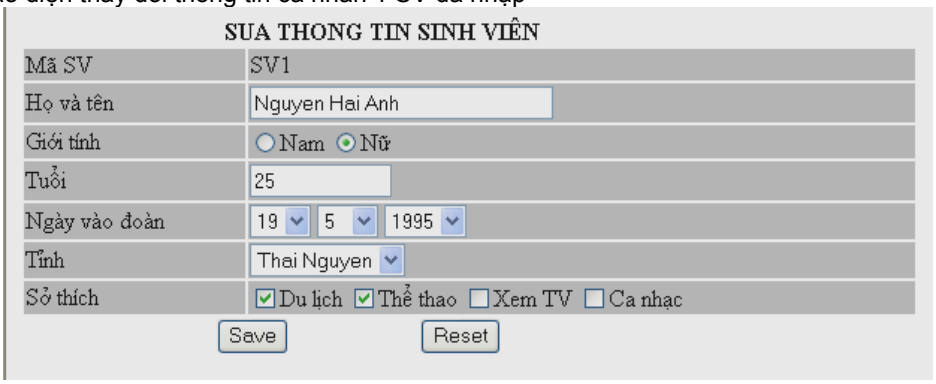
- *Sử dụng kỹ thuật phân trang*
- *Các chức năng ADD, EDIT, DELETE để thực hiên các chức năng thêm mới, sửa, xóa các bản ghi*
- *Nhấn vào Mã SV cho phép liên kết đến trang xem tổng hợp thông tin cá nhân về SV đó (Ví dụ: Profile.asp)*

3. Yêu cầu thiết giao diện các trang thêm mới và sửa như sau:

- Giao diện cập nhật SV mới



- Giao diện thay đổi thông tin cá nhân 1 SV đã nhập



MÃ NGUỒN MẪU

File Connection.inc

```
<%
FilePath=Server.Mappath("Hoso.mdb")
Set Conn=Server.CreateObject("ADODB.COnnection")
StrConn="Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" & FilePath
Conn.Open strConn
Set RS=Server.CreateObject("ADODB.Recordset")
%>
```

File Default.asp

```
<!--#include file=Connection.inc-->
<%
'Su dung Query trong Access de tao xau truy van nhu duoi day
StrSQL="SELECT SinhVien.*, [Tinh].[Tentinh] FROM Tinh INNER JOIN SinhVien ON
[Tinh].[MaTinh]=[SinhVien].[MaTinh];"

'Dat thuoc tinh cho RS co kha nang phan trang
RS.CursorLocation=3
RS.Pagesize=3
RS.Open StrSQL,Conn,0,1

'Xac dinh trang hien thoi
Session("CurrenPage")=1
If request("PageID")<>"" then
        Session("CurrenPage")=Cint(request("PageID"))
End if

If not(RS.eof or RS.bof) then
        RS.Absolutepage=Session("CurrenPage")
End if
%>

<html>
```

```
<head>
<meta http-equiv="Content-Language" content="en-us">
<meta http-equiv="Content-Type" content="text/html; charset=windows-1252">
<meta name="GENERATOR" content="Microsoft FrontPage 6.0">
<meta name="ProgId" content="FrontPage.Editor.Document">
<title>Phân trang với RS</title>
</head>

<body>
<p align="center"><b>DANH SACH SINH VIEN </b></p>
<form method="POST" action="Xuly_Default.asp">
<b>Page <%=Session("CurrenPage")%> of <%=RS.PageCount%></b>
  <table border="0" width="100%">
    <tr>
      <td>STT</td>
      <td>Sửa</td>
      <td>Xóa</td>
      <td>Mã SV</td>
      <td>Họ và tên</td>
      <td>Giới tính</td>
      <td>Tuổi</td>
      <td>Ngày vào đoàn</td>
      <td>Tên Tỉnh</td>
      <td>Sở thích</td>
    </tr>
<%
i=RS.Pagesize*(Session("CurrenPage")-1)
While not RS.eof and RS.AbsolutePage=Session("CurrenPage")
          i=i+1
%>
    <tr>
      <td><%=i%></td>
      <td><input type="radio" value=<%=RS.Fields("MaSV")%> checked name="RdoSua"></td>
      <td><input type="checkbox" name="chkXoa" value=<%=RS.Fields("MaSV")%>></td>
      <td>
          <a href=Profile.asp?MaSV_ID=<%=RS.Fields("MaSV")%>><%=RS.Fields("MaSV")%></a>
      </td>
      <td><%=RS.Fields("Hoten")%></td>
      <td width="6%" align="center">
      <%
      If RS.Fields("Gioitinh") then
                  Response.Write("Nam")
          Else
                  Response.Write("Nu")
      End if
      %>
      </td>
      <td width="6%" align="center"><%=RS.Fields("Tuoi")%></td>
      <td width="9%"><%=RS.Fields("NgayDoan")%></td>
      <td width="10%"><%=RS.Fields("Tentinh")%></td>
      <td width="29%"><%=RS.Fields("Sothich")%></td>
    </tr>
<%
RS.MoveNext
Wend
%>
  </table>
  <table border="0" width="100%">
    <tr>
      <td width="23%">
      <%If Session("CurrenPage")=1 then
                  Response.write("Back")
      Else%>
```

```
    <a href=Default.asp?PageID=<%=Session("CurrenPage")-1%>>Back</a>
    <%End if%>
    </td>
    <td width="54%">
      <p align="center">
      <%
      k1=Session("CurrenPage")-1
      k2=Session("CurrenPage")+1
      If k1<1 then k1=1
      If k2>RS.PageCount then k2=RS.PageCount
      For i=k1 to k2
          Response.write ("       ")
          If i=Session("CurrenPage") then
                  Response.write(i)
          Else%>
                      <a href=Default.asp?PageID=<%=i%>><%=i%></a>
                  <%End if
      Next
      %>
    </td>
    <td width="23%">
      <p align="right">
    <%If Session("CurrenPage")=RS.PageCount then
                          Response.write("Next")
         Else%>
                      <a href=Default.asp?PageID=<%=Session("CurrenPage")+1%>>Next</a>
    <%End if
    RS.Close
    %>
    </td>
   </tr>
  </table>
<input type="submit" value="Add" name="cmdAdd">
<input type="submit" value="Edit" name="cmdEdit">
<input type="submit" value="Delete" name="cmdDelete">
</form>
</body>
</html>
```

File Xuly_Default.asp

```
<%
if request.form("cmdAdd")<>"" then
          response.redirect("Them.asp")
end if
if request.form("cmdEdit")<>"" then
          session("MaSV")=request.form("rdoSua")
          response.redirect("Sua.asp")
end if
if request.form("cmdDelete")<>"" then
%>
<!--#include file=xoa.asp-->
<%
end if
%>
```

File Them.asp

```
<!--#include file=Connection.inc-->
<%
StrSQL="Select * from Tinh Order By Tentinh"
RS.Open StrSQL,Conn
%>
<html>
<head>
<meta http-equiv="Content-Language" content="en-us">
<meta http-equiv="Content-Type" content="text/html; charset=windows-1252">
```

```
<meta name="GENERATOR" content="Microsoft FrontPage 6.0">
<meta name="ProgId" content="FrontPage.Editor.Document">
<title>Thêm mới SV</title>
</head>

<body>

<form method="POST" action="Xuly_them.asp">
          <b>CẬP NHẬT SINH VIÊN</b>
<table border="0" width="61%">
   <tr><td>Mã SV</td><input type="text" name="txtMaSV" size="12"></td></tr>
   <tr><td>HỌ và tên</td><td><input type="text" name="txtHoten" size="30"></td></tr>
   <tr>
    <td>Giới tính</td>
    <td><input type="radio" value="Nam" checked name="rdoGT">Nam
          <input type="radio" name="rdoGT" value="Nu">Nữ
    </td>
   </tr> <tr><td>Tuổi</td><td><input type="text" name="txtTuoi" size="12"></td> </tr>
   <tr> <td>Ngày vào đoàn</td>
         <td>
    <select size="1" name="optNgay">
    <%For i=1 to 31%>
       <option <%if i=day(date()) then%> selected <%End if%> value=<%=i%>><%=i%></option>
    <%Next%>
    </select>
    <select size="1" name="optThang">
    <%For i=1 to 12%>
     <option <%if i=month(date()) then%> selected <%End if%> value=<%=i%>><%=i%></option>
    <%Next%>
    </select>
    <select size="1" name="optNam">
    <%For i=1970 to year(date())%>
       <option selected value=<%=i%>><%=i%></option>
    <%Next%>
    </select></td>
   </tr>
   <tr>
    <td>Tỉnh</td>
    <td>
   <select size="1" name="optTinh">
    <%
    While not Rs.eof
    %>
       <option value=<%=RS.Fields("Matinh")%>><%=RS.Fields("TenTinh")%></option>
    <%
    RS.MoveNext
    Wend
    RS.Close
    %>
    </select></td>
   </tr>
   <tr>
    <td>Sở thích</td>
    <td><input type="checkbox" name="chkST" value="Du lich">Du lịch
      <input type="checkbox" name="chkST" value="The thao">Thể thao
      <input type="checkbox" name="chkST" value="Xem TV">Xem TV
      <input type="checkbox" name="chkST" value="Ca nhac">Ca nhạc
    </td>
   </tr>
   <tr> <td><input type="submit" value="Save" name="CmdSave">
      <input type="reset" value="Reset" name="cmdReset"></td>
   </tr>
  </table>
```

```
</form>
</body>
</html>
```

File Xuly_them.asp

```
<!--#include file=Connection.inc-->
<%
StrSQL="Select * from Sinhvien"
RS.Open StrSQL,Conn,1,3
RS.AddNew
RS.Fields("MaSV")=request.form("txtMaSV")
RS.Fields("Hoten")=request.form("txtHoten")
RS.Fields("MaSV")=request.form("txtMaSV")
If Request.form("rdoGT")="Nam" then
        RS.Fields("Gioitinh")=True
else
        RS.Fields("Gioitinh")=False
End if
RS.Fields("Tuoi")=request.form("txtTuoi")
ND=request.form("optThang")&"/"&request.form("optNgay")&"/"&request.form("optNam")
RS.Fields("NgayDoan")=Cdate(ND)
RS.Fields("Matinh")=request.form("optTinh")
RS.Fields("Sothich")=request.form("chkST")
RS.Update
RS.Close
response.redirect("Default.asp")
%>
```

File Sua.asp

```
<!--#include file=Connection.inc-->
<%
StrSQL="Select * from Tinh Order By Tentinh"
RS.Open StrSQL,Conn
StrSQL1="Select * from Sinhvien Where MaSV='" & session("MaSV") & "'"
Set RS1=Server.CreateObject("ADODB.Recordset")

RS1.Open StrSQL1,Conn
%>
<html>
<head>
<meta http-equiv="Content-Language" content="en-us">
<meta http-equiv="Content-Type" content="text/html; charset=windows-1252">
<meta name="GENERATOR" content="Microsoft FrontPage 6.0">
<meta name="ProgId" content="FrontPage.Editor.Document">
<title>Sửa thông tin SV</title>
</head>

<body>

<form method="POST" action="Xuly_Sua.asp">
<b>SUA THÔNG TIN SINH VIÊN</b>
<table border="0" width="61%">
   <tr><td>Mã SV</td> <td><%=RS1.Fields("MaSV")%></td> </tr>
   <tr>
     <td>Họ và tên</td>
     <td>
          <input type="text" name="txtHoten" size="30" value=<%="""&RS1.Fields("Hoten")&"""%>>
      </td>
   </tr>
   <tr>
     <td>Giới tính</td>
     <td>
<input type="radio" value="Nam" <%if rs1.fields("Gioitinh") then%> checked <%End if%>
name="rdoGT">Nam
```

```
<input type="radio" name="rdoGT" <%if Not rs1.fields("Gioitinh") then%> checked <%End if%>
value="Nu">Nữ
    </td>
   </tr>
   <tr>
    <td>Tuổi</td>
    <td><input type="text" name="txtTuoi" size="12" value=<%=rs1.fields("Tuoi")%>></td>
   </tr>
   <tr>
    <td>Ngày vào đoàn</td>
    <td>
    <select size="1" name="optNgay">
    <%For i=1 to 31%>
   <option <%if i=day(rs1.fields("Ngaydoan")) then%> selected <%End if%>
value=<%=i%>><%=i%></option>
    <%Next%>
    </select>
    <select size="1" name="optThang">
    <%For i=1 to 12%>
   <option <%if i=Month(rs1.fields("Ngaydoan")) then%> selected <%End if%>
value=<%=i%>><%=i%></option>
    <%Next%>
    </select>
    <select size="1" name="optNam">
    <%For i=1970 to year(date())%>
   <option <%if i=year(rs1.fields("Ngaydoan")) then%> selected <%End if%>
value=<%=i%>><%=i%></option>
    <%Next%>
    </select></td>
   </tr>
   <tr><td>Tỉnh</td>
   <td>
    <select size="1" name="optTinh">
    <%
    While not Rs.eof
    %>
       <option <%if RS.Fields("Matinh")=RS1.Fields("Matinh") then%> Selected <%End if%>
value=<%=RS.Fields("Matinh")%>><%=RS.Fields("TenTinh")%></option>
    <%
    RS.MoveNext
    Wend
    %>
    </select></td>
   </tr>
   <tr>
    <td>Sở thích</td>
    <td>
<input type="checkbox" name="chkST" value="Du lich" <%if
instr(Ucase(rs1.fields("Sothich")),ucase("Du lich"))>0 then%> checked <%End if%>>Du lịch
<input type="checkbox" name="chkST" value="The thao" <%if
instr(Ucase(rs1.fields("Sothich")),ucase("The thao"))>0 then%> checked <%End if%>>Thể thao
<input type="checkbox" name="chkST" value="Xem TV" <%if
instr(ucase(rs1.fields("Sothich")),ucase("Xem TV"))>0 then%> checked <%End if%>>Xem TV
<input type="checkbox" name="chkST" value="Ca nhac" <%if
instr(ucase(rs1.fields("Sothich")),ucase("Ca nhac"))>0 then%> checked <%End if%>>Ca nhạc</td>
   </tr>
<tr>
<td><input type="submit" value="Save" name="CmdSave">
      <input type="reset" value="Reset" name="cmdReset">
</td>
</tr>
  </table>
</form>
```

```
<%
rs.Close
rs1.Close
%>
</body>
</html>
```

File Xuly_sua.asp

```
<!--#include file=Connection.inc-->
<%
StrSQL="Select * from Sinhvien Where MaSV='" & session("MaSV") & "'"
RS.Open StrSQL,Conn,1,3
RS.Fields("Hoten")=request.form("txtHoten")
If Request.form("rdoGT")="Nam" then
        RS.Fields("Gioitinh")=True
else
        RS.Fields("Gioitinh")=False
End if
RS.Fields("Tuoi")=request.form("txtTuoi")
ND=request.form("optThang")&"/"&request.form("optNgay")&"/"&request.form("optNam")
RS.Fields("NgayDoan")=Cdate(ND)
RS.Fields("Matinh")=request.form("optTinh")
RS.Fields("Sothich")=request.form("chkST")
RS.Update
RS.Close
response.redirect("Default.asp")
%>
```

File Xoa.asp

```
<!--#include file=Connection.inc-->
<%
For i=1 to request.form("chkXoa").Count
        If request.form("chkXoa")(i)<>"" then
                StrSQL="Select * from SinhVien Where MaSV='" & request.form("chkXoa")(i) & "'"
                RS.Open StrSQL,Conn,1,3
                RS.Delete
                RS.Close
        end if
next
Response.redirect("Default.asp")
%>
```

File Profile.asp

```
<!--#include file=Connection.inc-->
<%
StrSQL="SELECT SinhVien.*, Tinh.* FROM Tinh INNER JOIN SinhVien ON
[Tinh].[MaTinh]=[SinhVien].[MaTinh] WHERE MaSV=' " & request("MaSV_ID") & " ' “
RS.Open StrSQL,Conn
Response.write "Mã SV: " & RS.fields("MaSV") & "<BR>"
Response.write "Họ tên: " & RS.fields("Hoten") & "<BR>"
Response.write "Giới tính: "
If rs.fields("Gioitinh") then
        Response.write "Nam" & "<BR>"
else
        Response.write "Nữ" & "<BR>"
end if
Response.write "Tuổi: " & RS.fields("Tuoi") & "<BR>"
Response.write "Ngày vào đoàn: " & RS.fields("Ngaydoan") & "<BR>"
Response.write "Tỉnh/Tp: " & RS.fields("Tentinh") & "<BR>"
Response.write "Sở thích: " & RS.fields("Sothich") & "<BR>"
%>
<a href=Default.asp>Quay lại</a>
```

Bài giảng Lập trình mạng

LỜI NÓI ĐẦU

Ngày nay, hầu như việc viết một ứng dụng để chạy trên máy đơn cục bộ không còn được ưa chuộng và thích hợp nữa. Các chương trình và ứng dụng hiện đại phải tích hợp và triệu gọi lẫn nhau trên mạng Intranet (mạng cục bộ), mạng Internet (mạng toàn cầu) và ngôn ngữ lập trình Java là một trong những lựa chọn tốt nhất để làm việc này. Java là một ngôn ngữ lập trình không đơn giản, ngoài sự nổi tiếng về bản thân ngôn ngữ, nền tảng Java còn hướng đến các ứng dụng mạng như: giao tiếp trên mạng theo mô hình khách/chủ (client/server) ... So với lập trình thông thường, lập trình mạng đòi hỏi người lập trình những hiểu biết và kỹ năng chuyên sâu hơn để tạo giao tiếp và trao đổi dữ liệu giữa các máy tính với nhau.

Để giúp sinh viên chuyên ngành CNTT trong Nhà trường có thể tiếp cận được với những kỹ thuật mới này, chúng tôi đã mạnh dạn soạn thảo cuốn ***"Bài giảng Lập trình mạng"*** *để đưa vào giảng dạy cho sinh viên CNTT học năm thứ 3 trong Nhà trường. Cuốn bài giảng này được soạn thảo dựa trên nền tảng các sinh viên CNTT sau 2 năm học đầu trong trường đã được trang bị đầy đủ các kiến thức về Ngôn ngữ lập trình hướng đối tượng, Mạng máy tính, Thiết kế Web. Đây là một môn học với đặc thù là kiến thức luôn đổi mới và cập nhật, do đó yêu cầu với môn học này là sinh viên phải tự đọc thêm tài liệu, giáo viên chỉ là người hướng dẫn những vấn đề cơ bản nhất cho sinh viên.*

Lập trình mạng là môn học mới được đưa vào giảng dạy, nên trong quá trình soạn thảo bài giảng không tránh khỏi bỡ ngỡ và thiếu sót. Chúng tôi rất mong được sự giúp đỡ, đóng góp ý kiến của các đồng nghiệp và độc giả quan tâm để lần tái bản sau cuốn sách được hoàn thiện hơn. Mọi ý kiến đóng góp có thể gửi về theo địa chỉ email: qtmcn1@yahoo.com.

Chúng tôi xin chân thành cảm ơn!

Hà Nội 12/2005

Các tác giả

MỤC LỤC

địa chỉ IP (không có phương thức khởi dựng cho lớp này). Thường ta sẽ quan tâm đến các phương thức sau:

- public static InetAddress getLocalHost ()
  throws UnknownHostExceptiongetByName

  Trả về đối tượng InetAddress là địa chỉ của máy cục bộ (local host)

- public static InetAddress getByName (String host)
  throws UnknownHostException

Phương thức này nhận địa chỉ của một máy bằng kiểu chuỗi String và trả về đối tượng kiểu InetAddress thay mặt cho địa chỉ máy này.

- public static InetAddress[] getAllByName (String host)
  throws UnknownHostException

Phương thức này nhận địa chỉ của một máy bằng kiểu chuỗi và trả về tất cả các đối tượng InetAddress thay mặt cho địa chỉ máy này.

- public byte[] getAddress()

  Trả về địa chỉ IP của đối tượng InetAddress dưới dạng một dãy các byte. Vị trí byte cao nhất nằm ở byte 0.

- public String getHostAddress()

  Trả về địa chỉ IP của đối tượng InetAddress dưới dạng một chuỗi được dịnh dạng phân làm 4 nhóm %d.%d.%d.%d (Ví dụ: "172.16.11.12").

Dưới đây là ví dụ cho thấy cách dùng lớp InetAddress để lấy về các thông tin của địa chỉ máy chủ:

Ví dụ 1-0 AddrLookupApp.java

import java.net

b. Lớp Socket

Lớp Socket dùng tạo kết nối từ phía khách với máy chủ thường được khởi dựng bằng các phương thức sau:

- public Socket (String host, int port)
  throws UnknownHostException, IOException

Tạo ra một socket để kết nối máy có tên theo địa chỉ host và số cổng port.

- public Socket(InetAddress address, int port) throws IOException

Tạo ra một socket kết nối địa chỉ là đối tượng InetAddress và số cổng port.

- Public Socket(String host, int port, boolean stream) throws IOException

Tạo ra một socket kết nối theo địa chỉ host và số cổng port tham số stream cuối cùng để quy định kết nối theo TCP (stream=true) hay UDP (stream=false). Tuy nhiên nếu áp dụng để tạo socket cho giao thức UDP nên sử dụng lớp thay thế là DatagramSocket.

Các phương thức khác hỗ trợ cho lớp Socket từ phía máy khách bao gồm:

- InputStream get InputStream() throws IOException

Lấy về luồng nhập để máy khách có thể nhập dữ liệu trả về từ phía máy chủ.

- OutputStream get OutputStream() throws IOException

Lấy về luồng xuất để máy khách có thể ghi dữ liệu gửi đến máy chủ.

- InetAddress get InetAddress()

Lấy địa chỉ kết nối socket của máy chủ

- int getPort()

Lấy về số cổng dùng kết nối của máy chủ.

Ví dụ đoạn mã sau sẽ thực hiện kết nối với máy chủ có địa chỉ "my.testing.server" và mở ra hai luồng xuất nhập để đọc và gửi thông tin đến máy chủ theo số cổng 1234.

```
try{
Socket me=new Socket("my.testing.server", 1234);
// Luồng nhập thông tin để trả về máy chủ từ phía kết nối
DataInputstream in = new DataInputStream(me.getInputStream());
// Luồng xuất để ghi thông tin gửi đến máy chủ
DataOutputStream out = new DataOutputStream(me.getOutputStream());
Catch (Exception e) {
System.out.println(e);
}
```

c. Lớp ServerSocket

Lớp ServerSocket dùng tạo kết nối từ phía máy chủ với các máy khách. Đối tượng ServerSocket được tạo ra trên máy chủ và lắng nghe những kết nối từ phía máy khách gửi đến theo một số cổng định trước. Đối tượng ServerSocket được khởi dựng từ phương thức sau:

- public ServerSocket(int port) throws IOException

port là số hiệu cổng mà đối tượng ServerSocket phải lắng nghe để nhận biết những kết nối từ phía máy khách gửi đến

Để chờ đợi kết nối từ các máy khách gửi đến đối tượng ServerSocket thường nhờ đến phương thức accept như sau:

- Socket accept () throws IOException

Phương thức này thực sự dừng lại chờ đợi cho đến khi nhận được thông tin kết nối sẻ trả về đối tượng socket của máy khách nơi có yêu cầu nối vào máy chủ.

Cuối cùng máy chủ có thể cắt đứt mọi kết nối bằng cách gọi phương thức close của đối tượng ServerSocket

- Public void close () throws IOException

Ví dụ đoạn mã sau sẽ tạo một đối tượng ServerSocket trê máy chủ luôn lắng nghe kết nối từ phía máy khách gửi đến qua số cổng 1234

```
Try {
    ServerSocket server = new ServerSocket (1234);
    Socket client;
// Chương trình sẽ dựng lại ở đây để chờ đợi sự kết nối
    client = server.accept ();
// Có một kết nối gửi đến từ phía máy khách
    System.out.println("Accept connect");
// Xử lý các yêu cầu về dịch vụ
// …
// Cắt đứt các kết nối
    client.close ();
    server.close ();
  catch (Exception e) {
  System.out.println (e);
    }
```

d. Lớp DatagramSocket

Lớp này được dùng để chuyển đi một gói dữ liệu (biểu diễn bằng đối tượng DatagramPackage) theo giao thức UDP. Dữ liệu được gửi đi không bảo đảm được nhận đầy đủ và có thể bị lỗi trên đường truyền (cơ chế dùng DatagramSocket không an toàn bằng lớp Socket). Dưới đây là một số phương thức dùng của lớp DatagramSocket:

- public DatagramSocket () throws SocketException
  Phương thức khởi dựng để tạo kết nối UDP
- public DatagramSocket (int port) throws SocketException
  Phương thức khởi dựng để tạo kết nối UDP với số hiệu cổng port
- public void synchronized send (DatagramSocket p) throws IOException
  Gửi gói dữ liệu đi.
- Public void synchronized receive (DatagramSocket p) throws IOException

Nhận gói dữ liệu về.

- public void synchronized close ()

  Đóng kết nối.

e. Lớp DatagramPackage

Lớp này dùng cho một gói dữ liệu gửi đi trên mạng theo kết nối DatagramSocket. Một gói có thể chứa các thông tin như dữ liệu, chiều dài gói, các địa chỉ IP và số cổng mà từ đó các gói dữ liệu được gửi đi. Dưới đây là một số phương thức hữu dụng của lớp DatagramPackage này:

- *public DatagramPackage(byte buff[], int len)*

Phương thức khởi tạo ra gói có dữ liệu chứa trong bộ đệm *buff[]* và chiều dài gói dữ liệu là *len.*

- *public DatagramPackage (byte buff[], int len, InetAddress iaddr, int port)*

Phương thức khởi tạo ra gói có dữ liệu chứa trong bộ đệm *buff[]* cùng với chiều dài vùng đệm muốn lấy, địa chỉ máy đích và số hiệu cổng.

- *public InetAddress getAddress()*

Trả về địa chỉ chứa trong gói dữ liệu.

- *public byte[] getData()*

Trả về dữ liệu thật sự chứa trong gói.

- *public int getLength()*

Trả về kích thước hay chiều dài gói dữ liệu.

- *public int getPort()*

Trả về số hiệu cổng chứa trong gói dữ liệu.

f. Lớp URL

URL(Uniform Resource Locator) là địa chỉ định vị tài nguyên trên mạng, thường một một URL như đã đề cập bao gồm 3 phần: phần nghi thức(protocol), phần địa chỉ hay tên máy chủ(host name), và phần chỉ định tên tập tin hay tài liệu muốn lấy từ máy chủ về.

Java đóng gói tất cả những đặc điểm này vào một lớp URL. Đối tượng URL được tạo ra bằng một trong những phương thức khởi tạo sau:

- *public URL(String spec) throws MalformedURLException*

Tạo một đối tượng URL từ địa chỉ định vị là một chuỗi.

- *public URL(String protocol, String host, int port, String file) throws MalformedURLException*

Tạo một địa chỉ định vị tuyệt đối với đầy đủ nghi thức(protocol), máy chủ(server), cổng(port), đường dẫn(file) tới tập tin cần lấy trên máy chủ.

- *public URL(String protocol, String host, String file) throws MalformedURLException*

Tạo một địa chỉ định vị tuyệt đối với đầy đủ nghi thức(protocol), máy chủ(server), đường dẫn(file) tới tập tin cần lấy trên máy chủ(bỏ qua thành phần định vị cổng giao tiếp, nếu truy tìm trang web theo nghi thức http thì cổng chương trình web server mặc định là 80).

Các phương thức hỗ trợ khác dùng cho lớp URL là:

- *public final Object getContent() throws IOException*

lấy về nội dung mà kết nối theo địa chỉ URL có được.

- *String getFile()*

Lấy về tên tập tin hay tài liệu nằm trong chuỗi địa chỉ URL có được.

- *String getHost()*

Lấy tên máy chủ(thường là thành phần thức 2 của chuỗi URL)

- *String getPort()*

Lấy về số hiệu cổng.

- *String getProtocol()*

Lấy về tên giao thức(thường là thành phần đầu tiên trong chuỗi URL)

- *String getRef()*

Lấy về nội dung chuỗi tham khảo thêm trong chuỗi URL(được đặt sau dấu # của chuỗi)

- *Public final InputStream openStream() throws IOException*

Mở luồng nhập để đọc thông tin trả về từ máy chủ

Ví dụ đoạn mã sau đây dùng để lấy về nội dung trang web index.htm từ máy chủ java.sun.com:

```
try{
// Mở kết nối đến trang Web theo địa chỉ định vị URL
        URL o = new URL("http://java.sun.com/index.htm");
// Tạo luồng nhập để đọc nội dung trang Web trả về từ máy chủ
    BufferedReader inStream = new BufferedReader(new InputStreamReader(o.openStream()));
// In nội dung trang Web index.htm ra màn hình
        String line;
        while((line = in.readLine()) != null) {
```

```
            System.out.println (line);
        }
    }catch (Exception e) {
    // Quá trình mở và kết nối với trang web bị lỗi
            System.out.println (e) ;
    }
```

Cũng như các ngôn ngữ lập trình khác, Java cần một trình biên dịch để chuyển đổi mã lệnh cho người đọc (mã nguồn) sang ứng dụng thực thi được. Các trình biên dịch thông thường như Microsoft Visual C++ cho Windows 95 sẽ biên dịch chương trình sang mã lệnh thực hiện trên một loại phần cứng nhất định nào đó (trong trường hợp này là mã lệnh cho Intel x86). Trái lại, trình biên dịch Java lại chuyển chương trình nguồn Java thành các bytecode. Các bytecode này chỉ có thể chạy được trên máy ảo Java (Java Virtual Machine -JVM).

Lưu ý: Hiện nay, máy ảo Java (JVM) mới được tạo dựng bằng phần mềm chứ không phải phần cứng. Sun Microsystems và một số công ty điện tử khác đang tiến hành nghiên cứu phát triển chip picoJava, nhằm mục đích tạo máy ảo Java bằng phần cứng. Các chip này cho phép đưa Java vào các thiết bị điện tử một cách dễ dàng hơn, đồng thời làm tăng tốc độ tối đa cho các JVM viết bằng phần mềm.

Bộ Java Developers Kit (JDK) do Sun cung cấp bao gồm một số chương trình tiện ích cho phép bạn biên dịch, bắt lỗi và tạo tài liệu cho một ứng dụng Java. Hiện nay trên thị trường đang có rất nhiều môi trường phát triển Java của hãng thứ ba rất tiện lợi (như Visual J++, Symantec Cafe,..), nhưng tất cả các chương trình này đều dựa trên nền JDK. Các trình tiện ích của JDK bao gồm:

javac: Bộ biên dịch Java: Làm nhiệm vụ chuyển mã nguồn Java sang bytecode.

java: Bộ thông dịch Java: Thực thi các ứng dụng Java trực tiếp từ tập tin lớp (class).

appletviewer: Một trình thông dịch Java thực thi các Java applet từ tập tin HTML.

javadoc: Tạo tài liệu dạng HTML từ mã nguồn cùng với các chú thích bên trong.

jdb (Java debuger): Cho phép bạn thực hiện từng dòng trong chương trình, đặt các điểm dừng (breakpoint), xem giá trị các biến.

javah: Tạo ra tập tin header của C cho phép C gọi hàm Java hoặc ngược lại.

javap: Trình dịch ngược java (disassembler): Hiển thị các hàm và dữ liệu truy cập được bên trong một tập tin lớp đã dịch. Nó cũng cho phép hiển thị nghĩa của bytecode.

Quá trình biên dịch Java như sau: mã nguồn trong các tập tin *.java, qua trình biên dịch javac được chuyển thành các bytecode. Bytecode nằm trong tập tin *.class, được gọi là tập tin lớp (bởi mỗi tập tin chứa một lớp riêng biệt của Java). Các ứng dụng Java có thể bao gồm nhiều lớp khác nhau.

Chú ý: Một lớp (class) của Java cũng giống hệt như một lớp trong C++. Lớp chính là các biến dữ liệu và thủ tục kết hợp với nhau thành một khối.

Kiểm tra sự tiến bộ

1. Bạn có thể viết các chương trình dạng thủ tục bằng Java. **Đúng/Sai**
2. Java là ngôn ngữ có kiểu dữ liệu chặt chẽ. **Đúng/Sai**
3. .....là chương trình Java chạy độc lập, và sử dụng giao diện đồ họa để người sử dụng nhập dữ liệu.
4. .....sử dụng JDBC API để kết nối cơ sở dữ liệu.
5. .....hiểu một dòng các lệnh máy tính trừu tượng.
6. Coalescing và Compaction là gì?
7. Lệnh ..... được sử dụng để dịch file mã nguồn .java.
8. Lớp ..... phải là lớp cha của các applet, applet là chương có thể nhúng vào trang Web hay chạy bằng Java AppletViewer.

Bài tập

1. Cài đặt Java 2
2. Gõ các lệnh sau tại dấu nhắc và liệt kê các tham số khác nhau của chúng:
   - javac
   - java

```
        double y = 7.22;
        System.out.println("Variable values...");
        System.out.println("    i = " + i);
        System.out.println("    j = " + j);
        System.out.println("    x = " + x);
        System.out.println("    y = " + y);

        //adding numbers
        System.out.println("Adding...");
        System.out.println("    i + j = " + (i + j));
        System.out.println("    x + y = " + (x + y));

        //subtracting numbers
        System.out.println("Subtracting...");
        System.out.println("    i - j = " + (i - j));
        System.out.println("    x - y = " + (x - y));

        //multiplying numbers
        System.out.println("Multiplying...");
        System.out.println("    i * j = " + (i * j));
        System.out.println("    x * y = " + (x * y));

        //dividing numbers
        System.out.println("Dividing...");
        System.out.println("    i / j = " + (i / j));
        System.out.println("    x / y = " + (x / y));

        //computing the remainder resulting from dividing numbers
        System.out.println("Computing the remainder...");
        System.out.println("    i % j = " + (i % j));
        System.out.println("    x % y = " + (x % y));

        //mixing types
        System.out.println("Mixing types...");
        System.out.println("    j + y = " + (j + y));
        System.out.println("    i * x = " + (i * x));
    }
}
```

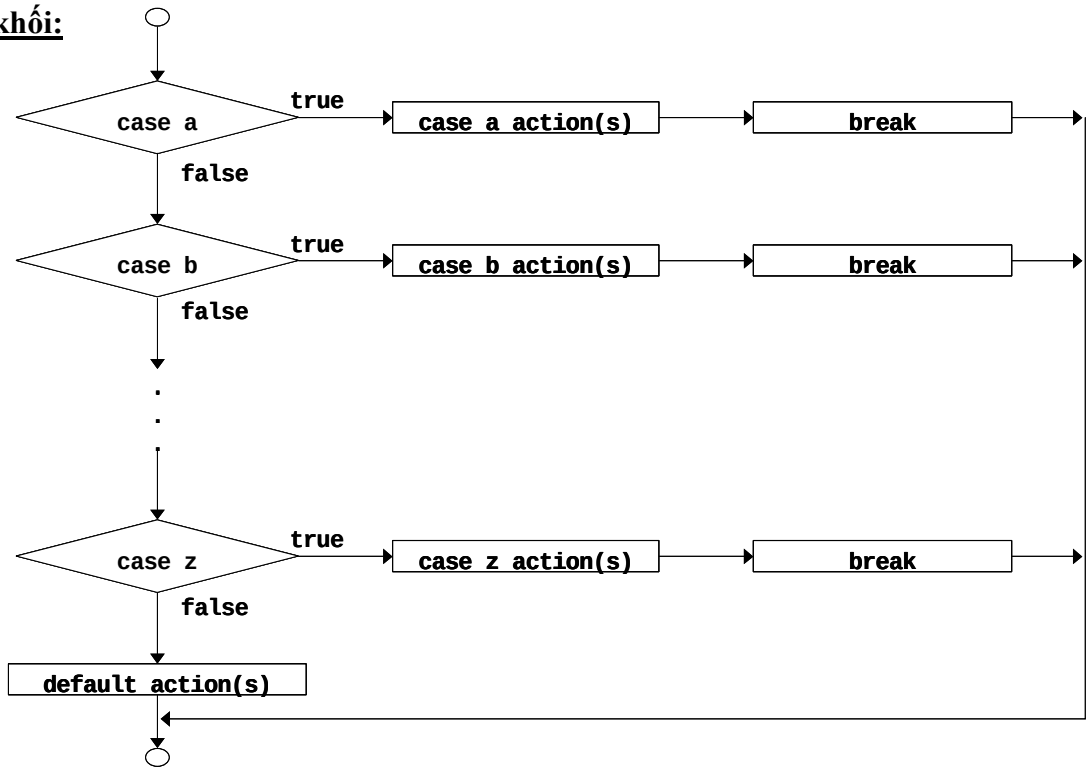
Kết quả của chương trình:

```
Variable values...
    i = 37
    j = 42
    x = 27.475
    y = 7.22
Adding...
    i + j = 79
    x + y = 34.695
```

```
            break;
        ...
        default:
            statement(s)
            break;
    }
```

Expression : Biểu thức điều kiện phụ thuộc vào kiểu dữ liệu cơ bản (byte, int ...)

Sơ đồ khối:



Ví dụ:

```
public class SwitchDemo {
    public static void main(String[] args) {

        int month = 2;
        int year = 2000;
        int numDays = 0;

        switch (month) {
            case 1:
            case 3:
            case 5:
            case 7:
            case 8:
            case 10:
            case 12:
                numDays = 31;
                break;
```

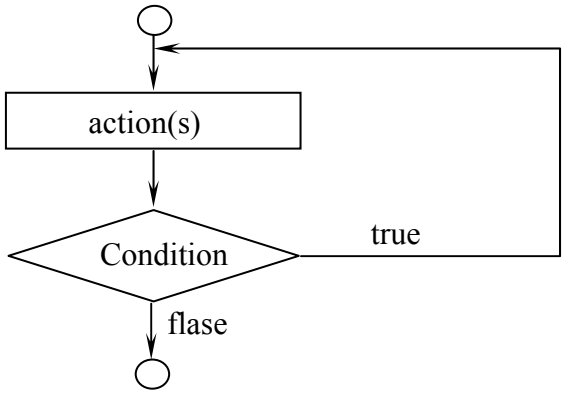
```
        char c = copyFromMe.charAt(i);

        while (c != 'g') {
            copyToMe.append(c);
            c = copyFromMe.charAt(++i);
        }
        System.out.println(copyToMe);
    }
}
```

Java cung cấp một cấu trúc khác tương tự như câu lệnh while.

Cú pháp:

```
do {
      Statement(s);
while (Condition);
```

Sơ đồ khối:



Ví dụ:

```
public class DoWhileDemo {
    public static void main(String[] args) {

        String copyFromMe = "Copy this string until you " +
                            "encounter the letter 'g'.";
        StringBuffer copyToMe = new StringBuffer();
        int i = 0;
        char c = copyFromMe.charAt(i);
        do {
            copyToMe.append(c);
            c = copyFromMe.charAt(++i);
        } while (c != 'g');
        System.out.println(copyToMe);
    }
}
```

6. **'static'** hàm ý rằng phương thức không có mã và được bổ sung trong các lớp con **Đúng/Sai** Khi bạn không định nghiã một hàm khởi tạo cho một lớp, JVM sẽ cung cấp một hàm mặc định hoặc một hàm khởi tạo ẩn (implicit). **Đúng/Sai** Vòng lặp while thực thi ít nhất một lần thậm chí nếu điều kiện được xác định là False **Đúng/Sai**

Bài tập

1. Hãy viết một đoạn chương trình để in ra dòng chữ " Welcome to the world of Java"
2. Hãy viết hai phương thức khởi tạo tường minh cho một lớp dùng để tính diện tích hình chữ nhật. Khi một giá trị được truyền vào phương thức khởi tạo, nó cho rằng độ dài và chiều rộng bằng nhau và bằng giá trị truyền vào. Lúc đó, nó sẽ tính diện tích tương ứng. Khi hai giá trị được truyền vào, nó sẽ tính diện tích hình chữ nhật.
3. Viết một chương trình thực hiện những việc sau đây:
    a. Khai báo và gán giá trị đầu cho các biến m và n là 100 và 200 tương ứng.
    b. Theo các điều kiện: nếu m bằng 0, hiển thị kết quả tương ứng.
    c. Nếu m lớn hơn n , hiển thị kết quả tương ứng.
    d. Kiểm tra giá trị n là chẵn hay lẻ.
4. Viết một chương trình hiển thị tổng các bội số của 7 nằm giữa 1 và 100.
5. Viết chương trình để cộng bảy số hạng của dãy sau:
   1!+2!+3!…….

Đây là chương trình đầy đủ của Sender:

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
import java.util.Enumeration;

public class Sender extends Applet
                    implements ActionListener {
    private String myName;
    private TextField nameField;
    private TextArea status;
    private String newline;

    public void init() {
        GridBagLayout gridBag = new GridBagLayout();
        GridBagConstraints c = new GridBagConstraints();

        setLayout(gridBag);

        Label receiverLabel = new Label("Receiver name:",
                                        Label.RIGHT);
        gridBag.setConstraints(receiverLabel, c);
        add(receiverLabel);

        nameField = new TextField(getParameter("RECEIVERNAME"),
                                              10);
        c.fill = GridBagConstraints.HORIZONTAL;
        gridBag.setConstraints(nameField, c);
        add(nameField);
        nameField.addActionListener(this);

        Button button = new Button("Send message");
        c.gridwidth = GridBagConstraints.REMAINDER; //end row
        c.anchor = GridBagConstraints.WEST; //stick to the
                                            //text field
        c.fill = GridBagConstraints.NONE; //keep the button
                                          //small
        gridBag.setConstraints(button, c);
        add(button);
        button.addActionListener(this);

        status = new TextArea(5, 60);
        status.setEditable(false);
        c.anchor = GridBagConstraints.CENTER; //reset to the default
```

```
        c.fill = GridBagConstraints.BOTH; //make this big
        c.weightx = 1.0;
        c.weighty = 1.0;
        gridBag.setConstraints(status, c);
        add(status);

        myName = getParameter("NAME");
        Label senderLabel = new Label("(My name is " + myName + ".)",
                                      Label.CENTER);
        c.weightx = 0.0;
        c.weighty = 0.0;
        gridBag.setConstraints(senderLabel, c);
        add(senderLabel);

    newline = System.getProperty("line.separator");
    }

    public void actionPerformed(ActionEvent event) {
        Applet receiver = null;
        String receiverName = nameField.getText(); //Get name to
                                                   //search for.
        receiver = getAppletContext().getApplet(receiverName);
        if (receiver != null) {
            //Use the instanceof operator to make sure the applet
            //we found is a Receiver object.
            if (!(receiver instanceof Receiver)) {
                status.append("Found applet named "
                              + receiverName + ", "
                              + "but it's not a Receiver object."
                          + newline);
            } else {
                status.append("Found applet named "
                              + receiverName + newline
                              + "  Sending message to it."
                          + newline);
                //Cast the receiver to be a Receiver object
                //(instead of just an Applet object) so that the
                //compiler will let us call a Receiver method.
                ((Receiver)receiver).processRequestFrom(myName);
            }
        } else {
            status.append("Couldn't find any applet named "
                          + receiverName + "." + newline);
        }
```

```
    }

    public Insets getInsets() {
        return new Insets(3,3,3,3);
    }

    public void paint(Graphics g) {
        g.drawRect(0, 0,
                   getSize().width - 1, getSize().height - 1);
    }

    public String getAppletInfo() {
        return "Sender by Kathy Walrath";
    }
}
```

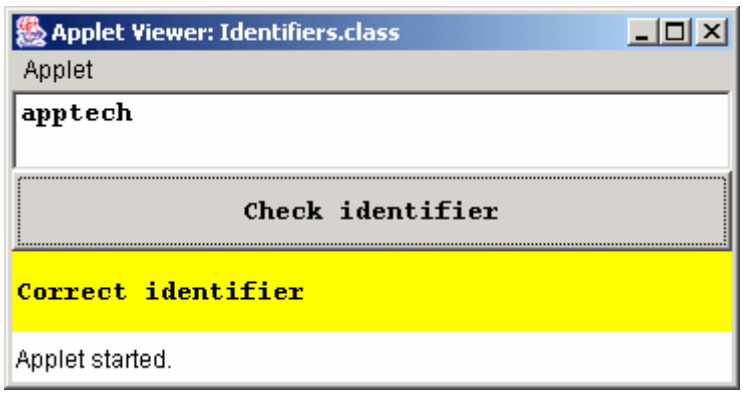
Dưới đây là chương trình Reciever:

```
import java.applet.*;
import java.awt.*;

public class Receiver extends Applet {
     private final String waitingMessage="Waiting for a message...";
     private Label label = new Label(waitingMessage, Label.RIGHT);

     public void init() {
         add(label);
         add(new Button("Clear"));
         add(new Label("(My name is " + getParameter("name") + ".)",
                       Label.LEFT));
         validate();
     }

     public boolean action(Event event, Object o) {
         label.setText(waitingMessage);
         repaint();
         return false;
     }

     public void processRequestFrom(String senderName) {
         label.setText("Received message from " + senderName + "!");
         repaint();
     }
```

5. Phương thức .....dùng để tạo tham chiếu đến đối tượng nền đồ hoạ (Graphics).
6. Trong java, điều khiển màu sắc được thực hiện thông qua hai màu cơ bản là trắng và đen. **Đúng/Sai**
7. Phương thức .....dùng để lấy tất cả các font chữ mà hệ thống hỗ trợ.
8. Trong lớp FontMetrics, 'ascent' là khoảng cách từ 'baseline' đến đáy của ký tự. **Đúng/Sai**

Bài tập

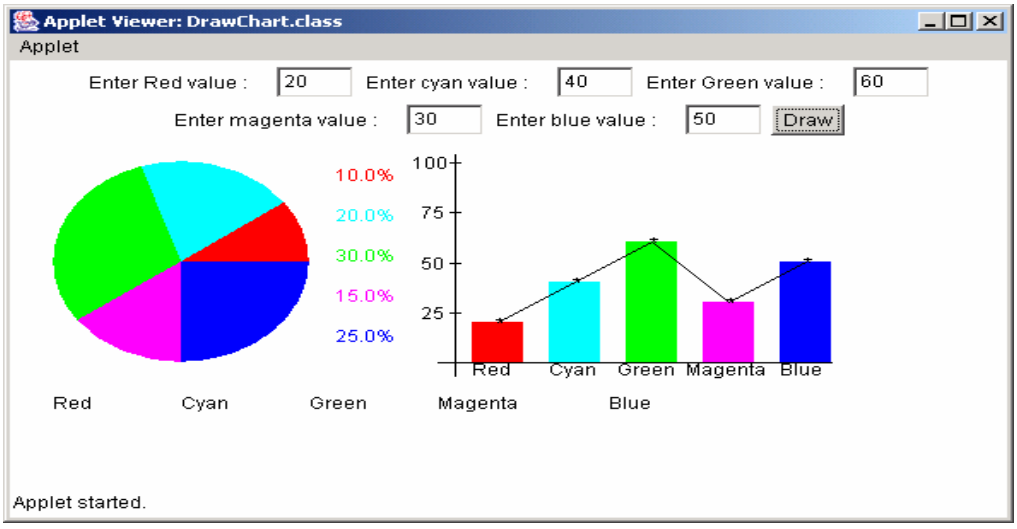
1. Viết applet như sau:



Khi người sử dụng nhập vào ô văn bản và click chuột vào nút 'Check identifier' applet phải kiểm tra xem từ có trong ô văn bản có phải là một từ khoá có trong java không.

Ghi chú: Sử dụng các font chữ khác nhau cho ô văn bản, nút lệnh và nhãn.

2. Viết applet như sau:



Người sử dụng được phép nhập vào giá trị màu trong các ô tương ứng. Khi click nút 'Draw' các biểu đồ dạng đường, bar, pie được hiện ra.

nghĩa trong một lớp mà lớp đó hiện thực giao diện này. Nói một cách khác, bạn cần chỉ ra hành vi của phương thức. Tất cả các phương thức trong các giao diện phải là kiểu **public**. Bạn không được sử dụng các bổ ngữ (modifers) chuẩn khác như protected, private,..khi khai báo các phương thức trong giao diện.

Đoạn mã Chương trình 4.2 biểu diễn một giao diện được cài đặt như thế nào:

Chương trình 4.2

```
import java.io.*;
class Demo implements myinterface
{
 public void add(int x,int y)
 {
   System.out.println(" "+(x+y));
   //Giả sử phương thức add được khai báo trong giao diện
 }
 public void volume(int x,int y,int z)
 {
   System.out.println(" "+(x*y*z));
   //Giả sử phương thức volume được khai báo trong giao diện
 }
 public static void main(String args[])
 {
  Demo d=new Demo();
  d.add(10,20);
  d.volume(10,10,10);
 }
}
```

Khi bạn định nghĩa một giao diện mới, có nghĩa là bạn đang định nghĩa một kiểu dữ liệu tham chiếu mới. Bạn có thể sử dụng các tên giao diện ở bất cứ nơi đâu như bất kỳ kiểu dữ liệu

- Đoạn mã phải bắt đầu với một phát biểu "package". Điều này nói lên rằng lớp được định nghĩa trong tập tin là một phần của gói xác định.
- Mã nguồn phải nằm trong cùng một thư mục, mà thư mục đó lại là tên gói của bạn.
- Quy ước rằng, các tên gói sẽ bắt đầu bằng một chữ thường để phân biệt giữa lớp và gói.
- Các phát biểu khác có thể xuất hiện sau khai báo gói là các câu lệnh nhập, sau chúng bạn có thể bắt đầu định nghĩa lớp của bạn.
- Tương tự tất cả các tập tin khác, mỗi lớp trong một gói cần được biên dịch.
- Để cho chương trình Java của bạn có khả năng sử dụng các gói đó, hãy nhập (import) chúng vào mã nguồn của bạn.
- Sự khai báo sau đây là hợp lệ và không hợp lệ :

Hợp lệ

package mypackage;

import java.io.;*

Không hợp lệ

import java.io.;*

package mypackage;

Bạn có các tuỳ chọn sau trong khi nhập vào một gói:

- Bạn có thể nhập vào một tập tin cụ thể từ gói:

import java.mypackage.calculate

- Bạn có thể nhập (import) toàn bộ gói:

import java.mypackage.*;

Máy ảo Java (JVM) sẽ quản lý các thành phần nằm trong các gói đã được nhập vào (import).

Bạn đã sẵn sàng làm việc với một lệnh nhập import -java.io.*. Bản thân Java đã được cài đặt sẵn một tập các gói, bảng dưới đây đề cập đến một vài gói có sẵn của Java:

Gói	Mô tả

- Khai báo và định nghĩa các lớp sẽ nằm trong gói đó. Tất cả các thành phần của gói sẽ là public, để có thể được truy cập từ bên ngoài. Máy ảo Java (JVM) quản lý tất cả các phần tử nằm trong gói đó.

```
package mypackage; //khai báo gói
import java.util.*;
public class Calculate //định nghĩa một lớp
{
int var;
Calculate(int n)
        {
 ...
                var = n;
                //các phương thức
                //...
                public class Display //định nghĩa một lớp
                {
                ...//Các phương thức
                }
        }
}
```

- Lưu các định nghĩa trên trong một tập tin với phần mở rộng .java, và dịch các lớp được định nghĩa trong gói. Việc dịch có thể thực hiện với tham số "-d". Chức năng này tạo một thư mục trùng với tên gói, và đặt tập tin .class vào thư mục được chỉ rõ.

javac –d d:\temp Calculate.java

Nếu khai báo gói không có trong chương trình, lớp hoặc giao diện đó sẽ nằm trong gói mặc định mà không có tên. Nói chung, gói mặc định này thì chỉ có nghĩa cho các ứng dụng nhỏ hoặc tạm thời.

Hãy ghi nhớ các điểm sau đây khi bạn khai thác các gói do người dùng định nghĩa trong các chương trình khác:

javac –classpath c:\temp Packagedemo.java

Thứ tự của các mục trong classpath rất quan trọng. Khi bạn thực thi đoạn mã của bạn, mày ảo Java sẽ tìm kiếm các mục trong classpath theo thứ tự các thư mục trong classpath, cho đến khi nó tìm thấy lớp cần tìm.

Ví dụ của một gói

Chương trình 4.3

```
package mypackage;
public class calculate
{
 public double volume(double height, double width,double depth) {
            return (height*width*depth);
      }
      public int add(int x,int y) {
            return (x+y);
      }
      public int divide(int x,int y) {
            return (x/y);
      }
}
```

Để sử dụng gói này, bạn cần phải:

- Nhập lớp được sử dụng.
- Nhập toàn bộ gói.
- Sử dụng các thành phần của gói.

Bạn cần dịch tập tin này. Nó có thể được dịch với tuỳ chọn –d, nhờ đó javac nó tạo một thư mục với tên của gói và đặt tập tin .class vào thư mục này.

javac –d c:\temp calculate.java

Chương trình biên dịch tạo một thư mục được gọi là “mypackage” trong thư mục temp, và lưu trữ tập tin calculate.class vào thư mục này.

String strname = "Java Language";
boolean flag = strname.endsWith("Java");

Biến "flag" chứa giá trị false.

- ***copyValueOf( )***

Phương thức này trả về một chuỗi được rút ra từ một mảng ký tự được truyền như một đối số. Phương thức này cũng lấy hai tham số nguyên. Tham số đầu tiên chỉ định vị trí từ nơi các ký tự phải được rút ra, và tham số thứ hai chỉ định số ký tự được rút ra từ mảng. Ví dụ:

char name[] = {'L','a','n','g','u','a','g','e'};

String subname = String .copyValueOf(name,5,2);

Bây giờ biến "subname" chứa chuỗi "ag".

- ***toCharArray( )***

Phương thức này chuyển chuỗi thành một mảng ký tự. Ví dụ:

String text = new String("Hello World");

char textArray[] = text.toCharArray( );

- ***indexOf( )***

Phương thức này trả về thứ tự của một ký tự nào đó, hoặc một chuỗi trong phạm vi một chuỗi. Các câu lệnh sau biểu diễn các cách khác nhau của việc sử dụng hàm.

String day = new String("Sunday");

int index1 = day.indexOf('n');

//chứa 2

int index2 = day.indexOf('z',2);

//chứa –1 nếu "z" không tìm thấy tại vị trí 2.

int index3 = day.indexOf("Sun");

//chứa mục 0

- ***toUpperCase( )***

Phương thức này trả về chữ hoa của chuỗi.

String lower = new String("good morning");

System.out.println("Uppercase: "+lower.toUpperCase( ));

```
        StringBuffer s3 = new StringBuffer("StringBuffer");


        System.out.println("s3 = "+ s3);
        System.out.println(s2.length());   //chứa 0
        System.out.println(s3.length());  //chứa 12
        System.out.println(s1.capacity());  //chứa 16
        System.out.println(s2.capacity());  //chứa 20
 System.out.println(s3.capacity());  //chứa 28
}
}
```

"length()" và "capacity()" của StringBuffer là hai phương thức hoàn toàn khác nhau. Phương thức "length()" đề cập đến số các ký tự mà đối tượng thực chứa, trong khi "capacity()" trả về tổng dung lượng của một đối tượng (mặc định là 16) và số ký tự trong đối tượng StringBuffer.

Dung lượng của StringBuffer có thể thay đổi với phương thức "ensureCapacity()". Đối số int đã được truyền đến phương thức này, và dung lượng mới được tính toán như sau:

NewCapacity = OldCapacity * 2 + 2

Trước khi dung lượng của StringBuffer được đặt lại, điều kiện sau sẽ được kiểm tra:

- Nếu dung lượng(NewCapacity) mới lớn hơn đối số được truyền cho phương thức "ensureCapacity()", thì dung lượng mới (NewCapacity) được đặt.
- Nếu dung lượng mới nhỏ hơn đối số được truyền cho phương thức "ensureCapacity()", thì dung lượng được đặt bằng giá trị tham số truyền vào.

Chương trình 4.7 minh hoạ dung lượng được tính toán và được đặt như thế nào.

Chương trình 4.7

```
class test{
    public static void main(String args[]){
        StringBuffer s1 = new StringBuffer(5);
        System.out.println("Dung lượng của bộ nhớ đệm = "+s1.capacity()); //chứa 5
```

chèn vào. Vị trí chèn sẽ lớn hơn hay bằng 0, và nhỏ hơn hay bằng chiều dài của đối tượng StringBuffer. Bất kỳ đối số nào, trừ ký tự hoặc chuỗi, được chuyển sang chuỗi và sau đó mới được chèn vào. Ví dụ:

StringBuffer str = new StringBuffer("Java sion");

str.insert(1,'b');

Biến "str" chứa chuỗi "Jbava sion".

- ***charAt()***

Phương thức này trả về một giá trị ký tự trong đối tượng StringBuffer tại vị trí được chỉ định.Ví dụ:

StringBuffer str = new StringBuffer("James Gosling");

char letter = str.charAt(6); //chứa "G"

- ***setCharAt()***

Phương thức này được sử dụng để thay thế ký tự trong một StringBuffer bằng một ký tự khác tại một vị trí được chỉ định.

StringBuffer name = new StringBuffer("Jawa");

name.setCharAt(2,'v');

Biến "name" chứa "Java".

- ***setLength()***

Phương thức này thiết lập chiều dài của đối tượng StringBuffer. Nếu chiều dài được chỉ định nhỏ hơn chiều dài dữ liệu hiện tại của nó, thì các ký tự thừa sẽ bị cắt bớt. Nếu chiều dài chỉ định nhiều hơn chiều dài dữ liệu thì các ký tự null được thêm vào phần cuối của StringBuffer

StringBuffer str = new StringBuffer(10);

str.setLength(str.length() +10);

- ***getChars()***

Phương thức này được sử dụng để trích ra các ký tự từ đối tượng StringBuffer, và sao chép chúng vào một mảng. Phương thức getChars() có bốn tham số sau:

Chỉ số đầu: vị trí bắt đầu, từ nơi mà ký tự được lấy ra.

Chỉ số kết thúc: vị trí kết thúc

System.out.println(Math.ceil(8.02)); //trả về 9.0

System.out.println(Math.ceil(-1.3)); //trả về -1.0

System.out.println(Math.ceil(100)); //trả về 100.0

System.out.println(Math.floor(-5.6)); //trả về -6.0

System.out.println(Math.floor(201.1)); //trả về 201

System.out.println(Math.floor(100)); //trả về 100

- ***max()***

Phương thức này tìm giá trị lớn nhất trong hai giá trị được truyền vào. Các đối số được truyền vào có thể là kiểu int, long, double, và float.

- ***min()***

Phương thức này tìm giá trị nhỏ nhất trong hai giá trị được truyền vào. Các đối số được truyền vào có thể là kiểu int, long, double và float.

- ***round()***

Phương thức này làm tròn đối số có dấu phẩy động. Ví dụ, câu lệnh Math.round(34.5) trả về 35.

- ***random()***

Phương thức này trả về một số ngẫu nhiên kiểu double giữa 0.0 và 1.0.

- ***sqrt()***

Phương thức này trả về căn bậc hai của một số. Ví dụ, câu lệnh Math.sqrt(144) trả về 12.0.

- ***sin()***

Phương thức này trả về sine của một số, nếu góc được truyền đến bằng radian. Ví dụ: **Math.sin(Math.PI/2)** trả về 1.0, giá trị của sin 45^0.

PI/2 radian = 90 độ. Giá trị của "PI" được định nghĩa trong lớp Math (Math.PI).

- ***cos()***

Phương thức này trả về cosine của một góc tính bằng radian.

setProperties()	Thiết lập các thuộc tính hệ thống hiện hành.
currentTimeMillis()	Trả về thời gian hiện tại bằng mili giây (ms), được tính từ lúc 0 giờ ngày 01 tháng 01 năm 1970.
arrayCopy(Object, int, Object, int, int)	Sao chép mảng.

Bảng 4.5 Lớp System.

Lớp System không thể tạo thể hiện (instance) được.

Đoạn mã trong chương trình sau đọc và hiển thị một vài các thuộc tính môi trường Java.

Chương trình 4.9

```
class SystemDemo
{
 public static void main(String args[])
 {
  System.out.println(System.getProperty("java.class.path"));
  System.out.println(System.getProperty("java.home"));
  System.out.println(System.getProperty("java.class.version"));
        System.out.println(System.getProperty("java.specification.vendor"));
        System.out.println(System.getProperty("java.specification.version"));
 System.out.println(System.getProperty("java.vendor"));
        System.out.println(System.getProperty("java.vendor.url"));
 System.out.println(System.getProperty("java.version"));
  System.out.println(System.getProperty("java.vm.name"));
 }
}
```

Mỗi thuộc tính cần in ra cần được cung cấp như một tham số (dạng chuỗi) đến phương thức System.getProperty(). Phương thức này sẽ trả về thông tin tương ứng và phương thức System.out.println() in ra màn hình.

Kết quả chương trình trên như sau:

Phương thức	Mục đích
clear()	Xoá tất cả các phần tử từ bảng băm.
clone()	Tạo một bản sao của Hashtable.
contains(Object)	Trả về True nếu bảng băm chứa các đối tượng được chỉ định.
containsKey(Object)	Trả về True nếu bảng băm chứa khoá được chỉ định.
elements()	Trả về một tập hợp phần tử của bảng băm.
get(Object key)	Trả về đối tượng có khoá được chỉ định.
isEmpty()	Trả về true nếu bảng băm rỗng.
keys()	Trả về tập hợp các khoá trong bảng băm.
put(Object, Object)	Thêm một phần tử mới vào bảng băm (Object, Object) là khoá và giá trị.
rehash()	Thay đổi bảng băm thành một bảng băm lớn hơn.
remove(Object key)	Xoá một đối tượng được cho bởi khoá được chỉ định.
size()	Trả về số phần tử trong bảng băm.
toString()	Trả về đại diện chuỗi được định dạng cho bảng băm.

Bảng 4.8 Các phương thức lớp Hashtable.

Chương trình sau sử dụng lớp Hashtable. Trong chương trình này, tên của các tập ảnh là các khoá, và các năm là các giá trị.

"contains" được sử dụng để tra cứu phần tử nguyên 1969, để thấy có danh sách chứa bất kỳ các tập ảnh từ 1969.

"containsKey" được sử dụng để tìm kiếm cho khoá "Animals", để tìm tập ảnh đó trong bảng băm.

Phương thức "get()" được sử dụng để tìm tập ảnh "Wish You Were Here" có trong bảng băm không. Phương thức get() trả về phần tử cùng với khoá (tên và năm).

Chương trình 4.12

```
import java.util.*;
public class HashTableImplementer
{
 public static void main(String args[])
 {
```

```
//tạo một bảng băm mới
Hashtable ht = new Hashtable();
//thêm các tập ảnh tốt nhất của Pink Floyd
ht.put("Pulse", new Integer(1995));
ht.put("Dark Side of the Moon", new Integer(1973));
ht.put("Wish You Were Here", new Integer(1975));
ht.put("Animals", new Integer(1997));
ht.put("Ummagumma", new Integer(1969));
//Hiển thị bảng băm
System.out.println("Initailly: "+ht.toString());
//kiểm tra cho bất kỳ tập ảnh nào từ 1969
if(ht.contains(new Integer(1969)))
System.out.println("An album from 1969 exists");
//kiểm tra cho tập ảnh các con thú
if(ht.containsKey("Animals"))
        System.out.println("Animals was found");
//Tìm ra
Integer year = (Integer)ht.get("Wish You Were Here");
System.out.println("Wish you Were Here was released in"+year.toString());
//Xoá một tập ảnh
System.out.println("Removing Ummagumma\r\n");
ht.remove("Ummagumma");
//Duyệt qua tất cả các khoá trong bảng.
System.out.println("Remaining:\r\n");
for(Enumeration enum = ht.keys(); enum.hasMoreElements();)
        System.out.println((String)enum.nextElement());
}
}
```

Một phần được thêm vào một lớp Vector bằng cách sử dụng phương thức addElement(). Tương tự, một phần tử có thể được thay thế bằng cách sử dụng phương thức setElementAt(). Một lớp Vector có thể tìm kiếm bằng cách sử dụng phương thức contains(), phương thức này đơn giản chỉ tìm sự xuất hiện của một đối tượng trong Vector. Phương thức elements() trả về một tập hợp các đối tượng được lưu trữ trong lớp Vector. Các phương thức này và các phương thức thành viên khác của lớp Vector được tóm tắt trong bảng dưới đây:

Phương thức	Mục đích
addElement(Object)	Thêm phần tử được chỉ định vào lớp Vector.
capacity()	Trả về dung lượng hiện thời của lớp Vector.
clone()	sao chép lớp vector, nhưng không phải là các phần tử của nó.
contains(Object)	Trả về True nếu lớp Vector chứa đối tượng được chỉ định.
copyInto(Object [])	Sao chép các phần tử của lớp Vector vào mảng được chỉ định.
elementAt(int)	Lấy phần tử vị trí được chỉ định.
elements()	Trả về một bảng liệt kê của các phần tử trong lớp Vector.
ensureCapacity(int)	Đảm bảo rằng lớp Vector có thể lưu trữ ít nhất dung lượng tối thiểu được chỉ định.
firstElement()	Trả về phần tử đầu tiên trong lớp Vector.
indexOf(Object)	Tìm kiếm lớp Vector, và trả về chỉ mục đầu tiên tìm thấy đối tượng.
indexOf(Object, int)	Tìm kiếm lớp Vector bắt đầu từ vị trí chỉ định, trả về vị trí đầu tiên tìm thấy.
insertElementAt(Object, int)	Chèn đối tượng được chỉ định tại vị trí được chỉ định.
isEmpty()	Trả về True nếu lớp Vector không có phần tử.
lastElement()	Trả về phần tử cuối cùng trong lớp Vector.
lastIndexOf(Object)	Tìm kiếm lớp Vector, và trả về chỉ mục của đối tượng tìm thấy cuối cùng.
lastIndexOf(Object, int)	Tìm kiếm lớp Vector bắt đầu tại số chỉ mục được chỉ định, và trả về chỉ mục của phần tử cuối cùng tìm thấy.
removeAllElements()	Xoá tất cả các phần tử từ lớp Vector.
removeElement(Object)	Xoá đối tượng được chỉ định từ lớp Vector.
removeElementAt(int)	Xoá đối tượng tại chỉ mục được chỉ định.
setElementAt(Object, int)	Thay thế đối tượng tại chỉ mục được chỉ định với đối tượng được chỉ định.
setSize(int)	Thiết lập kích thước của lớp Vector thành kích thước mới được chỉ định.
Size()	Trả về số của các phần tử hiện thời trong lớp Vector.
toString()	Trả về một chuỗi chứa nội dung của lớp Vector.

trimToSize()	Định lại kích thước của lớp Vector để di chuyển dung lượng thừa trong nó.

Bảng 4.11 Các phương thức lớp Vector

Chương trình sau tạo ra một lớp Vector vect. Nó chứa 6 phần tử: “Numbers In Words”, “One”, “Two”, “Three”, “Four”, “Five”. Phương thức removeElement()được sử dụng để xoá các phần tử từ vect.

Chương trình 4.13

```
import java.util.*;
public class VectorImplementation
{
 public static void main(String args[])
 {
  Vector vect = new Vector();
  vect.addElement(“One”);
  vect.addElement(“Two”);
  vect.addElement(“Three”);
  vect.addElement(“Four”);
  vect.addElement(“Five”);
  vect.insertElementAt(“Numbers In Words”,0);
  vect.insertElementAt(“Four”,4);
  System.out.println(“Size: “+vect.size());
  System.out.println(“Vector “);
  for(int i = 0; i<vect.size(); i++)
  {
    System.out.println(vect.elementAt(i)+” , “);
  }
          vect.removeElement(“Five”);
           System.out.println(“”);
```

StringTokenizer(String, Strìng)	Tạo ra một đối tượng StringTokenizer mới dựa trên (String, String) chuỗi được chỉ định và một tập các dấu phân cách.
StringTokenizer(String, String, boolean)	Tạo ra một đối tượng StringTokenizer dựa trên chuỗi được chỉ định, một tập các dấu phân cách, và một cờ hiệu cho biết nếu các dấu phân cách sẽ được trả về như các token hay không.

Bảng 4.12 Các phương thức xây dựng của lớp StringTokenizer.

Các phương thức xây dựng ở trên được sử dụng trong các ví dụ sau:

StringTokenizer st1 = new StringTokenizer("A Stream of words");

*StringTokenizer st2 = new StringTokenizer("4*3/2-1+4", "+-*/", true);*

StringTokenizer st3 = new StringTokenizer("aaa,bbbb,ccc", ",");

Trong câu lệnh đầu tiên, StringTokenizer của "st1" sẽ được xây dựng bằng cách sử dụng các chuỗi được cung cấp và dấu phân cách mặc định. Dấu phân cách mặc định là khoảng trắng, tab, các ký tự xuống dòng. Các dấu phân cách này thì chỉ sử dụng khi phân tách văn bản, như với "st1".

Câu lệnh thứ hai trong ví dụ trên xây dựng một đối tượng StringTokenizer cho các biểu thức toán học bằng cách sử dụng các ký hiệu *, +, /, và -.

Câu lệnh thứ 3, StringTokenizer của "st3" sử dụng dấu phẩy như một dấu phân cách.

Lớp StringTokenizer cài đặt giao diện Enumeration. Vì thế, nó bao gồm các phương thức hasMoreElements() và nextElement(). Các phương thức có thể sử dụng của lớp StringTokenizer được tóm tắt trong bảng sau:

Phương thức	**Mục đích**
countTokens()	Trả về số các token còn lại.
hasMoreElements()	Trả về True nếu còn có token đang được đánh dấu trong chuỗi. Nó thì giống hệt như hasMoreTokens.
hasMoreTokens()	Trả về True nếu còn có token đang được đánh dấu trong chuỗi. Nó giống hệt như hasMoreElements.
nextElement()	Trả về token kế tiếp trong chuỗi. Nó thì giống như nextToken.

nextToken()	Trả về Token kế tiếp trong chuỗi. Nó thì giống như nextElement.
nextToken(String)	Thay đổi bộ dấu phân cách bằng chuỗi được chỉ định, và sau đó trả về token kế tiếp trong chuỗi.

Bảng 4.13 Các phương thức lớp StringTokenizer.

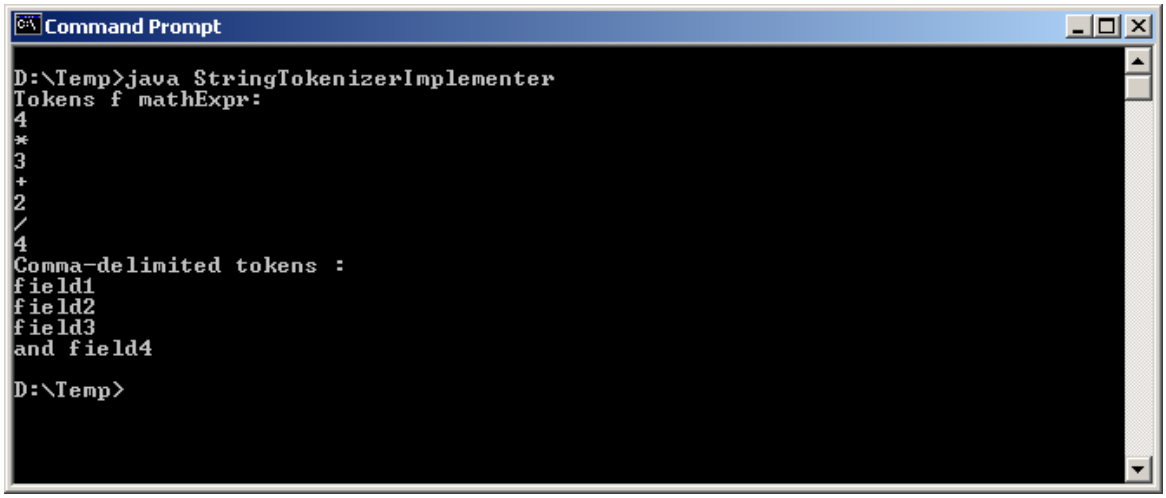
Hãy xem xét chương trình đã cho ở bên dưới. Trong ví dụ này, hai đối tượng StringTokenizer đã được tạo ra. Đầu tiên, "st1" được sử dụng để phân tách một biểu thức toán học. Thứ hai, "st2" phân tách một dòng của các trường được phân cách bởi dấu phẩy. Cả hai tokenizer, phương thức hasMoreTokens() và nextToken() được sử dụng đế duyệt qua tập các token, và hiển thị các token.

Chương trình 4.13

```
import java.util.*;
public class StringTokenizerImplementer
{
        public static void main(String args[])
 {
         // đặt một biểu thức toán học và tạo một tokenizer cho chuỗi đó.
         String mathExpr = "4*3+2/4";
         StringTokenizer st1 = new StringTokenizer(mathExpr,"*+/-", true);
         //trong khi vẫn còn các token, hiển thị  System.out.println("Tokens of  mathExpr: ");
 while(st1.hasMoreTokens())
         System.out.println(st1.nextToken());
         //tạo một chuỗi của các trường được phân cách bởi dấu phẩy và tạo //một tokenizer
        cho chuỗi.
         String commas = "field1,field2,field3,and field4";
 StringTokenizer st2 = new StringTokenizer(commas,",",false);
//trong khi vẫn còn token, hiển thị.
         System.out.println("Comma-delimited tokens : ");
        while (st2.hasMoreTokens())
```

```
            System.out.println(st2.nextToken());
    }
}
```

Kết quả chạy chương trình được mô tả như hình dưới.



Hình 4.6 Kết quả chạy chương trình minh hoạ lớp StringTokenizer.

Kiểm tra sự tiến bộ

1. ……luôn là lệnh đầu tiên trước các lệnh: import, class trong chương trình Java.
2. Một giao diện có thể chứa nhiều các phương thức. **Đúng/Sai**
3. Trong khi tạo gói, thì mã nguồn phải nằm trong thư mục có tên như tên gói. **Đúng/Sai**
4. ……..là một danh sách của các thư mục, mà JVM sẽ tìm kiếm các tập tin lớp.
5. Lớp bao bọc (wrapper) cho các kiểu dữ liệu double và long cung cấp hai hằng số là …….và……...
6. ……phương thức được sử dụng để thay thế một ký tự trong lớp StringBuffer bằng một ký tự khác tại vị trí được chỉ định.
7. Phương thức……. của lớp StringTokenizer trả về số token còn lại.

Bài tập

1. Tạo một giao diện và sử dụng nó trong một chương trình của Java để hiển thị bình phương và luỹ thừa 3 của một số.
2. Tạo một gói và viết một hàm, hàm đó trả về giai thừa của một đối số được truyền vào trong một chương trình.
3. Viết một chương trình bằng cách sử dụng các hàm của lớp Math để hiển thị bình phương của các số lớn nhất và nhỏ nhất của một tập các số được nhập vào bởi người sử dụng tại dòng lệnh.

4. Hãy tạo ra sổ ghi nhớ của chính bạn, nơi mà những con số được nhập vào như sau:

Joy	34543
Jack	56765
Tina	34567

Chương trình phải làm như sau:

- Kiểm tra xem số 3443 có tồn tại trong sổ ghi nhớ của bạn hay không.
- Kiểm tra xem mẫu tin của Jack có hiện hữu trong sổ ghi nhớ của bạn hay không.
- Hiển thị số điện thoại của Tina.
- Xoá số điện thoại của Joy.
- Hiển thị các mẫu tin còn lại.

5. Viết một chương trình mà nhập vào một số điện thoại tại dòng lệnh, như một chuỗi có dạng (091) 022-6758080. Chương trình sẽ hiển thị mã quốc gia (091), mã vùng (022), và số điện thoại (6758080) (Sử dụng lớp StringTokenizer).

```
    }
    Socket t = new Socket(host, 8008);
    BufferedReader in
          = new BufferedReader(new InputStreamReader(t.getInputStream()));
    PrintWriter out
          = new PrintWriter(new OutputStreamWriter(t.getOutputStream()));

    for (int i = 1; i <= 10; i++) {
          System.out.println("Sending: line " + i);
          out.println("line " + i);
          out.flush();
    }
    out.println("BYE");
    out.flush();

    for (;;) {
      String str = in.readLine();
      if (str == null) {
        break;
      } else {
        System.out.println(str);
      }
    }
   } catch (Exception e) {
    System.out.println("Error: " + e);
   }
  }

}
```

Biên dịch chương trình: *javac EchoClient.java*

Hãy chạy chương trình EchoServer trên một máy và EchoClient trên máy khác nếu hệ thống có mạng. Nếu thử nghiệm trên một máy đơn ta có thể mô phỏng mô hình khách/chủ bằng cách mở hai cửa sổ dòng lệnh DOS. EchoClient sẽ chạy trên một cửa sổ còn EchoServer chạy trên cửa sổ khác, quan sát kết quả trả về để thấy được mô hình khách/chủ làm việc (Chú ý là chạy EchoServer trước rồi mới đến EchoClient).

- Lớp ExchangeRateTable

Là lớp chương trình dùng tạo một cửa sổ trình bày tỷ giá của các thị trường chứng khoán trong một vùng căn bản TextArea.

- Lớp ExchangeThread

Đây là một phân tuyến chạy song song với chương trình chịu trách nhiệm mỗi giây sẽ gửi yêu cầu đến máy chủ và lấy số liệu về cập nhật.

- Lớp ExchangeData

Là lớp thực hiện kết nối với máy chủ, lớp này thực sự cài đặt phương thức getRates dùng để gửi yêu cầu và nhận dữ liệu trả về từ máy chủ.

Ví dụ 2.2: ExchangeRateTable.java

```
import java.io.*;
import java.awt.*;
import java.awt.event.*;
import java.net.*;
public class ExchangeRateTable {
      public static void main (String args[]) {
            Frame myWindow = new Frame("Stock Exchange Application");
            TextArea rateTable = new TextArea("Wait ...");
            Label rateLabel = new Label("Exchange Rate Table");
            rateTable.setBounds(new Rectangle(16,33,240,100));
            rateLabel.setBounds(new Rectangle(16,6,158,21));
            myWindow.setLayout(null);
            myWindow.add(rateTable, null);
            myWindow.add(rateLabel,null);

            myWindow.addWindowListener(new WindowAdapter(){
            public void windowClosing(WindowEvent event){
          System.exit(0);
            }
            });

      myWindow.setSize(new Dimension(300,150));
      myWindow.show();

      ExchangeThread exRate = new ExchangeThread(rateTable);
      exRate.start();
    }
    }
```

```
class ExchangeThread extends Thread {
TextArea rateTable;
ExchangeData rate = new ExchangeData();

public ExchangeThread(TextArea rateTable){
this.rateTable = rateTable;
}
public void run(){
while (true)
{
String data = rate.getRates();
rateTable.setText(data);
delay(1000);
}
}
private void delay(int miliSeconds){
try
{
this.sleep(miliSeconds);
}
catch (Exception e)
{
System.out.println("Sleep error!");
}
}
}
class ExchangeData
{
DatagramSocket socket;
InetAddress serverAddress;
String localHost;
int bufferLength = 256;
byte inBuffer[] = new byte[bufferLength];
byte outBuffer[];
DatagramPacket outDatagram;
DatagramPacket inDatagram;

public ExchangeData(){
try
{
socket = new DatagramSocket();
inDatagram = new DatagramPacket(inBuffer, inBuffer.length);
```

```
<HTML>
<H1>
    Hello JSP!
</H1> <P>
<%
    out.println("Bay gio la: "+new java.util.Date()+"<br>");
    for(int i=6;i>0;i--){
    out.println("<FONT Size="+i+">Hello</FONT>");
    }
%>
</HTML>
```

Chúng ta có thể sử dụng trình soạn thảo Notepad của Windows, hoặc EditPlus, UltraEdit … để tạo ra file FirstPage.jsp ở trên.

Để chạy được file FirstPage.jsp chúng ta cần đến một trình chủ Web server hiểu và diễn dịch được JSP/Servlet. Hiện có rất nhiều trình chủ có khả năng này như: TomCat, Java Web Server, Jrun, WebLogic … Tuy nhiên cách cài đặt và cấu hình mỗi trình chủ rất khác biệt, ở đây chúng ta sử dụng trình chủ Jrun 3.1. Đây là trình chủ Web server hỗ trợ rất nhiều công nghệ Java, có cấu hình đơn giản, chạy được hầu hết trên các hệ điều hành Unix, Linux, Windows. Tốc độ thông dịch trang JSP của trình chủ JSP khá nhanh và hiệu quả, giao diện đẹp. Với Jrun, gọi thư mục cài đặt trình chủ là [JRUN_HOME] (thường là thư mục C:/Program Files/Allaire/JRun), chép file FirstPage.jsp vào thư mục [JRUN_HOME]/servers/default/default-app. Sau khi khởi tạo dịch vụ Web của trình chủ Jrun hoạt động trên cổng 8100 (khởi tạo trong quá trình cài đặt JRun), trang FirstPage.jsp được triệu gọi từ trình duyệt phía máy khách theo địa chỉ URL như sau: http://localhost:8100/FirstPage.jsp, kết quả sẽ được hiện thị trên trình duyệt.

```
<%
 // lấy biến dữ liệu mang tên user từ đối tượng session
String username=session.getAttribute("user");
%>
Welcome <%= username%>
<%
  /*
     Tạo đối tượng Date
      Hiển thị đối tượng Date cho biết ngày giờ cập nhật của trang Web
*/
java.util.Date date= new java.util.Date();
%>
This page last update on <%=date%>
</HTML>
```

JSP còn cung cấp thêm cho bạn cú pháp chú thích <%-- --%>. Tất cả các khối lệnh Java và HTML nằm giữa hai dấu chú thích này sẽ được trình biên dịch trang bỏ qua không quan tâm đến. Ví dụ:

```
<%--
      out.println("You will never see this line");
--%>
```

Dấu chú thích này rất hiệu quả. Nó giúp bạn tạm thời cô lập hoặc che bỏ tác dụng của một đoạn mã Java nào đó đang bị lỗi trong trang JSP. Chúng ta chỉ tạm thời làm mất tác dụng của chúng chứ không cần xóa bỏ. Ví dụ:

```
<HTML>
<%--
<%
 String username=session.getAttribute("user");
%>
Welcome <%= username%>
--%>
```

```
<%!
        private ResultSet query(String driverName, String connectionURL, String query)
{
        Connection con = null;
        Statement stmt = null;
        ResultSet rs = null;
        try {
        Class.forName(driverName).newInstance();
        con = DriverManager.getConnection(connectionURL);
        stmt = con.createStatement();
        rs = stmt.executeQuery(query);
        return rs;
}
        catch(Exception es){
        out.println("Connection error!");
        }
        return rs;
}
%>
<%
        ResultSet rs = query("sun.jdbc.odbc.JdbcOdbcDriver",
"jdbc:odbc:MyAccessDataSource",
         "SELECT Empno, Name, Position FROM Employee");
        if (rs==null){
        out.println("<center><table border>");
        out.println("<tr>");
        for (int i=0; i<columnCount;i++){
        out.println("<th>" + rmsd.getColumnLabel(i+1) + "</th>");
        }
        out.println("</tr>");
```

```
        while (rs.next()){
        out.println("<tr>");
        for (int i=0; i<columnCount;i++){
        out.println("<td>" + rs.getString(i+1) + "</td>");
        }
        out.println("</table></center>");
}
%>
</HTML>
```

Chạy thử chương trình, gõ địa chỉ URL: http://localhost:8100/EmployeeList.jsp

Lập Trình Mạng Căn Bản

MỞ ĐẦU

Ngày nay, cho dù là hệ thống mạng nhỏ, cỡ vừa và lớn thì mỗi máy tính đều sử dụng địa chỉ IP động được cấp phát từ dịch vụ DHCP server.

Mạng nhỏ thì DHCP cấp IP động cho các máy trạm (làm việc trong môi trường Workgroup) nằm ở các thiết bị mạng (Switch, Modem, Router, AP ...). Còn các mạng lớn (mô hình mạng sẽ phức tạp hơn), các máy trạm nằm trong môi trường Domain. Khi đó các nhà quản trị mạng họ dùng ngay dịch vụ DHCP server có sẵn trên Windows Server 2003, 2008 để cấp phát IP động cho các máy trạm trong hệ thống mạng thay vì sử dụng DHCP server tích hợp sẵn trong các thiết bị mạng phần cứng. Vậy DHCP là gì? Nó có ưu điểm gì?

Dynamic Host Configuration Protocol (DHCP - giao thức cấu hình động máy chủ) là một giao thức cấu hình tự động địa chỉ IP. Máy tính được cấu hình một cách tự động vì thế sẽ giảm việc can thiệp vào hệ thống mạng. Nó cung cấp một database trung tâm để theo dõi tất cả các máy tính trong hệ thống mạng. Mục đích quan trọng nhất là tránh trường hợp hai máy tính khác nhau lại có cùng địa chỉ IP.
Nếu không có DHCP, các máy có thể cấu hình IP thủ công. Ngoài việc cung cấp địa chỉ IP, DHCP còn cung cấp thông tin cấu hình khác, cụ thể như DNS. Hiện nay DHCP có 2 version: cho IPv4 và IPv6.

Nói một cách tổng quan hơn DHCP là dich vụ mang đến cho chúng ta nhiều lợi điểm trong công tác quản trị và duy trì một mạng TCP/IP như:
+ Tập chung quản trị thông tin về cấu hình IP.
+ Cấu hình động các máy.
+ Cấu hình IP cho các máy một cách liền mạch
+ Sự linh hoạt
+ Khả năng mở rộng.

Nhiệm vụ của từng thành viên trong nhóm:

1. Nguyễn Xuân Toàn 09520312

   - Xây dựng DHCP Server

2. Nguyễn Anh Vũ 09520358

   - Xây dựng DHCP Server

3. Phạm Xuân Mạnh 09520422

   - Xây dựng DHCP Client và thiết kế giao diện.

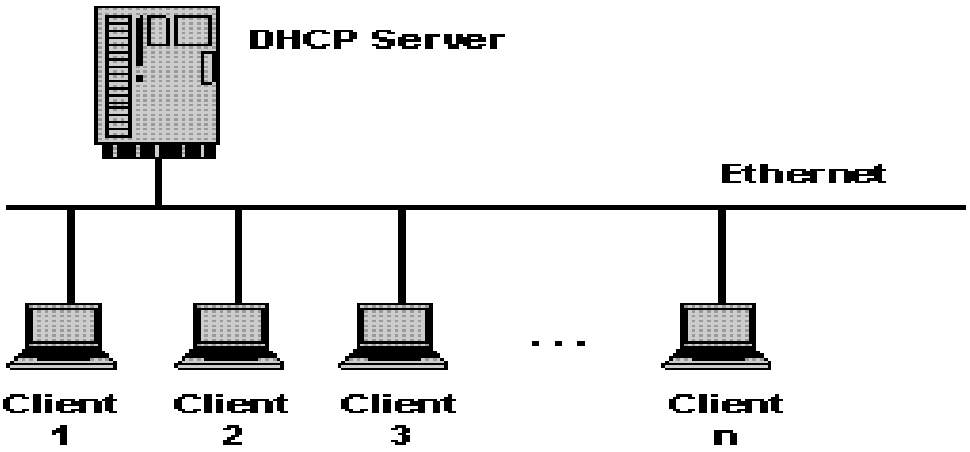
MỤC LỤC

Trang

Chương I: KHẢO SÁT BÀI TOÁN

1.1. DHCP là gì?

❖ DHCP là viết tắt của Dynamic Host Configuration Protocol, là giao thức Cấu hình Host Động được thiết kế làm giảm thời gian chỉnh cấu hình cho mạng TCP/IP bằng cách tự động gán các địa chỉ IP cho khách hàng khi họ vào mạng. Dich vụ DHCP là một thuận lới rất lớn đối với người điều hành mạng. Nó làm yên tâm về các vấn đề cố hữu phát sinh khi phải khai báo cấu hình thủ công. Nó bao gồm DHCP Server và DHCP Client.



Hình 1.1. Mô hình DHCP

- DHCP server là một máy chủ có cài đặt dịch vụ DHCP server. Nó có chức năng quản lý sự cấp phát địa chỉ IP động và các dữ liệu cấu hình TCP/IP. Ngoài ra còn có nhiệm vụ trả lời khi DHCP Client có yêu cầu về hợp đồng thuê bao.
- DHCP client là dịch vụ nằm cục bộ trên máy tính (kể cả máy trạm và server). Nó dùng để đăng ký, cập nhật thông tin về địa chỉ IP và các bản ghi DNS cho chính bản thân nó. DHCP client sẽ gửi yêu cầu đến DHCP server khi nó cần đến 1 địa chỉ IP và các tham số TCP/IP cần thiết để làm việc trong mạng nội bộ và trên Internet.

1.2. DHCP làm việc như thế nào?

❖ DHCP tự động quản lý các địa chỉ IP và loại bỏ được các lỗi có thể làm mất liên lạc. Nó tự động gán lại các địa chỉ chưa được sử dụng và cho thuê địa chỉ trong một khoảng thời gian.

❖ Trình tự thuê Địa chỉ IP DHCP là một giao thức Internet có nguồn gốc ở BOOTP (bootstrap protocol), được dùng để cấu hình các trạm không đĩa. DHCP khai thác ưu điểm của giao thức truyền tin và các kỹ thuật khai báo cấu hình được định nghĩa trong BOOTP, trong đó có khả năng gán địa chỉ. Sự tương tự này cũng cho phép các bộ định tuyến hiện nay chuyển tiếp các thông điệp BOOTP giữa các mạng con cũng có thể chuyển tiếp các thông điệp DHCP. Vì thế, máy chủ DHCP có thể đánh địa chỉ IP cho nhiều mạng con.

1.3. Lợi ích của việc sử dụng DHCP

1.3.1. Quản lý TCP/IP tập trung

Thay vì phải quản lý địa chỉ IP và các tham số TCP/IP khác vào một cuốn sổ nào đó (đây là việc mà quản trị mạng phải làm khi cấu hình TCP/IP bằng tay) thì DHCP server sẽ quản lý tập trung trên giao diện của nó. Giúp các nhà quản trị vừa dễ quản lý, cấu hình, khắc phục khi có lỗi xảy ra trên các máy trạm.

1.3.2. Giảm gánh nặng cho các nhà quản trị hệ thống

Thứ nhất, trước đây các nhà quản trị mạng thường phải đánh cấu hình IP bằng tay (gọi là IP tĩnh) nhưng nay nhờ có DHCP server nó sẽ cấp IP một cách tự động cho các máy trạm. Nhất là trong môi trường mạng lớn thì sự cần thiết và hữu ích của dịch vụ mạng này mới thấy rõ ràng nhất.

Thứ hai, trước đây với kiểu cấu hình bằng tay thì người dùng họ có thể thay đổi IP. Có người thay đổi lung tung DNS server sau đó quên không nhớ IP của DNS server là gì để đặt lại cho đúng lại với quản trị mạng, có người đặt IP làm trùng với IP của người khác, người khác lại đặt IP trùng với Defaul Gateway ... làm cho quản trị mạng khốn khổ vì phải chỉnh sửa. Nhưng vấn đề này không có ở IP động. Chỉ có người quản trị DHCP server họ mới có quyền làm những việc trên.

1.3.3. Giúp hệ thống mạng luôn được duy trì ổn định

Địa chỉ IP cấp phát động cho các máy trạm lấy từ dải IP cấu hình sẵn trên DHCP server. Các tham số (DG, DNS server ...) cũng cấp cho tất cả các máy trạm là chính xác. Sự trùng lặp IP không bao giờ xảy ra. Các máy trạm luôn luôn có một cấu hình TCP/IP chuẩn. Làm cho hệ thống hoạt động liên tục, vừa giảm gánh nặng cho người quản trị vừa tăng hiệu quả làm việc cho user nói riêng và doanh nghiệp nói chung.

1.3.4. Linh hoạt và khả năng mở rộng

Người quản trị có thể thay đổi cấu hình IP một cách dễ dàng khi cơ sở hạ tầng mạng thay đổi. Do đó làm tăng sự linh hoạt cho người quản trị mạng. Ngoài ra DHCP phù hợp từ mạng nhỏ đến mạng lớn. Nó có thể phục vụ 10 máy khách cho đến hàng ngàn máy khách.

Chương II: CƠ SỞ LÝ THUYẾT

2.1. Địa chỉ IP động

2.1.1. Địa chỉ IP động đặc biệt là gì ?

- Địa chỉ IP động đặc biệt (Automatic private IP Addressing) hay APIPA là một dặc trưng của hệ điều hành Microsoft windows cho phép gán một dải địa chỉ IP tự động trên các máy Client dải địa chỉ này có giá trị trong khoảng từ: 169.254.0.0 đến 169.254.255.255. khi mà dịch vụ DHCP server không được phép cấp phát IP cho các máy Client.

2.1.2. Cách thức cấp phát địa chỉ IP động

- Dịch vụ DHCP sẽ thiết lập Hợp đồng thuê địa chỉ IP và ra gạn hợp đồng cho thuê địa chỉ IP nhằm cấp phát địa chỉ IP động cho các máy Client.

2.2. Cấu hình phạm vi cấp phát của dịch vụ DHCP

2.2.1. Phạm vi cấp phát DHCP là gì ?

- Phạm vi cấp phát là dải địa chỉ IP hợp lệ được dùng để thuê hoặc gán cho các máy Client trên cùng một Subnet. Ta cấu hình một pham vi cấp phát trên DHCP Server để xác định dải địa chỉ IP mà các Server có thể gán cho các máy Client.

2.2.2. Tại sao phải sử dụng phạm vi cấp phát DHCP ?

- Phạm vi cấp phát sẽ xác định xem những địa chỉ IP nào sẽ được phép cấp phát cho các máy client.

2.3. Cấu hình địa chỉ DHCP giành sẵn (Reservations) và các tùy chọn của DHCP

2.3.1. Địa chỉ DHCP giành sẵn là gì ?

- Địa chỉ IP dành sẵn là một dải địa chỉ IP được gán cố định. Nó là dải địa chỉ IP được tạo ra trong một phạm vi (scope) dành riêng, dải địa chỉ IP dành riêng này được dùng để gàn cho các máy Client (Chúng là các địa chỉ IP tĩnh trong mạng được gán cho các máy client).

2.3.2. Các thông tin của một địa chỉ DHCP giành sẵn

- Reservation Name :Là tên được gán bởi nhà quản trị.
- IP Address : Là phạm vi dải địa chỉ IP được gán cho các máy Client.
- MAC Address: Là địa chỉ MAC của mỗi thiết bị mà bạn muốn dành sẵn một địa chỉ IP cho nó.
- Description: Những mô tả do nhà quản trị đưa ra.
- Supported Type:Kiểu hỗ trợ này có thế là: DHCP dành sẵn, BOOTP dành sẵn, hoặc cả hai.

2.3.3. Tùy chọn DHCP là gì ?

Các tùy chọn DHCP là các tham số cấu hình máy khách bổ sung mà một máy chủ DHCP có thể gán khi phục vụ các máy khách DHCP.

2.3.4. Tại sao phải sử dụng tùy chọn DHCP ?

Các tùy chọn DHCP được cấu hình sử dụng bảng điều khiển DHCP và có thể được áp dụng cho nhiều phạm vi và sự dành sẵn.Một tùy chọn

DHCP sẽ làm tăng thêm các chức năng cho hệ thống mạng. Các tùy chọn DHCP cho phép bạn thêm dữ liệu cấu hình IP các client.

2.3.5. Một số tùy chọn chung của DHCP

- Router (Default Gateway): Địa chỉ của bất cứ cổng ra mặc định (default gateway) hay bộ định tuyến (router) nào.
- DNS Domain Name: Tên miền DNS xác định miền mà máy khách sẽ phụ thuộc. Máy khách có thể sử dụng thông tin này để cập nhật thông tin lên máy chủ DNS để các máy tính khác có thể tìm thấy nó.
- DNS Servers: Địa chỉ của bất cứ máy chủ DNS nào mà máy khách có thể sử dụng trong quá trình truyền thông.
- WINS Servers: Địa chỉ của bất cứ máy chủ WINS nào mà máy khách có thể sử dụng trong quá trình truyền thông.
- WINS Node Type: Là một kiểu phương thức phân giải tên NetBIOS mà các máy khách (Client) có thế sử dụng.

2.4. Cấu hình DHCP - DHCP Relay Agent

2.4.1. DHCP relay agent là gì ?

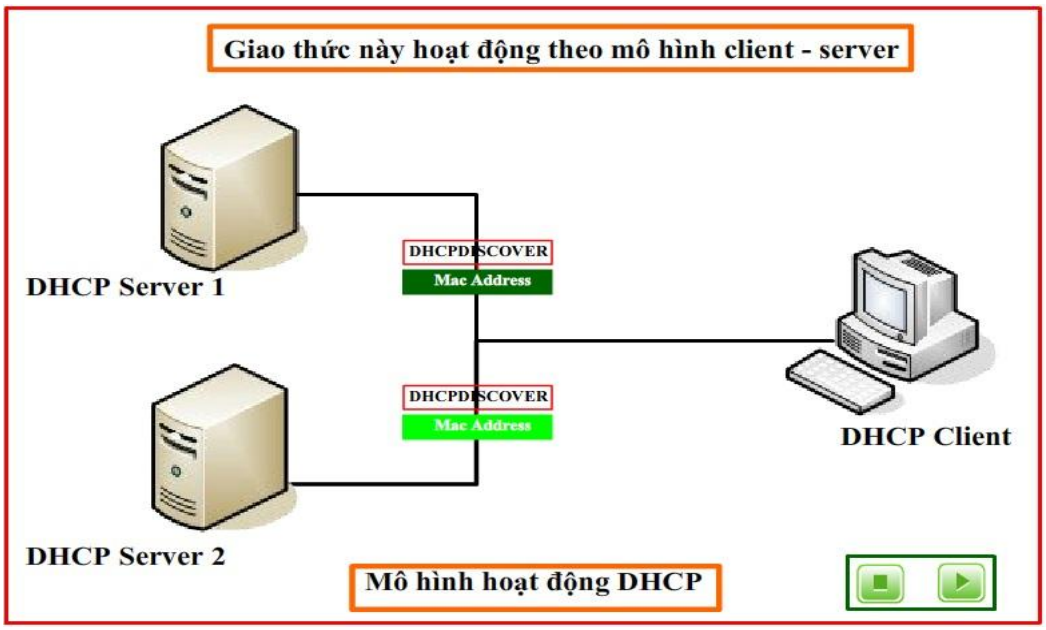
- DHCP relay agent là một máy tính hoặc một Router được cấu hình để lắng nghe các thông điệp quảng bá DHCP/BOOTP từ DHCP client, chuyển tiếp các thông điệp quảng bá từ một giao tiếp này đến giao tiếp khác. và hướng chúng tới một hoặc nhiều máy chủ DHCP cụ thể.

2.4.2. Tại sao phải sử dụng DHCP relay agent ?

- DHCP clients sử dụng một địa chỉ quảng bá (broadcast) để bảo mật việc thuê một địa chỉ IP từ DFHCP server. Một Router thong thường không thể thông qua một broadcast trừ khi nó được cấu hình riêng biệt.
- Tiết kiệm được số địa chỉ IP thật (Public IP)
- Phù hợp với các máy tính thường xuyên di chuyển giữa các lớp mạng.
- Kết hợp với hệ thống mạng không dây (Wireless) cung cấp tại các điểm. Hotspot như: nhà ga, sân bay, khách sạn, trường học.
- Thuận tiện cho việc bổ xung các thiết bị mới vào lớp mạng.

2.5. Phương thức hoạt động của dịch vụ DHCP

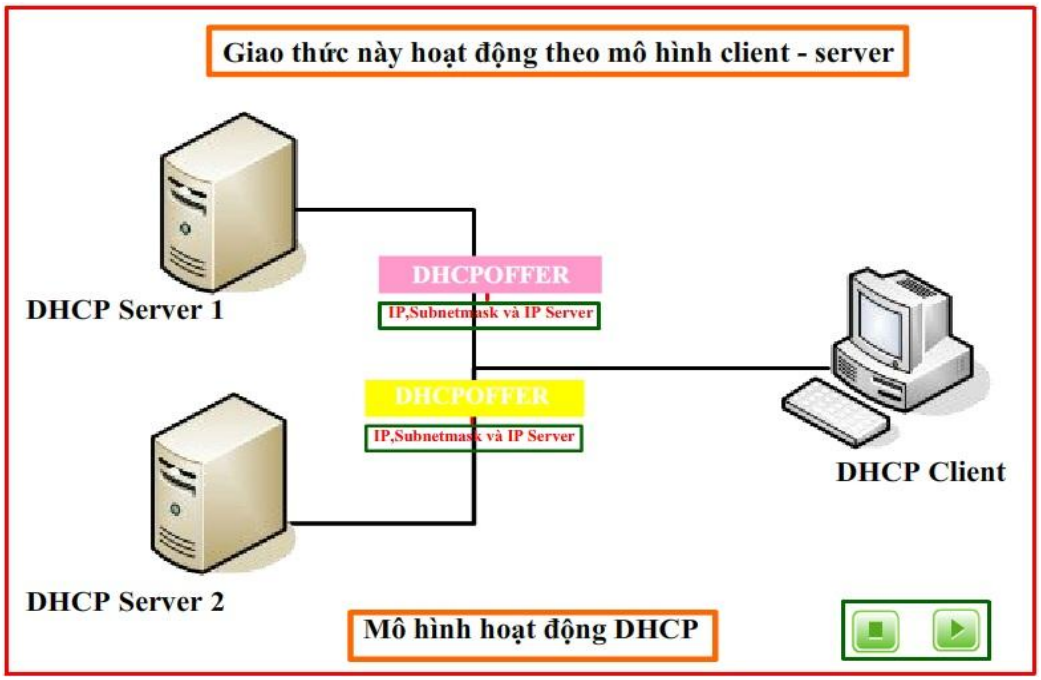
- **Bước 1**: Khi máy Client khởi động, máy sẽ gửi broadcast gói tin DHCP DISCOVER, yêu cầu một Server phục vụ mình. Gói tin này cũng chứa địa chỉ MAC của client.



Hình 2.5.1. Mô phỏng quá trình Client gửi DHCP Discover cho Server

Nếu client không liên lạc được với DHCP Server thì sau 4 lần truy vấn không thành công nó sẽ tự động phát sinh ra 1 địa chỉ IP riêng cho chính mình nằm trong dãy 169.254.0.0 đến 169.254.255.255 dùng để liên lạc tạm thời. Và client vẫn duy trì việc phát tín hiệu Broad cast sau mỗi 5 phút để xin cấp IP từ DHCP Server.

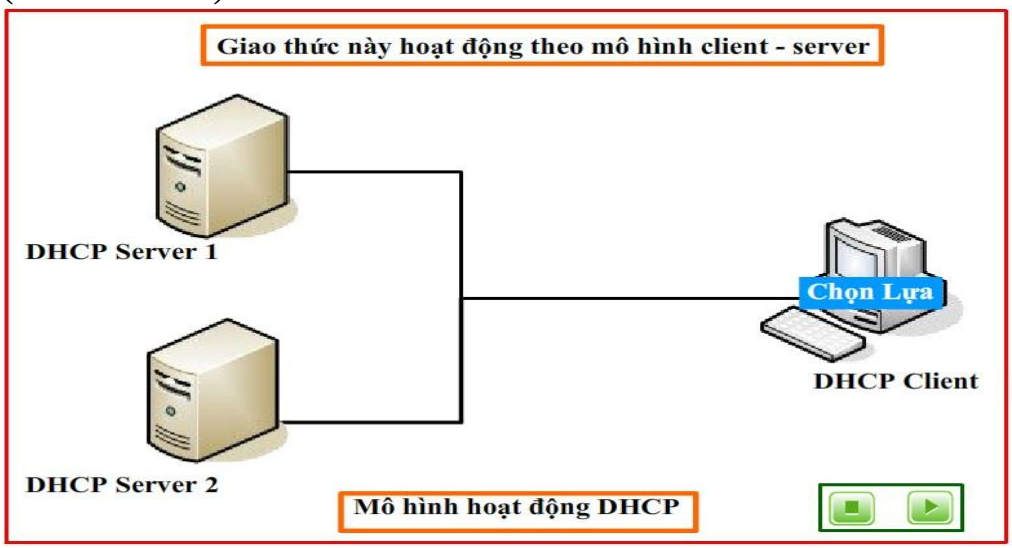
❖ **Bước 2**: Các máy Server trên mạng khi nhận được yêu cầu đó. Nếu còn khả năng cung cấp địa chỉ IP, đều gửi lại cho máy Client một gói tin DHCP OFFER, đề nghị cho thuê một địa chỉ IP trong một khoảng thời gian nhất định, kèm theo là một Subnet Mask và địa chỉ của Server.



Hình 2.5.2. Mô phỏng quá trình Server gửi lại DHCPOFFER cho client

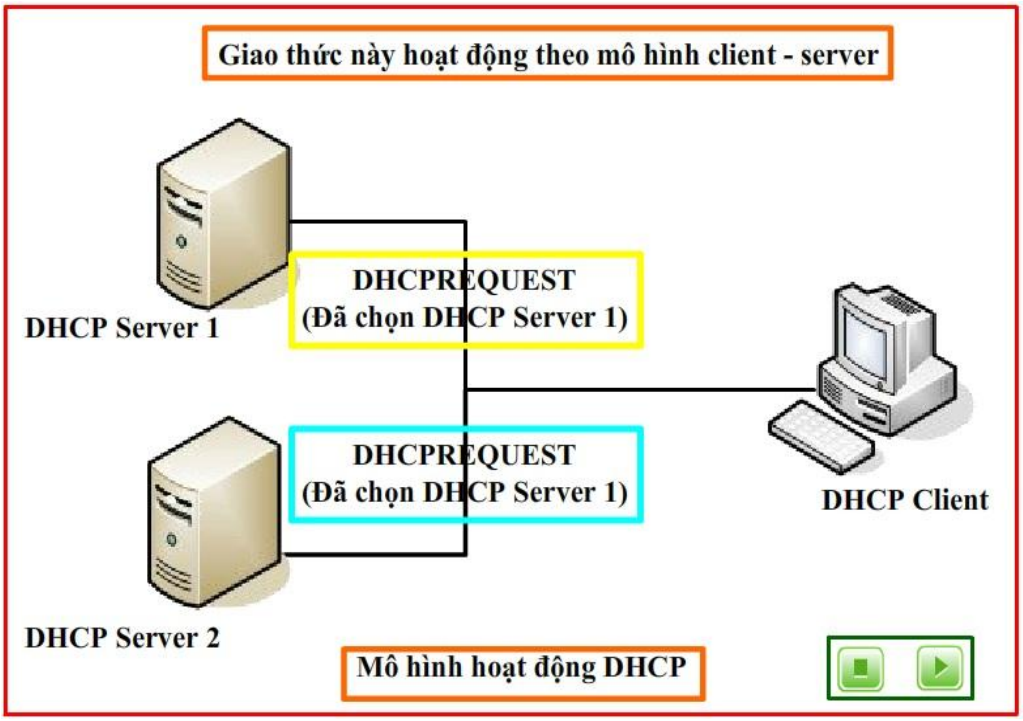
Server sẽ không cấp phát đia chỉ IP vừa đề nghị cho client thuê trong suốt thời gian thương thuyết.

❖ **Bước 3**:Máy Client sẽ lựa chọn một trong những lời đề nghị (DHCPOFFER)



Hình 2.5.3. Mô phỏng quá trình Client lựa chọn Server

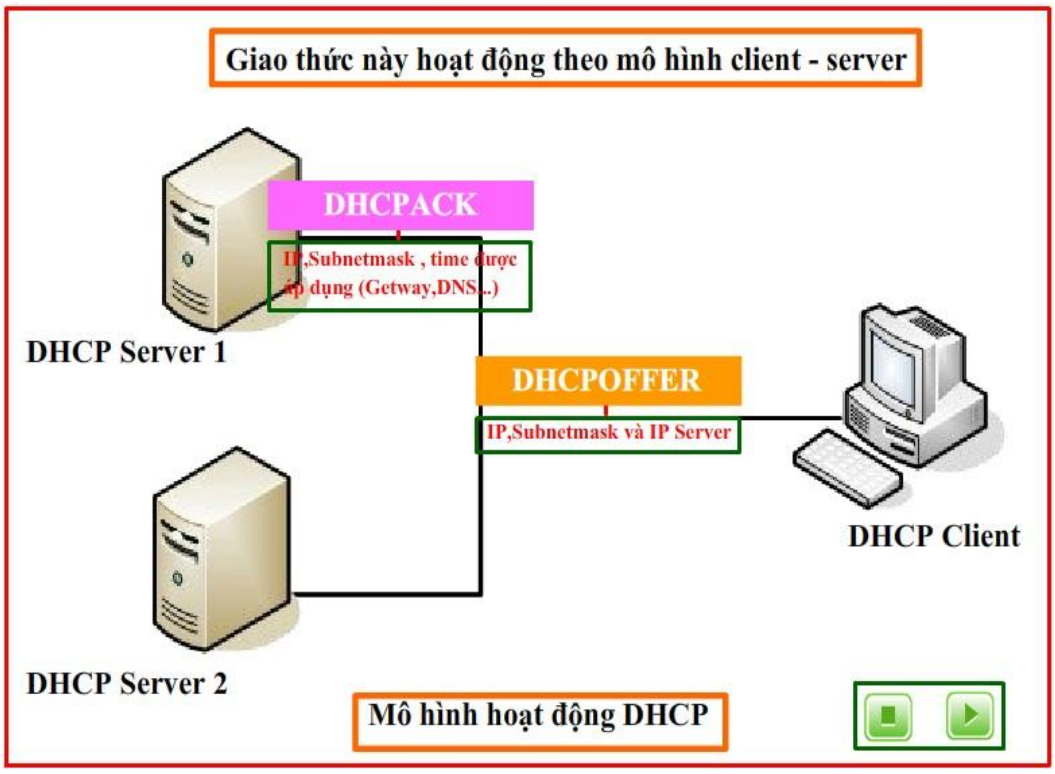
và gửi broadcast lại gói tin DHCPREQUEST và chấp nhận lời đề nghị đó.



Hình 2.5.4. Mô phỏng quá trình Client gửi DHCPREQUEST cho Server

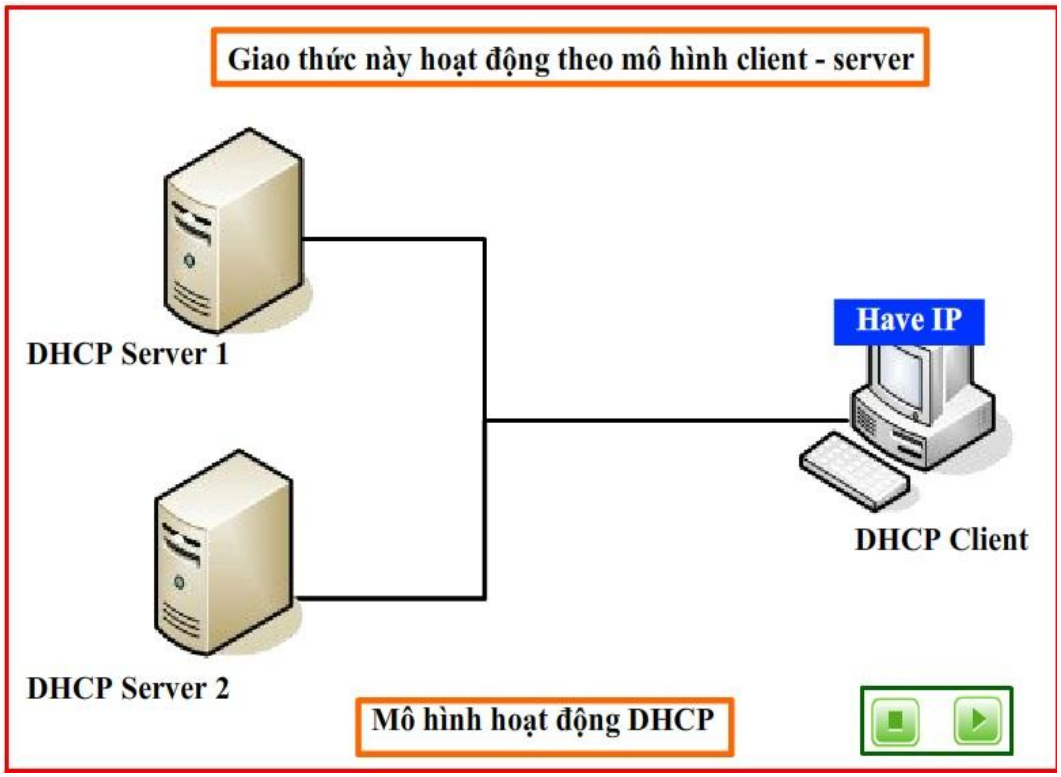
Điều này cho phép các lời đề nghị không được chấp nhận sẽ được các Server rút lại và dùng để cấp phát cho các Client khác.

❖ **Bước 4**: Máy Server được Client chấp nhận sẽ gửi ngược lại một gói tin DHCP ACK như một lời xác nhận, cho biết địa chỉ IP đó, Subnet Mask đó và thời hạn cho sử dụng đó sẽ chính thức được áp dụng.



Hình 2.5.5. Mô phỏng quá trình Server gửi DHCPACK cho client

Ngoài ra server còn gửi kèm những thông tin bổ sung như địa chỉ Gateway mặc định, địa chỉ DNS Server...



Hình 2.5.6. Mô phỏng quá trình Client nhận được IP mới từ Server

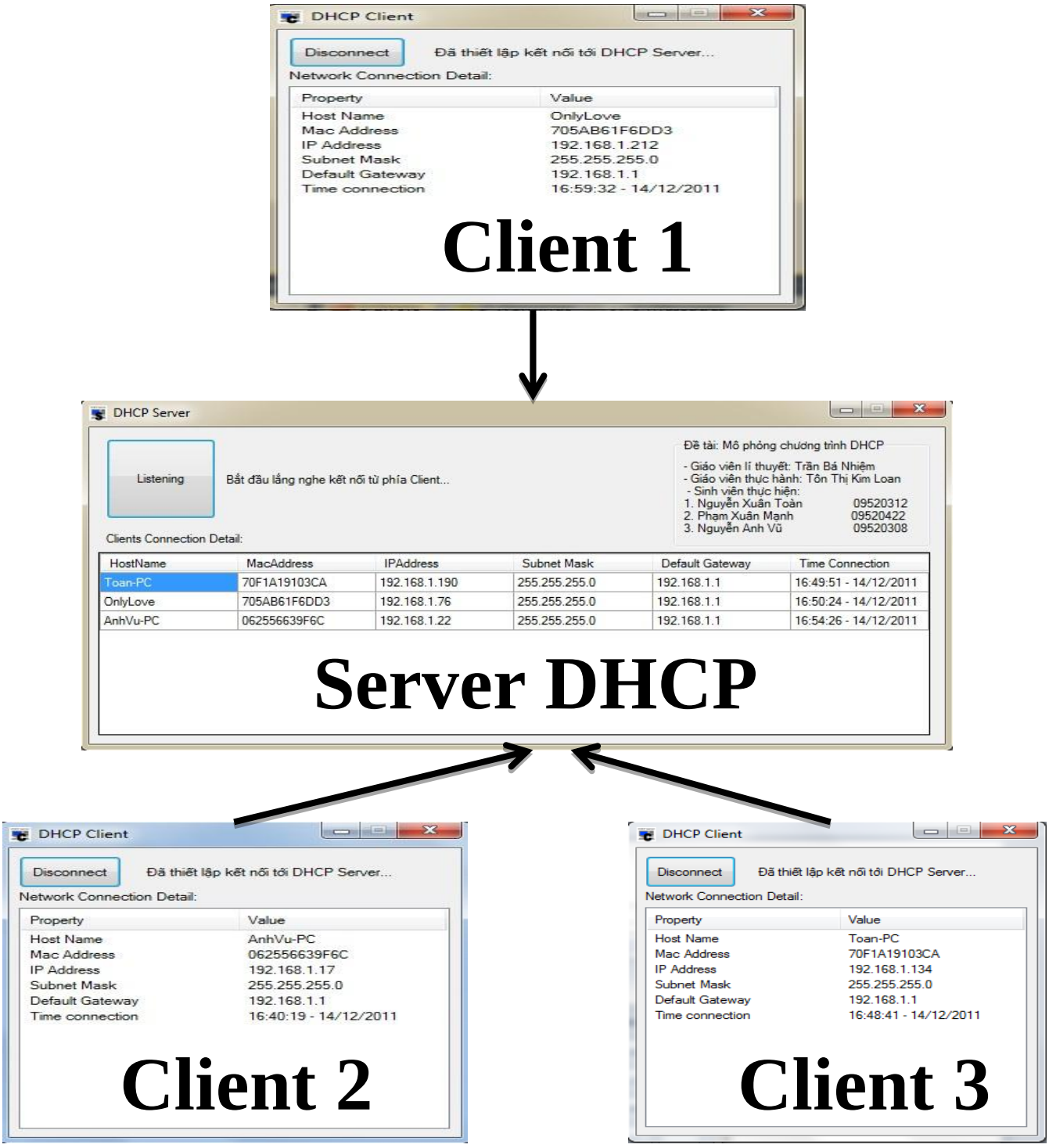
(Nhưng cũng có thể Server sẽ gửi gói tin DHCP NAK (Negative Acknowledgment) nếu lời đề nghị lúc đầu không chính xác nữa hoặc thông

số IP đó đã có máy tính khác sử dụng. Và lúc này Client lại phải bắt đầu lại quy trình xin cấp IP từ đầu)

CHƯƠNG III: PHÂN TÍCH THIẾT KẾ CHƯƠNG TRÌNH

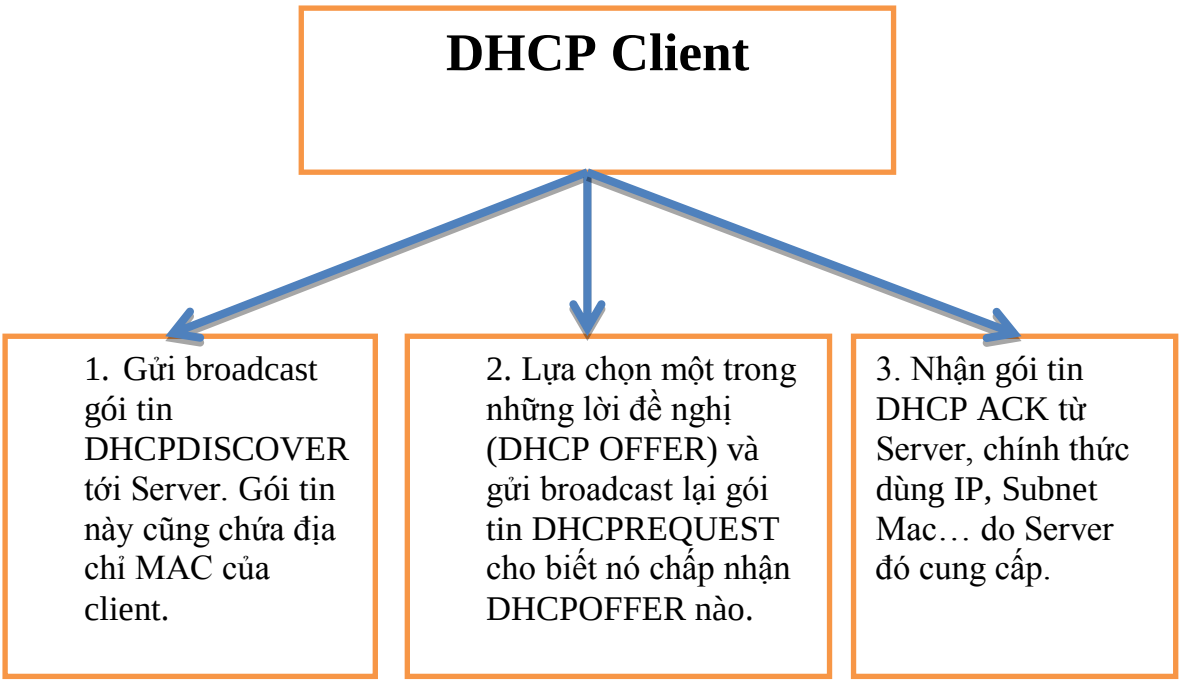
3.1. Mô hình thực nghiệm của chương trình mô phỏng chương trình DHCP

- Hình dưới đây mô tả 3 client kết nối tới DHCP server

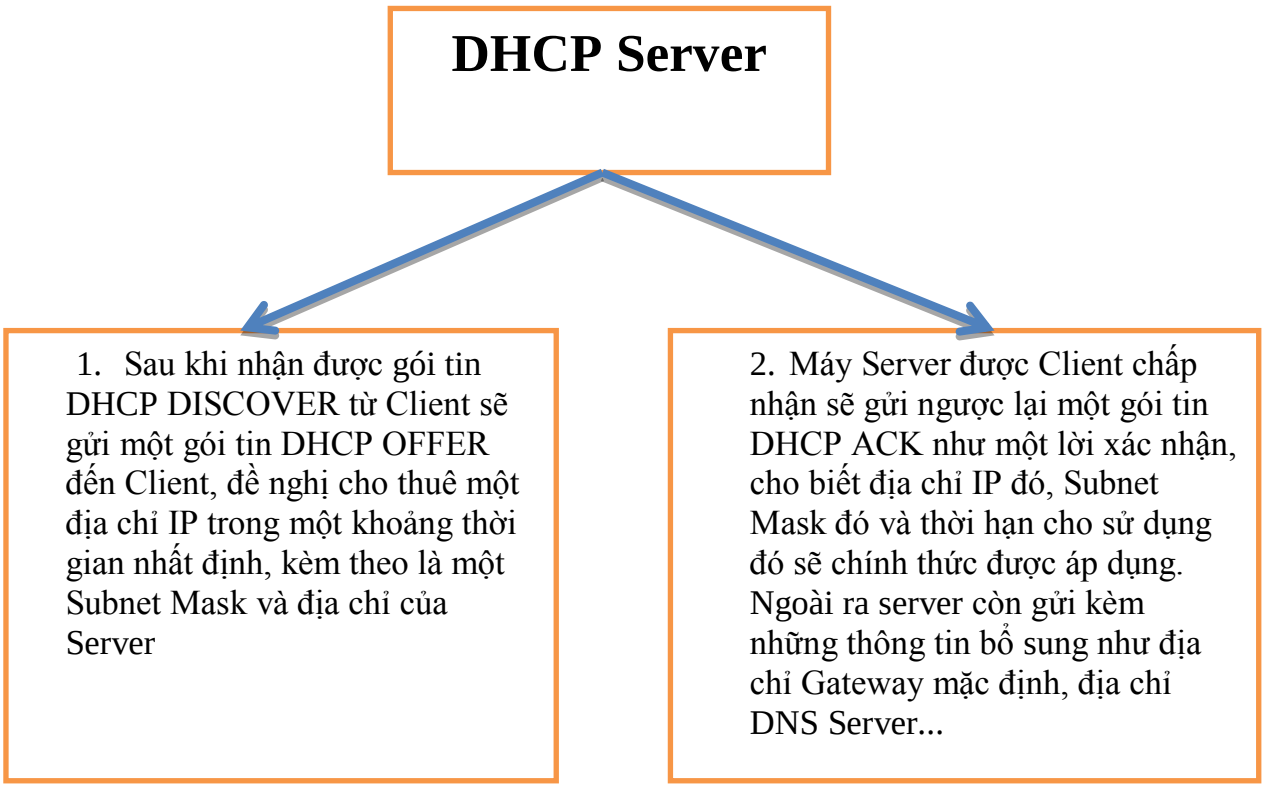


Hình 3.1.1. Mô hình thực nghiệm của chương trình mô phỏng

3.2. Thiết kế chương trình về mặt chức năng



Hình 3.2.1. Mô tả các chức năng của DHCP Client



Hình 3.2.2. Mô tả các chức năng của DHCP Server

3.3. Thiết kế về mặt lập trình

- Đầu tiên bên Server DHCP sẽ bắt đầu lắng nghe khi click vào button Listen:

```
private void btnlisten_Click(object sender, EventArgs e)
        {
            DHCPServer mainServer = new
DHCPServer(IPAddress.Any);
            DHCPServer.StatusChanged += new
StatusChangedEventHandler(mainServer_StatusChanged);
            mainServer.StartListening();
            btnlisten.Text = "Listening";
            lblStatus.Text = "Bắt đầu lắng nghe kết nối từ phía
Client...";
        }
```

- Sau đó bên Client DHCP sẽ kết nối tới Server DHCP để bắt đầu gửi và nhận dữ liệu khi click vào button Connect

```
private void btnConnect_Click(object sender, EventArgs e)
        {

            if (Connected == false)
            {

                InitializeConnection();

            }
            else
            {
                CloseConnection("Disconnected at user's
request.");
            }
        }
```

- Nó sẽ gọi hàm `InitializeConnection()` để kết nối tới Server, bắt đầu nhận thông tin, gửi Hostname và Mac Address đến cho Server

```
private void InitializeConnection()
        {

            tcpServer = new TcpClient();
            tcpServer.Connect(IPAddress.Parse("192.168.1.33"),
8080);
            Connected = true;
            btnConnect.Text = "Disconnect";
            swSender = new StreamWriter(tcpServer.GetStream());
            NetworkInterface[] nics =
NetworkInterface.GetAllNetworkInterfaces();
            PhysicalAddress mac = nics[0].GetPhysicalAddress();
            UserName = Dns.GetHostName();
            swSender.WriteLine(UserName);
            swSender.WriteLine(mac);
            swSender.Flush();
            ReceiveMessages();

        }
```

- Server khởi tạo hàm DHCPDISCOVER để chứa thông tin từ Client:

```
public static void DHCPdiscover(TcpClient tcpUser, string strUsername, string
Message)
        {
            DHCPServer.htUsers.Add(strUsername, tcpUser);
```

```
            DHCPServer.htConnections.Add(tcpUser, strUsername);
            DHCPoffer(htConnections[tcpUser] + Message);
        }
```

- Server gửi cho Clien gói tin DHCPOFFER chứa thông tin về IP Address, Subnet Mask cho Client: (IP Adress sẽ được cho trong khoảng từ 192.168.1.2 cho đến 192.168.1.255, Subnet Mask mặc định sẽ là 255.255.255.0)

```
public static void DHCPoffer(string Message)
        {
            StreamWriter swSenderSender;
            TcpClient[] tcpClients = new
      TcpClient[DHCPServer.htUsers.Count];
            DHCPServer.htUsers.Values.CopyTo(tcpClients, 0);
            Random r = new Random();

            byte[] b = new Byte[4];
            b[0] = 192;
            b[1] = 168;
            b[2] = 1;
            b[3] = Byte.Parse((r.Next(2, 255)).ToString());
            IPAddress ip = new IPAddress(b);
            byte[] d = new Byte[4];
            d[0] = 255;
            d[1] = 255;
            d[2] = 255;
            d[3] = 0;
            IPAddress dg = new IPAddress(d);
            e = new StatusChangedEventArgs(ip + " ");
            OnStatusChanged(e);
            e = new StatusChangedEventArgs(dg+ " ");
            OnStatusChanged(e);
            for (int i = 0; i < tcpClients.Length; i++)
            {
                try
  {
                    if (Message.Trim() == "" || tcpClients[i] ==
null)
                    {
                        continue;
                    }
                    swSenderSender = new
StreamWriter(tcpClients[i].GetStream());
                    swSenderSender.WriteLine("DHCPoffer:");
                    swSenderSender.WriteLine(ip);
                    swSenderSender.WriteLine(dg);
                    swSenderSender.Flush();
                    swSenderSender = null;

                }
                catch
                {
                    RemoveUser(tcpClients[i]);
                }
            }
        }
```

Sau đó bên Client sẽ tiếp nhận gói tin DHCPOFFER:

```
public static void DHCPoffer(string Message)
        {
            StreamWriter swSenderSender;
            TcpClient[] tcpClients = new TcpClient[DHCPServer.htUsers.Count];
            DHCPServer.htUsers.Values.CopyTo(tcpClients, 0);
```

```
            Random r = new Random();

            byte[] b = new Byte[4];
            b[0] = 192;
            b[1] = 168;
            b[2] = 1;
            b[3] = Byte.Parse((r.Next(2, 255)).ToString());
            IPAddress ip = new IPAddress(b);
            byte[] d = new Byte[4];
            d[0] = 255;
            d[1] = 255;
            d[2] = 255;
            d[3] = 0;
            IPAddress dg = new IPAddress(d);
            e = new StatusChangedEventArgs(ip + " ");
            OnStatusChanged(e);
            e = new StatusChangedEventArgs(dg+ " ");
            OnStatusChanged(e);
            for (int i = 0; i < tcpClients.Length; i++)
            {
                try
                {
                    if (Message.Trim() == "" || tcpClients[i] == null)
                    {
                        continue;
                    }
                    swSenderSender = new
StreamWriter(tcpClients[i].GetStream());
                    swSenderSender.WriteLine("DHCPoffer:");
                    swSenderSender.WriteLine(ip);
                    swSenderSender.WriteLine(dg);
                    swSenderSender.Flush();
                    swSenderSender = null;

                }
                catch
                {
                    RemoveUser(tcpClients[i]);
                }
            }
        }
```

- Server gửi cho Client gói tin DHCPACK và Client nhận về gói tin này, sau đó gán IP nhận được vào:

```
public void ganiptinh()
        {
            string chuoiganip = ("netsh interface ip set address
name=\"Local Area Connection\" static " + ip + " " + sm + " " +
dg);
            ProcessStartInfo p = new ProcessStartInfo("cmd.exe",
"/c" + chuoiganip);
            p.RedirectStandardOutput = true;
            p.UseShellExecute = false;
            p.CreateNoWindow = true;
            Process pro = new Process();
            pro.StartInfo = p;
            pro.Start();


        }
```

- Khi tắt kết nối, Client sẽ gán lại IP động (trường hợp thử nghiệm trên máy có nối mạng, sau khi tắt kết nối tới DHCP Server, sẽ tự động gán IP động sẵn có của máy)

```
public void ganipdong()
        {
            string chuoiganip = ("netsh interface ip set address name=\"Local
Area Connection\" dhcp");
            ProcessStartInfo p = new ProcessStartInfo("cmd.exe", "/c" +
chuoiganip);
            p.RedirectStandardOutput = true;
            p.UseShellExecute = false;
            p.CreateNoWindow = true;
            Process pro = new Process();
            pro.StartInfo = p;
            pro.Start();
        }
```

- Client đóng kết nối:

```
private void CloseConnection(string Reason)
        {

            Connected = false;
            lblStatus.Text = "Đã ngắt kết nối tới Server DHCP!";
            listView1.Items.Clear();
            btnConnect.Text = "Connect";
            swSender.Close();
            srReceiver.Close();
            tcpServer.Close();
            ganipdong();
\        }
```

Trên đây chỉ là một số đoạn code trong chương trình lấy ra để minh họa một số quá trình mô phỏng chương trình DHCP.

CHƯƠNG IV: KẾT LUẬN

4.1. Các chức năng đã đạt được của chương trình

- Trong thời gian qua, chúng em đã cố gắng tìm hiểu về các yêu cầu của đề tài. Những phần chúng em đã làm được bao gồm việc xây dựng các hàm tạo kết nối giữa Client và Server, cho phép Client và Server gửi và nhận thông tin qua lại với nhau.
- Client có thể gửi thông tin về Hostname, Mac Address trong gói DHCP REQUEST cho Server, và nhận về gói DHCP ACK bao gồm các thông tin về IP Address, Subnet Mask, Defaul Gateway từ Server sau đó gán những thông tin đó cho Client.
- Server nhận được Hostname, Mac Address từ Client, sau đó gửi cho Client gói DHCP ACK bao gồm các thông tin Server cấp cho Client như: IP Address, Subnet Mask, Defaul Gateway.

4.2. Các vấn đề còn tồn tại

- Bên cạnh những điều đạt được thì vẫn còn những tồn tại:
  - Chưa quản lí được thời gian cho thuê IP của Server giành cho Client.
  - Chưa khắc phục được lỗi thử nghiệm trên một số máy sau khi Client nhận được gói tin DHCP ACK thì nó không được gán vào cho Client.
  - Chưa broadcast được gói tin DHCP REQUEST từ Client cho nhiều Server…..

PHỤ LỤC 1
CÁC HÌNH ẢNH SỬ DỤNG TRONG BÀI

PHỤ LỤC 2
TÀI LIỆU THAM KHẢO

1. Tài liệu học môn lập trình mạng do thầy Nhiệm cung cấp.
2. http://tools.ietf.org/html/rfc2131
3. http://vi.wikipedia.org/wiki/DHCP
4. http://www.java2s.com/Code/CSharp/CatalogCSharp.htm
5. http://social.msdn.microsoft.com/Forums/en-AU/wsk/thread/836c2150-583c-43a6-93b3-0e3202c2e2f5
6. http://support.microsoft.com/kb/169289/vi-vn#mtDisclaimer
7. http://forum.duytan.edu.vn/sites/index.aspx?p=forum_thread&forum=688&thread=2119#p0
8. http://yinyangit.wordpress.com/2011/06/22/socket-communication-with-tcp-client-server/
9. http://yinyangit.wordpress.com/2011/07/24/csharp-simple-example-multiconnection-tcp-server/#
10. http://baomathethong.blogspot.com/2010/07/dhcp-va-su-huu-ich-danh-cho-quan-tri.html
11. http://125.234.239.108/~phuocnt/TKQL%20MANG%20LAN/Bai Giang/CH03_BOOTROM/mo%20hinh%20DHCP.swf

BÀI GIẢNG
MÔN: LẬP TRÌNH MẠNG
(Cập nhật: 3/2006)

Hiệu chỉnh: Nguyễn Cao Đạt

E-mail: dat@hcmut.edu.vn

GIỚI THIỆU

- **Mục tiêu**
  - Cung cấp kiến thức nền tảng về lập trình mạng (Client/Server).
  - Cung cấp kỹ năng lập trình mạng trên các môi trường phát triển phần mềm trực quan như VC++, JBuilder.
  - Nắm vững các giao thức cấp ứng dụng của các ứng dụng phổ biến như DNS, E-mail, FTP, HTTP,... để có khả năng lập trình các ứng dụng mạng.
- **Tài liệu tham khảo**

  [1] Computer Networks, A.S. Tanenbaum, Prentice-Hall, Edition 3.

  [2] Unix network programming.

  [3] Winsock Programming
- **Thông tin liên lạc**

  E-mail : dat@hcmut.edu.vn

  Telephone : 8647256 – 5200

GIỚI THIỆU

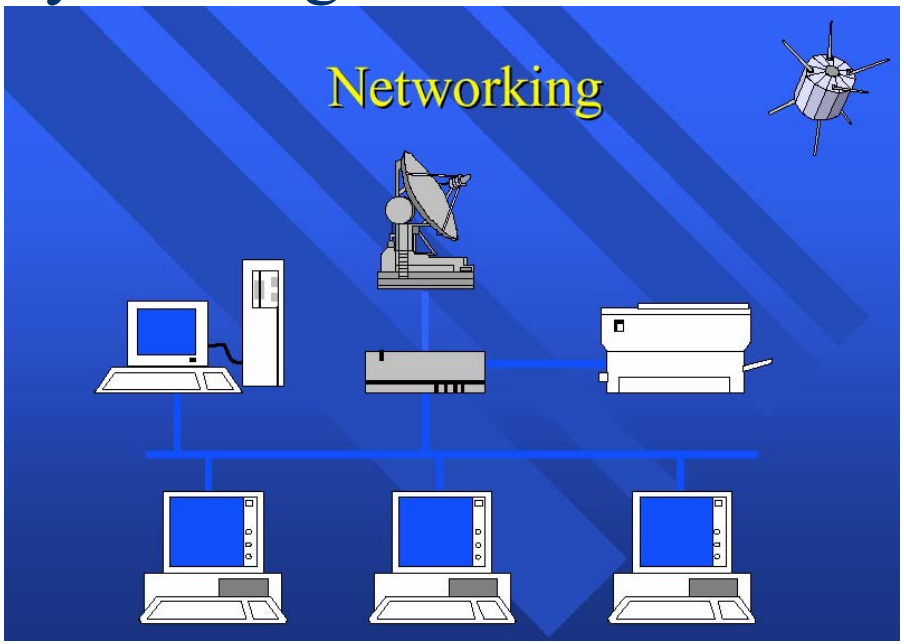
- **Chương trình học chi tiết**
  - **Chương 0:** Khái quát về mạng máy tính, TCP/IP
  - **Chương 1:** Lập trình mạng dùng socket
  - **Chương 2:** Lập trình MiniChat dùng VC++ bằng cơ chế xử lý sự kiện
  - **Chương 3:** Lập trình MiniChat dùng JBuilder bằng cơ chế xử lý sự kiện
  - **Chương 4:** DHMTL và lập trình Web chạy ở client
  - **Chương 5:** Lập trình Web chạy ở server
  - **Chuyên đề:**
    - Lập trình mạng với các giao thức khác.
    - XML-RPC
    - SOAP – WebService

KHÁI QUÁT VỀ MẠNG MÁY TÍNH, TCP/IP

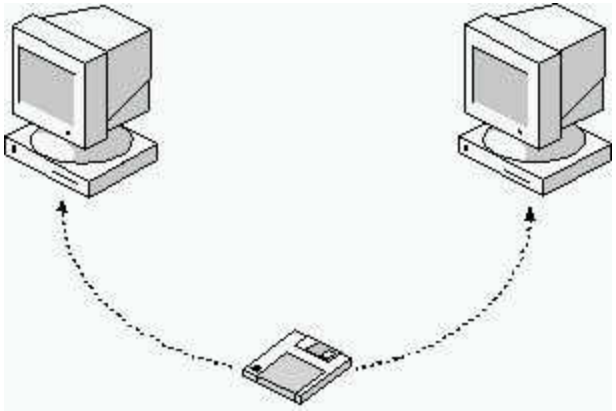
CHƯƠNG 0

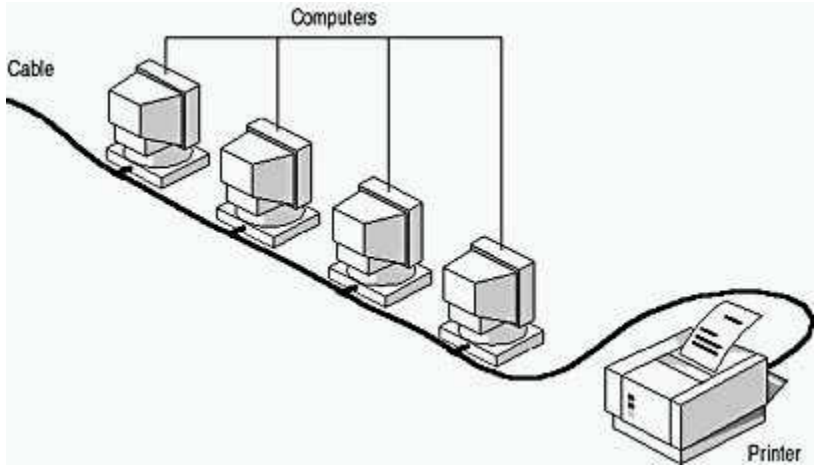
GIỚI THIỆU

- **Mạng máy tính** đề cập đến việc kết nối những máy tính hoạt động độc lập lại với nhau thông qua môi trường truyền thông.



VÌ SAO PHẢI SỬ DỤNG MẠNG MÁY TÍNH



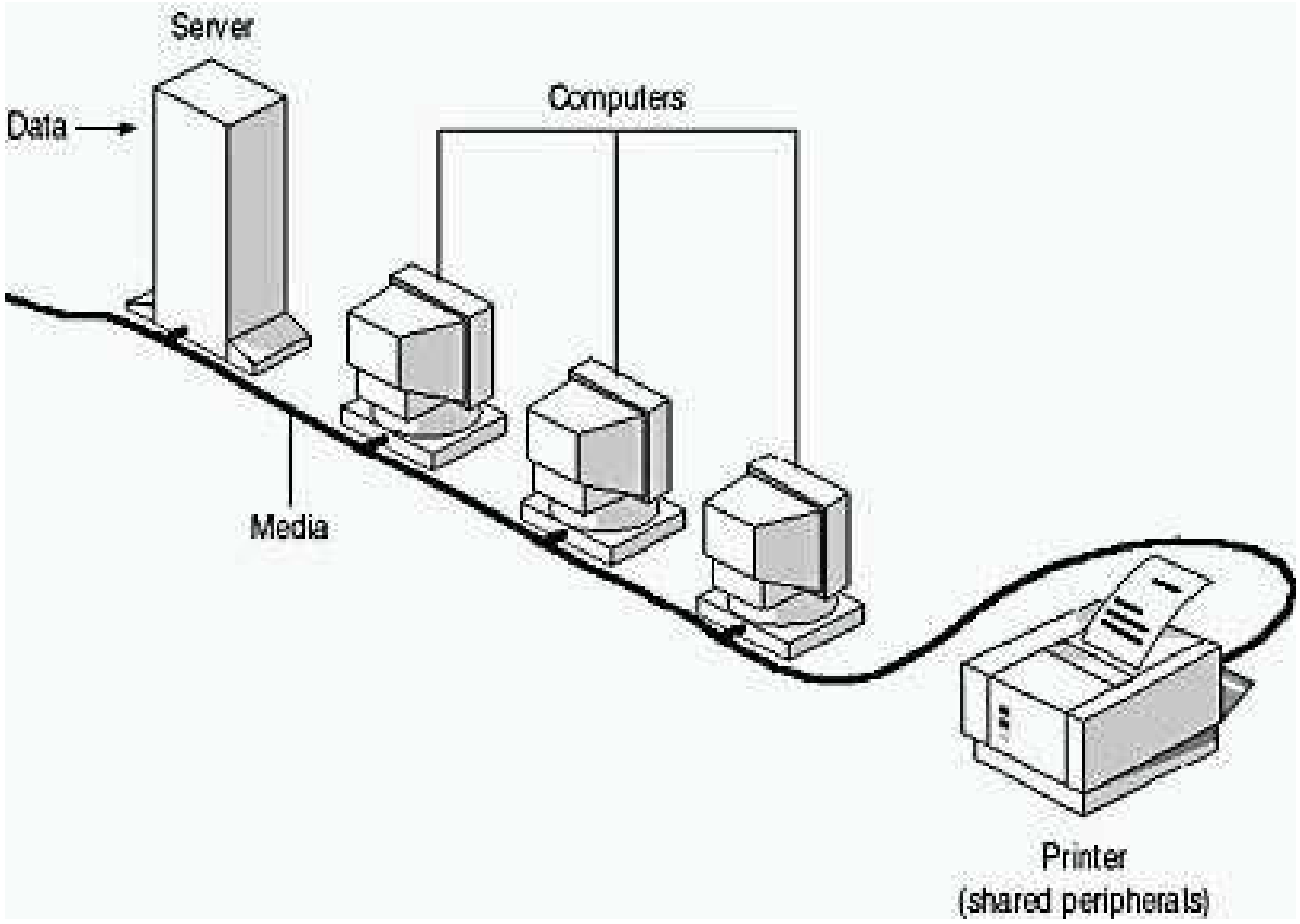


Chia xẻ thông tin.

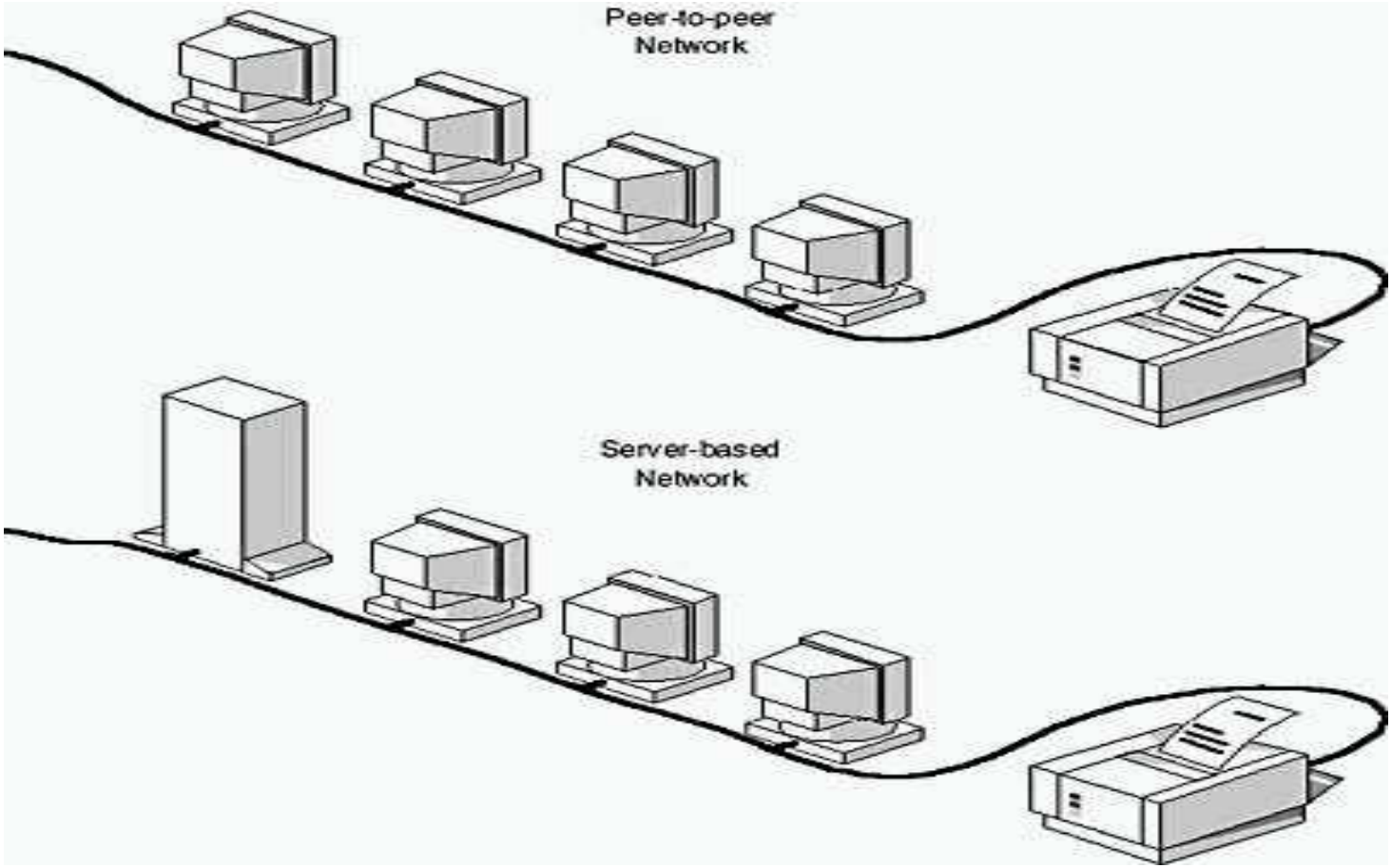
Chia xẻ phần cứng và phần mềm.

Hỗ trợ và quản lý tập trung.

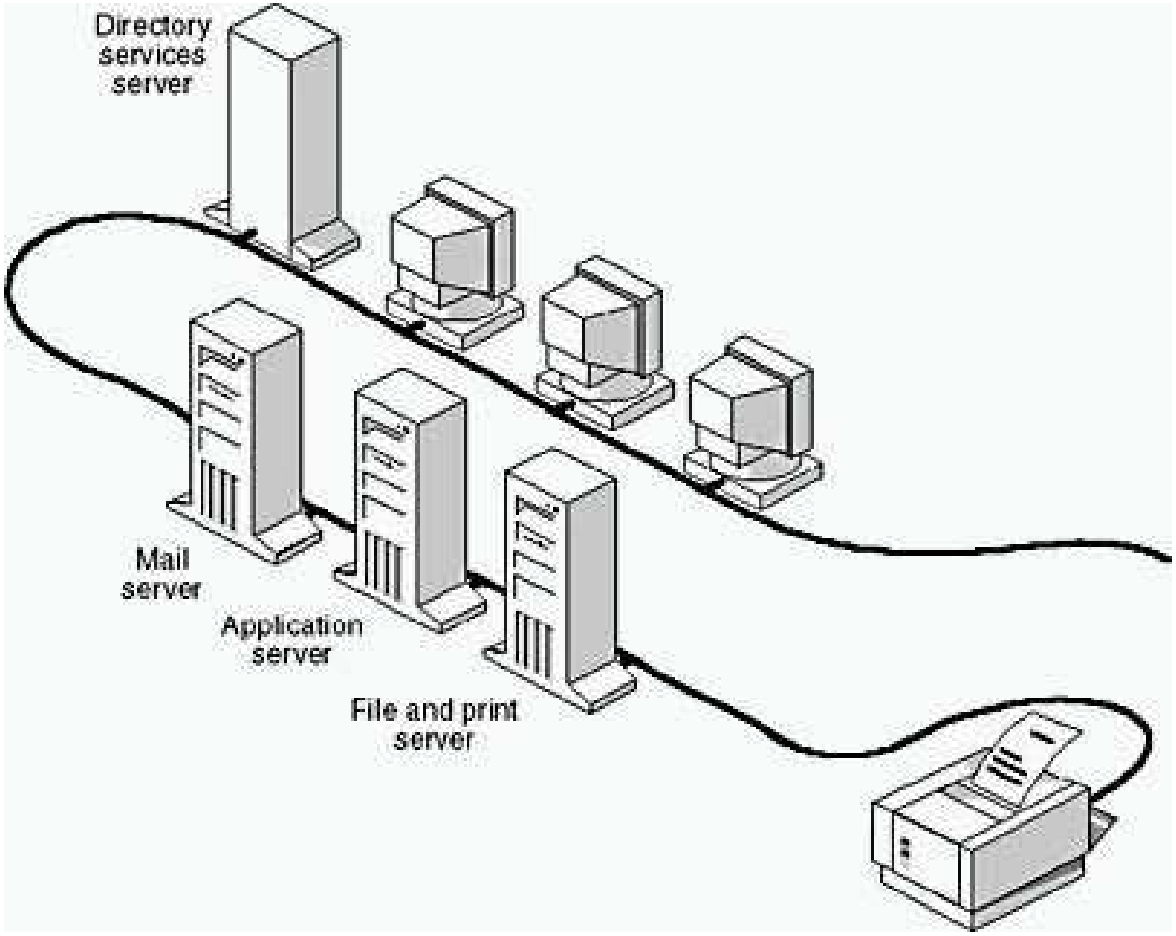
CÁC CẤU HÌNH MẠNG MÁY TÍNH



CÁC CẤU HÌNH MẠNG MÁY TÍNH



MẠNG KHÁCH CHỦ (Server –based)

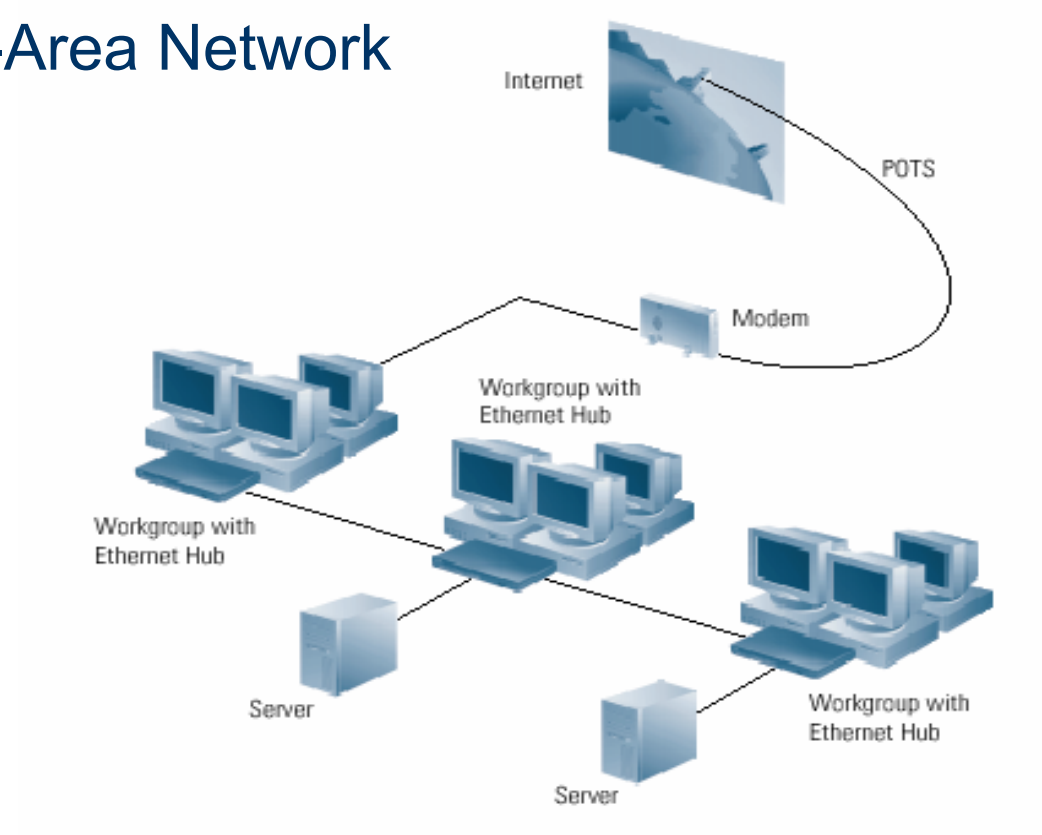


CÁC THÀNH PHẦN CƠ BẢN CỦA MẠNG MÁY TÍNH

- **Thiết bị phần cứng và môi trường truyền**
  - **Các môi trường truyền tín hiệu**
  - **Các thiết bị phần cứng**
  - **Kết nối các môi trường truyền lại với nhau (Hub, Switch).**
  - **Điều khiển quá trình truyền dữ liệu (Switch, Router, Gateway).**
  - **Chạy các phần mềm (máy tính cá nhân, máy chủ).**
- **Phần mềm mạng**
  - Truyền nhận dữ liệu tin cậy giữa hai tiến trình, hai máy tính
  - Nền tảng như PPP, Frame Relay, TCP/IP, IPX/SPX, NetBEUI.
  - Các phần mềm mạng (Web Browser/ Web Server, E-mail Client/Server,..).

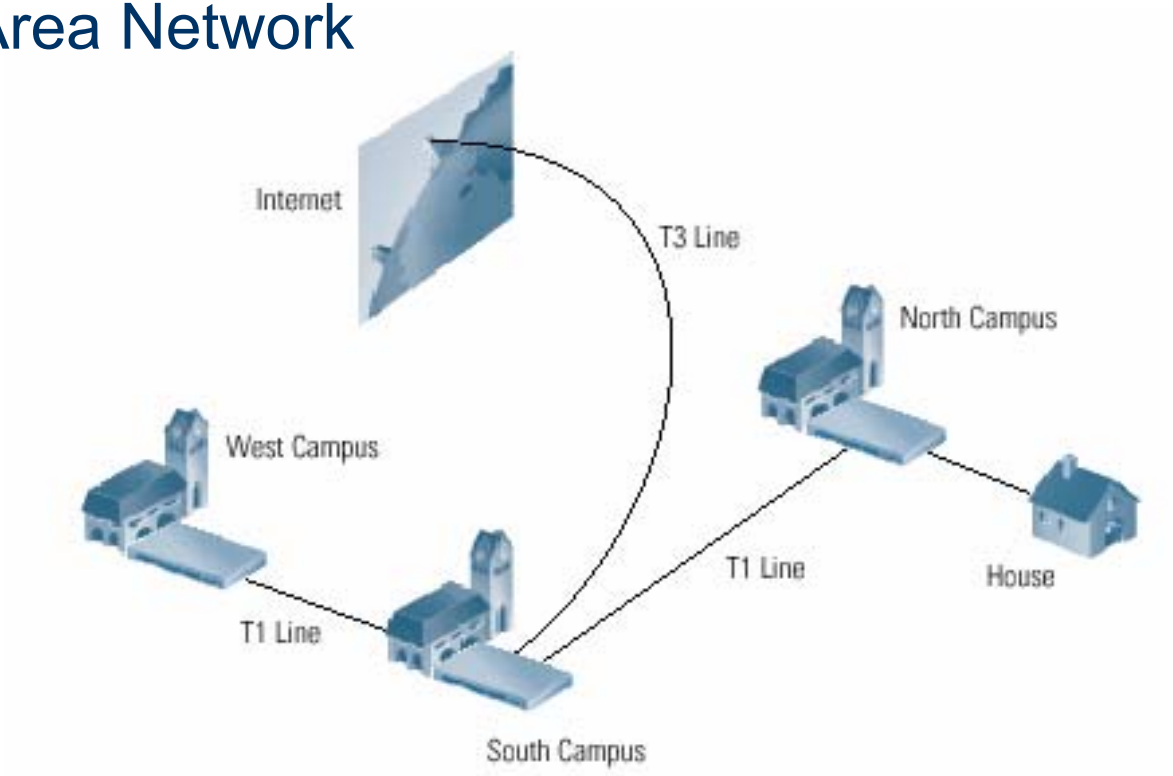
MẠNG CỤC BỘ VÀ ỨNG DỤNG MẠNG

Local-Area Network



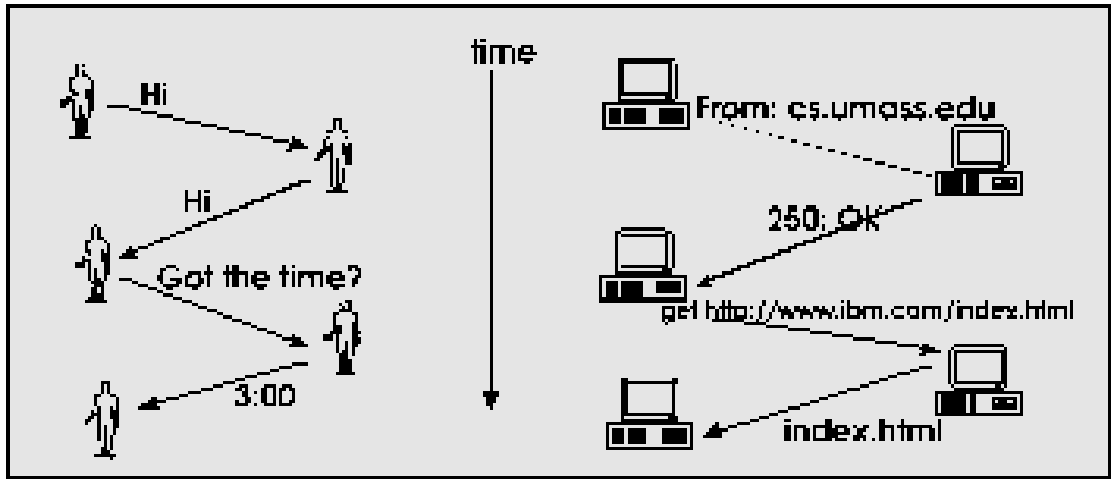
MẠNG DIỆN RỘNG VÀ ỨNG DỤNG MẠNG

Wide-Area Network



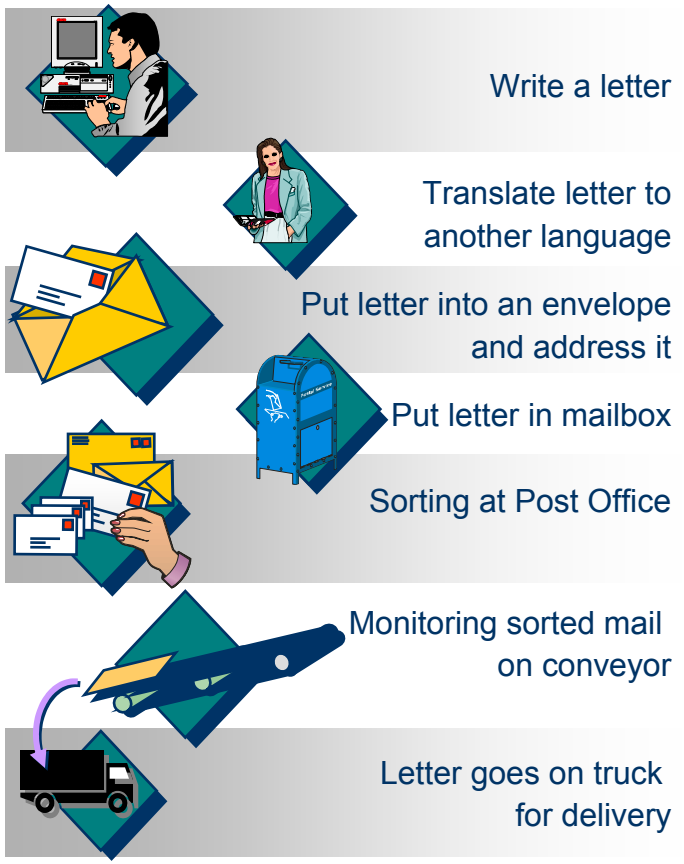
PROTOCOL, LAYER, OSI MODEL

- **Giao thức (Protocol)**
  - Qui tắc các thành phần liên lạc nhau.
- **Cần quan tâm**
  - Định dạng hay thứ tự của message trao đổi.
  - Hành động khi nhận message



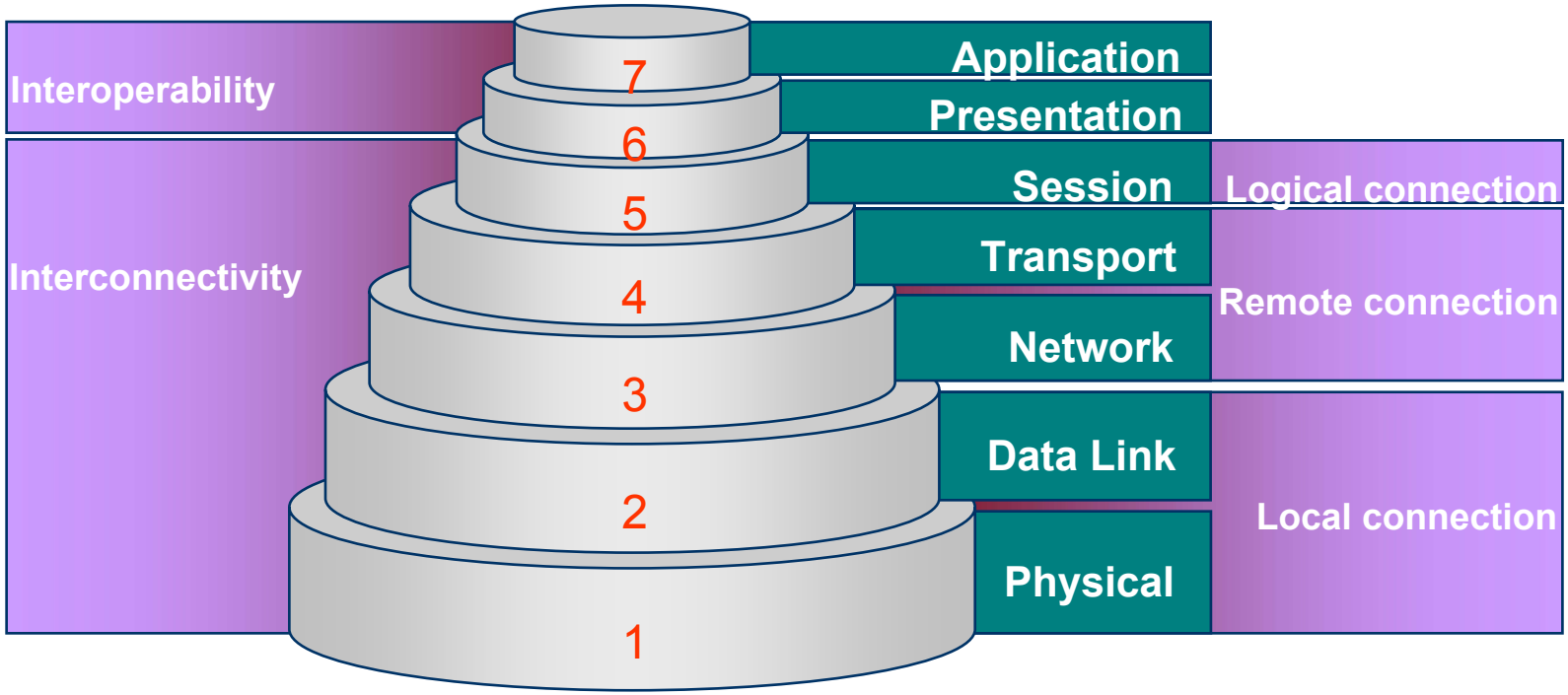
PROTOCOL, LAYER, OSI MODEL

- **Lớp (Layer)**



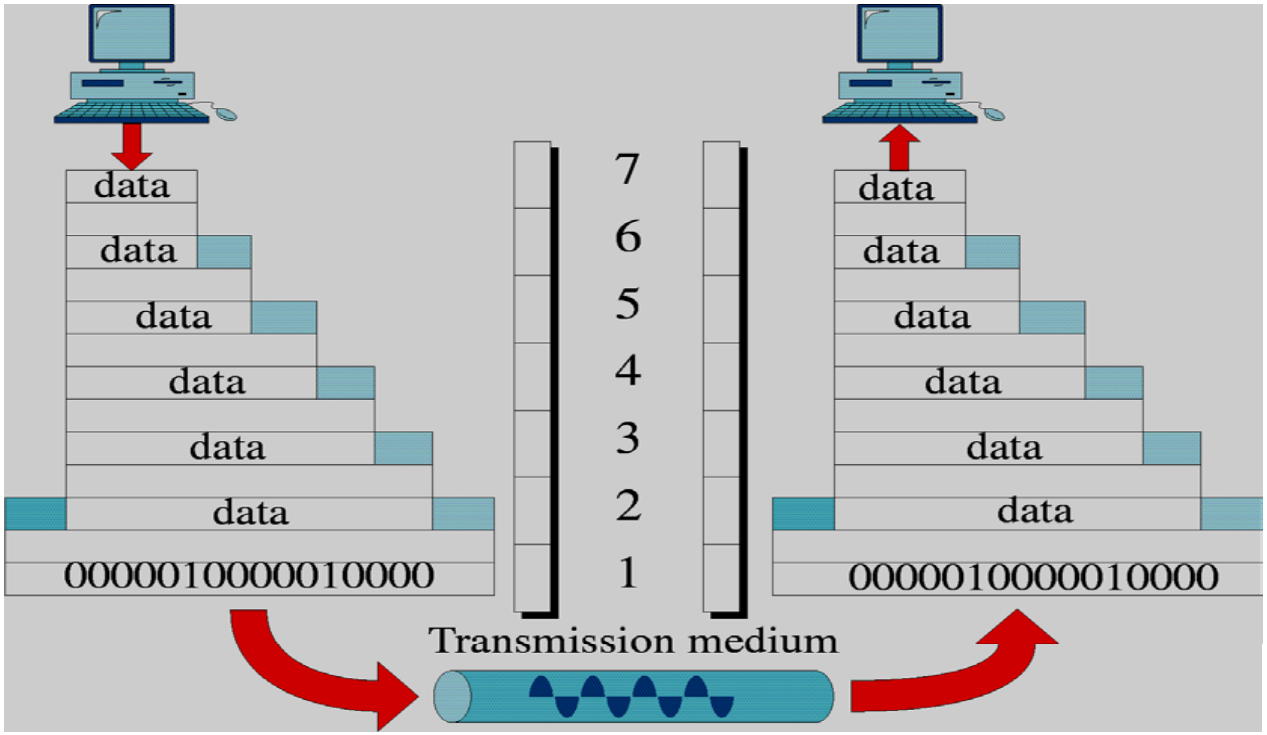
PROTOCOL, LAYER, OSI MODEL

- **Mô hình tham khảo OSI**



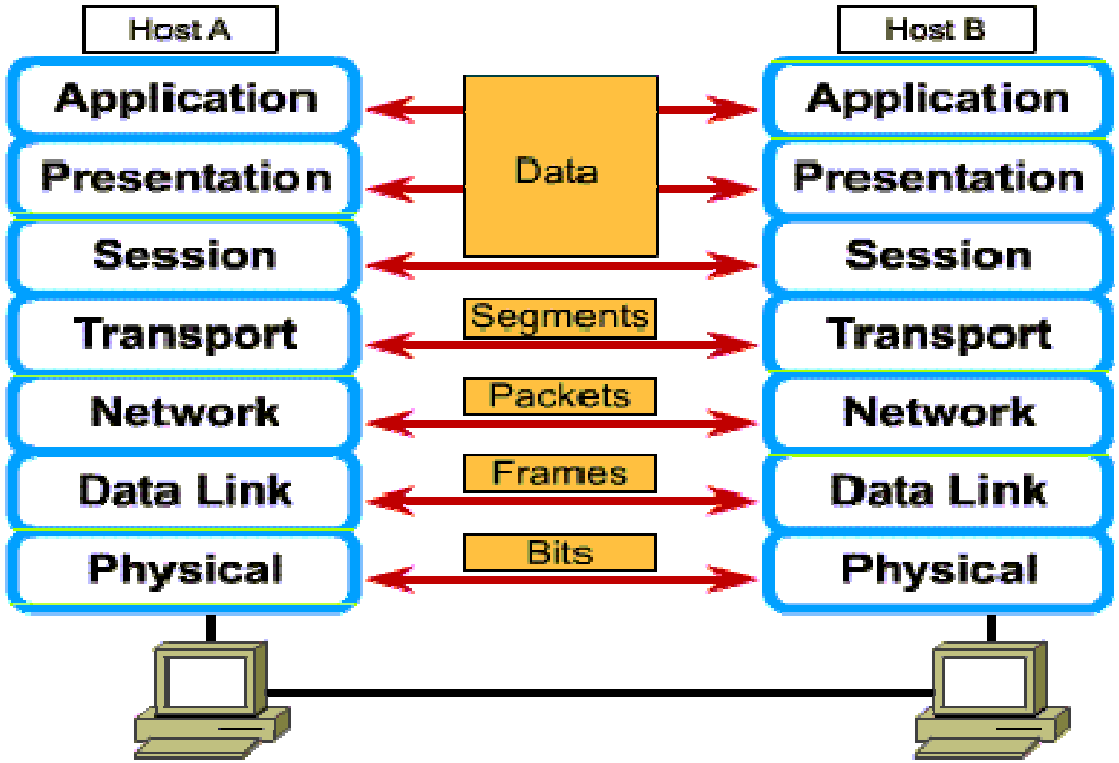
PROTOCOL, LAYER, OSI MODEL

- **Cơ chế trao đổi thông tin**



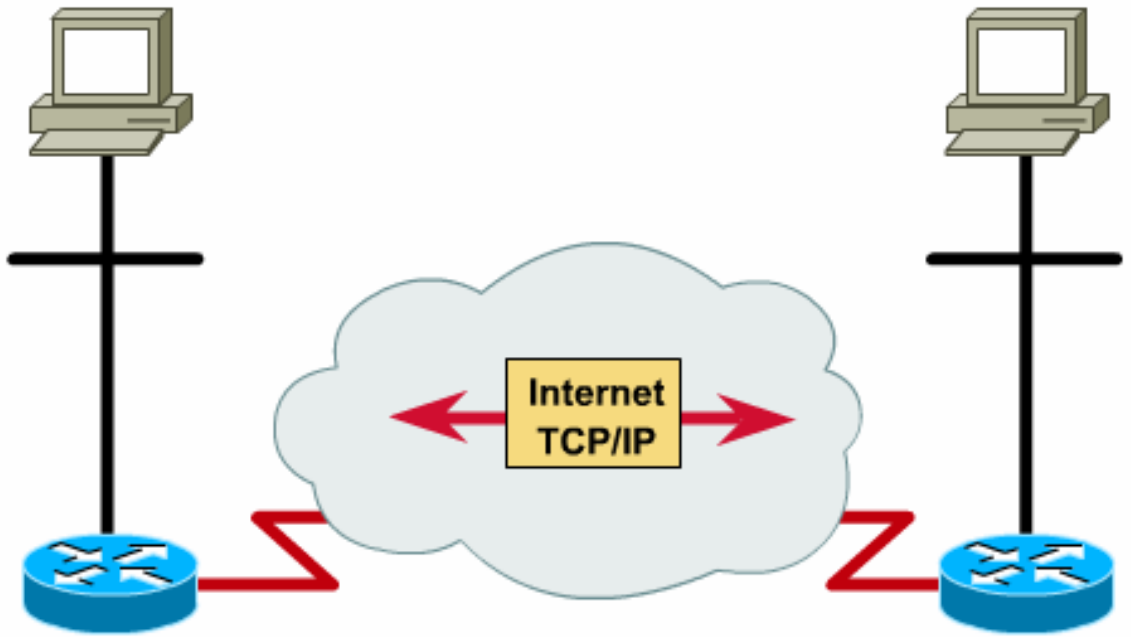
PROTOCOL, LAYER, OSI MODEL

- **Định dạng thông tin**

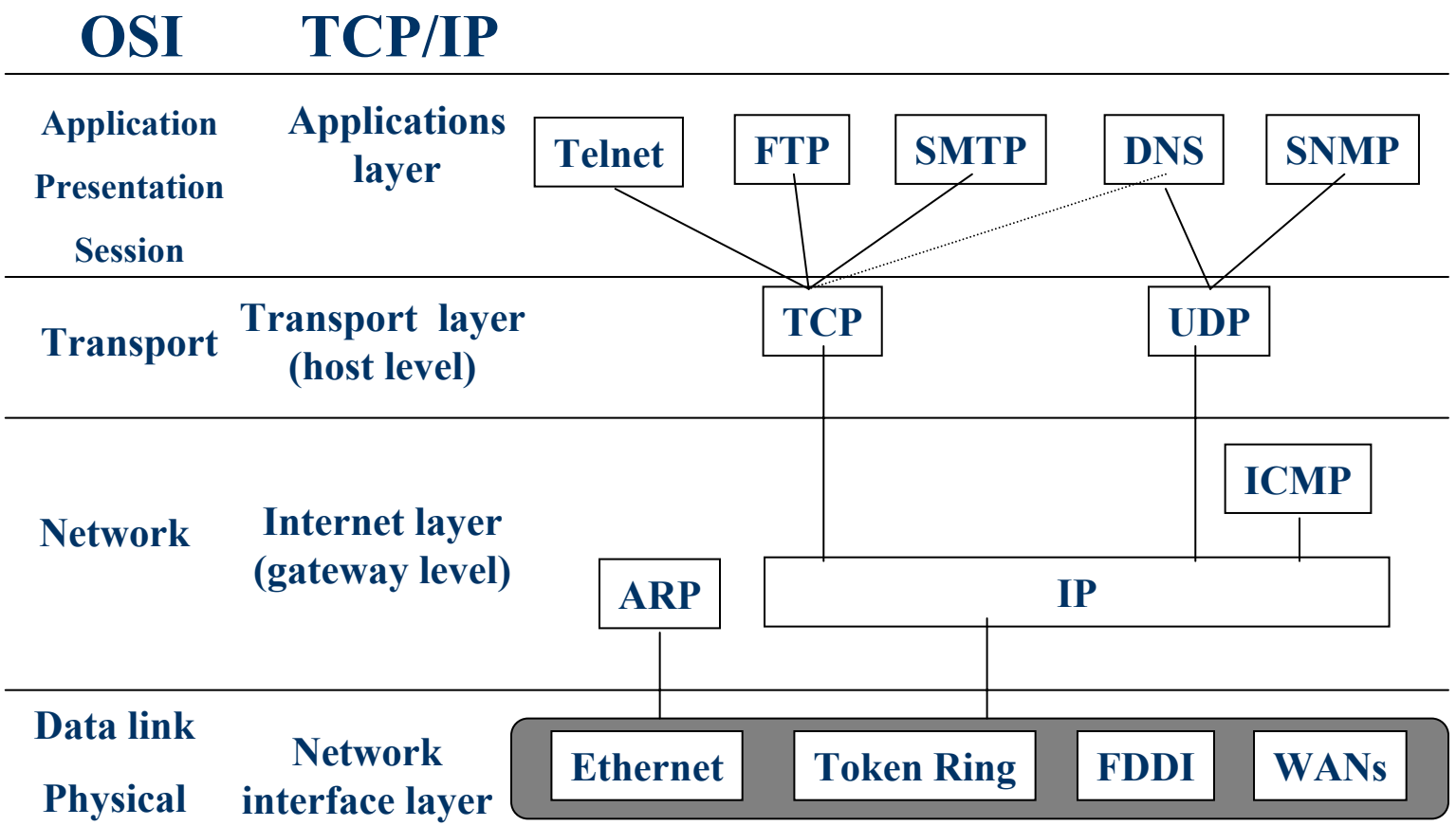


BỘ GIAO THỨC TCP/IP

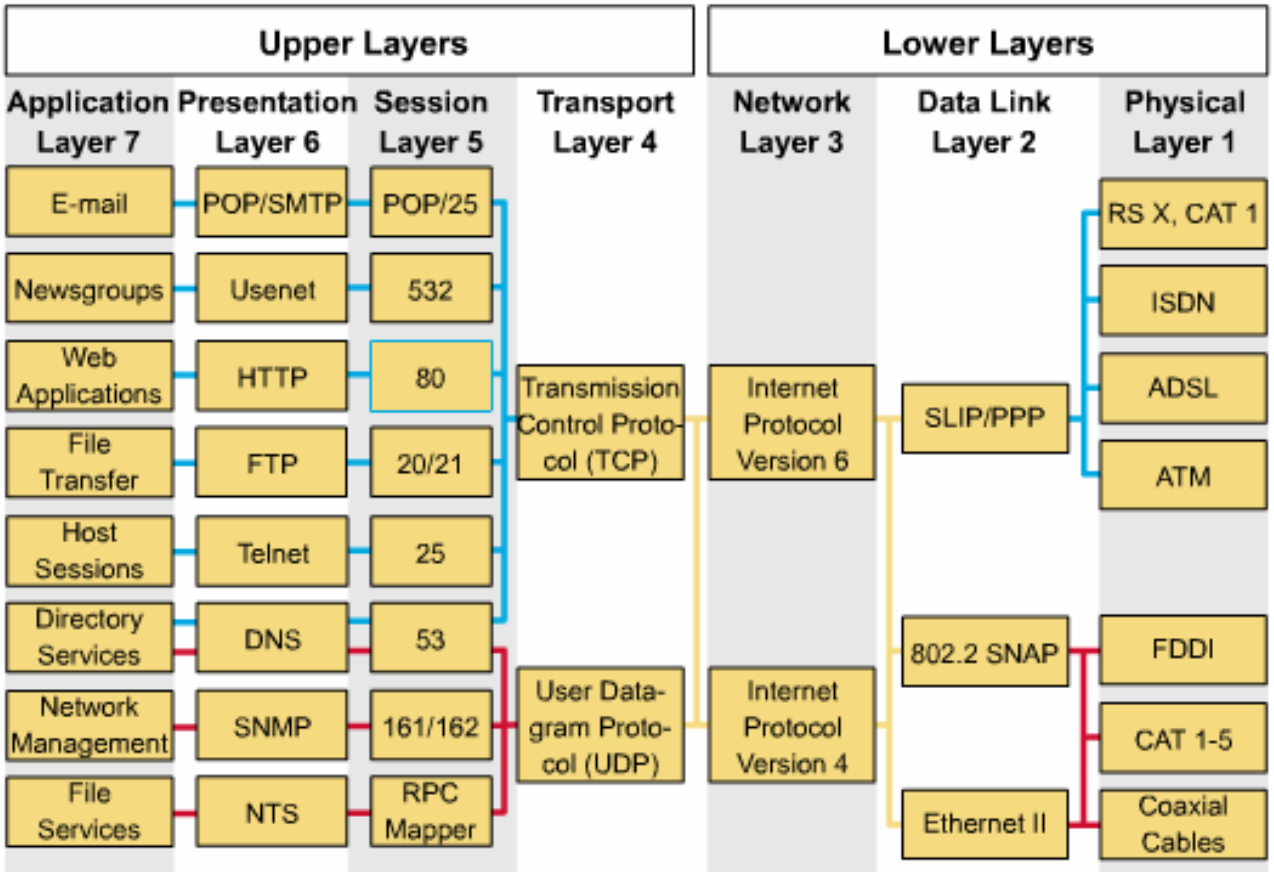
- **Khái niệm**
  - Transmission Control Protocol/ Internet Protocol
  - Được phát triển bởi Defense Advanced Research Projects Agency (DARPA).
  - Đưa vào chuẩn 1983.



SO SÁNH GIỮA OSI VÀ TCP/IP

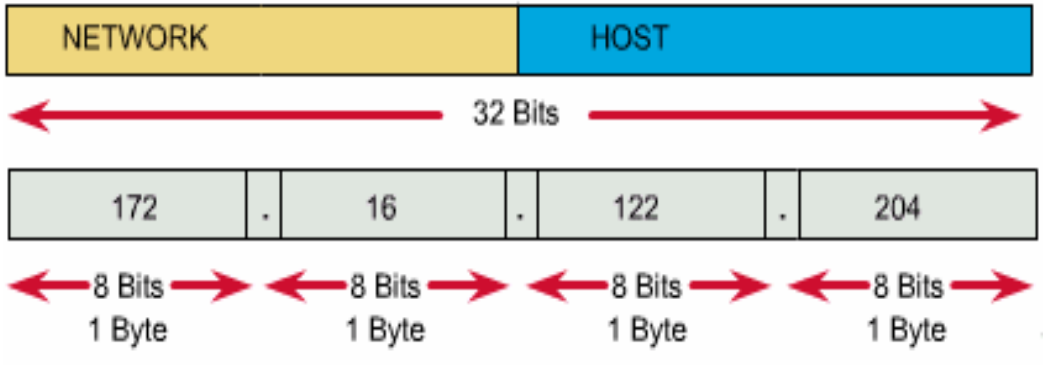


SO SÁNH GIỮA OSI VÀ TCP/IP



TẦNG INTERNET TRONG TCP/IP

- **Các giao thức**
  - IP, ARP, RARP, ICMP.
- **Địa chỉ toàn cục**
  - Các máy tính giao tiếp với nhau thông qua **địa chỉ IP** và **tên gợi nhớ.**
  - *Hãy suy nghĩ địa chỉ IP như thông tin nhà của bạn !!!!*
  - Địa chỉ IP : Mỗi địa chỉ là một cặp (netid, hostid).



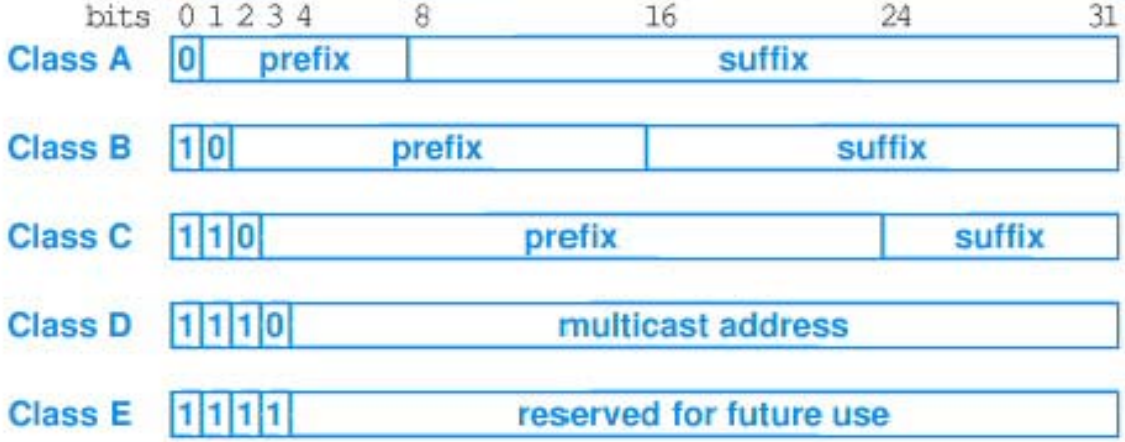
ĐỊA CHỈ IP

- **Địa chỉ IP (version 4)**
  - Địa chỉ IP có chiều dài 4 bytes (32 bits)
  - Địa chỉ IP thường biểu biễn dạng thập phân :
    - **xxx.xxx.xxx.xxx** (x là số thập phân 0-9)
    - Ví dụ : 172.28.11.100
- **Netmask**
  - Địa chỉ mạng tượng trưng như thông tin con đường, phường, quận … (một vùng)
  - Dùng mặt nạ mạng (**Netmask**) để khai báo số bit dùng cho địa chỉ mạng.
  - Ví dụ: 255.255.255.0 (24 bits dành cho địa chỉ mạng, 8 bits còn lại dành cho địa chỉ máy).
  - Địa chỉ máy tương trưng cho chỉ số nhà (duy nhất trong 1 mạng con)

32-bit Binary Number	Equivalent Dotted Decimal
10000001 00110100 00000110 00000000	129.52.6.0
11000000 00000101 00110000 00000011	192.5.48.3
00001010 00000010 00000000 00100101	10.2.0.37
10000000 00001010 00000010 00000011	128.10.2.3
10000000 10000000 11111111 00000000	128.128.255.0

ĐỊA CHỈ IP

- **Phân lớp địa chỉ**
  - Xác định bởi những bit nhận dạng (Class ID).



  - Sự tương quan giữa lớp và kích thước mạng

Address Class	Bits In Prefix	Maximum Number of Networks	Bits In Suffix	Maximum Number Of Hosts Per Network
A	7	128	24	16777216
B	14	16384	16	65536
C	21	2097152	8	256

ĐỊA CHỈ IP

- **Các địa chỉ IP đặc biệt**

Prefix	Suffix	Type Of Address	Purpose
all-0s	all-0s	this computer	used during bootstrap
network	all-0s	network	identifies a network
network	all-1s	directed broadcast	broadcast on specified net
all-1s	all-1s	limited broadcast	broadcast on local net
127	any	loopback	testing

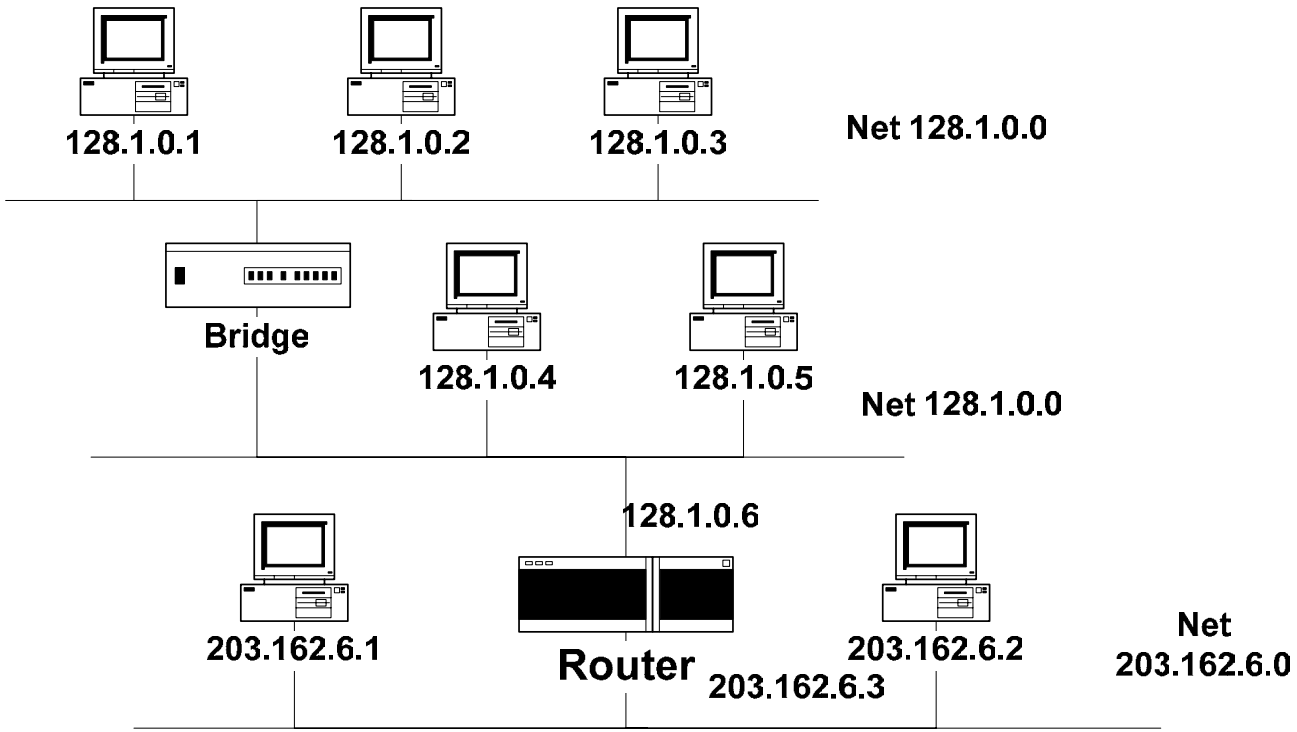
- **Các vùng địa chỉ IP dành riêng (Private Network)**

10.0.0.0 -> 10.255.255.255.255

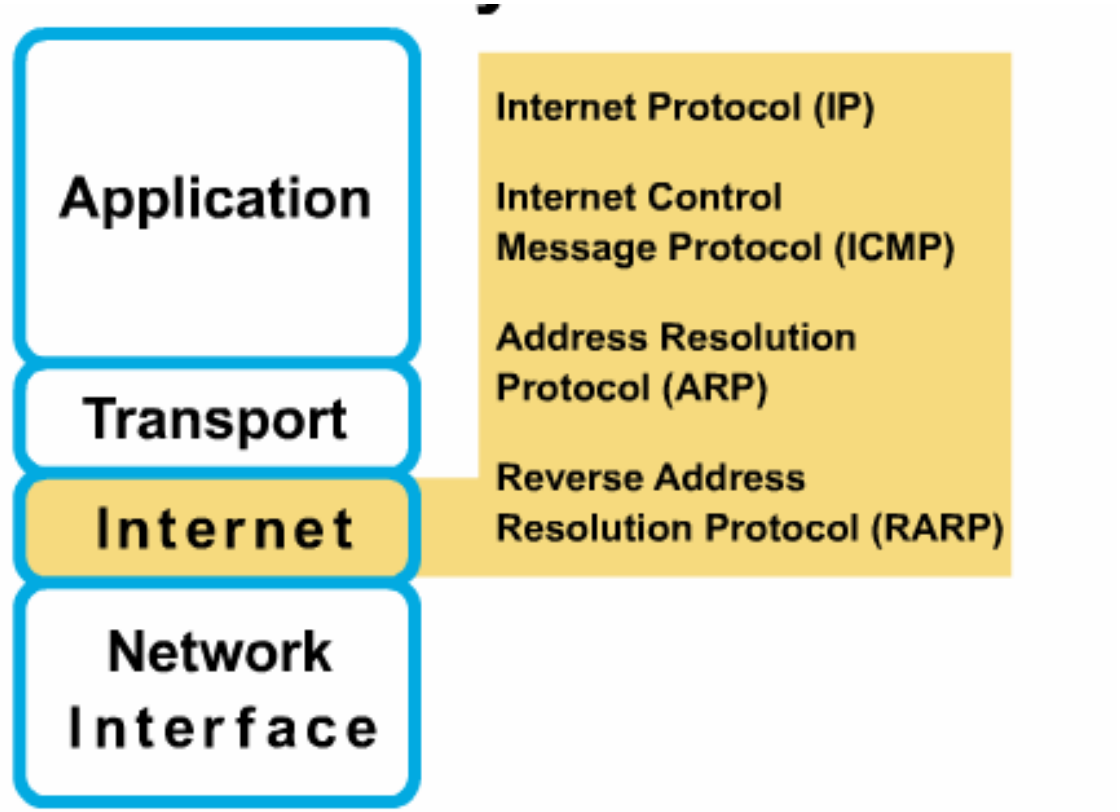
172.16.0.0 -> 172.31.255.255

192.168.0.0 -> 192.168.255.255

ĐỊA CHỈ IP


128.1.0.1
128.1.0.2
128.1.0.3
Net 128.1.0.0
Bridge
128.1.0.4
128.1.0.5
Net 128.1.0.0
128.1.0.6
203.162.6.1
Router
203.162.6.3
203.162.6.2
Net
203.162.6.0

CÁC GIAO THỨC XEM XÉT



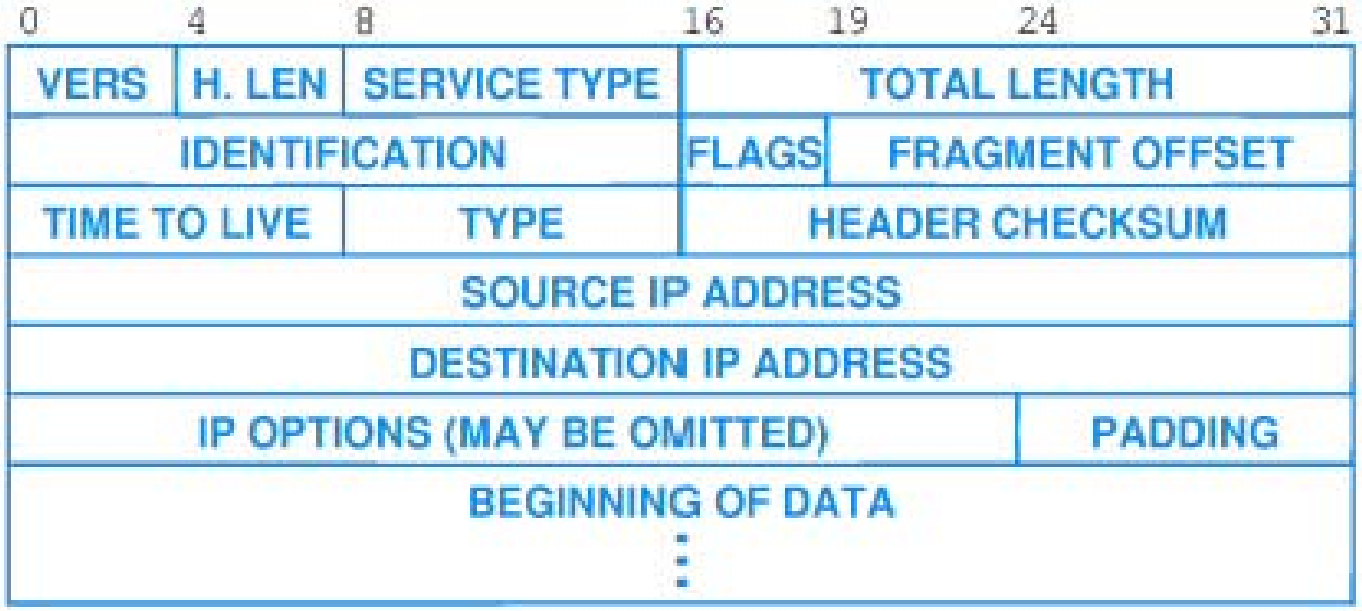
◆OSI network layer corresponds to the TCP/IP Internet layer

INTERNET PROTOCOL

- **Khái niệm**
  - RFC 791
  - Giao thức ở lớp 3 phổ biến.
  - Giao thức nền tảng tạo mạng Internet.
- **Chức năng**
  - Định nghĩa cấu trúc các gói dữ liệu là đơn vị cơ sở cho việc truyền dữ liệu trên Internet.
  - Định tuyến để chuyển các gói dữ liệu trong mạng.
- **Đặc tính**
  - Có khả năng phát hiện lỗi trong phần header.
  - Nỗ lực tối đa(Best-effort): không tin cậy và không có kết nối.
  - Phân mảnh và hợp nhất.

INTERNET PROTOCOL

- **Định dạng của IP Datagram**



INTERNET PROTOCOL

- **Định dạng của IP Datagram (tiếp theo)**
  - Version(VER): IPv4 hay IPv6.
  - Flags, Fragmentation offset : dùng cho vấn đề phân mảnh.
  - Time To Live.
  - Protocol:

Value	Protocol
1	ICMP
2	IGMP
6	TCP
8	EGP
17	UDP
41	IPv6
89	OSPF

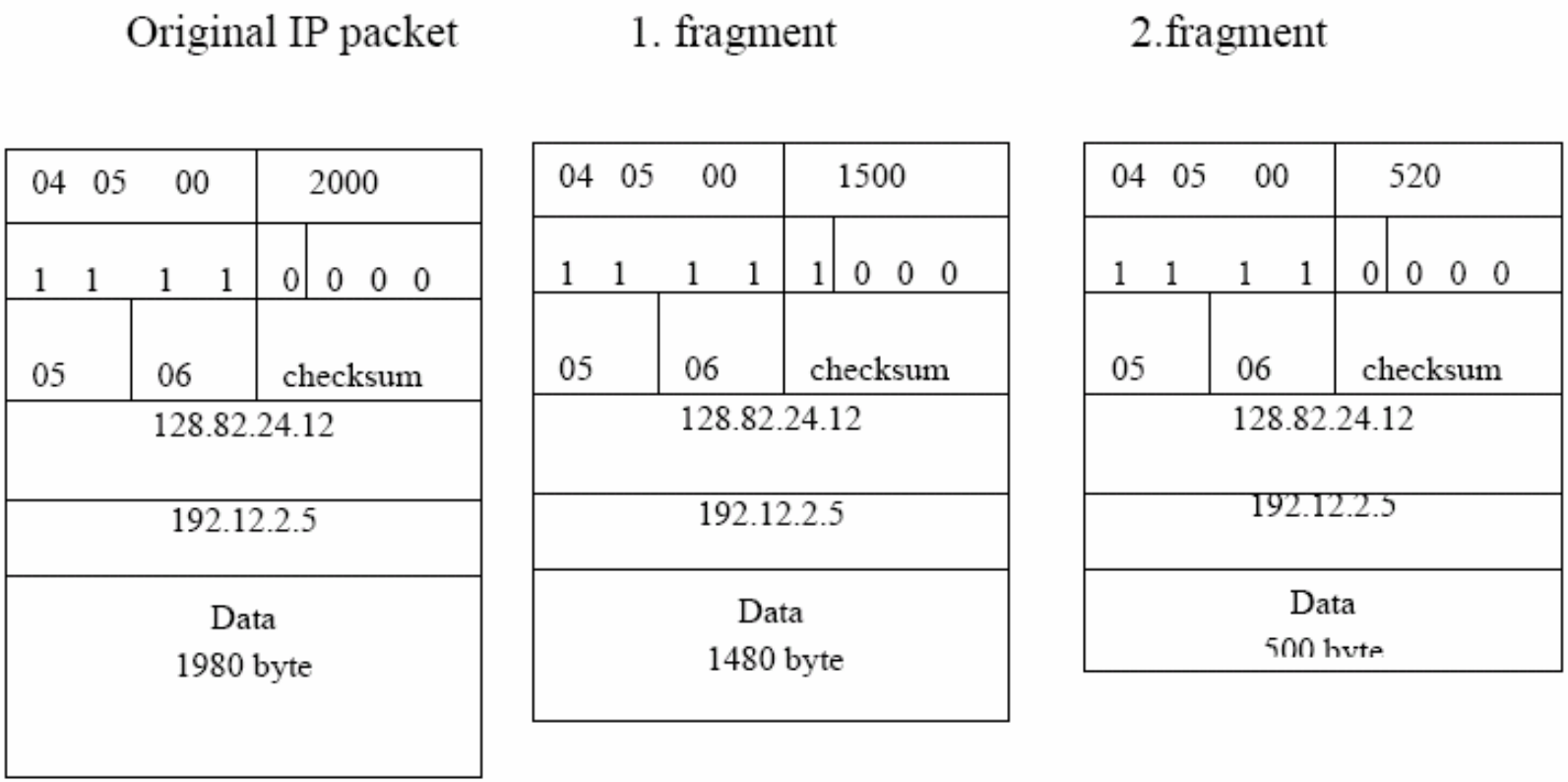
INTERNET PROTOCOL

- **Phân mảnh (Fragment)**
  - MTU (Maximum Transfer Unit)
  - Chiều dài tối đa của IP Datagram là: 65.535 Bytes.
  - Khi đi qua các mạng có MTU nhỏ hơn thì cần phân mảnh.
  - Khi đến đích sẽ thực hiện hợp nhất.

Protocol	MTU(Bytes)
Hyperchannel	65.535
Token ring (16Mbps)	17.914
Token ring (4Mbps)	4.464
FDDI	4.352
Ethernet	1500
X.25	576
PPP	296

INTERNET PROTOCOL

- **Ví dụ về phân mảnh đối với Ethernet**



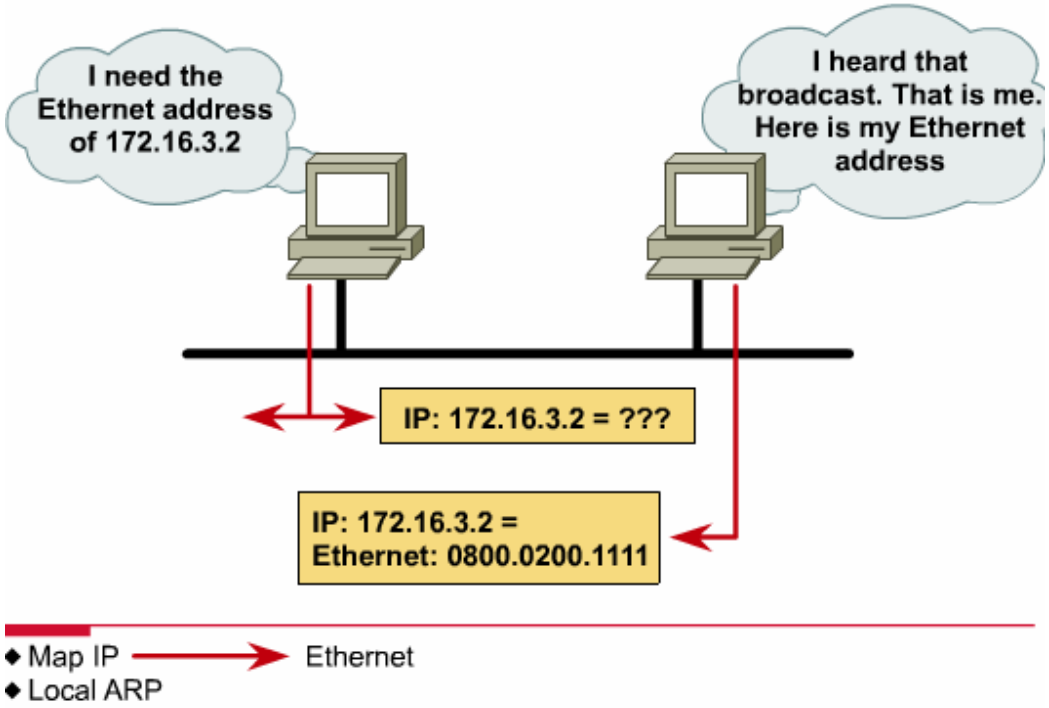
INTERNET CONTROL MESSAGE PROTOCOL

- **Khái niệm**
  - Gởi các thông báo lỗi và các thông báo điều khiển.
  - Các loại sau được dùng để định nghĩa thông điệp:

Destination Unreachable	Time to Live Exceeded
Parameter Problem	Source Quench
Redirect	
Echo	Echo reply
Timestamp	Timestamp Reply
Information Request	Information Reply
Address Request	Address Reply

ADDRESS RESOLUTION PROTOCOL

- **Khái niệm**
  - Phân giải hay ánh xạ từ địa chỉ IP thành địa chỉ MAC.



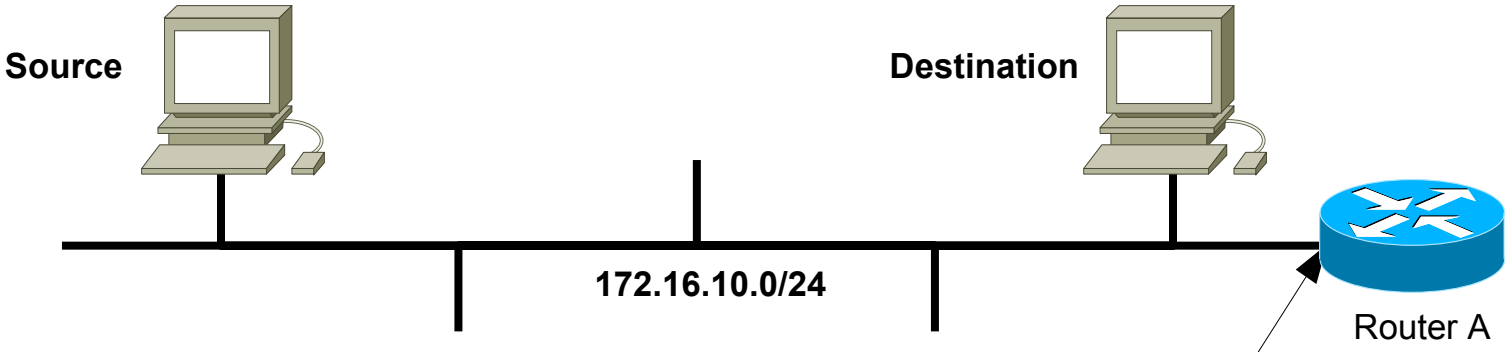
  - Cơ chế hoạt động.

ADDRESS RESOLUTION PROTOCOL

- **Ví dụ cùng subnet**

Host Stevens
172.16.10.10
255.255.255.0
MAC 00-0C-04-17-91-CC

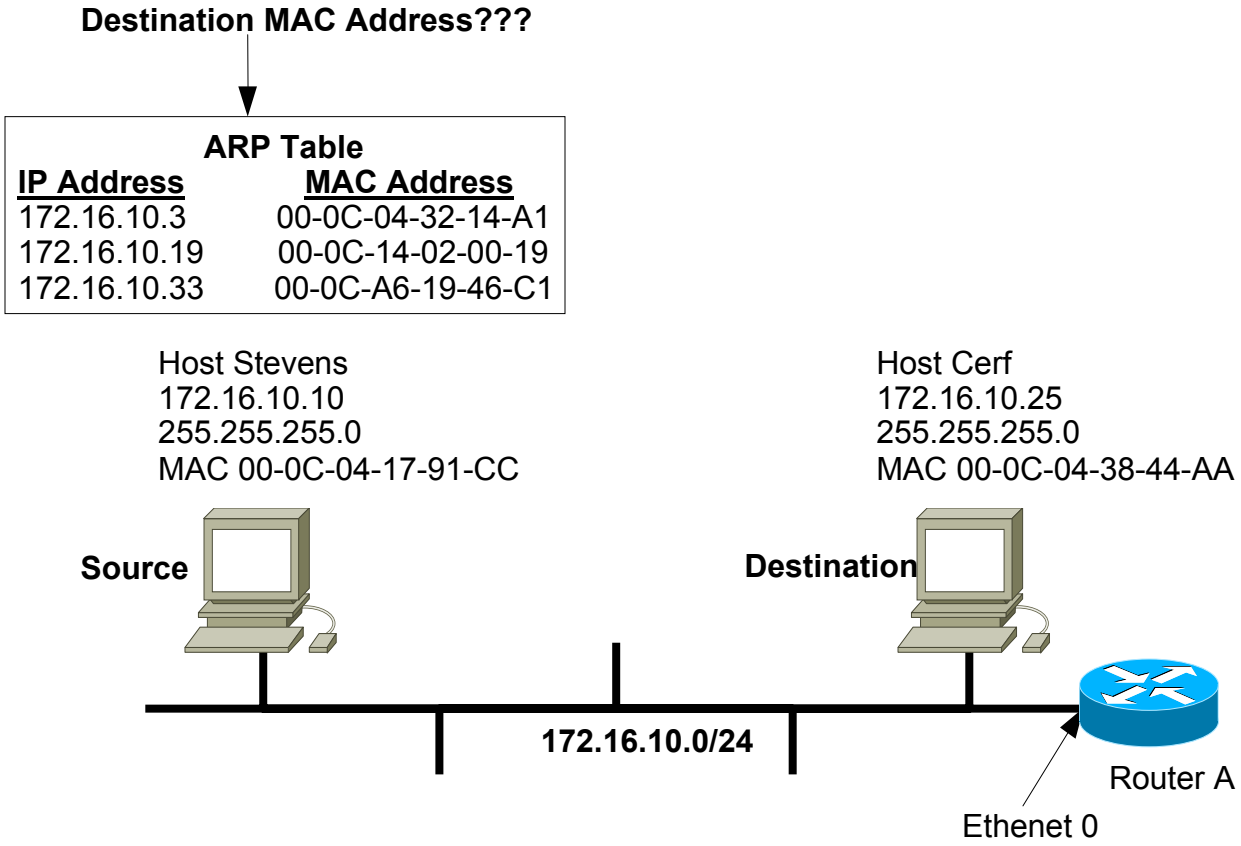
Host Cerf
172.16.10.25
255.255.255.0
MAC 00-0C-04-38-44-AA



Host Stevens ở địa chỉ IP là 172.16.10.10 muốn gởi IP packet đến Host Cerf ở địa chỉ IP là 172.16.10.25.

ADDRESS RESOLUTION PROTOCOL

- **Ví dụ cùng subnet (tiếp theo)**



ADDRESS RESOLUTION PROTOCOL

- **Ví dụ cùng subnet (tiếp theo)**

  - ARP Request từ host Stevens ở địa chỉ IP 172.16.10.10

"Hey everyone! I have this IP Address, 172.28.10.25, and I need the device this belongs to, to send me their MAC address."

ARP Request from 172.16.10.10

Ethernet Header			**Ethernet Data – 28 byte ARP request/reply**				
Ethernet Destination Address (MAC)	Ethernet Source Address (MAC)	Frame Type	ARP headers, i.e. op field	Sender's Ethernet Address (MAC)	Sender's IP Address	Target's Ethernet Address (MAC)	Target's IP Address
FF-FF-FF-FF-FF-FF	00-0C-04-17-91-CC	0x806	op = 1	00-0C-04-17-91-CC	172.16.10.10		172.16.10.25

op field – ARP request = 1
ARP reply = 2
RARP request = 3
RARP reply = 4

ADDRESS RESOLUTION PROTOCOL

- **Ví dụ cùng subnet (tiếp theo)**

  - ARP Reply từ Cerf ở địa chỉ IP 172.16.10.25

  "Hey sender of ARP Request! Here is my MAC address that you wanted for that IP address."

ARP Reply from 172.16.10.25

Ethernet Header			Ethernet Data – 28 byte ARP request/reply				
Ethernet Destination Address (MAC)	Ethernet Source Address (MAC)	Frame Type	ARP headers, i.e. op field	Sender's Ethernet Address (MAC)	Sender's IP Address	Target's Ethernet Address (MAC)	Target's IP Address
00-0C-04-17-91-CC	00-0C-04-38-44-AA	0x806	op = 2	00-0C-04-38-44-AA	172.16.10.25	00-0C-04-17-91-CC	172.16.10.10

Here it is!

ADDRESS RESOLUTION PROTOCOL

- **Ví dụ cùng subnet (tiếp theo)**

  - Host Stevens nhận được ARP Reply và đưa địa chỉ IP, và địa chỉ MAC của Host Cerf vào bảng ARP của nó.

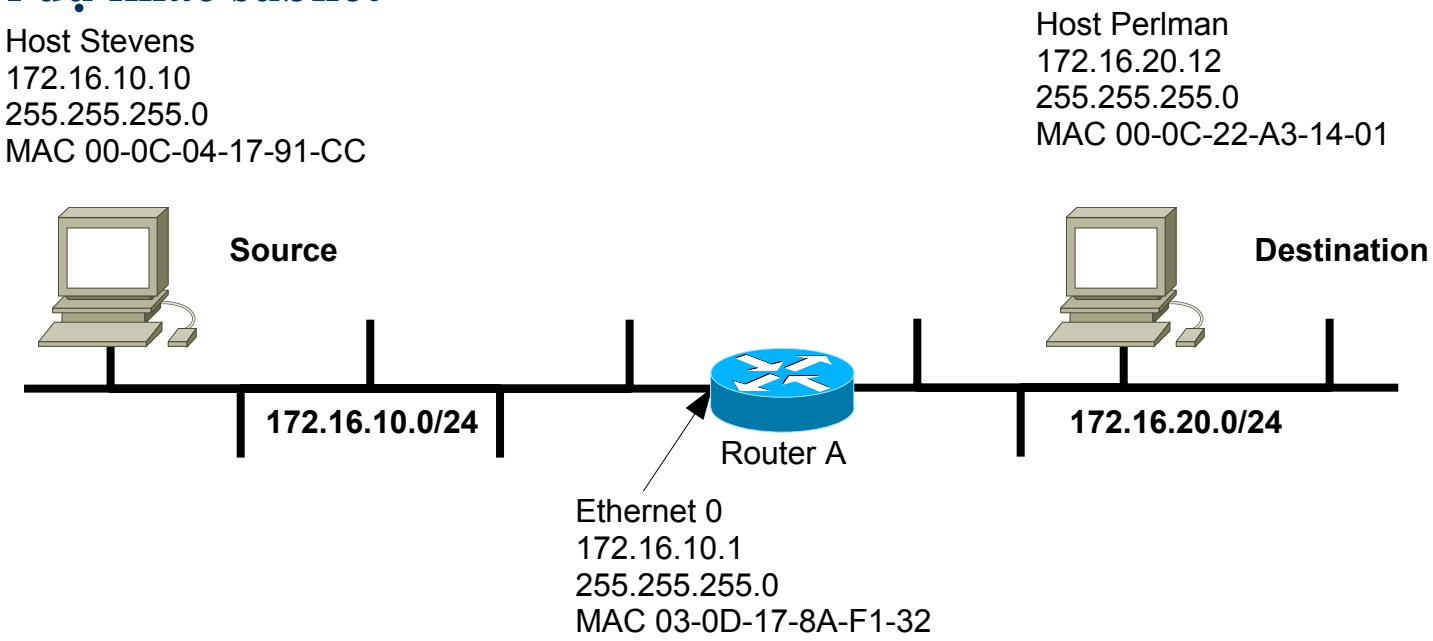
  - Host Stevens bây giờ đã có tất cả các thông tin cần thiết để đóng gói IP packet vào Ethernet frame và gởi frame này trực tiếp đến Host Cerf.

Ethernet Frame

Ethernet Header			IP Datagram from above				Ethernet Trailer
MAC Destination Address **00-0C-04-38-44-AA**	MAC Source Address 00-0C-04-17-91-CC	Other Header Info	IP Header Info	IP Original Source Address 172.17.10.10	IP Final Destination Address **172.16.10.25**	Data	FCS

ADDRESS RESOLUTION PROTOCOL

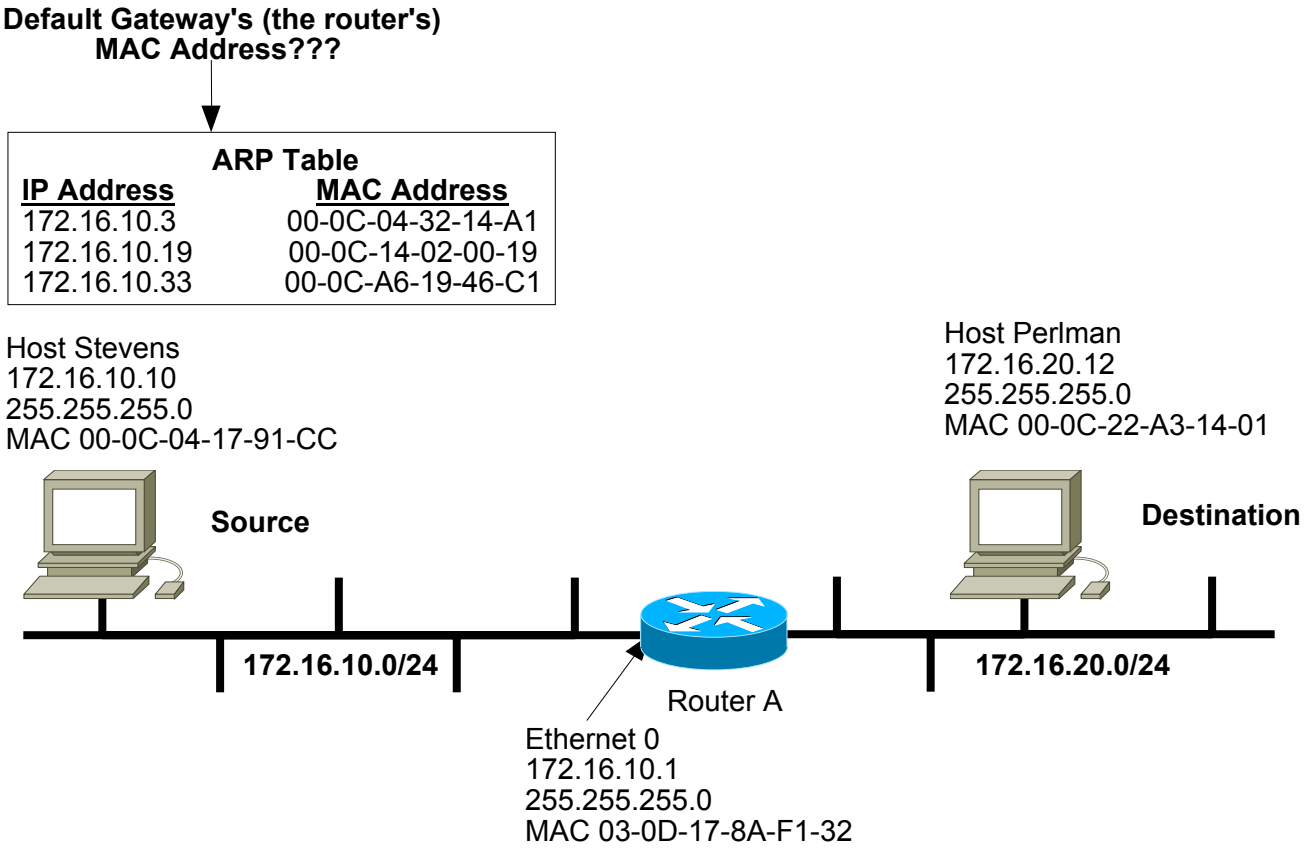
- **Ví dụ khác subnet**



Host Stevens ở địa chỉ IP là 172.16.10.10 muốn gởi một IP packet đến Host Perlman ở địa chỉ IP là 172.16.20.12

ADDRESS RESOLUTION PROTOCOL

- **Ví dụ khác subnet (tiếp theo)**



ADDRESS RESOLUTION PROTOCOL

- **Ví dụ khác subnet (tiếp theo)**

  - ARP Request từ host Stevens ở địa chỉ IP 172.16.10.10

"Hey everyone! I have this IP Address, 172.16.10.1, and I need the device this belongs to, to send me their MAC address."

ARP Request from 172.16.10.10

Ethernet Header			**Ethernet Data – 28 byte ARP request/reply**				
Ethernet Destination Address (MAC)	Ethernet Source Address (MAC)	Frame Type	ARP headers, i.e. op field	Sender's Ethernet Address (MAC)	Sender's IP Address	Target's Ethernet Address (MAC)	Target's IP Address
FF-FF-FF-FF-FF-FF	00-0C-04-17-91-CC	0x806	op = 1	00-0C-04-17-91-CC	172.16.10.10		**172.16.10.1**

op field – ARP request = 1
ARP reply = 2
RARP request = 3
RARP reply = 4

ADDRESS RESOLUTION PROTOCOL

- **Ví dụ khác subnet (tiếp theo)**

  - ARP Reply từ Router A ở địa chỉ IP 172.16.10.1

  "Hey sender of ARP Request! Here is my MAC address that you wanted for that IP address."

ARP Reply from 172.16.10.1

Ethernet Header			Ethernet Data – 28 byte ARP request/reply				
Ethernet Destination Address (MAC)	Ethernet Source Address (MAC)	Frame Type	ARP headers, i.e. op field	Sender's Ethernet Address (MAC)	Sender's IP Address	Target's Ethernet Address (MAC)	Target's IP Address
00-0C-04-17-91-CC	03-0D-17-8A-F1-32	0x806	op = 2	03-0D-17-8A-F1-32	172.16.10.1	00-0C-04-17-91-CC	172.16.10.10

Here it is!

ADDRESS RESOLUTION PROTOCOL

- **Ví dụ khác subnet (tiếp theo)**

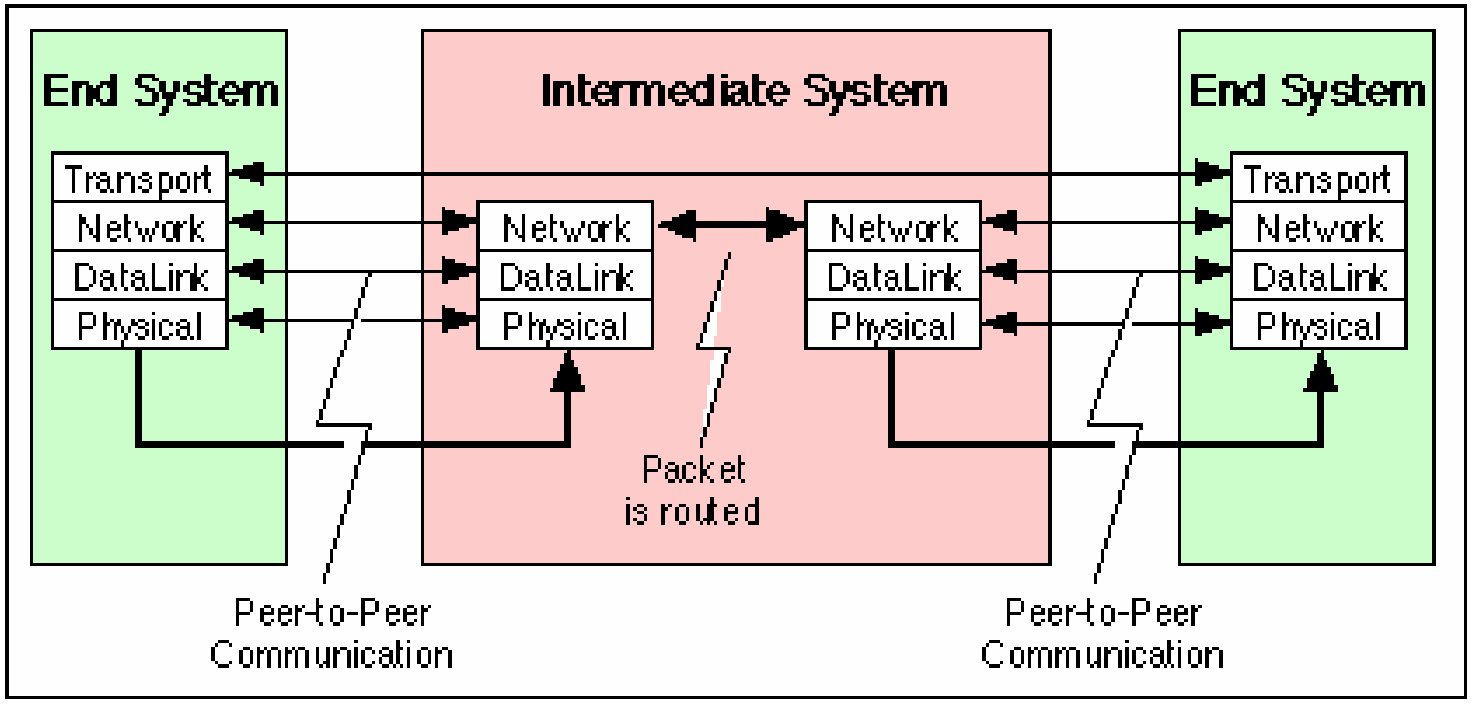
- Host Stevens nhận được ARP Reply và đưa địa chỉ IP, địa chỉ MAC của Host Cerf vào bảng ARP của nó.

- Host Stevens bây giờ đã có tất cả các thông tin cần thiết để đóng gói IP packet vào Ethernet frame và gởi frame này đến Router A.

Ethernet Frame

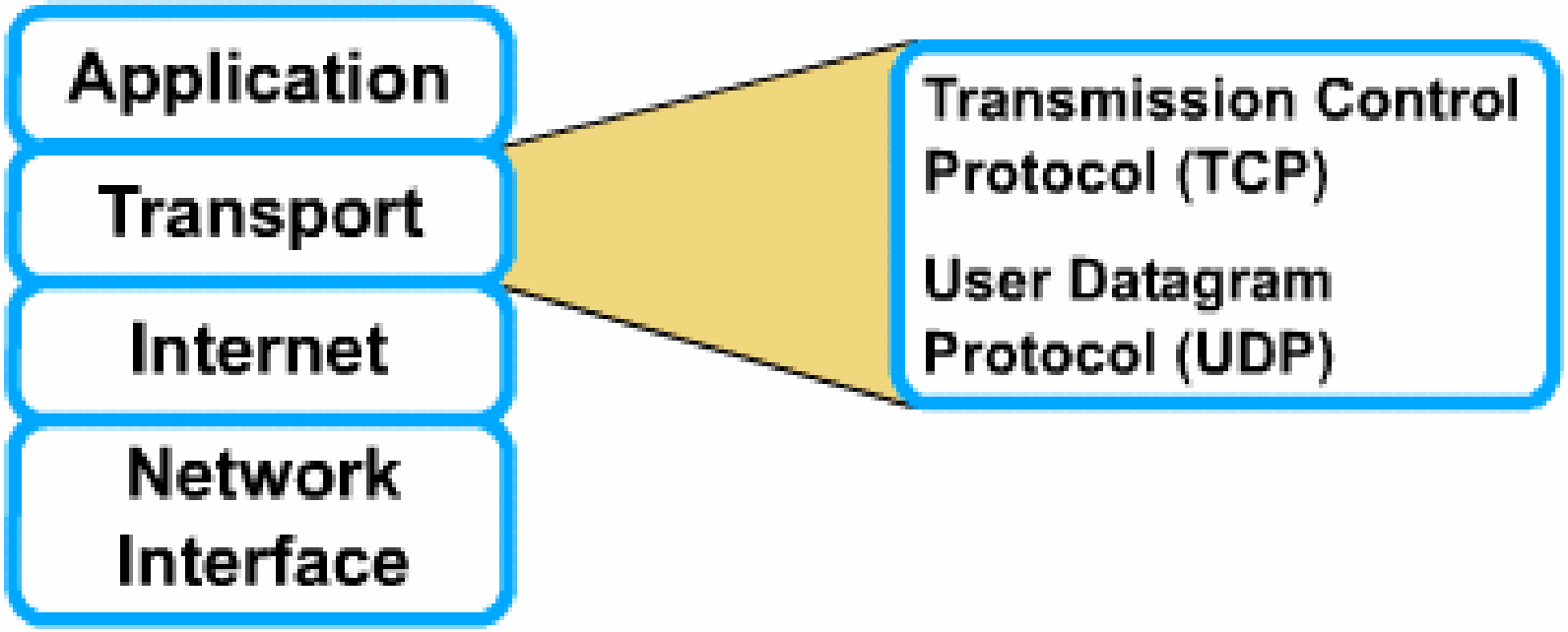
Ethernet Header			IP Datagram from above				Ethernet Trailer
MAC Destination Address **03-0D-17-8A-F1-32**	MAC Source Address 00-0C-04-17-91-CC	Other Header Info	IP Header Info	IP Original Source Address 172.17.10.10	IP Final Destination Address **172.16.10.1**	Data	FCS

TẦNG VẬN CHUYỂN



- Cung cấp việc vận chuyển dữ liệu trong suốt giữa các hệ thống đầu cuối (end systems).

TẦNG VẬN CHUYỂN TRONG TCP/IP

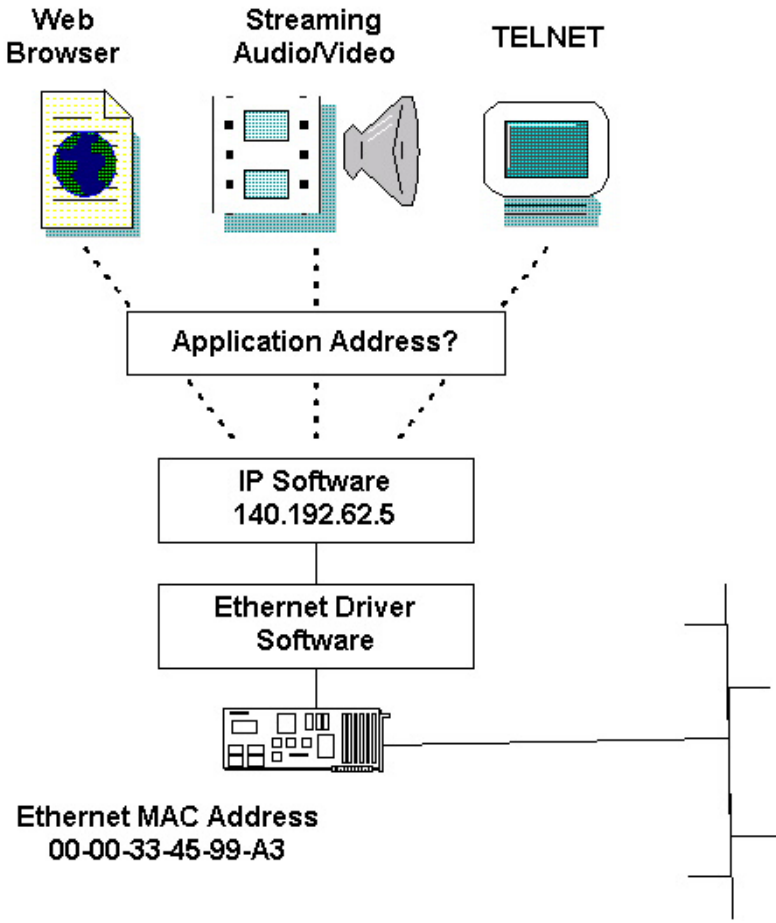


TCP (Transmission Control Protocol)

- Một giao thức phổ biến ở tầng vận chuyển
  - Được chuẩn hoá trên RFC 793
- **Các đặc điểm chính**
  - Tin cậy.
  - Hướng kết nối (Connection oriented).
  - Hoạt động hai chiều đồng thời.
  - Phân mảnh thông điệp và ráp lại ở đích.

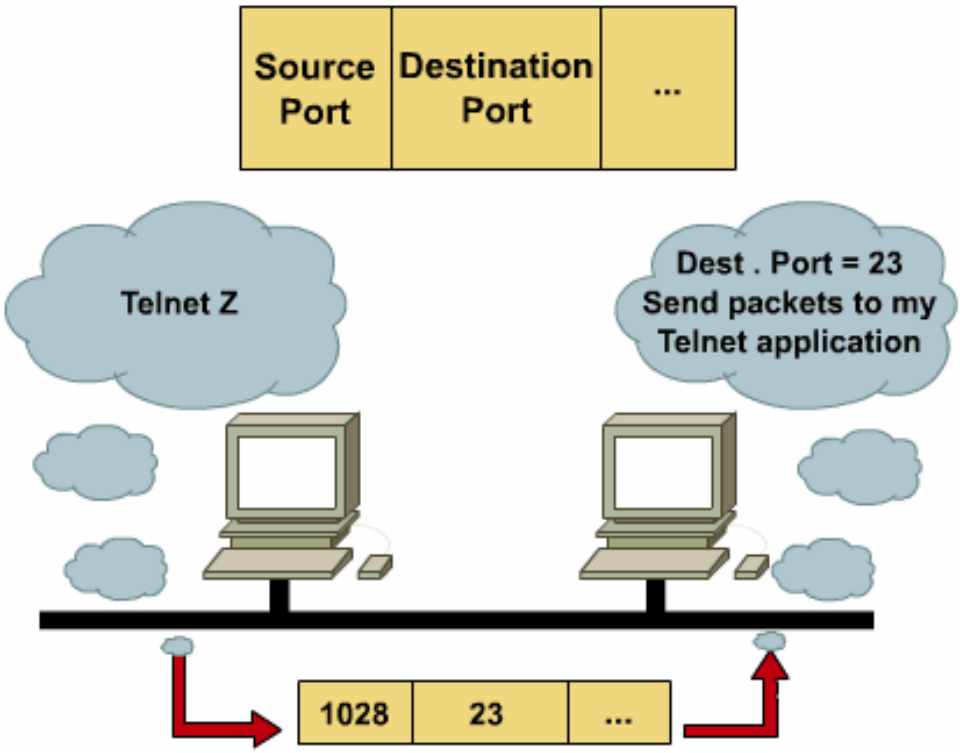
Địa chỉ ứng dụng

- Không thể dùng địa chỉ IP.
- Phải làm việc trên tất cả các hệ thống máy tính.
- Không thể sử dụng các chỉ số trên hệ điều hành:
  - Process ID
  - Task number
  - Job name



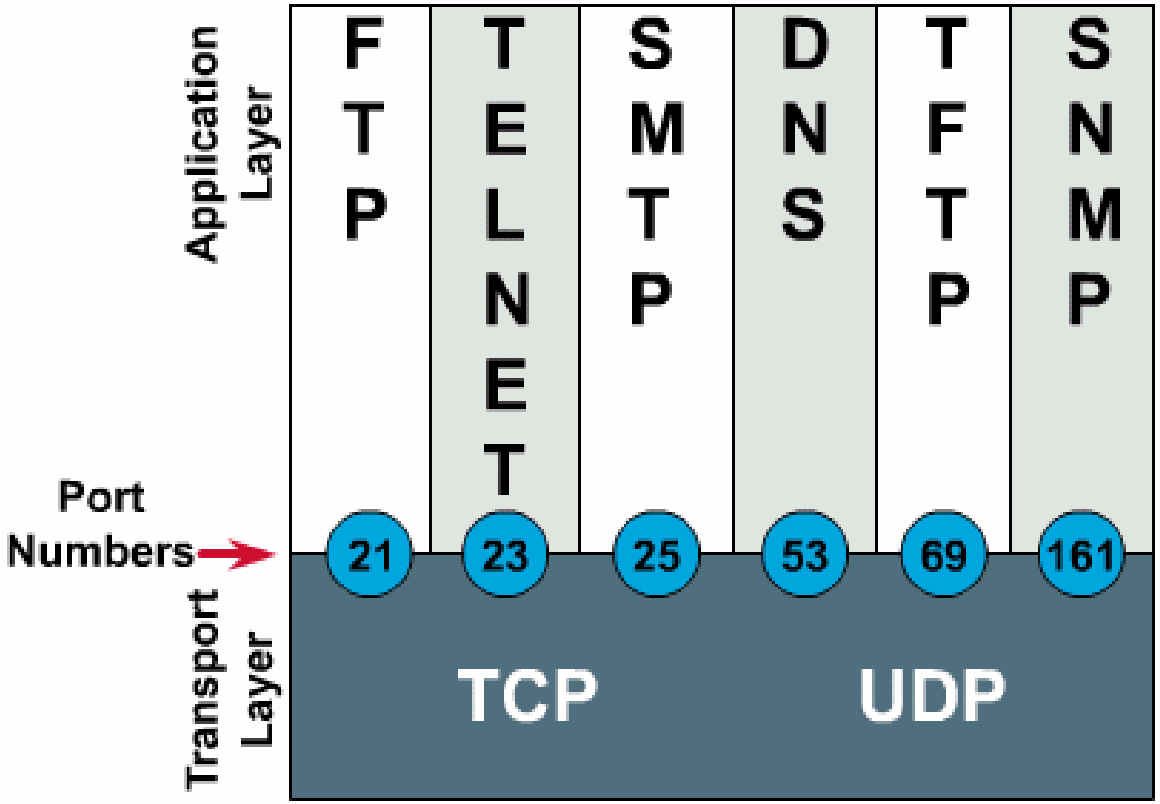
Địa chỉ ứng dụng

- **Chỉ số port**
  - Mỗi ứng dụng được gán một chỉ số nguyên.
- **Mô hình Client-Server.**
  - Server: Luôn dùng mộ chỉ số port đã biết (well-known port)
  - Client: Lấy chỉ số port chưa dùng từ hệ thống.



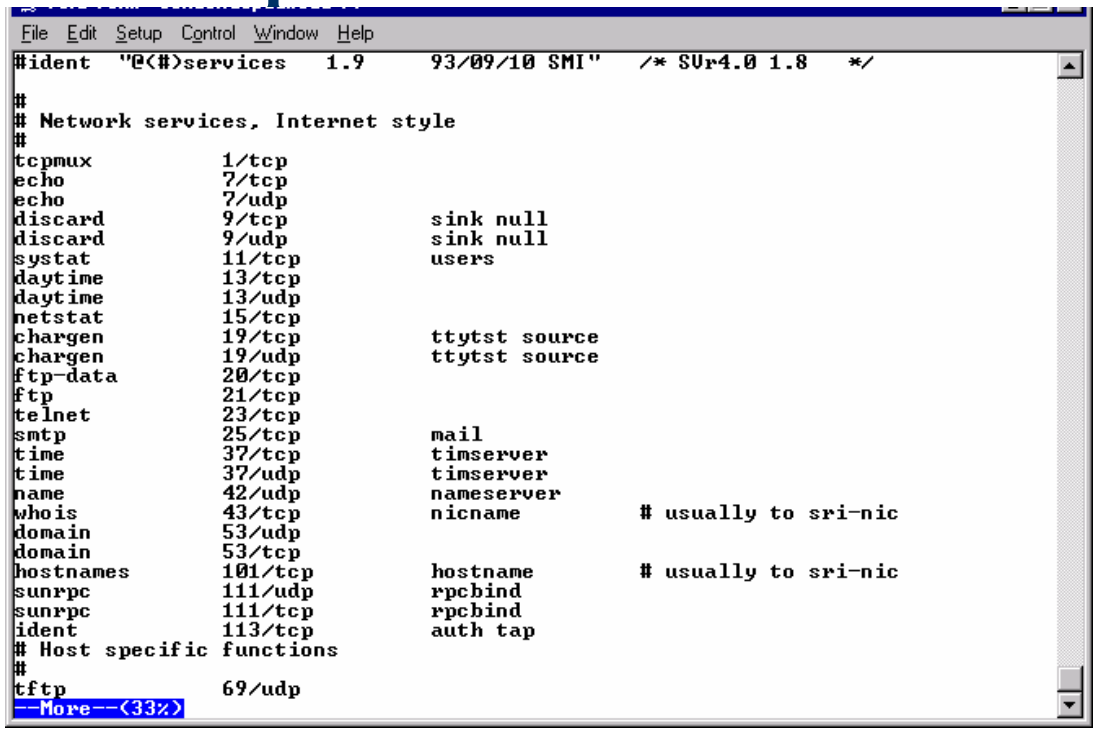
Địa chỉ ứng dụng

- Chỉ số port và các dịch vụ



Địa chỉ ứng dụng

- Chỉ số các port chuẩn



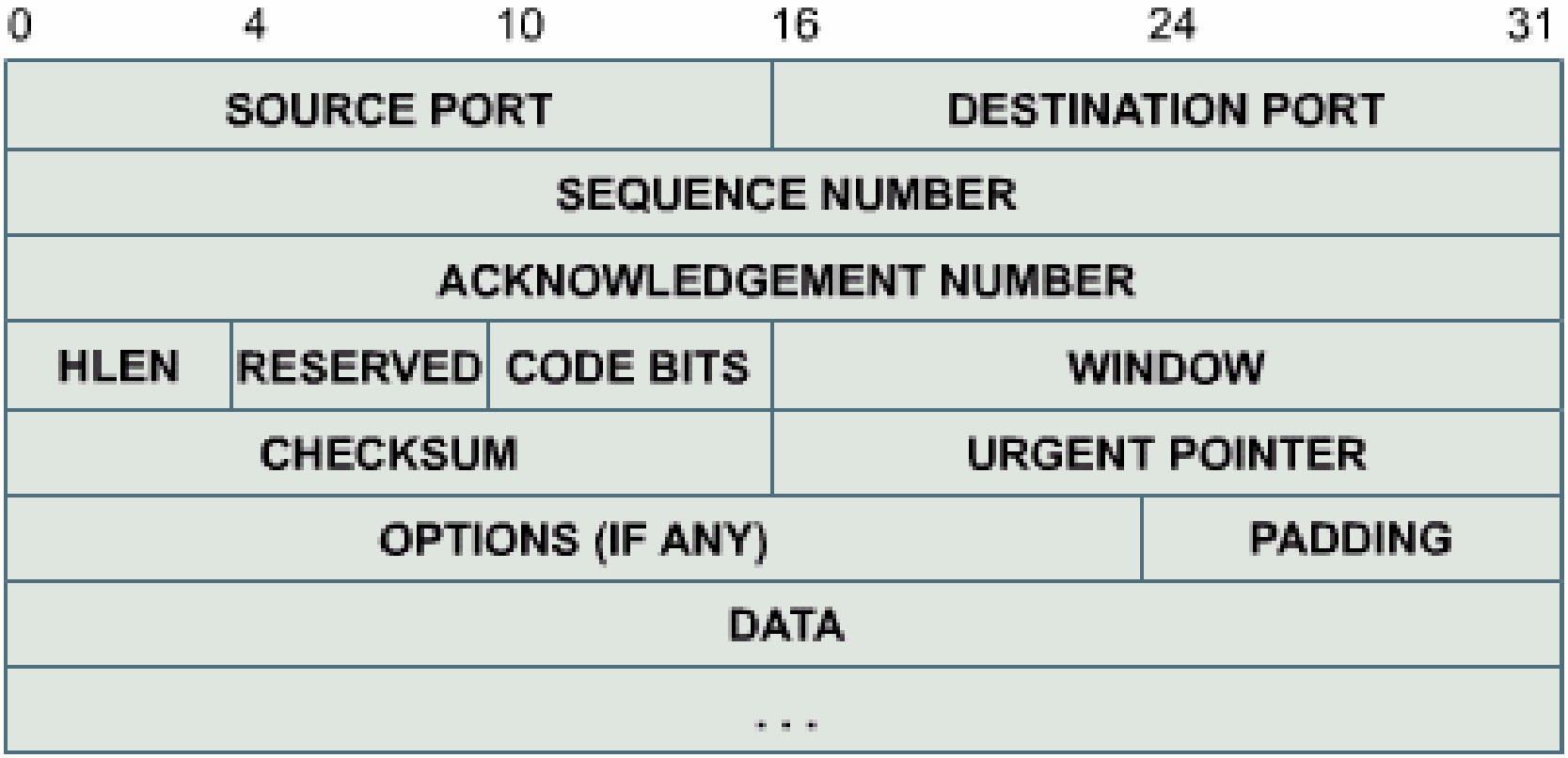
File Edit Setup Control Window Help

```
#ident  "@(#)services   1.9     93/09/10 SMI"   /* SVr4.0 1.8   */

#
# Network services, Internet style
#
tcpmux          1/tcp
echo            7/tcp
echo            7/udp
discard         9/tcp           sink null
discard         9/udp           sink null
systat          11/tcp          users
daytime         13/tcp
daytime         13/udp
netstat         15/tcp
chargen         19/tcp          ttytst source
chargen         19/udp          ttytst source
ftp-data        20/tcp
ftp             21/tcp
telnet          23/tcp
smtp            25/tcp          mail
time            37/tcp          timserver
time            37/udp          timserver
name            42/udp          nameserver
whois           43/tcp          nicname         # usually to sri-nic
domain          53/udp
domain          53/tcp
hostnames       101/tcp         hostname        # usually to sri-nic
sunrpc          111/udp         rpcbind
sunrpc          111/tcp         rpcbind
ident           113/tcp         auth tap
# Host specific functions
#
tftp            69/udp
--More--(33%)
```

  - Xem tập tin /etc/services trên các hệ thống UNIX và \winnt\system32\drivers\etc\services trên Windows NT

TCP (Transmission Control Protocol)

- **Định dạng của TCP Segment**

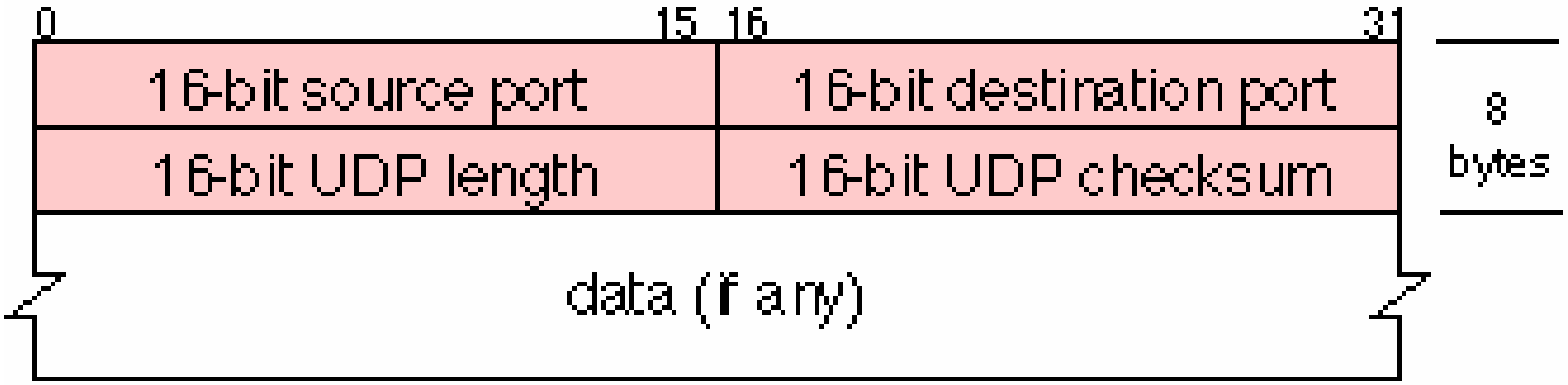


UDP (User Datagram Protocol)

- Được chuẩn hoá trên RFC 768
- **Các đặc điểm chính:**
  - Giao thức không kết nối (Connectionless protocol)
  - Phân phối thông điệp không tin cậy nhưng best effort
  - Có khả năng phát hiện lỗi (trường checksum)
  - Không điều khiển dòng (không window)
  - Không điều khiển lỗi (không ACK)
  - Cung cấp địa chỉ ứng dụng (chỉ số port)

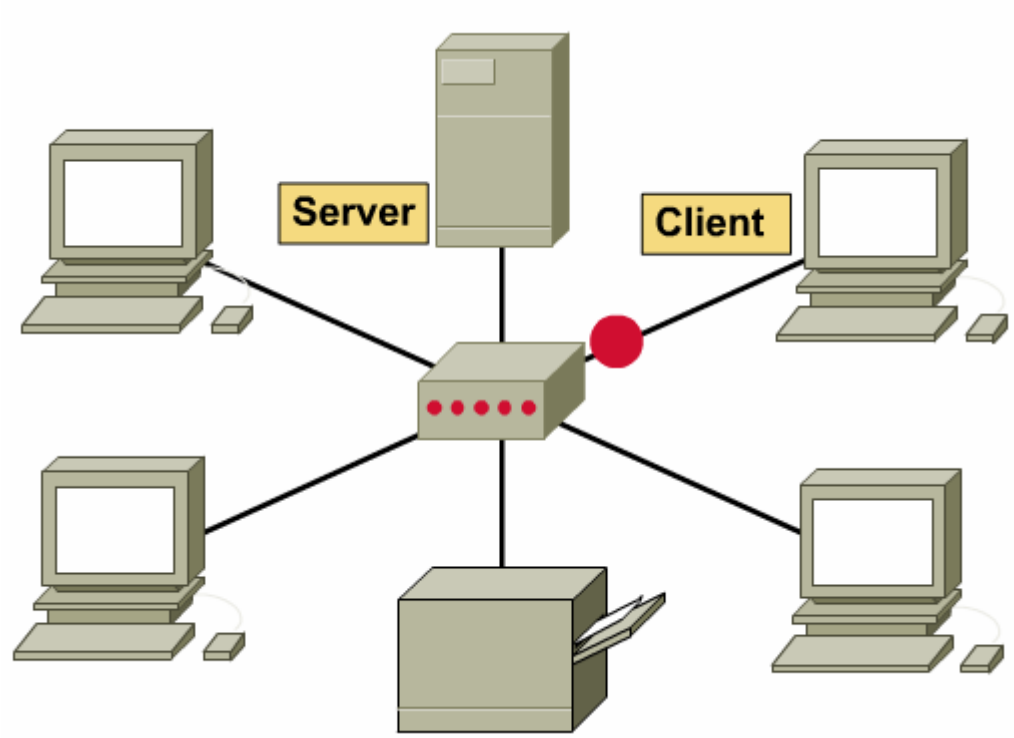
UDP (User Datagram Protocol)

- **Định dạng thông điệp UDP**



Ứng dụng mạng

- Các ứng dụng dạng Client – Server.
- Các ứng dụng FTP, WWW, E-mail.



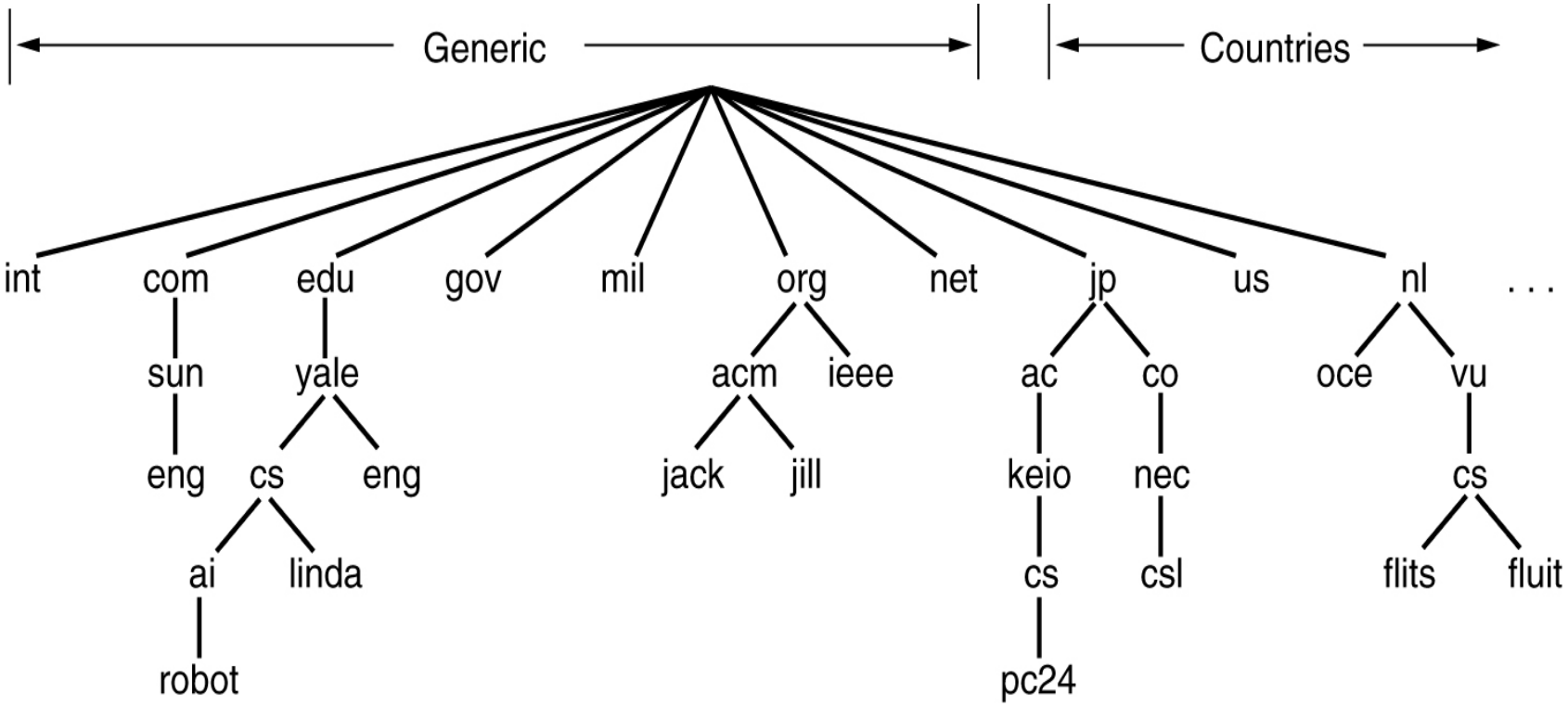
Hệ thống tên miền (DNS)

- **Khái niệm**
  - Ánh xạ tên gợi nhớ thành địa chỉ IP và ngược lại.

192.31.7.130	CISCO.COM
204.71.177.35	YAHOO.COM
152.163.210.7	AOL.COM
198.150.15.234	MAT-MADISON.COM
207.46.131.15	MICROSOFT.COM
192.233.80.9	NOVELL.COM

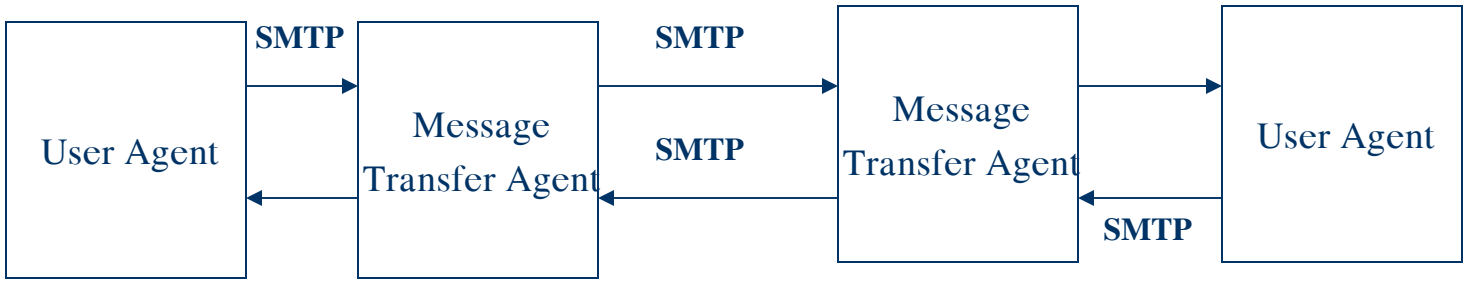
Hệ thống tên miền (DNS)

- **DNS Name Space:**



Hệ thống thư tín điện tử (E-mail)

- **Khái niệm**
  - Hệ thống Email hiện nay là hệ thống email cuả ARPANET được xây dựng năm 1982 dựa trên RFC 821 (Transmission protocol) và RFC 822 (message format).
- **Cấu trúc và dịch vụ**
  - User agent
  - Message transfer agent



Hệ thống thư tín điện tử (E-mail)

- **Chuyển message**
  - Trên Internet, email được chuyển bằng cách máy nguồn thiết lập một cầu nối TCP qua port 25 của máy đích. Chương trình được chạy trên port này là SMTP (Simple Mail Transfer Protocol)
  - Nó sẽ lắng nghe tại port 25 và cho phép thiết lập các cầu nối để từ đó đọc các message và chuyển đến các địa chỉ tương ứng.
- **Nhận message từ mail server đến mail client**
  - Trên Internet, email được nhận bằng cách máy nguồn thiết lập một cầu nối TCP qua port 110 của máy mail server. Chương trình được chạy trên port này là POP3 (Post Office Protocol Ver 3)
  - Nó sẽ lắng nghe tại port 110 và cho phép thiết lập các cầu nối để từ đó đọc các message và chuyển đến các mail client.
  - POP2 & IMAP

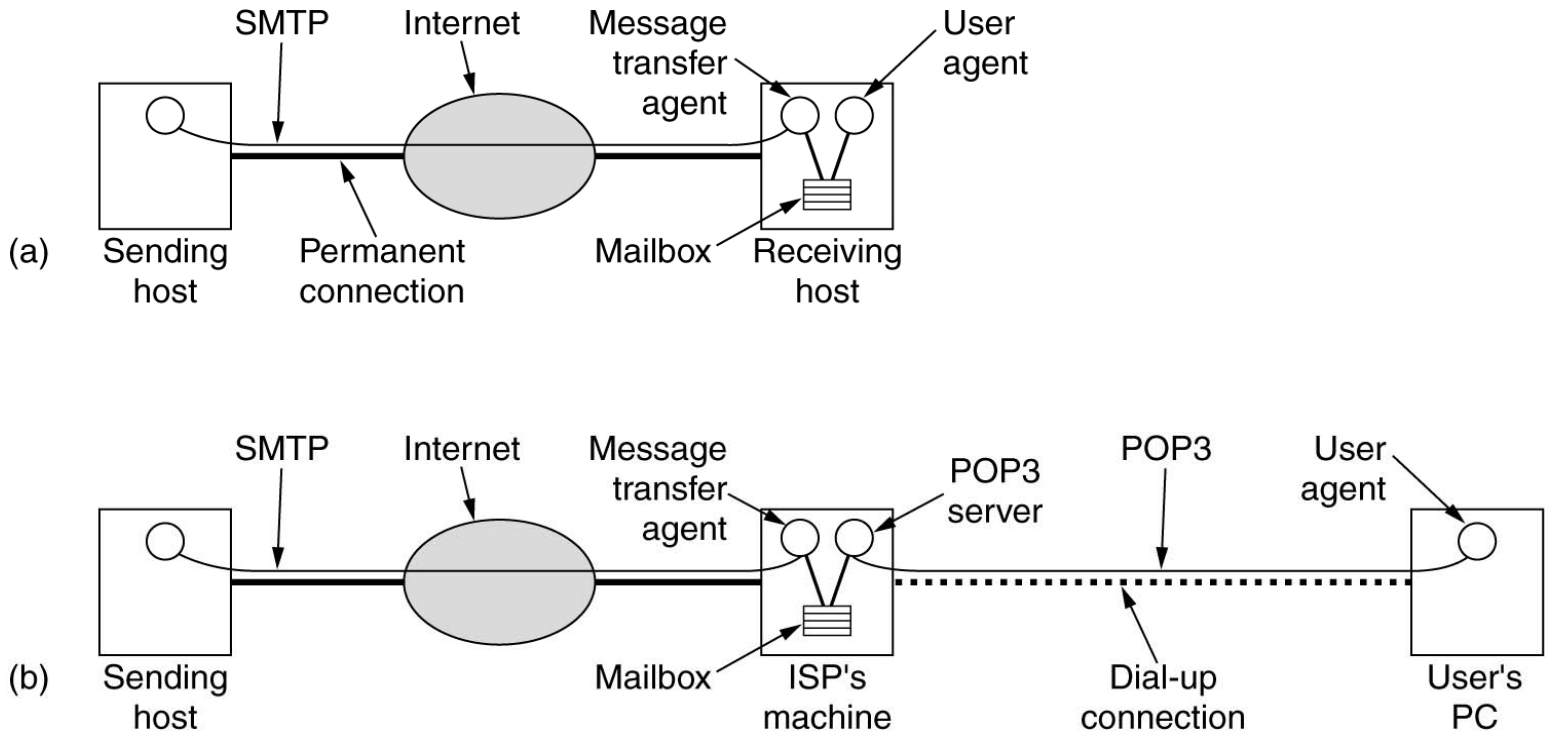
Hệ thống thư tín điện tử (E-mail)

- SMTP

```
                S: 220 xyz.com SMTP service ready
C: HELO abcd.com
                S: 250 xyz.com says hello to abcd.com
C: MAIL FROM: <elinor@abcd.com>
                S: 250 sender ok
C: RCPT TO: <carolyn@xyz.com>
                S: 250 recipient ok
C: DATA
                S: 354 Send mail; end with "." on a line by itself
C: From: elinor@abcd.com
C: To: carolyn@xyz.com
C: MIME-Version: 1.0
C: Message-Id: <0704760941.AA00747@abcd.com>
C: Content-Type: multipart/alternative; boundary=qwertyuiopasdfghjklzxcvbnm
C: Subject: Earth orbits sun integral number of times
C:
C: This is the preamble. The user agent ignores it. Have a nice day.
C:
C: --qwertyuiopasdfghjklzxcvbnm
C: Content-Type: text/enriched
C:
C: Happy birthday to you
C: Happy birthday to you
C: Happy birthday dear <bold> Carolyn </bold>
C: Happy birthday to you
C:
C: --qwertyuiopasdfghjklzxcvbnm
C: Content-Type: message/external-body;
C:          access-type="anon-ftp";
C:          site="bicycle.abcd.com";
C:          directory="pub";
C:          name="birthday.snd"
C:
C: content-type: audio/basic
C: content-transfer-encoding: base64
C: --qwertyuiopasdfghjklzxcvbnm
C: .
                S: 250 message accepted
C: QUIT
                S: 221 xyz.com closing connection
```

Hệ thống thư tín điện tử (E-mail)

- **Phân phối thư**



Hệ thống thư tín điện tử (E-mail)

- **Nhận thư bằng POP3**

```
                    S: +OK POP3 server ready
C: USER carolyn
                    S: +OK
C: PASS vegetables
                    S: +OK login successful
C: LIST
                    S: 1 2505
                    S: 2 14302
                    S: 3 8122
                    S: .
C: RETR 1
                    S: (sends message 1)
C: DELE 1
C: RETR 2
                    S: (sends message 2)
C: DELE 2
C: RETR 3
                    S: (sends message 3)
C: DELE 3
C: QUIT
                    S: +OK POP3 server disconnecting
```

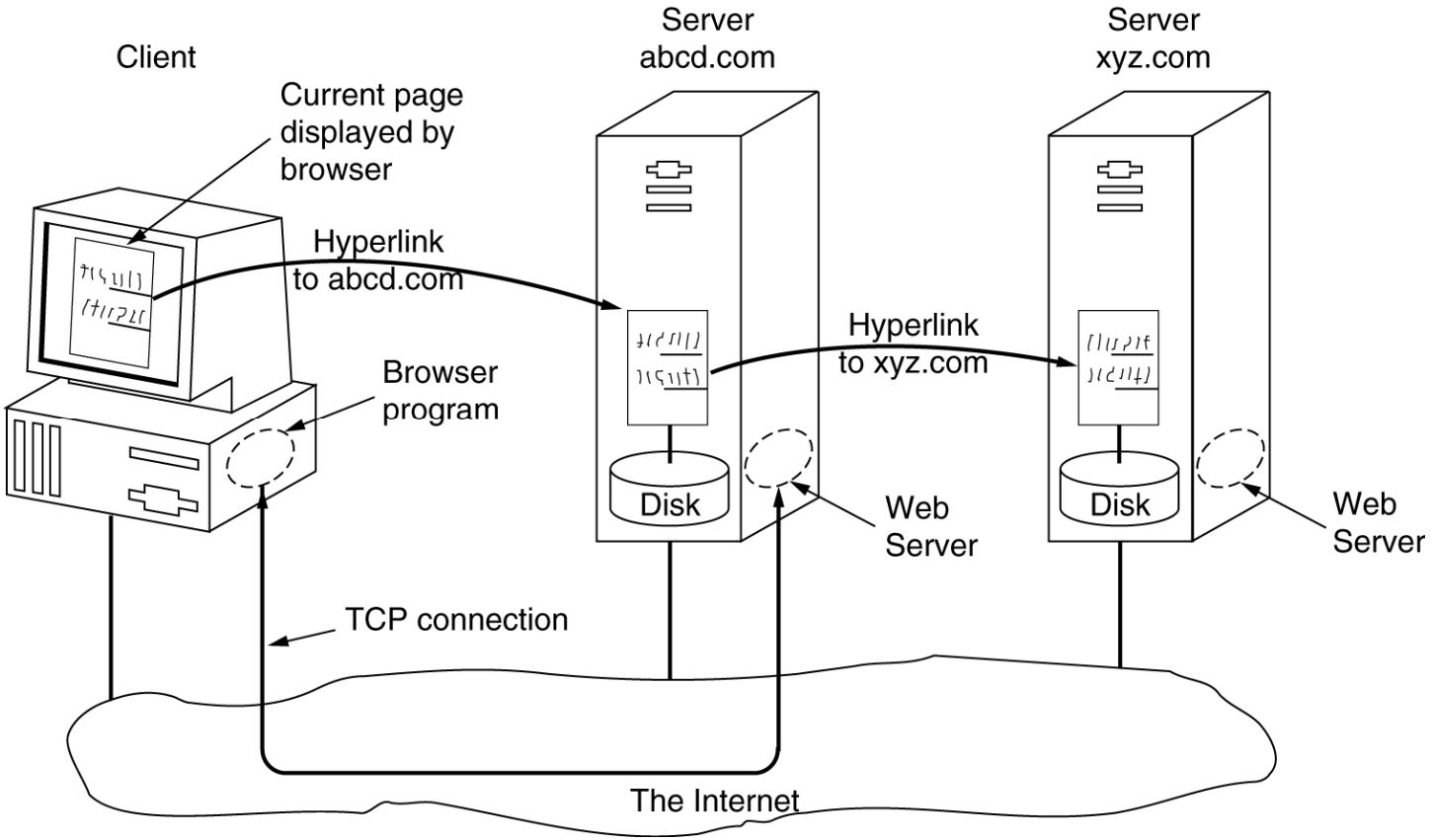
Hệ thống thư tín điện tử (E-mail)

- **POP3 & IMAP**

Feature	POP3	IMAP
Where is protocol defined?	RFC 1939	RFC 2060
Which TCP port is used?	110	143
Where is e-mail stored?	User's PC	Server
Where is e-mail read?	Off-line	On-line
Connect time required?	Little	Much
Use of server resources?	Minimal	Extensive
Multiple mailboxes?	No	Yes
Who backs up mailboxes?	User	ISP
Good for mobile users?	No	Yes
User control over downloading?	Little	Great
Partial message downloads?	No	Yes
Are disk quotas a problem?	No	Could be in time
Simple to implement?	Yes	No
Widespread support?	Yes	Growing

World Wide Web

- **Mô hình**

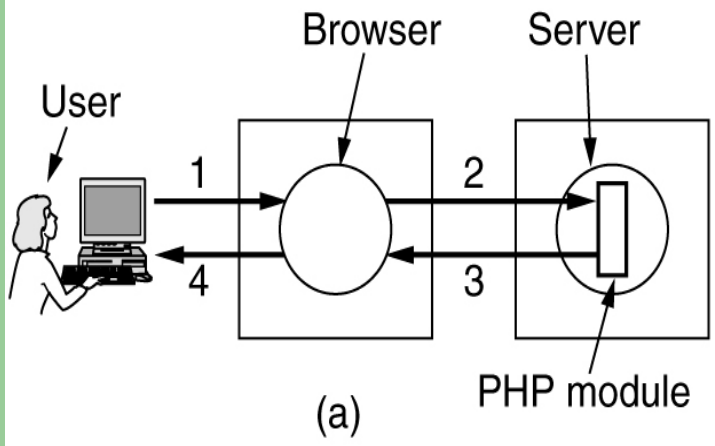


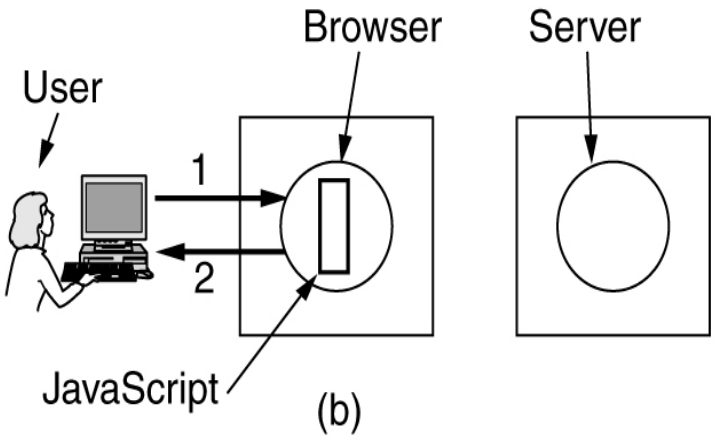
World Wide Web

- **Mô hình**
  - WWW là một hệ thống có cấu trúc để truy cập các văn bản được đặt khắp nơi trên hàng ngàn cái máy tính trên toàn thế giới.
- **Server :**
  - Web Server : lắng nghe tại port TCP 80
  - Giao thức sử dụng : HTTP (HyperText Transfer Protocol)

World Wide Web

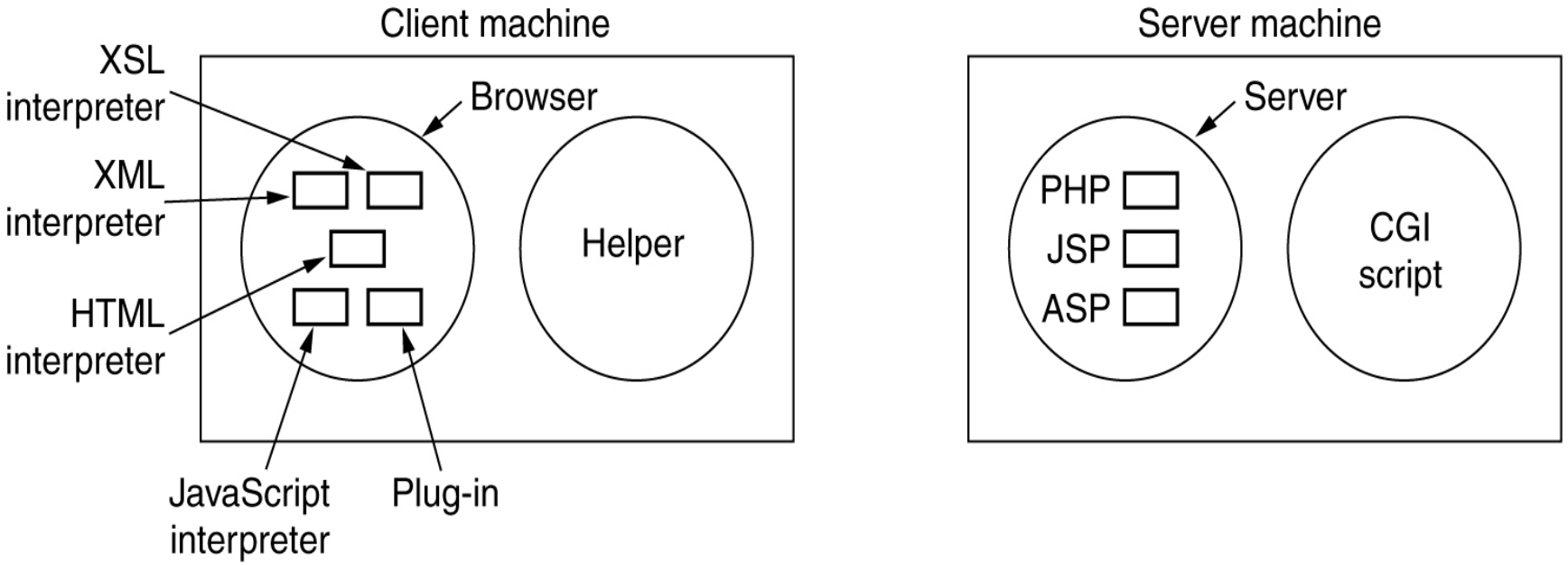
- **Server Side & Client Side**





World Wide Web

- **Server Side & Client Side**



World Wide Web

- **Các phương thức của HTTP**

Method	Description
GET	Request to read a Web page
HEAD	Request to read a Web page's header
PUT	Request to store a Web page
POST	Append to a named resource (e.g., a Web page)
DELETE	Remove the Web page
TRACE	Echo the incoming request
CONNECT	Reserved for future use
OPTIONS	Query certain options

World Wide Web

- **Ví dụ**

```
Trying 4.17.168.6...
Connected to www.ietf.org.
Escape character is '^]'.
HTTP/1.1 200 OK
Date: Wed, 08 May 2002 22:54:22 GMT
Server: Apache/1.3.20 (Unix) mod_ssl/2.8.4 OpenSSL/0.9.5a
Last-Modified: Mon, 11 Sep 2000 13:56:29 GMT
ETag: "2a79d-c8b-39bce48d"
Accept-Ranges: bytes
Content-Length: 3211
Content-Type: text/html
X-Pad: avoid browser bug

<html>
<head>
<title>IETF RFC Page</title>

<script language="javascript">
function url() {
 var x = document.form1.number.value
 if (x.length == 1) {x = "000" + x }
 if (x.length == 2) {x = "00" + x }
 if (x.length == 3) {x = "0" + x }
 document.form1.action = "/rfc/rfc" + x + ".txt"
 document.form1.submit
}
</script>

</head>
```

TỔNG KẾT

- **Khái quát về mạng máy tính**
  - Khái niệm
  - Cấu hình
  - Các thành phần cơ bản
- **Mô hình OSI**
  - Protocol
  - Layer
  - OSI Model
- **Bộ giao thức TCP/IP**
  - IP
  - TCP, UDP
  - DNS, SMTP, POP3, HTTP

Nội dung môn học

CHƯƠNG 1: GIỚI THIỆU VỀ TCP/IP

CHƯƠNG 2: THIẾT KẾ GIẢI THUẬT CHO CHƯƠNG TRÌNH CLIENT/SERVER

CHƯƠNG 3: LẬP TRÌNH MẠNG TRÊN CÁC MÔI TRƯỜNG PHỔ DỤNG

CHƯƠNG 4: LẬP TRÌNH MẠNG VỚI JAVA

Nội dung môn học(tt)

CHƯƠNG 5: LẬP TRÌNH WEB – CGI

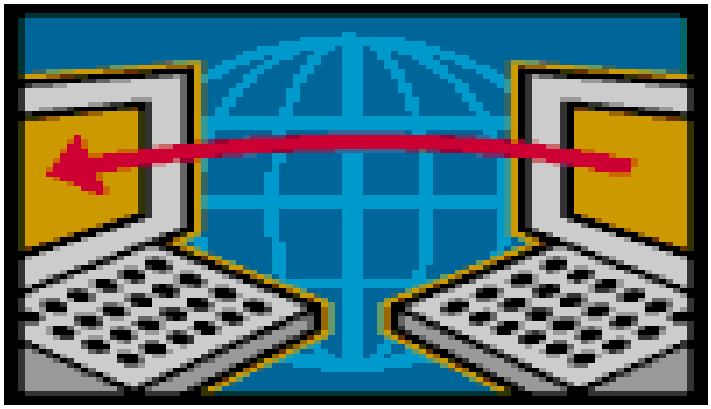
CHƯƠNG 6: LẬP TRÌNH WEB VỚI CÁC CÔNG NGHỆ PHỔ BIẾN

CHƯƠNG 7: ỨNG DỤNG XML TRONG LẬP TRÌNH MẠNG

CHƯƠNG 8: BẢO MẬT DỮ LIỆU TRUYỀN

Tài liệu tham khảo

- [1] Douglas E. Comer, *Internetworking with TCP/IP*, Prentice-Hall,1993.
- [2] W. Richard Stevens, *Unix Network Programming*, Prentice-Hall,1990.
- [3] Arthur Dumas, *Programming Winsock*, Sams Publishing,1995.
- [4] Merlin, Conrad Hughes ..., *Java Network Programming*, Manning Publications Co., 1997.
- [5] D. Travis Dewire, *Second-Generation Client/Server Computing*, Mc Graw-Hill, 1997.
- [6] John Shapley Gray, *Interprocess Comunication in UNIX*, Prentice-Hall,1997.
- [7] Deitel & Deitel. *Java How to program, 3th edition*, Prentice-Hall,1999.
- [8] Richard Anderson, ..., *Professional Active Server Pages 3.0*, Wrox Press, 1999.
- [9] Marty Hall, *Core Servlet and Java Server Pages*, Prentice-Hall PTR, 2000
- [10] MSDN.
- [11] Tập tài liệu RFC.



CHƯƠNG 1
GIỚI THIỆU VỀ TCP/IP

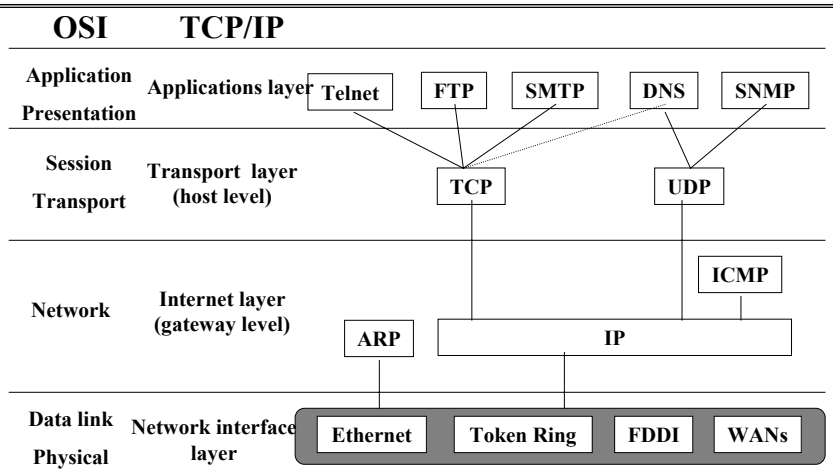
1.1 Tổng quát về TCP/IP.

1.2 Các giao thức và dịch vụ trên TCP/IP.

1.3 Khái niệm về Socket.

1.4 Một số ứng dụng mạng.

1.1 Tổng quát về TCP/IP.



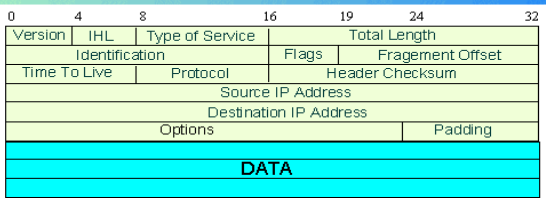
1.1 Tổng quát về TCP/IP (tt)

- Một số đặc tính :
  - Độc lập về hình thái của mạng.
  - Độc lập về phần cứng của mạng.
  - Các chuẩn giao thức mở.
  - Mô hình địa chỉ toàn cầu.
  - Nền tảng client/server mạnh mẽ.
  - Các chuẩn về giao thức ứng dụng mạnh mẽ.

1.1 Tổng quát về TCP/IP (tt)

4. IP

IP Frame



- **Flags (3): control fragmentation.**
  - Bit_0 =0. reserved Bit_1=may(0)/may-not(1) fragment . Bit_2= last (0)/more(1) fragment
- **Fragment Offset(13): pointer to fragment location**
- **Time to live: number of seconds for datagram to remain alive**
- **Protocol: identify higher level protocol user (UDP, TCP..)**
- **Header checksum: error detection bits**
- **Source/destination addresses: contain 32 bits Internet address**

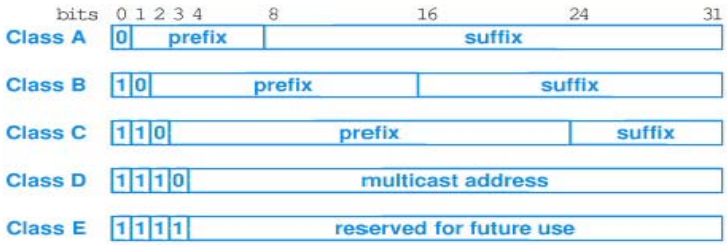
1.1 Tổng quát về TCP/IP (tt)

- Địa chỉ Internet:
  - Định vị duy nhất một máy
  - Chiều dài 32 bit
  - Cấu trúc IP (netid, hostid), các máy trên một mạng có netid giống nhau.
  - Do NIC cấp
  - Cách biểu diễn:

    10101100 00011100 00010000 00000101
    172 28 16 5
    172.28.16.5

1.1 Tổng quát về TCP/IP (tt)

- Phân lớp địa chỉ:
  - Để xác định netid (Network Identifier) và hostid (Host Identifier)



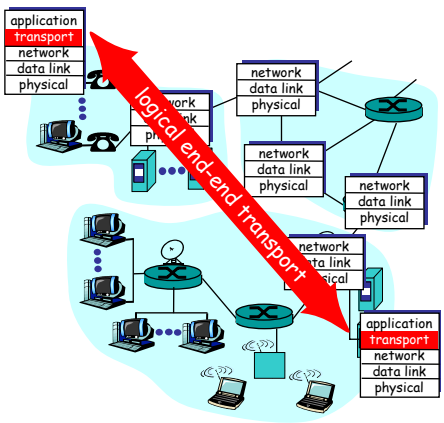
1.1 Tổng quát về TCP/IP (tt)

- Một số địa chỉ IP đặc biệt

Prefix	Suffix	Type Of Address	Purpose
all-0s	all-0s	this computer	used during bootstrap
network	all-0s	network	identifies a network
network	all-1s	directed broadcast	broadcast on specified net
all-1s	all-1s	limited broadcast	broadcast on local net
127	any	loopback	testing

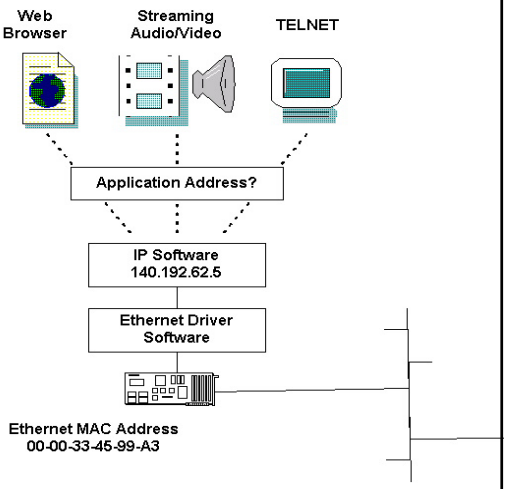
1.1 Tổng quát về TCP/IP (tt)

- Lớp Transport
  - Cung cấp giao tiếp luận lý giữa các processes trên các hosts khác nhau
  - Có hai dạng dịch vụ:
    - TCP (Transmittion Control Protocol)
    - UDP (User Datagram Protocol)



1.1 Tổng quát về TCP/IP (tt)

- Lớp Transport (tt)
  - Mở rộng cách đánh địa chỉ cho process.
  - Địa chỉ port : xác định ứng dụng mạng trên mỗi máy.
  - Địa chỉ của một ứng dụng mạng (IP,port)

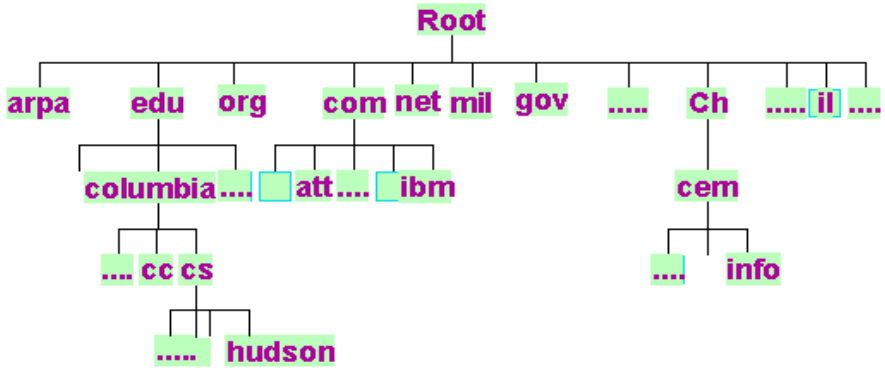


1.2 Các giao thức và dịch vụ

- Hệ thống tên miền DNS (Domain Name System)
  - Dùng chuỗi ký tự để đánh địa chỉ, không phân biệt chữ hoa, thường, mỗi thành phần có thể 63 ký tự và tên đầy đủ không dài quá 255, dưới đây gọi là tên.
  - Tên được đặt theo cây phân cấp
  - Địa chỉ tài nguyên biểu diễn dạng tên được hình thành từ nó cho đến root

1.2 Các giao thức và dịch vụ (tt)

- Hệ thống tên miền DNS (tt)

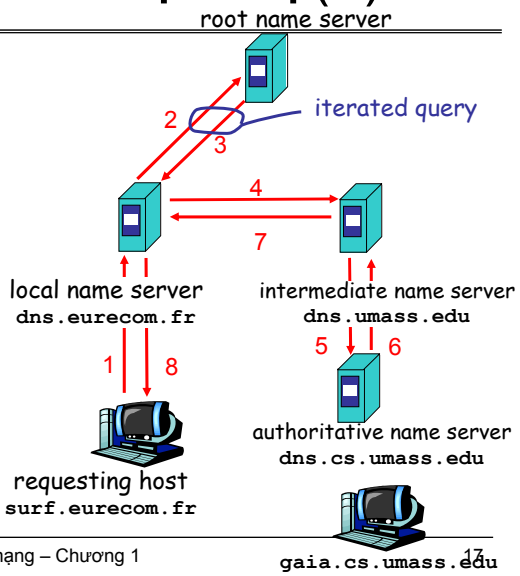


1.2 Các giao thức và dịch vụ(tt)

- Hệ thống tên miền DNS (tt)
  - Network chỉ hiểu địa chỉ IP (binary) => ánh xạ giữa địa chỉ IP và tên.
  - Hệ thống tên miền được hiện thực theo *distributed database*, quản lý theo dạng phân cấp với *name servers*
  - Network chỉ hiểu địa chỉ IP (binary) => ánh xạ giữa địa chỉ IP và tên.
  - Mỗi ứng dụng mạng phải chuyển tên sang địa chỉ IP

1.2 Các giao thức và dịch vụ(tt)

- DNS (tt)
  - Ứng dụng giao tiếp với local name server để hỏi địa chỉ ánh xạ.
  - Local name server sẽ trả lời hoặc request tiếp…



1.2 Các giao thức và dịch vụ(tt)

- Giao thức ở lớp ứng dụng
  - Ứng dụng mạng : trao đổi thông tin giữa các processes trên mạng.
  - Các ứng dụng phải định nghĩa protocol để giao tiếp với nhau.
  - Protocol qui định thứ tự các thông điệp trao đổi, hành động khi nhận mỗi loại thông điệp.
  - Ứng dụng cũng phải hiện thực phần giao tiếp với người dùng.

1.2 Các giao thức và dịch vụ(tt)

- Giao thức ở lớp ứng dụng(tt)
  - User agent là giao tiếp giữa người sử dụng và ứng dụng mạng.
    - Web:browser
    - E-mail: mail reader
    - streaming audio/video: media player

1.2 Các giao thức và dịch vụ(tt)

- Mô hình mạng client/server
  - Server : là phần tử thụ động
    - Chờ yêu cầu từ client, xử lý và trả kết quả cho client
  - Client : là phần tử chủ động
    - Kết nối đến server để gởi yêu cầu.
    - Chờ nhận kết quả trả về và xử lý kết quả.

1.2 Các giao thức và dịch vụ(tt)

- State và Stateless
  - State : lưu giữ trạng thái giữa các lần kết nối (request/response).
  - Stateless : Mỗi lần request/response thì cầu nối hủy bỏ. Không giữ trạng trái trước đó.

1.3 Khái niệm về Socket.

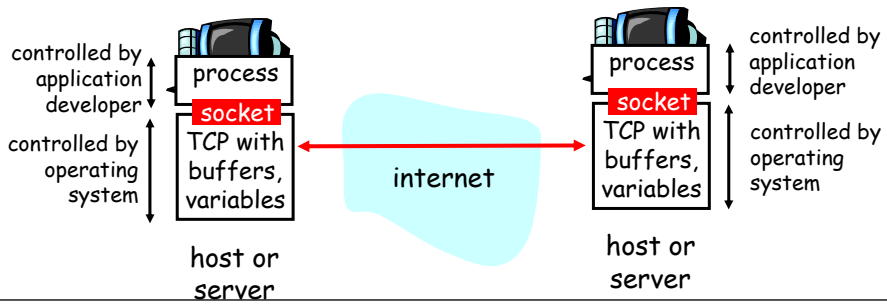
- Socket API
  - Được giới thiệu ở BSD4.1 UNIX, 1981
  - Được ứng dụng khởi tạo, sử dụng hay hủy bỏ
  - Dùng cơ chế client/server
  - Cung cấp hai dịch vụ chuyển dữ liệu thông qua socket API:
    - unreliable datagram
    - reliable, byte stream-oriented

1.3 Khái niệm về Socket(tt)

- Socket :
  - Là môi trường để các process ứng dụng giao tiếp với nhau, process ứng dụng có thể chạy trên cùng một máy hoặc trên hai máy khác nhau.
  - Được ứng dụng tạo ra và sử dụng tuy nhiên được hệ thống (hệ điều hành) kiểm soát.

1.3 Khái niệm về Socket(tt)

- Socket: "cửa" nằm giữa process ứng dụng và end-end-transport protocol (UCP or TCP)
- TCP service: dịch vụ truyền tin cậy chuỗi bytes giữa hai process



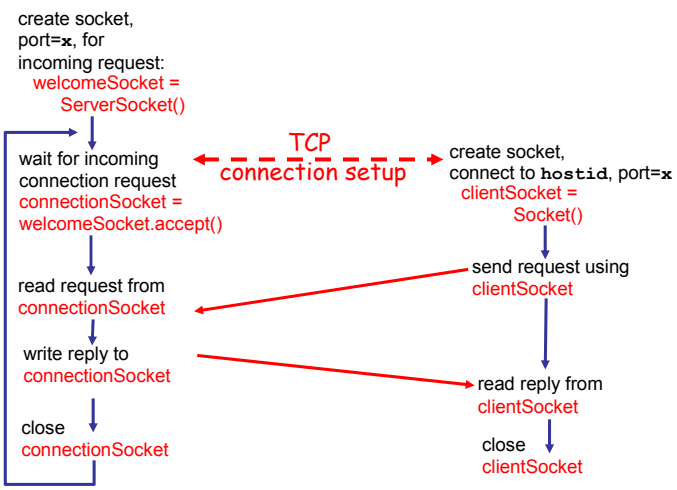
1.3 Khái niệm về Socket(tt)

- Lập trình socket với TCP
  - Client phải kết nối đến server
    - server process phải chạy trước (phần tử thụ động)
    - server phải tạo một socket để lắng nghe và chấp nhận các kết nối từ client
  - Client kết nối đến server bằng cách:
    - Khởi tạo TCP socket ở local
    - Xác định IP address, port number của server process và kết nối đến

1.3 Khái niệm về Socket(tt)

- Lập trình socket với TCP(tt)
  - Sau khi client khởi tạo socket, nó sẽ thiết lập kết nối đến server
  - Khi server nhận yêu cầu kết nối, nó sẽ chấp nhận yêu cầu và khởi tạo socket mới để giao tiếp với client.
    - Cho phép server chấp nhận nhiều client tại một thời điểm.

1.3 Khái niệm về Socket(tt)



Example: Java client (TCP)

```
import java.io.*;
import java.net.*;
class TCPClient {

   public static void main(String argv[]) throws Exception
   {
      String sentence;
      String modifiedSentence;

      BufferedReader inFromUser =
        new BufferedReader(new InputStreamReader(System.in));

      Socket clientSocket = new Socket("hostname", 6789);

      DataOutputStream outToServer =
        new DataOutputStream(clientSocket.getOutputStream());
```

Create input stream

Create client socket, connect to server

Create output stream attached to socket

Example: Java client (TCP), cont.

Create input stream attached to socket

```
BufferedReader inFromServer =
  new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));

sentence = inFromUser.readLine();
```

Send line to server

```
outToServer.writeBytes(sentence + '\n');
```

Read line from server

```
modifiedSentence = inFromServer.readLine();

System.out.println("FROM SERVER: " + modifiedSentence);

clientSocket.close();

}
}
```


Example: Java server (TCP)

```
import java.io.*;
import java.net.*;

class TCPServer {

  public static void main(String argv[]) throws Exception
   {
     String clientSentence;
     String capitalizedSentence;
```

Create welcoming socket at port 6789

```
     ServerSocket welcomeSocket = new ServerSocket(6789);
```

Wait, on welcoming socket for contact by client

```
     while(true) {

       Socket connectionSocket = welcomeSocket.accept();
```

Create input stream, attached to socket

```
      BufferedReader inFromClient =
        new BufferedReader(new
        InputStreamReader(connectionSocket.getInputStream()));
```

Example: Java server (TCP), cont

Create output stream, attached to socket

```
DataOutputStream  outToClient =
  new DataOutputStream(connectionSocket.getOutputStream());
```

Read in line from socket

```
clientSentence = inFromClient.readLine();

capitalizedSentence = clientSentence.toUpperCase() + '\n';
```

Write out line to socket

```
outToClient.writeBytes(capitalizedSentence);
            }
        }
    }
```

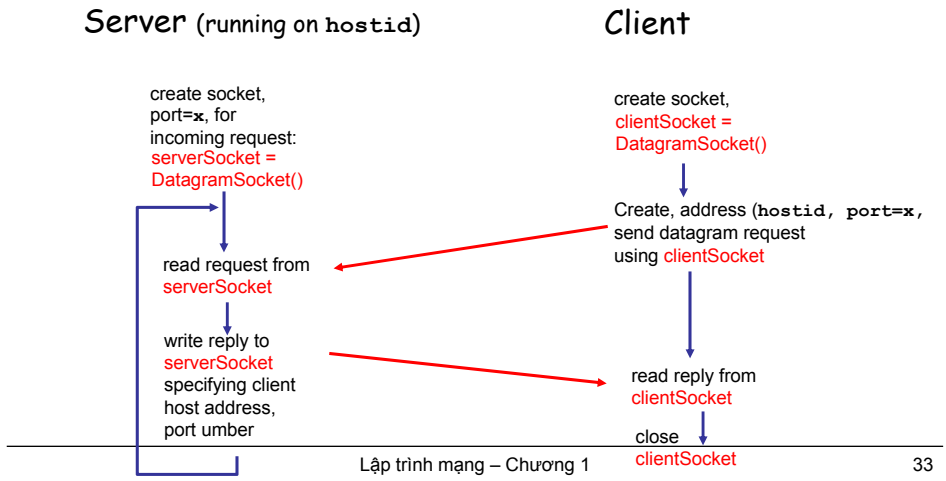
End of while loop, loop back and wait for another client connection

1.3 Khái niệm về Socket(tt)

- Lập trình socket với UTP
  - Cung cấp cơ chế truyền không tin cậy các nhóm các byte (datagrams) giữa client và server.
  - Không cần thiết lập kết nối giữa client với server.
  - Sender phải gởi kèm địa chỉ IP và port đích
  - Server khi nhận dữ liệu sẽ phân tích địa chỉ của sender để truyền lại.

1.3 Khái niệm về Socket(tt)

- Lập trình socket với UTP(tt)



Example: Java client (UDP)

```
import java.io.*;
import java.net.*;

class UDPClient {
   public static void main(String args[]) throws Exception
   {
```

Create input stream →

```
BufferedReader inFromUser =
  new BufferedReader(new InputStreamReader(System.in));
```

Create client socket →

```
DatagramSocket clientSocket = new DatagramSocket();
```

Translate hostname to IP address using DNS →

```
InetAddress IPAddress = InetAddress.getByName("hostname");

byte[] sendData = new byte[1024];
byte[] receiveData = new byte[1024];

String sentence = inFromUser.readLine();
sendData = sentence.getBytes();
```

Example: Java client (UDP), cont.

Create datagram with data-to-send, length, IP addr, port

Send datagram to server

Read datagram from server

```
                DatagramPacket sendPacket =
                  new DatagramPacket(sendData, sendData.length, IPAddress, 9876);

                clientSocket.send(sendPacket);

                DatagramPacket receivePacket =
                   new DatagramPacket(receiveData, receiveData.length);

                clientSocket.receive(receivePacket);

                String modifiedSentence =
                    new String(receivePacket.getData());

                System.out.println("FROM SERVER:" + modifiedSentence);
                clientSocket.close();
                }
            }
```


Example: Java server (UDP)

Create datagram socket at port 9876

Create space for received datagram

Receive datagram

```
import java.io.*;
import java.net.*;

class UDPServer {
  public static void main(String args[]) throws Exception
    {
      DatagramSocket serverSocket = new DatagramSocket(9876);

      byte[] receiveData = new byte[1024];
      byte[] sendData  = new byte[1024];

      while(true)
       {
        DatagramPacket receivePacket =
            new DatagramPacket(receiveData, receiveData.length);
        serverSocket.receive(receivePacket);
```

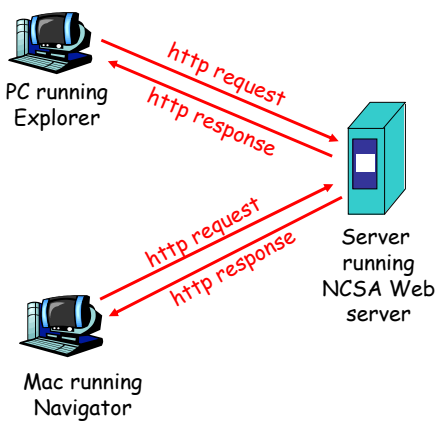
Example: Java server (UDP), cont

```
                String sentence = new String(receivePacket.getData());
Get IP addr
port #, of
sender          InetAddress IPAddress = receivePacket.getAddress();

                int port = receivePacket.getPort();

                        String capitalizedSentence = sentence.toUpperCase();

Create datagram
to send to client   sendData = capitalizedSentence.getBytes();

                DatagramPacket sendPacket =
                  new DatagramPacket(sendData, sendData.length, IPAddress,
                                port);
Write out
datagram
to socket       serverSocket.send(sendPacket);
              }
            }
          }
```

End of while loop, loop back and wait for another datagram

1.4 Một số ứng dụng mạng.

- World Wide Web (W W W)
  - Dùng giao thức **http**: hypertext transfer protocol
  - Web's application layer protocol
  - Mô hình client/server
    - *client:* browser gởi yêu cầu, nhận và hiển thị kết quả.
    - *server:* Web server gởi kết quả cho client đối với mỗi request.
  - http1.0: RFC 1945
  - http1.1: RFC 2068



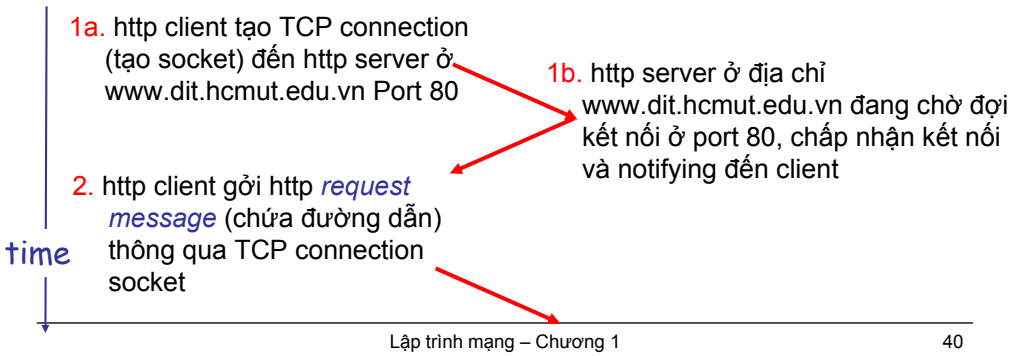
1.4 Một số ứng dụng mạng(tt)

- W W W (tt)
  - http: TCP transport service:
    - client khởi tạo TCP connection (tạo socket) đến server, port 80 (default)
    - server chấp nhận kết nối từ client
    - http messages (application-layer protocol messages) được trao đổi giữa browser (http client) và Web server (http server)
    - đóng TCP connection

1.4 Một số ứng dụng mạng(tt)

- W W W (tt) – Ví dụ
  - User đánh địa chỉ URL lên browser

http://www.dit.hcmut.edu.vn/~phu/courses/net-programming/index.html



1.4 Một số ứng dụng mạng(tt)

- W W W (tt) – Ví dụ

time

3. http server nhận yêu cầu, tạo *response message* (/~phu/courses/net-programming/index.html) trả kết quả, gởi message qua socket

4. http server đóng cầu nối TCP

5. http client nhận response message chứa file html và hiển thị. Phân tích file html để tìm các liên kết

6. Lặp lại các bước 1-5 cho mỗi liên kết(object)

1.4 Một số ứng dụng mạng(tt)

- W W W (tt)
  - Có hai dạng message trong http : *request, response*
  - http request message:
    - ASCII (human-readable format)

1.4 Một số ứng dụng mạng(tt)

- W W W (tt)
  - http request message:

request line (GET, POST, HEAD commands)

```
GET /~phu/index.html HTTP/1.0
User-agent: Mozilla/4.0
Accept: text/html, image/gif,image/jpeg
Accept-language:vn
```

header lines

(extra carriage return, line feed)

Carriage return, line feed indicates end of message

1.4 Một số ứng dụng mạng(tt)

- W W W (tt)
  - http response message:

status line (protocol status code status phrase)

```
HTTP/1.0 200 OK
Date: Thu, 06 Aug 1998 12:00:15 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Mon, 22 Jun 1998 …...
Content-Length: 6821
Content-Type: text/html
```

header lines

```
data data data data data ...
```

data, e.g., requested html file

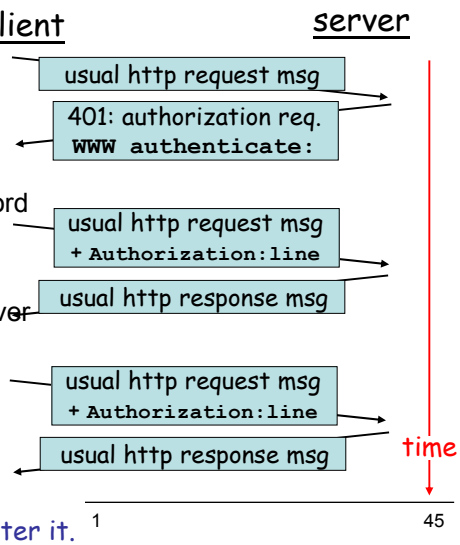
1.4 Một số ứng dụng mạng(tt)

Authentication goal: control access to server documents

- stateless: client must present authorization in each request
- authorization: typically name, password
  - `authorization:` header line in request
  - if no authorization presented, server refuses access, sends

    `WWW authenticate:`

    header line in response



Browser caches name & password so that user does not have to repeatedly enter it.

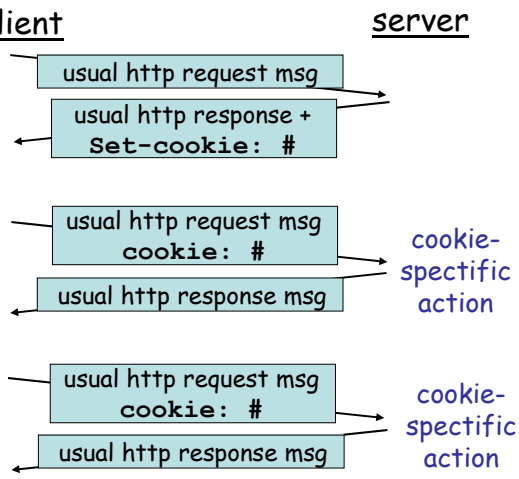
1.4 Một số ứng dụng mạng(tt)

Cookies

- server sends "cookie" to client in response mst

  `Set-cookie: 1678453`
- client presents cookie in later requests

  `cookie: 1678453`
- server matches presented-cookie with server-stored info
  - authentication
  - remembering user preferences, previous choices



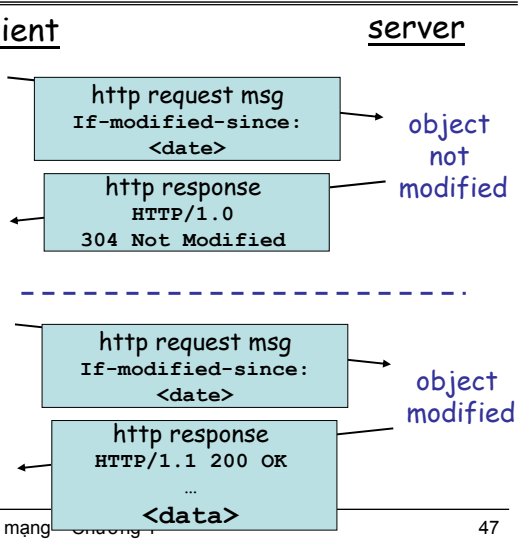
1.4 Một số ứng dụng mạng(tt)

Conditional GET

- Goal: don't send object if client has up-to-date stored (cached) version
- client: specify date of cached copy in http request

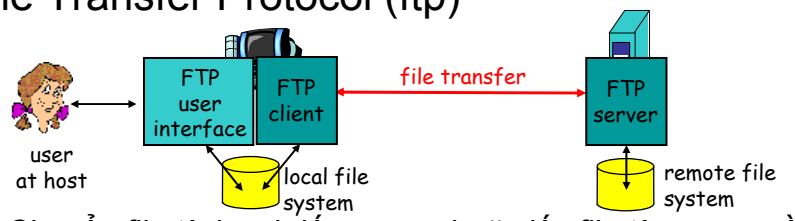
  `If-modified-since: <date>`
- server: response contains no object if cached copy up-to-date:

  `HTTP/1.0 304 Not Modified`



1.4 Một số ứng dụng mạng(tt)

- File Transfer Protocol (ftp)



  - Chuyển file từ local đến server hoặc lấy file từ server về local.
  - Hoạt động theo cơ chế client/server
  - FTP server chạy ở port 21.
  - Tham khảo : RFC 959

1.4 Một số ứng dụng mạng(tt)

- FTP (tt)
  - ftp client giao tiếp đến ftp server qua TCP ở port 21
  - Hai cầu nối TCP được thiết lập:
    - control: exchange commands, responses between client, server. “out of band control”
    - data: file data to/from server
  - ftp server hiện thực cơ chế “state”: current directory, earlier authentication

1.4 Một số ứng dụng mạng(tt)

Sample commands:

- sent as ASCII text over control channel
- **`USER`** ***`username`***
- **`PASS`** ***`password`***
- **`LIST`** return list of file in current directory
- **`RETR filename`** retrieves (gets) file
- **`STOR filename`** stores (puts) file onto remote host

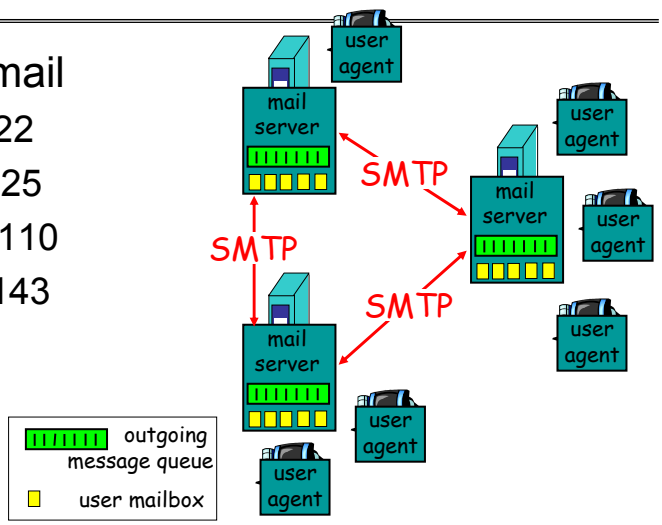
1.4 Một số ứng dụng mạng(tt)

Sample return codes

- status code and phrase (as in http)
- **331 Username OK, password required**
- **125 data connection already open; transfer starting**
- **425 Can't open data connection**
- **452 Error writing file**

1.4 Một số ứng dụng mạng(tt)

- Hệ thống E-mail
  - RFC 821, 822
  - SMTP: port 25
  - POP3: port 110
  - IMAP: port 143



1.4 Một số ứng dụng mạng(tt)

Hệ thống E-mail – Ví dụ về SMTP

S: 220 hamburger.edu
C: HELO crepes.fr
S: 250 Hello crepes.fr, pleased to meet you
C: MAIL FROM: <alice@crepes.fr>
S: 250 alice@crepes.fr... Sender ok
C: RCPT TO: <bob@hamburger.edu>
S: 250 bob@hamburger.edu ... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: Do you like ketchup?
C: How about pickles?
C: .
S: 250 Message accepted for delivery
C: QUIT
S: 221 hamburger.edu closing connection

Ví dụ về POP3

- client commands:
  - **`user:`** declare username
  - **`pass:`** password
- server responses
  - **`+OK`**
  - **`-ERR`**

transaction phase, client:

- **`list:`** list message numbers
- **`retr:`** retrieve message by number
- **`dele:`** delete
- **`quit`**

```
S: +OK POP3 server ready
C: user alice
S: +OK
C: pass hungry
S: +OK user successfully logged on

C: list
S: 1 498
S: 2 912
S: .
C: retr 1
S: <message 1 contents>
S: .
C: dele 1
C: retr 2
S: <message 1 contents>
S: .
C: dele 2
C: quit
S: +OK POP3 server signing off
```

CHƯƠNG 2
THIẾT KẾ GIẢI THUẬT CHO CHƯƠNG TRÌNH CLIENT/SERVER

2.1 Giao tiếp socket (Socket Interface)

2.2 Thiết kế giải thuật cho chương trình client

2.3 Thiết kế giải thuật cho chương trình server

2.1 Giao tiếp socket

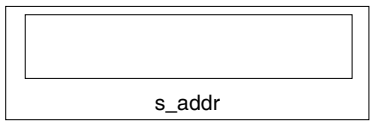
- Giao tiếp socket (Socket Interface) là các API dùng cho việc lập trình các ứng dụng mạng.
- Socket Interface được định nghĩa trong UNIX BSD, dựa trên việc mở rộng tập các system calls (access files).

  => Phần này chỉ giới thiệu các khái niệm, ý tưởng và các hàm, kiểu dữ liệu dùng cho lập trình mạng với Socket Interface.

2.1 Giao tiếp socket (tt)

- Một số cấu trúc dữ liệu
  - Cấu trúc địa chỉ Internet : định nghĩa dạng dữ liệu cấu trúc trong ngôn ngữ C. Cấu trúc này chỉ có 1 field kiểu u_long chứa địa chỉ IP 32 bit.

```
struct    in_addr
   {
              u_long    s_addr;
   };
```



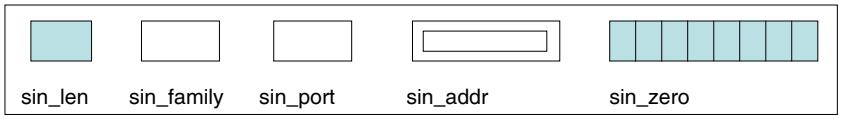
in_addr

Hình - cấu trúc địa chỉ Internet

2.1 Giao tiếp socket (tt)

- Một số cấu trúc dữ liệu (tt)
  - Cấu trúc địa chỉ socket :
    - địa chỉ này lưu trữ địa chỉ IP, chỉ số port, và dạng (family protocol)
    - Tên cấu trúc là *sockaddr_in* được biểu diễn ở hình trong slide kế. Trong đó:
      - *sin_len:* lưu trữ chiều dài cấu trúc của sockaddr_in
      - *sin_family:* dạng protocol của socket
      - *sin_port:* chỉ số port
      - *sin_addr:* địa chỉ in Internet của socket
      - *sin_zero[8]:* không dùng, đặt giá trị = 0

2.1 Giao tiếp socket (tt)



sockaddr_in

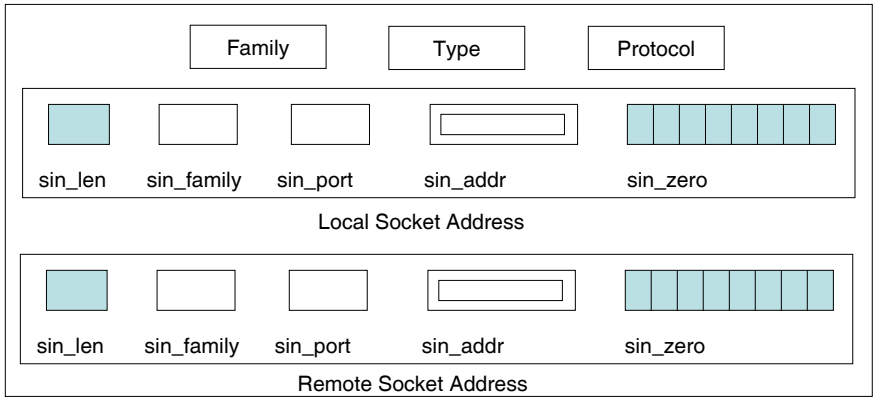
Hình - Cấu trúc địa chỉ socket

```
struct   sockaddr_in
{
        u_char          sin_len;
        u_short         sin_family;
        u_short         sin_port;
        struct  in_addr sin_addr;
        char            sin_zero[8];
};
```


2.1 Giao tiếp socket (tt)

- Một số cấu trúc dữ liệu (tt)
  - Cấu trúc socket :
    - socket được định nghĩa trong hệ điều hành bằng một cấu trúc, được xem như điểm nối để hai procceses giao tiếp với nhau.
    - Cấu trúc socket gồm 5 field được mô tả như hình trong slide kế:
      - *Family :* xác định protocol group
      - *Type :* xác loại socket, stream, datagram hay raw socket.
      - *Protocol :* là field thường gán giá trị bằng 0
      - *Local Socket Address* và *Remote Socket Address :* là địa chỉ socket của process cục bộ và từ xa.

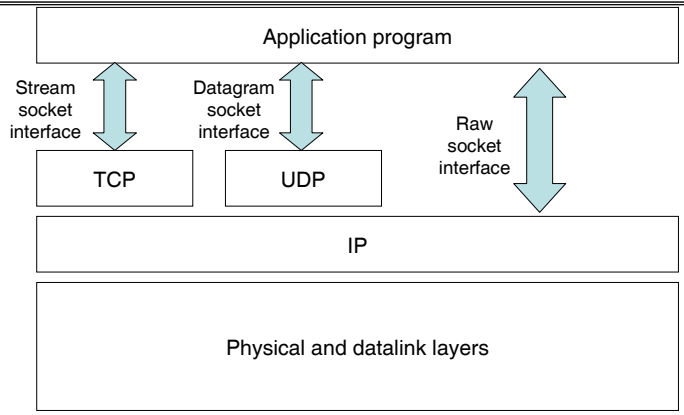
2.1 Giao tiếp socket (tt)



2.1 Giao tiếp socket (tt)

- Một số cấu trúc dữ liệu (tt)
  - Loại socket :
    - Giao tiếp socket định nghĩa 3 loại socket có thể dùng trên môi trường TCP/IP (hình ở slide kế).
    - Các loại socket gồm:
      - *Stream Socket:* dùng cho connection-oriented protocol như TCP.
      - *Datagram Socket:* dùng cho connectionless protocol như UDP.
      - *Raw Socket:* dùng cho một số protocol của một số ứng dụng đặc biệt, dùng các dịch vụ trực tiếp của lớp IP.

2.1 Giao tiếp socket (tt)



Hình - Các loại socket

- Một số cấu trúc dữ liệu (tt)
  - Thông tin remote host :
    - Thông tin được lưu trữ trong một cấu trúc *hostent* được trả về khi ứng dụng muốn ánh xạ địa chỉ tên miền bằng cách gọi hàm *gethostbyname():*

```
struct hostent * gethostbyname(const char * hostname);
struct hostent {
   char      *h_name;     char      **h_aliases;
   int       h_addrtype;  int       h_length;
   char      **h_addr_list;
}
```

2.1 Giao tiếp socket (tt)

- Một số cấu trúc dữ liệu (tt)
  - Byte Ordering
    - Big-Endian Byte Order : byte có trọng số lớn lưu trước.
    - Little -Endian Byte Order : byte có trọng số nhỏ lưu trước.
    - Tuỳ cấu trúc của mỗi máy, lưu trữ số theo một trong hai cách trên => khi giao tiếp mạng sẽ không đồng nhất.

2.1 Giao tiếp socket (tt)

- Một số cấu trúc dữ liệu (tt)
  - Byte Ordering (tt)
    - Network Byte Order : thứ tự lưu trữ dùng cho giao tiếp mạng.
    - Giao tiếp socket định nghĩa một số hàm để thực hiện các thao tác chuyển đổi :
      - *htons* và *htonl* : chuyển từ dạng lưu trữ của máy sang Network
      - *ntohs* và *ntohl* : chuyển từ dạng lưu trữ của Network sang dạng lưu trữ của máy.

2.1 Giao tiếp socket (tt)

- Các hàm dùng cho lập trình socket
  - Hàm *socket()* để tạo mới một socket

```
int socket (int family, int type, int protocol);
```

Hàm này tạo một socket, kết quả trả về là một số nguyên nhận dạng (socket descriptor), nếu có lỗi giá trị trả về là -1. Các thông số :

- *family:* họ socket
- *type:* kiểu socket (stream hay datagram)
- *protocol:* giao thức, thường đặt bằng 0

2.1 Giao tiếp socket (tt)

- Các hàm dùng cho lập trình socket (tt)
  - Hàm *bind()* để đăng ký với hệ thống

```
int bind (int sockfd, const struct sockaddr_in
  *localaddr, int localaddrlen);
```

Đăng ký socket đã khởi tạo với địa chỉ socket local. Trả về 0 nếu thành công, -1 nếu thất bại. Các thông số :

- *sockfd:* mô tả socket đã tạo bởi hàm ***socket()***
- *localaddr:*con trỏ chỉ đến địa chỉ socket của local
- *localaddrlen:* chiều dài của địa chỉ socket

2.1 Giao tiếp socket (tt)

- Các hàm dùng cho lập trình socket (tt)
  - Hàm *connect()* để kết nối đến server

```
int connect(int sockfd, const struct sockaddr_in
  *serveraddr, int serveraddrlen);
```

Dùng cho chương trình client thiết lập kết nối đến server. Trả về 0 nếu thành công, -1 nếu thất bại. Các thông số :

- *sockfd:* mô tả socket đã tạo bởi hàm ***socket()***
- *serveraddr:*con trỏ địa chỉ socket của server
- *serveraddrlen:* chiều dài của địa chỉ socket server

2.1 Giao tiếp socket (tt)

- Các hàm dùng cho lập trình socket (tt)
  - Hàm *listen()* để kết nối đến server

```
int listen(int sockfd, int backlog);
```

Hàm này dùng cho chương trình server connection-oriented để đặt socket ở trạng thái chờ, lắng nghe kết nối từ phía client. Trả về 0 nếu thành công, -1 nếu thất bại. Các thông số:

- *sockfd:* mô tả socket đã tạo bởi hàm ***socket()***
- *backlog:* số request có thể queued.

2.1 Giao tiếp socket (tt)

- Các hàm dùng cho lập trình socket (tt)
  - Hàm *accept()* : chấp nhận kết nối từ client đến.

```
int accept(int sockfd, const struct sockaddr_in
  *clientaddr, int *clientaddrlen);
```

Chấp nhận kết nối từ client, tạo socket mới. Giá trị là một socket descriptor của socket mới. Các thông số :

- *sockfd:* mô tả socket đã tạo bởi hàm ***socket()***
- *clientaddr:*con trỏ địa chỉ socket của client kết nối đến.
- *clientaddrlen:* chiều dài của clientaddr

2.1 Giao tiếp socket (tt)

- Các hàm dùng cho lập trình socket (tt)
  - Hàm *read()* để đọc dữ liệu từ socket

```
int read(int sockfd, const void *buf, int len);
```

Đọc dữ liệu từ connection vào bộ nhớ. Trả về số bytes đọc được nếu thành công, trả về 0 nếu không có dữ liệu, trả về -1 nếu thất bại. Các thông số :

- *sockfd:* mô tả socket đã tạo bởi hàm ***socket()***
- *buf:* con trỏ đến bộ đệm để lưu thông tin đọc được
- *len:* chiều dài của bộ đệm

2.1 Giao tiếp socket (tt)

- Các hàm dùng cho lập trình socket (tt)
  - Hàm *write()* để ghi dữ liệu

```
int write(int sockfd, const void *buf, int len);
```

Ghi dữ liệu từ bộ nhớ lên connection. Trả về số bytes ghi được nếu thành công, trả về -1 nếu thất bại. Các thông số :

- *sockfd:* mô tả socket đã tạo bởi hàm ***socket()***
- *buf:* con trỏ đến bộ đệm để lưu thông tin đọc được
- *len:* chiều dài của bộ đệm

2.1 Giao tiếp socket (tt)

- Các hàm dùng cho lập trình socket (tt)
  - Hàm *sendto()* để gởi dữ liệu

```
int sendto(int sockfd, const void *buf, int len, int
  flags, const struct sockaddr_in *toaddr, int
  toaddrlen);
```

Gởi dữ liệu đến một địa chỉ socket từ xa. Trả về số bytes gởi được nếu thành công, trả về -1 nếu thất bại. Các thông số :

- *sockfd, buf, len:* giống các hàm đã giới thiệu
- *flags: thường đặt bằng 0*
- *toaddr, toaddrlen:* địa chỉ socket đến và chiều dài.

2.1 Giao tiếp socket (tt)

- Các hàm dùng cho lập trình socket (tt)
  - Hàm *recvfrom()* để nhận dữ liệu

```
int recvfrom(int sockfd, const void *buf, int len, int
  flags, const struct sockaddr_in *fromaddr, int
  fromaddrlen);
```

Nhận dữ liệu từ một địa chỉ socket từ xa. Trả về số bytes gởi được nếu thành công, trả về -1 nếu thất bại. Các thông số :

- *sockfd, buf, len:* giống các hàm đã giới thiệu
- *fromaddr, fromaddrlen:* địa chỉ socket gởi đến và chiều dài.

2.1 Giao tiếp socket (tt)

- Các hàm dùng cho lập trình socket (tt)
  - Một số hàm dùng cho việc chuyển đổi

```
u_short     htons(u_short     host_short);
u_short     ntohs(u_short     network_short);
u_long      htonl(u_long      host_long);
u_long      ntohl(u_long      network_long);
char        *inet_ntoa(struct  in_addr  inaddr)
int         inet_aton(const  char *strptr,
                      struct  in_addr *addptr)
```

2.1 Giao tiếp socket (tt)

- Các hàm dùng cho lập trình socket (tt)
  - Một số hàm dùng cho việc thao tác dữ liệu

```
void   *memset ( void *dest, int chr, int len);
void   *memcopy ( void *dest, void *src, int len)
int      memcmp ( const void *first,
                  const void *second, int len)
```


2.2 Thiết kế giải thuật cho chương trình client

- Giải thuật cho chương trình client dùng TCP
  - Xác định địa chỉ server
  - Tạo socket.
  - Kết nối đến server.
  - Gởi/nhận dữ liệu theo giao thức lớp ứng dụng đã thiết kế.
  - Đóng kết nối.

2.2 Thiết kế giải thuật cho chương trình client (tt)

- Giải thuật cho chương trình client dùng UCP
  - Xác định địa chỉ server
  - Tạo socket.
  - Đăng ký socket với hệ thống.
  - Gởi/nhận dữ liệu theo giao thức lớp ứng dụng đã thiết kế đến server theo địa chỉ đã xác định.
  - Đóng kết nối.

2.3 Thiết kế giải thuật cho chương trình server

- Chương trình server có hai loại đơn giản : lặp (iterative) và đồng thời (concurrent).
- Hai dạng giao thức chương trình server có thể sử dụng là connection-oriented hoặc connectionless.
- Các slide kế tiếp trình bày cách thiết kế giải thuật cho các loại server kết hợp các đặc điểm trên

2.3 Thiết kế giải thuật cho chương trình server (tt)

- Giải thuật cho chương trình server iterative, connection-oriented:
  - Tạo socket, đăng ký địa chỉ socket với hệ thống.
  - Đặt socket ở trạng thái lắng nghe, chờ và sẵn sàng cho việc kết nối từ client.
  - Chấp nhận kết nối từ client, gởi/nhận dữ liệu theo giao thức lớp ứng dụng đã thiết kế.
  - Đóng kết nối sau khi hoàn thành, trở lại trạng thái lắng nghe và chờ kết nối mới.

2.3 Thiết kế giải thuật cho chương trình server (tt)

- Giải thuật cho chương trình server iterative, connectionless:
  - Tạo socket và đăng ký với hệ thống.
  - Lặp công việc đọc dữ liệu từ client gởi đến, xử lý và gởi trả kết quả cho client theo đúng giao thức lớp ứng dụng đã thiết kế.

2.3 Thiết kế giải thuật cho chương trình server (tt)

- Các yêu cầu cho concurrent Server:
  - Tại một thời điểm có thể xử lý nhiều yêu cầu từ client.
  - Chương trình concurrent server có thể chạy trên máy chỉ có 1 CPU.
  - Hệ thống phải hỗ trợ multi-tasking

2.3 Thiết kế giải thuật cho chương trình server (tt)

- Giải thuật cho chương trình concurrent, connectionless server:
  - Tạo socket, đăng ký với hệ thống.
  - Lặp việc nhận dữ liệu từ client, đối với một dữ liệu nhận, tạo mới một process để xử lý. Tiếp tục nhận dữ liệu mới từ client.
  - Công việc của process mới :
    - Nhận thông tin của process cha chuyển đến, lấy thông tin socket
    - Xử lý và gởi thông tin về cho client theo giao thức lớp ứng dụng đã thiết kế.
    - Kết thúc.

2.3 Thiết kế giải thuật cho chương trình server (tt)

- Giải thuật cho chương trình concurrent, connection-oriented server:
  - Tạo socket, đăng ký với hệ thống.
  - Đặt socket ở chế độ chờ, lắng nghe kết nối.
  - Khi có request từ client, chấp nhận kết nối, tạo một process con để xử lý. Quay lại trạng thái chờ, lắng nghe kết nối mới.
  - Công việc của process mới gồm:
    - Nhận thông tin kết nối của client.
    - Giao tiếp với client theo giao thức lớp ứng dụng đã thiết kế.
    - Đóng kết nối và kết thúc process con.

2.3 Thiết kế giải thuật cho chương trình server (tt)

- Multi-protocol Server (TCP,UDP)
  - Dùng một chương trình , mở một master socket cho cả TCP và UDP.
  - Dùng hàm hệ thống (*select*)để chọn lựa TCP socket hay UDP socket sẵn sàng.
  - Tùy vào protocol (TCP, UDP) để xử lý gởi nhận thông điệp theo đúng giao thức của lớp ứng dụng.
  - Tham khảo thêm RFC 1060

2.3 Thiết kế giải thuật cho chương trình server (tt)

- Multi-service Server
  - Tạo một điểm giao tiếp chung.
  - Với mỗi request, xem loại dịch vụ cần xử lý.
  - Với mỗi loại dịch vụ, xử lý riêng biệt
  - Có thể kết hợp Multi-service và Multi-protocol để thiết kế cho chương trình server.

CHƯƠNG 3
LẬP TRÌNH MẠNG TRÊN CÁC MÔI TRƯỜNG PHỔ DỤNG

3.1 Lập trình mạng trong UNIX

3.2 Các hàm hỗ trợ lập trình mạng trong UNIX

3.3 Lập trình mạng trong Windows với TCP/IP

3.4 Các hàm hỗ trợ lập trình mạng trong Windows

2.1 Lập trình mạng trong UNIX

- Lập trình mạng trong môi trường UNIX dùng socket có các hàm giống BSD Socket Interface đã giới thiệu.

3.2 Các hàm hỗ trợ lập trình mạng trong UNIX

- Địa chỉ socket trên Internet và địa chỉ IP:

```
#include <netinet/in.h>
struct sockaddr_in {
  short   sin_family;
  u_short sin_port;
  struct  in_addr sin_addr;
  char    sin_zero[8];
};
struct in_addr{
  u_long  s_addr;
}
```

3.2 Các hàm … (tt)

- Địa chỉ socket tổng quát:

```
#include <sys/socket.h>
struct sockaddr {
  short   sa_family;
  char    sa_data[14];
};
```

- Họ địa chỉ socket được định nghĩa trong <sys/socket.h>:

```
#define AF_UNIX 1/* local to host (pipes, portals) */
#define AF_INET 2 /* internetwork: UDP, TCP, etc. */
```


3.2 Các hàm … (tt)

- Cấu trúc địa chỉ máy từ xa.

```
struct hostent {
  char    *h_name;
  char    **h_aliases;
  int     h_addrtype;
  int     h_length;
  char    **h_addr_list;
  #define h_addr h_addr_list[0];
}
```

3.2 Các hàm … (tt)

- Tạo socket:

  #include <sys/types.h>

  #include <sys/socket.h>

  int **socket**(int *family*, int *type*, int *protocol*);

  Ví dụ tạo socket:

```
int sockfd;
//Tạo stream socket
sockfd = socket(AF_INET,SOCK_STREAM,0);
//Tạo datagram socket
sockfd = socket(AF_INET,SOCK_DGRAM,0);
```


3.2 Các hàm … (tt)

- Liên kết socket với địa chỉ socket(đăng ký)

  int **bind**(int *sockfd*, struct sockaddr **myaddr*,
  int *myaddrlen*);

  Ví dụ bind socket vừa tạo với địa chỉ socket:

```
struct sockaddr_in myaddr;
bzero((char*)&myaddr,sizeof(myaddr));
myaddr.sin_family = AF_INET;
myaddr.sin_port = htons(portno);
myaddr.sin_addr.s_addr = htonl(INADDR_ANY);
if (bind(sockfd, (struct sockaddr *) &myaddr,
  sizeof(myaddr)) < 0) error("ERROR on binding");
```

3.2 Các hàm … (tt)

- Chuyển socket về trạng thái chờ kết nối.

  int **listen**(int *sockfd*, int *backlog*);

- Chấp nhận yêu cầu kết nối từ client.

  int **accept**(int *sockfd*, struct sockaddr_in **peer*,
  int **addrlen*);

```
struct sockaddr_in cli_addr;int newsockfd,clilen;
listen(sockfd,5);
clilen = sizeof(cli_addr);
newsockfd = accept(sockfd,
         (struct sockaddr*)&cli_addr, &clilen);
if (newsockfd < 0) error("ERROR on accept");
```


3.2 Các hàm … (tt)

- Hàm kết nối đến server

  int **connect**(int *sockfd*, struct sockaddr *s*ervaddr*,
  int **addrlen*);

  Ví dụ:

```
struct sockaddr_in servaddr;
bzero((char*)&servaddr,sizeof(myaddr));
servaddr.sin_family = AF_INET;
servaddr.sin_port = htons(portno);
servaddr.sin_addr.s_addr = inet_addr(serverIP);
if (connect(sockfd, (struct sockaddr *) &servaddr,
  sizeof(servaddr)) < 0)
  error("Can not connect to server");
```

3.2 Các hàm … (tt)

- Các hàm truyền nhận dữ liệu:

int **read**(int *fd*,char **buf*, int *nbytes*);

int **write**(int *fd,char *buf,*int *nbytes*);

int **send**(int *sockfd*, char **buf,*int *nbytes,*int *flags*);

int **recv**(int *sockfd*, char **buf,*int *nbytes,*int *flags*);

int **sendto**(int *sockfd*, char **buf*, int *len*, int *flags*,
struct sockaddr_in **toaddr*, int *toaddrlen*);

int **recvfrom**(int *sockfd*, char **buf*, int *len*, int *flags*,
struct sockaddr_in **fromaddr*, int *fromaddrlen*);

3.2 Các hàm … (tt)

- Tạo process con để xử lý từng kết nối:

int fork(void);

```
int pid;
while(1){
    newsockfd = accept(sockfd, (struct sockaddr*)
                &cli_addr, &clilen);
    if ((pid=fork())==0){
          close(sockfd);
          process(newsockfd);
          close(newsockfd)
          exit(0);
    }
    close(newsockfd);
}
```

3.3 Lập trình mạng trong Windows với TCP/IP

- Dùng thư viện WinSock API (Windows Sockets Application Programming Interface) để hiện thực.
- Cần có thư viện WINSOCK.DLL hoặc WINSOCK32.DLL (32-bit Windows).
- Cần include các hàm và cấu trúc từ WINSOCK.H hoặc WINSOCK2.H
- Có thể biên dịch dạng dòng lệnh :

cl –o dest-file src-file ws2_32.lib

3.3 Lập trình mạng trong Windows với TCP/IP

- WinSock hiện thực Berkeley Sockets Interface trên môi trường Windows.
- WinSock có nhiều mở rộng thêm so với Berkeley Sockets.
  - Hỗ trợ kiến trúc Windows Message-Driven hay event-driven.
  - Hỗ trợ kiến trúc nonpreemptive của Windows

3.4 Các hàm hỗ trợ lập trình mạng trong Windows

- Khởi tạo WinSock:

  int WSAStartup(WORD wVersionRequired,
  LPWSADATA lpWSAData);

  Ví dụ :

```
WORD wVerRequested = MAKEWORD(0,1);
WSADATA wsaData;
if(WSAStartup(wVerRequested,&wsaData)!=0){
// process error
}
```


3.4 Các hàm WinSock (tt)

- Kết thúc WinSock

  int WSAClearup();
- Hàm lấy thông tin lỗi :

  int WSAGetLastError(void);

3.4 Các hàm WinSock (tt)

- Cám hàm dùng cho chuyển đổi:
  - Chuyển địa chỉ IP dạng chuỗi sang nhị phân:

```
unsigned long inet_addr(const char
                        FAR *cp);
```

  - Chuyển địa chỉ IP dạng nhị phân sang dạng chuỗi:

```
char FAR *inet_ntoa(struct in_addr in);
```

  - Lấy địa chỉ máy cục bộ:

```
int gethostname(char FAR*name,int len);
```

  - Lấy địa chỉ máy từ xa:

```
struct hostent FAR *gethostbyname(const
 char FAR *name);
```


3.4 Các hàm WinSock (tt)

Ví dụ về lấy địa chỉ

```
PHOSTENT phe =
   gethostbyname(condlg.m_remotehost);
char szTemp[128];
if (phe == NULL) {
   wsprintf(szTemp,"Not exist '%s'",
       condlg.m_remotehost);
   MessageBox(szTemp); return;
}
memcpy((char FAR *)&(ser_addr.sin_addr), phe-
>h_addr,phe->h_length);
```

3.4 Các hàm WinSock (tt)

- Hàm tạo socket

SOCKET socket (int *af*, **int** *type*, **int** *protocol* **);**

af :họ socket (thường dùng AF_INET : Internet)

type : loại socket (SOCK_STREAM, SOCK_DGRAM)

protocol : giao thức, thường đặt = 0 để lấy giá trị default

trả về giá trị INVALID_SOCKET nếu có lỗi

Ví dụ về hàm tạo socket :

```
ser_sock=socket(AF_INET,SOCK_STREAM,0);
if(ser_sock==INVALID_SOCKET) {
 MessageBox("Can not create socket");
 return TRUE;
}
```


3.4 Các hàm WinSock (tt)

- Hàm đăng ký địa chỉ socket với hệ thống

int bind (SOCKET *s*, **const struct sockaddr FAR** **addr*,
int *addrlen* **);**

s : mô tả socket đã được khởi tạo.

addr : địa chỉ socket.

addrlen : chiều dài *addr*

Nếu có lỗi trả về giá trị SOCK_ERROR

3.4 Các hàm WinSock (tt)

Ví dụ về lệnh bind :

```
//…
char message[100];
SOCKADDR_IN addr;
addr.sin_family=AF_INET;
addr.sin_port=htons(2000);
addr.sin_addr.s_addr=htonl(INADDR_ANY);
if(bind(s,(LPSOCKADDR)&addr,sizeof(addr))==
SOCKET_ERROR){
    wsprintf(message, "Can not bind socket : %d",
    WSAGetLastError());
    MessageBox(message);
    return TRUE;
}
```


3.4 Các hàm WinSock (tt)

- Hàm chuyển socket về trạng thái chờ

int listen (SOCKET *s*, **int** *backlog* **);**

backlog : chiều dài hàng đợi

trả về giá trị SOCKET_ERROR nếu có lỗi

Ví dụ về hàm listen:

```
if(listen(s,5)==SOCKET_ERROR) {
 wsprintf(message, "Can not listen : %d",
WSAGetLastError());
 MessageBox(message);
 return TRUE;
}
```

3.4 Các hàm WinSock (tt)

- Hàm chấp nhận kết nối từ client.

  SOCKET accept (SOCKET *s*, **struct sockaddr FAR ***addr**,**
  int FAR ***addrlen* **);

  s : là mô tả socket của server.

  addr : con trỏ địa chỉ socket của client kết nối đến.

  addrlen : chiều dài của *addr*

3.4 Các hàm WinSock (tt)

Ví dụ về hàm accept :

```
SOCKADDR_IN client_addr;SOCKET cli_s;
IN_ADDR clientIP;
int len=sizeof(client_addr);
cli_S=accept(s,(LPSOCKADDR)&client_addr,&len);
   if(sock==INVALID_SOCKET) {
     MessageBox("Can not eccept");
     return TRUE;    }
   else {
     memcopy(&clientIP,&client_addr.sin_addr.s_addr,4);
     wsprintf(message,"Client IP= %s and port= %d",
       inet_ntoa(clientIP),ntohs(cli_s.sin_port));
       …
   }
```

3.4 Các hàm WinSock (tt)

- Hàm thiết lập kết nối đến server.

int connect (SOCKET *s*, **const struct sockaddr FAR** ***name*, **int** *namelen* **);**

s : socket của chương trình local

name : địa chỉ socket của server.

namelen : chiều dài của name

Trả về giá trị SOCKET_ERROR nếu có lỗi

3.4 Các hàm WinSock (tt)

Ví dụ về hàm connect ➔

```
…
SOCKADDR_IN ser_addr;
ser_addr.sin_family=AF_INET;
ser_addr.sin_port=htons(2000);
ser_addr.sin_addr.s_addr=
          inet_addr("172.28.10.20");
if(connect(s,(LPSOCKADDR)&ser_addr,
   sizeof(ser_addr))==SOCKET_ERROR){
  MessageBox("Can not connect to server");
}
```

3.4 Các hàm WinSock (tt)

Lệnh gởi dữ liệu
int send (SOCKET *s*, **const char FAR *** *buf*, **int** *len*,
int *flags* **);**
buf : chuỗi dữ liệu cần gởi
len : chiều dài của *buf*
flags : thường đặt giá trị 0
Trả về số byte dữ liệu gởi được, nếu lỗi trả về SOCKET_ERROR

```
//...
char buf[255];
lstrcpy(mesg,"Hello World");
if (send(s,buf,strlen(buf),0) ==SOCKET_ERROR) {
  MessageBox("Can not send data");
  return;
}
```


3.4 Các hàm WinSock (tt)

int recv (SOCKET *s*, **char FAR*** *buf*, **int** *len*, **int** *flags* **);**
Các thông số tương tự hàm *send*

```
//…
#define BUFSIZE (100)
char buf[BUFSIZE];
int nByteRecv;
nByteRecv = recv(s,buf, BUFSIZE,0);
if (nByteRecv == SOCKET_ERROR) {
  MessageBox("Error receive data");
  return;
} //…
```

3.4 Các hàm WinSock (tt)

- Các hàm dùng cho UDP

int sendto (SOCKET *s*, **const char FAR** **buf*, **int** *len*,
int *flags,* **const struct sockaddr FAR** **to*, **int** *tolen*);

to : địa chỉ socket của process muốn gởi đến

int recvfrom (SOCKET *s*, **char FAR*** *buf*, **int** *len*,
int *flags,***const struct sockaddr FAR**
***from*, int FAR *fromlen ***);**

from : địa chỉ socket của process gởi dữ liệu đến

3.4 Các hàm WinSock (tt)

Ví dụ về hàm sendto:

```
#define BUFSIZE (100)
char buf[BUFSIZE];
int nByteSend;
SOCKADDR_IN to;
to.sin_family = AF_INET;
to.sin_port = 2000;
to.sin_addr.s_addr = inet_addr("127.0.0.1");
lstrcpy(buf,"Hello World");
nByteSend = sendto(s,buf,lstrlen(buf),0,
                   (LPSOCKADDR)&to,sizeof(to));
if(nByteSend == SOCKET_ERROR )
//…
```

3.4 Các hàm WinSock (tt)

Ví dụ về hàm recvfrom

```
#define BUFSIZE (100)
char buf[BUFSIZE];
int nByteRecv;
SOCKADDR_IN from;
int fromlen;
nByteRecv = recvfrom(s,buf,BUFSIZE,0,
                (LPSOCKADDR)&from,&fromlen);
if(nByteRecv == SOCKET_ERROR )
//…
```


3.4 Các hàm WinSock (tt)

Hàm khai báo nhận event từ network cho socket.

int WSAAsyncSelect (SOCKET *s*, **HWND** *hWnd*, **unsigned int** *wMsg*, **long** *lEvent*);

hWnd : cửa sổ nhận sự kiện.

wMsg: thông điệp gởi đến.

lEvent : sự kiện của socket cần xử lý.

• Khi dùng hàm này, socket sẽ được chuyển về trạng thái nonblocking.

• Đối với mỗi socket thì chỉ khai báo một thông điệp đến. Có thể khai báo nhiều sự kiện bằng phép OR (|)

3.4 Các hàm WinSock (tt)

Ví dụ về hàm **WSAAsyncSelect**

```
BOOL CServerDlg::OnInitDialog() {
  //s là socket đã được tạo,
  //đã sử dụng các hàm bind và listen
  if(WSAAsyncSelect(s,m_hWnd,WM_USER+1,
                FD_ACCEPT)==SOCKET_ERROR) {
     return TRUE;
  }
  return FALSE;
}
```


3.4 Các hàm WinSock (tt)

- Sau khi dùng hàm WSAAsyncSelect, ta phải khai báo hàm để xử lý biến cố tương ứng.

```
BEGIN_MESSAGE_MAP(CServerDlg, CDialog)
   //{{AFX_MSG_MAP(CServerDlg)
  ON_MESSAGE(WM_USER+1,OnAccept)
  //...
  //}}AFX_MSG_MAP
END_MESSAGE_MAP()
```

- Viết hàm xử lý biến cố tương ứng

```
LONG CServerDlg::OnAccept(WPARAM wParam,
                          LPARAM lParam){
}
```

3.4 Các hàm WinSock (tt)

- Có thể viết code cho hàm WindowProc để xử lý sự kiện network.

```
LRESULT CServerDlg::WindowProc(UINT message, WPARAM wParam,
  LPARAM lParam) {
switch (message) {
  case WM_USER+1 :
     OnAccept(); return 1;
  case WSA_RDCLOSE :
     if (WSAGETSELECTEVENT(lParam) == FD_READ)
          Read_Process(wParam);
```

Socket desciptor

3.4 Các hàm WinSock (tt)

- Hàm đóng socket :

  int closesocket (SOCKET s);

  Hàm trả về giá trị 0 nếu thành công, nếu thất bại trả về giá trị SOCKET_ERROR

3.4 Các hàm WinSock (tt)

- Lập trình trên mạng trên Windows bằng MFC : dùng các lớp CWinSock, CDatagramSocket, CStreamSocket.
- Tham khảo thêm MSDN

CHƯƠNG 4
LẬP TRÌNH MẠNG VỚI JAVA

4.1 Giới thiệu ngôn ngữ Java
4.2 Ví dụ về lập trình mạng bằng Java
4.3 Khái niệm Stream và Multithreading
4.4 Thư viện java.net.*

4.1 Giới thiệu ngôn ngữ Java

- Là ngôn ngữ lập trình hướng đối tượng trong sáng, ra đời vào khoảng năm 1995 do Sun MicroSystems xây dựng.
- Là ngôn ngữ thông dịch, chạy trên kiến trúc máy ảo (Java Virtual Machine).
- Hỗ trợ mạng mẽ lập trình mạng, bảo mật, multi-thread…
- Sử dụng thư viện chuẩn JDK

4.1 Giới thiệu ngôn ngữ Java

- JVM hỗ trợ trên nhiều flatform => Java có tính portable “write one-run everywhere”.
- Hiện có rất nhiều công cụ hỗ trợ lập trình Java như : JBuilder (5), Visual Café, Microsoft Visual J++…
- JDK (Java Development Kit) phiên bản mới 1.4.1 trên http://java.sun.com/j2se/1.4.1/index.html

4.1 Giới thiệu ngôn ngữ Java

- Cài đặt : download chương trình và cài đặt lên máy tính theo hướng dẫn. VD:
  - Windows : C:\JDK
  - Unix : /usr/local/jdk
- Đặt biến môi trường PATH đến thư mục BIN trong thư mục cài đặt:
  - Windows : SET PATH=c:\jdk\bin;%PATH%
  - Unix :
    - PATH=$PATH:/usr/local/jdk/bin
    - export PATH

4.1 Giới thiệu ngôn ngữ Java

- Đặt biến môi trường CLASSPATH đến các package có sử dụng trong chương trình. VD:
  - set CLASSPATH=c:\lib\jdbc.zip;c\lib\xml4j.jar;.
- Lập trình : có thể dùng trình soạn thảo bất kỳ, lưu với tên file .java
- Biên dịch :
  - javac file-name.java
- Chạy :
  - java file-name

4.2 Ví dụ LTM với Java

Chương trình client/server Echo

- Chương trình Client

```
1. //file Client.java
2. import java.net.*;
3. import java.io.*;
4. public class Client{
5.      public static void main(String args[]) throws Exception{
6.              Socket clientsock;
7.              DataOutputStream output;
8.              BufferedReader input;//bộ đệm đọc dữ liệu
9.              clientsock = new Socket("127.0.0.1",2000);
10.             input = new BufferedReader(new
11.             InputStreamReader(clientsock.getInputStream()));
12.             output = new DataOutputStream(
13.                             clientsock.getOutputStream());
```


4.2 Ví dụ… (tt)

- Chương trình Client

```
14.             BufferedReader keyInput = new BufferedReader(new
15.                              InputStreamReader(System.in));
16.     System.out.print("Enter sentence to send to server:");
17.             String data = keyInput.readLine();
18.             output.writeBytes(data+"\n");
19.             int recvByte;
20.             System.out.print("Data receicved: ");
21.             System.out.println(input.readLine());
22.             clientsock.close();
23.     }//main
24. }//class
```

- Biên dịch: javac Client.java
- Thực thi : java Client

4.2 Ví dụ… (tt)

- Chương trình Server

```
1. //file Server.java
2. import java.net.*;
3. import java.io.*;
4. public class Server{
5.      public static void main(String args[]) throws Exception{
6.              ServerSocket serversock = new ServerSocket(2000);
7.              DataOutputStream output;//stream xuat du lieu
8.              BufferedReader input;//stream doc du lieu
9.              for(;;){
10.               Socket client = serversock.accept();
11.               output = new DataOutputStream(
12.                           client.getOutputStream());
```


4.2 Ví dụ… (tt)

- Chương trình Server (tt)

```
13.               input = new BufferedReader(new
14.             InputStreamReader(client.getInputStream()));
15.               String data = input.readLine();
16.               System.out.println("Recv from client: "+data);
17.               output.writeBytes(data+"\n");
18.               output.flush();
19.             }//for
20.     } //main
21. }//class
```

- Dịch : javac Server.java
- Chạy : java Server

4.3 Stream và Multithreading

- Khái niệm stream trong ngôn ngữ Java:
  - Stream : hỗ trợ chức năng truy xuất I/O trong ngôn ngữ Java.
  - Các công việc truy xuất I/O có thể kể đến như file, kết nối mạng, bàn phím(thiết bị nhập chuẩn), màn hình (tb xuất chuẩn)…
  - Stream là môi trường dẫn dữ liệu, không quan tâm đến định dạng của dữ liệu
  - Các lớp stream được cung cấp ở gói java.io.*;

4.3 Stream và Multithreading (tt)

- Khái niệm stream ...(tt):
  - Được chia ra làm hai loại chính input stream là stream chứa dữ liệu nhập; output stream là stream chứa dữ liệu xuất.
  - Hai lớp cơ bản trong Java xử lý nhập xuất là InputStream và OutputStream.
  - Các lớp dẫn xuất thường dùng của InputStream : BufferedInputStream, DataInputStream, ByteArrayInputStream, StringBufferInputStream.

4.3 Stream và Multithreading (tt)

- Khái niệm stream ...(tt):
  - Các lớp dẫn xuất thường dùng của OutputStream : BufferedOutputStream, DataOutputStream, ByteArrayOutputStream
  - Các lớp thường dùng cho truy xuất tập tin : File, RadomAcessFile…
  - Chi tiết lập trình xem thêm Java docs của JDK

4.3 Stream và Multithreading (tt)

- Thread và Multithread trong Java.
  - Thread : là một đối tượng có thể chạy nhiều phiên bản đồng thời.
  - Java hỗ trợ lập trình thread trong bản thân ngôn ngữ.
  - Có hai cách để tạo Thread :
    - Xây dựng class extends Thread.
    - Implements interface Runable

4.3 Stream và Multithreading (tt)

- Multithreading
  - Ví dụ :

```
public static void main(String args[]){
  //...
  while(true){
    Socket newsock = server.accept();
    ClientThread ct = new ClientThread(newsock);
    ct.start();
  }
}
```


4.3 Stream và Multithreading (tt)

```
class ClientThread extends Thread{
  Socket sock;
  public ClientThread(Socket sock){
    this.sock = sock;
  }
  public void run(){
    //xu ly
  }
}
```

4.4 Thư viện java.net.*

- Lớp InetAddress : dùng để thao tác về địa chỉ Internet, các phương thức thường dùng:
  - public byte[] **getAddress**(): Returns the raw IP address of this object
  - public static InetAddress[] **getAllByName**(String host) throws UnknownHostException
  - public String getHostAddress()
    - Returns the IP address string "%d.%d.%d.%d".
  - public static InetAddress getByName(String host) throws UnknownHostException

4.4 Thư viện java.net.*

- Lớp Socket : dùng cho chương trình client kết nối đến máy chủ
  - public **Socket**(String host, int port) throws UnknownHostException, IOException
    - Creates a stream socket and connects it to the specified port number on the named host.
  - public **Socket**(InetAddress address, int port) throws IOException
    - Creates a stream socket and connects it to the specified port number at the specified IP address.
  - public **Socket**(String host, int port, boolean stream) throws IOException
    - Creates a stream socket and connects it to the specified port number on the named host.

4.4 Thư viện java.net.*

- Lớp Socket (tt):
  - public InputStream **getInputStream**() throws IOException
  - public InetAddress **getInetAddress**()
    - Returns the address to which the socket is connected.
  - int **getPort**()
    - Returns the remote port to which this socket is connected.
  - public OutputStream **getOutputStream**() throws IOException
    - mReturns an output stream for this socket.

4.4 Thư viện java.net.*

- Lớp ServerSocket : dùng cho chương trình server tao socket và chấp nhận kết nối.
  - **ServerSocket**(int port)
    - Creates a server socket on a specified port.
  - **ServerSocket**(int port, int backlog)
    - Creates a server socket and binds it to the specified local port number, with the specified backlog.
  - **ServerSocket**(int port, int backlog, InetAddress bindAddr)
    - Create a server with the specified port, listen backlog, and local IP address to bind to.

4.4 Thư viện java.net.*

- Lớp ServerSocket (tt)
  - public Socket **accept**() throws IOException
    - Listens for a connection to be made to this socket and accepts it. The method blocks until a connection is made.
  - public void **close**() throws IOException
    - Closes this socket.
  - public void **setSoTimeout**(int timeout) throws SocketException
    - Enable/disable SO_TIMEOUT with the specified timeout, in milliseconds.
  - public String **toString**()
    - Returns the implementation address and implementation port of this socket as a String.

4.4 Thư viện java.net.*

- Lớp DatagramSocket : sử dụng cho chương trình dùng UDP
  - public **DatagramSocket**() throws SocketException
    - Constructs a datagram socket and binds it to any available port on the local host machine.
  - public **DatagramSocket**(int port) throws SocketException
    - Constructs a datagram socket and binds it to the specified port on the local host machine.
  - public **DatagramSocket**(int port, InetAddress laddr) throws SocketException
    - Creates a datagram socket, bound to the specified local address. The local port must be between 0 and 65535 inclusive.

4.4 Thư viện java.net.*

- Lớp DatagramSocket (tt) :
  - public void **connect**(InetAddress address, int port)
    - Connects the socket to a remote address for this socket.
  - public void **disconnect**()
    - Disconnects the socket. This does nothing if the socket is not connected.
  - public void **receive**(DatagramPacket p) throws IOException
    - Receives a datagram packet from this socket
  - public void **send**(DatagramPacket p) throws IOException
    - Sends a datagram packet from this socket.

4.4 Thư viện java.net.*

- Lớp DatagramPacket : dùng xây dựng các gói tin để trao đổi theo giao thức UDP.
  - public **DatagramPacket**(byte[] buf, int length)
    - Constructs a DatagramPacket for receiving packets of length length.
  - public **DatagramPacket**(byte[] buf, int length, InetAddress address, int port)
    - Constructs a datagram packet for sending packets of length length to the specified port number on the specified host.
  - public **DatagramPacket**(byte[] buf, int offset, int length)
    - Constructs a DatagramPacket for receiving packets of length length, specifying an offset into the buffer

4.4 Thư viện java.net.*

- Lớp DatagramPacket (tt)
  - public InetAddress **getAddress**()
    - Returns the IP address of the machine to which this datagram is being sent or from which the datagram was received.
  - public byte[] **getData**()
    - Returns the data received or the data to be sent.
  - public int **getLength**()
    - Returns the length of the data to be sent or the length of the data received.
  - public int **getPort**()
    - Returns the port number on the remote host.

4.4 Thư viện java.net.*

- Lớp DatagramPacket (tt)
  - public void **setAddress**(InetAddress iaddr)
    - Sets the IP address of the machine to which this datagram is being sent.
  - public void **setPort**(int iport)
    - Sets the port number on the remote host to which this datagram is being sent.
  - public void **setData**(byte[] buf)
    - Set the data buffer for this packet.
  - public void **setData**(byte[] buf, int offset, int length)
    - Set the data buffer for this packet.

4.4 Thư viện java.net.*

- Lớp URL : kết nối đến một tài nguyên Internet.
  - public **URL**(String spec) throws MalformedURLException
    - Creates a URL object from the String representation.
  - public **URL**(String protocol, String host, String file) throws MalformedURLException
    - Creates a URL from the specified protocol name, host name, and file name. The default port for the specified protocol is used.
  - public **URL**(String protocol, String host, int port, String file) throws MalformedURLException
    - Creates a URL object from the specified protocol, host, port number, and file. Specifying a port number of -1 indicates that the URL should use the default port for the protocol.

4.4 Thư viện java.net.*

- Lớp URL(tt)
  - public final Object **getContent**() throws IOException
    - Returns the contents of this URL.
  - public String **getFile**()
    - Returns the file name of this URL.
  - public URLConnection **openConnection**() throws IOException
    - Returns a URLConnection object that represents a connection to the remote object referred to by the URL.
  - public final InputStream **openStream**() throws IOException
    - Opens a connection to this URL and returns an InputStream for reading from that connection.

HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

LẬP TRÌNH MẠNG

(Dùng cho sinh viên hệ đào tạo đại học từ xa)

Lưu hành nội bộ

2010

LẬP TRÌNH MẠNG

Biên soạn : NINH XUÂN HẢI

CHƯƠNG I
NGÔN NGỮ JAVA

I. LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

1. Các phương pháp lập trình

a) *Lập trình tuyến tính*

Toàn bộ chương trình chỉ là một đơn thể duy nhất, các lệnh được thực hiện tuần tự theo thứ tự xuất hiện trong chương trình. Lập trình tuyến tính đơn giản nhưng khó sửa lỗi, khó mở rộng.

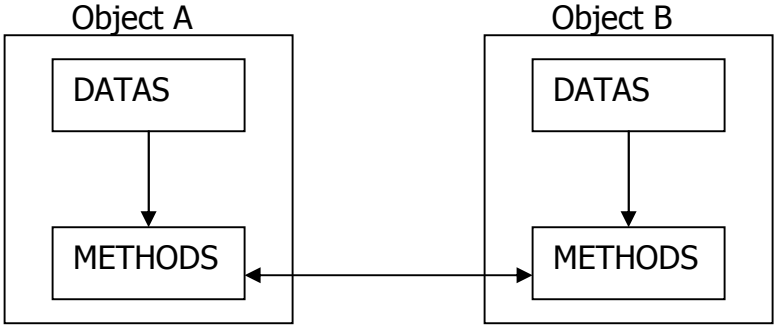
b) *Lập trình hướng thủ tục*

Chương trình được tách thành nhiều phần gọi là hàm hay thủ tục. Mỗi hàm sẽ thực hiện một chức năng của chương trình. Trong chương trình thường có một hàm chính (main), khi chương trình thực thi sẽ gọi hàm main, hàm main có thể gọi các hàm khác, các hàm khác lại có thể gọi lẫn nhau. Lập trình hương thủ tục dễ sửa lỗi, dễ mở rộng, nhưng vì dữ liệu và hàm tách biệt nên khó bảo vệ dữ liệu và hàm, để không bị truy xuất bởi các hàm không mong đợi. Khi sửa đổi dữ liệu các hàm truy xuất phải thay đổi theo, ngoài ra khó sử dụng lại các hàm đã viết sẵn.

c) *Lập trình hướng đối tượng*

Chương trình sẽ được tách thành nhiều lớp, mỗi lớp gồm có dữ liệu (biến) và phương thức (hàm) xử lý dữ liệu. Do dữ liệu và hàm được đóng gói thành lớp nên LTHĐT sẽ có 3 đặc tính sau:

- Tính đóng gói (Encapsulation): Việc tổ chức dữ liệu và phương thức trong một lớp gọi là tính đóng gói, tính đóng gói cho phép bảo vệ dữ liệu, che dấu chi tiết cài đặt.
- Tính thừa kế (Inheritance): Sử dụng lớp có trước (lớp cha) để xây dựng lớp mới (lớp con) gọi là tính thừa kế. Lớp con được thừa hưởng những thuộc tính, phương thức của lớp cha và có thể có thêm những thuộc tính, phương thức riêng. Tính thừa kế giúp người lập trình dễ dàng sử dụng lại mã chương trình đã viết trước đó.
- Tính đa hình (Polymorphism): Một phương thức có thể thực hiện theo nhiều cách khác nhau trên các lớp khác nhau gọi là tính đa hình. Tính đa hình giúp cho việc lập trình trở nên đơn giản, dễ mở rộng.



MÔ HÌNH CỦA LTHĐT

2. Các khái niệm về LTHĐT

2.1 Đối tượng (object)

Đối tượng dùng để biểu diễn một thực thể của thế giới thực. Mỗi đối tượng được xác định bởi thuộc tính (dữ liệu, biến) và hành vi (phương thức, hàm). Thuộc tính để xác định tính chất riêng của đối tượng, hành vi là hành động tác động lên đối tượng.

Object = Variables + Methods

Ví dụ : Một đối tượng sinh viên có thể có thuộc tính là: họ tên, đlt, đtb và hành vi tác động lên đối tượng sinh viên là tính đtb của sv đó

* Đối tượng sinh viên thứ 1	* Đối tượng sinh viên thứ 2
- Thuộc tính: họ tên: Trần văn An đlt= 1 đth= 2 - Hành vi: tính đtb của sv: dtb=(dlt+dth)/2=1.5	- Thuộc tính: họ tên: Đoàn Dự đlt= 2 đth= 3 - Hành vi: tính đtb của sv: dtb=(dlt+dth)/2=2.5

Ví dụ: Một đối tượng hcn có thể có thuộc tính là chiều dài, chiều rộng và hành vi tác động lên đối tượng hcn là tính diện tích của hcn đó

Đối tượng hcn thứ 1	Đối tượng hcn thứ 2
- Thuộc tính: chiều dài=3 chiều rộng=4 - Hành vi: Tính dt: dt=cd*cr=12	- Thuộc tính: chiều dài=5 chiều rộng=6 - Hành vi: Tính dt: dt=cd*cr=30

2.2 Lớp (class)

Là cấu trúc mô tả các đối tượng có cùng thuộc tính và hành vi. Mỗi lớp sẽ khai báo các thuộc tính, hành vi của các đối tượng thuộc lớp. Các đối tượng thuộc lớp sẽ có cùng tên thuộc tính nhưng có giá trị thuộc tính khác nhau. Thuộc tính còn gọi là dữ liệu hay là biến, hành vi còn gọi là hàm hay phương thức.

II. Ngôn ngữ lập trình Java

1. Giới thiệu:

NNLT Java do hãng Sun Microsystem thiết kế năm 1991 tên là Oak, mục đích lập trình cho các thiết bị điện tử, 1995 được mở rộng để viết ứng dụng trên Internet và lấy tên là Java.

Ưu điểm của ngôn ngữ Java là ngôn ngữ lập trình hướng đối tượng, không phụ thuộc phần cứng và hệ điều hành, hỗ trợ lập trình mạng rất mạnh và đơn giản và dễ học hơn C++ nhiều.

Java có thể soạn bằng bất cứ trình soạn thảo văn bản nào (nodepad, wordpad, Jbuilder,…), sau đó được dịch sang file .class dạng bytecode. Mã bytecode không phụ thuộc phần cứng và hệ điều hành nhưng muốn thực thi trên một máy có hệ điều hành cụ thể thì máy đó cần cài đặt

JVM (Java Virtual Machine) tương ứng, JVM là môi trường để thực thi file dạng bytecode trên một máy cụ thể.

Để biên dịch, thực thi chương trình Java ta có thể cài đặt chương trình JDK (Java Development Toolkit) gồm có trình biên dịch (javac.exe) để dịch file .java thành file .class, trình thông dịch (java.exe) để thực thi file .class, thư viện chứa các hàm chuẩn (APIs) và một số tiện ích khác nhưng do JDK không có trình soạn thảo Java và không có giao tiếp đồ họa với người dùng nên khi viết Java thường ta sử dụng phần mềm thân thiện hơn như là Jbuilder, Visual J++,...

Sơ đồ hoạt động của một ct ứng dụng Java:

2. Sử dụng ngôn ngữ

* Các kiểu cơ bản

Từ khóa	kích thước
Số nguyên có dấu	
byte	1 byte
short	2 bytes
int	4 bytes
Số thực	
float	8 bytes
double	8 byte
Các kiểu khác	
char	kiểu kí tự 2 byte
boolean	kiểu luận lý (true, false)

* Hằng số:

123 (int), 123L (long), 123.45F (float), 123.45 (double), 'c' (char)

* Kiểm soát việc truy xuất biến, phương thức của lớp:

Specifier	class	subclass	package	world
private	X			
protected	X	X	X	
public	X	X	X	X

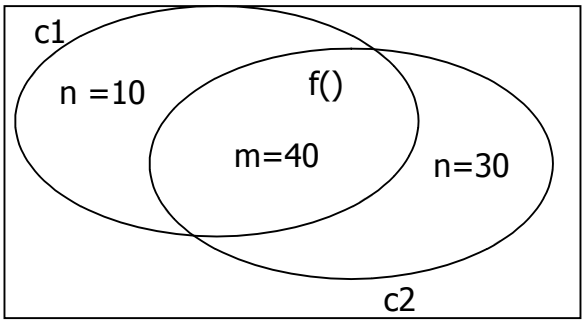
public: có thể truy xuất ở gói khác
protected: chỉ truy xuất ở cùng gói (mặc định là protected)
private: chỉ truy xuất trong lớp

*** Biến tĩnh, phương thức tĩnh**

- Biến tĩnh: khai báo static, dùng chung cho mọi đt thuộc lớp. Không có static gọi là biến thể hiện, mỗi đối tượng có biến thể hiện riêng.
- Phương thức tĩnh: khai báo static, được gọi bằng tên lớp hoặc tên đối tượng, trong pt tĩnh chỉ được truy xuất biến tĩnh.

Ví dụ:

```
public class TestStatic
{
  int n; //bien the hien
  static int m;//bien tĩnh
  void f()
  {
    int x=5;    //bien cuc bo
  }
  public static void main(String[] args)
  {
    TestStatic c1=new TestStatic(); c1.n=10; c1.m=20;
    TestStatic c2=new TestStatic(); c2.n=30; c2.m=40;
    System.out.println("kq: "+c1.n+","+c1.m+","+c2.n+","+c2.m+","+TestStatic.m);
  }
}
```



Lop TestStatic

Ket qua:
kq: 10,40,30,40,40

*** Lớp trừu tượng, pt trừu tượng**

Lớp trừu tượng và phương thức trừu tượng được khai báo abstract. Pt trừu tượng chỉ có phần khai báo chưa có cài đặt. Lớp trừu tượng có những pt trừu tượng và các pt khác. Không thể tạo đt từ pt trừu tượng, lớp thừa kế lớp trừu tượng cần phải cài đặt các pt trừu tượng nếu không sẽ trở thành lớp trừu tượng. Pt trừu tượng sẽ có tính đa hình.

Ví dụ:

```
public abstract class Nguoi //lop trừu tượng
{
        String hoten;
        public void NhapHoTen() throws IOException
        {
           BufferedReader stdIn = new BufferedReader(new InputStreamReader(System.in));
           System.out.print("Ho Ten:");    hoten=stdIn.readLine();
         }
         public void XuatHoTen()
         {
```

```
        System.out.println("Ten: "+ten);
    }
    public abstract boolean Thuong(); //pt xet dieu kien duoc thuong la phương thức trừu tượng.
}
```

*** Giao diện:**

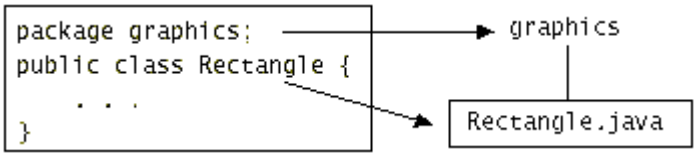
Dùng từ khoá interface. Trong giao diện chỉ có các biến hằng số (final) và các pt trong giao diện chỉ có phần khai báo, chưa có cài đặt. Một lớp có thể sử dụng nhiều giao diện, giao diện dùng để giải quyết vấn đề đa thừa kế. Một lớp có thể sử dụng (thừa kế) nhiều giao diện và phải cài đặt tất cả các pt trong các giao diện nếu không lớp phải khai báo trừu tượng.

Ví dụ:

```
public interface SoSanh //giao diện
{
  public static final int LESS = -1;
  public static final int EQUAL = 0;
  public static final int GREATER = 1;
  public int sosanh(SoSanh obj);
}
```

*** Gói:** là tập hợp các lớp và các giao diện có liên quan, mục đích của gói là bảo vệ sự truy cập và giải quyết vấn đề trùng tên. Khai báo lớp thuộc gói dùng lệnh package, sử dụng lớp của gói dùng lệnh import. Java cung cấp sẵn rất nhiều gói, khi sử dụng lớp thuộc gói nào thì cần import gói tương ứng, các lớp thuộc gói java.lang mặc định đã import.

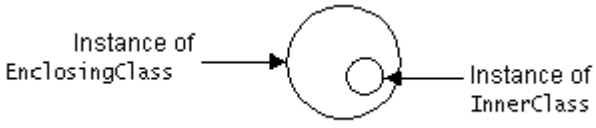
Ví dụ:



*** Lớp lồng nhau:**

Đối tượng thuộc lớp trong có thể truy xuất biến và phương thức không tĩnh của lớp ngoài.

```
class EnclosingClass{
   . . .
   class ANestedClass {
      . . .
   }
}
```



BÀI TẬP CHƯƠNG I:
GIỚI THIỆU NGÔN NGỮ JAVA

*** Bài 1**: Viết chương trình giải pt bậc nhất, tham số đưa vào từ dòng lệnh, nếu thiếu tham số phải thông báo cách sử dụng, nếu tham số a, b không phải là số phải báo lỗi. Dùng trình biên dịch, thông dịch của JDK để biên dịch và thực thi chương trình.

*** Bài 2** : Xây dựng lớp hình chữ nhật, có thuộc tính là chiều dài, chiều rộng và có phương thức tính diện tích, chu vi hcn, phương thức khởi tạo; viết hàm main để thử lớp này. Soạn và thực thi chương trình bằng Jbuilder.

*** Bài 3:** Xây dựng lớp sinh viên, có thuộc tính là họ tên, điểm lý thuyết, điểm thực hành, điểm trung bình= (đlt+dth)/2, và có phương thức nhập thông tin sv, tính đtb, xuất thông tin sv. Viết hàm main để thử lớp này.

*** Bài 4:** Xây dưng lớp hình vuông thừa kế lớp hình chữ nhật. Viết ct tính dt, cv hình vuông.

*** Bài 5:** Xây dựng lớp điểm có thuộc tính x1,y1 và pt hiện toạ độ điểm. Xây dựng lớp đường thẳng, thừa kế lớp điểm có pt tính chiều dài đoạn thẳng. Xây dựng lớp tam giác thừa kế lớp đoạn thẳng, có pt tính diện tích, chu vi tam giác.

*** Bài 6:** Nhập một danh sách gồm giảng viên và sinh viên, in ra danh sách những người được thưởng. Biết rằng điều kiện được thưởng là giảng viên có số bài báo >3, sinh vien có điểm thi tốt nghiệp >8.

*** Bài 7:** Thiết kế các lớp cho các danh mục cần quản lý trong thư viện bao gồm bài báo, sách, luận văn. Mỗi danh mục có nhan đề, tác giả, với mỗi loại danh mục ta có thêm các thông tin khác nhau:

a. Bài báo phải có tên tạp chí đăng bài báo và số phát hành của tạp chí.
b. Sách thì phải có nhà xuất bản.
c. Luận văn phải có tên trường.

Viết chương trình cho phép nhập, xuất thông tin về các danh mục này.

*** Bài 8:** Xây dựng lớp ma trận có phương thức cộng hai ma trận sao cho có thể sử dụng phương thức này cộng các ma trận có dữ liệu khác nhau như là ma trận số nguyên, ma trân phân số.

*** Bài 9:** Xây dựng lớp sắp xếp có phương thức sắp xếp sao cho có thể sử dụng phương thức này sắp xếp các dãy khác nhau như là dãy hình tròn sắp theo bán kính, dãy hình hình vuông sắp theo cạnh (biết rằng lớp hình vuông đã được thiết kế là thừa kế lớp hình chữ nhật).

- Hết -

Chương II

SỬ DỤNG GIAO DIỆN VÀ TRUY XUẤT CSDL

(GRAPHICAL USER INTERFACES – DATABASE)

I. GRAPHICAL USER INTERFACES (GUI)

1. Giới thiệu:

- Để người dùng có thể giao tiếp với chương trình ở dạng trực quan (dạng đồ hoạ), Java cung cấp nhiều thành phần giao tiếp như là: button, text box, menu,…
- Các gói java.awt (JDK 1.0) (AWT: Abstract Window Toolkit), javax.swing (JDK 2.0) chứa các lớp để tạo và quản lý các thành phần giao tiếp.
- Các thành phần trong gói swing bắt đầu bằng chữ J, và các thành phần này có nhiều chức năng hơn các thành phần trong gói awt và độc lập hoàn toàn với phần cứng.

2. Tạo khung cho cửa sổ ứng dụng

- Khi viết ứng dụng có dùng giao diện, ta cần tạo khung cửa sổ cho ứng dụng để chứa các thành phần giao tiếp.
- Trong gói swing có ba lớp dùng để tạo khung là: Jframe (khung chính), Jdialog (khung phụ, phụ thuộc vào khung khác) và JApplet (khung chứa applet). Khi viết applet không cần tạo khung vì applet thừa kế Frame.
- Thường các thành phần giao diện được ghép thành nhóm trong panel, trong panel lại có thể chứa

panel con và frame có thể chứa nhiều panel.

3. Sắp xếp các thành phần giao tiếp

- Muốn đặt các thành phần giao tiếp ở vị trí theo ý muốn mà không phụ thuộc vào độ phân giải của màn hình, ta dùng lớp quản lý bố cục (Layout Manager).
- Có 5 lớp quản lý bố cục là: FlowLayout, GridLayout, BorderLayout, CardLayout, GridBagLayout

  + FlowLayout: bố trí các thành phần từ trái sang phải, hết dòng sẽ xuống dòng.

  + GridLayout: bố trí các thành phần theo dạng bảng, theo thứ tự từ trái sang phải, từ trên xuống dưới.

  + BorderLayout: chia khung cửa sổ thành 5 phần: East, West, South, North, Center. North, South kéo dài theo phương ngang, West, East kéo dài theo phương dọc, Center kéo dài theo hai phương.
- Khi cần bố trí phức tạp, ta dùng các panel, mỗi panel có thể dùng trình quản lý bố cục riêng để bố trí những thành phần trong panel. Đối tượng Jpanel mặc định sử dụng FlowLayout, containers trong đối tượng JApplat,JDialog, JFrame mặc định dùng BorderLayout, nhưng có thể dùng phương thức setLayout() để thay đổi lớp quản lý bố cục.

4. Xử lý sự kiện (Event)

- Khi người dùng tác động lên thành phần giao tiếp (ví dụ nhấn chuột, nhấn phím,…) sẽ sinh ra một sự kiện, Java sẽ gởi sự kiện này cho đối tượng lắng nghe sự kiện, đối tượng lắng nghe sự kiện sẽ gọi một phương thức đặc biệt để xử lý sự kiện.
- Mỗi thành phần giao tiếp (đối tượng nguồn) cần phải đăng ký các đối tượng lắng nghe sự kiện (đối tượng đích) và trong mỗi đối tượng đích cần cài đặt phương thức xử lý sự kiện (đối tượng nguồn và đối tượng đích có khi chỉ là một)

HÀNH ĐỘNG	ĐỐI TƯỢNG NGUỒN	SỰ KIỆN ĐƯỢC TẠO RA
Nhấn nút button	JButton	ActionEvent
Thay đổi văn bản của textfield	JTextComponent	TextEvent
Nhấn enter trên textfield	JTextFeild	ActionEvent
Chọn mục của combobox	JComboBox	ItemEvent, ActionEvent
Chọn các mục của list	JList	ListSelectionEvent
Chọn mục của menu	JMenuItem	ActionEvent
Dời focus đến hoặc đi khỏi thành phần	Component	FocusEvent
Nhấn hoặc thả phím trên thành phần	Component	KeyEvent
Di chuyển chuột trên thành phần	Component	MouseEvent
Mở, đóng, phóng to, thu nhỏ cửa sổ	Window	WindowEvent

Đối tượng lắng nghe sự kiện (đối tượng đích) cần dùng giao diện phù hợp để cung cấp phương thức xử lý sự kiện tương ứng. Thông thường sự kiện tên là XEvent thì đối tượng đích phải dùng giao diện XListener và phương thức đăng ký là addXListener.

Ví dụ: Khi nhấn nút button (đối tượng nguồn) sẽ sinh ra sự kiện ActionEvent, sự kiện ActionEvent được gởi cho đối tượng lắng nghe sự kiện (đối tượng đích). Đối tượng nguồn dùng phương thức addActionEvent để đăng ký đối tượng đích, đối tượng đích dùng giao diện ActionListener cung cấp phương thức actionPerformed(ActionEvent e) để xử lý sự kiện ActionEvent.

Sự kiện	Dùng giao diện	Phương thức xử lý sự kiện
ActionEvent	ActionListener	actionPerformed(ActionEvent e)
TextEvent	TextListener	textValueChanged(TextEvent e)
ItemEvent	ItemListener	itemStateChanged(ItemEvent e)
ListSelectionEvent	ItemListener	itemStateChanged(ItemEvent e)
FocusEvent	FocusListener	focusGained(FocusEvent e) focusLost(FocusEvent e)
KeyEvent	KeyListener	keyPressed(KeyEvent e) keyReleased(KeyEvent e) keyTyped(KeyEvent e)
MouseEvent	MouseListener MouseMotionListener	mousePressed(MouseEvent e) mouseReleased(MouseEvent e) mouseClicked(MouseEvent e) mouseMoved(MouseEvent e) mouseDragged(MouseEvent e)
WindowEvent	WindowListener	windowClosed(WindowEvent e) windowOpened(WindowEvent e)

*** <u>Cài đặt đối tượng lắng nghe sự kiện</u>**

<u>Ví dụ</u>: xử lý sự kiện nhấn nút button

- xây dựng lớp dùng để tạo đối tượng lắng nghe sự kiện

```
public class MyClass  implements ActionListener{
        public void actionPerformed(ActionEvent e) {
        //các lệnh xử lý sự kiện
        }
}
```

- Đăng ký đối tượng lắng nghe sự kiện sinh ra từ thành phần giao tiếp

```
button.addActionListener(instanceOfMyClass);
```

- Nếu phương thức xử lý sự kiện đơn giản ta có thể khai báo là lớp vô danh bên trong (anonymous inner class), không cần xây dựng lớp lắng nghe sự kiện

<u>Ví dụ:</u>

```
button.addActionListener(new ActionListener()
{
   public void actionPerformed(ActionEvent e)
   {
      …
   }
});
```

<u>Ghi chú</u>: mã xử lý sự kiện được thực thi trong một tiểu trình riêng

5. Chương trình mẫu GUI

Dang 1: không xử lý sự kiện

```
import javax.swing.*;
public class SwingApplication
{
    public SwingApplication() //phuong thuc khoi tao co nhiem vu tao GUI
        {
                JFrame frame = new JFrame("Tieu de cua so");
                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                frame.getContentPane().add(<thanh phan giao tiep>);
                frame.setSize(200,100);//hoac frame.pack(); vua du chua cac thanh phan
giao tiep
                frame.setVisible(true); //Hien cua so
        }

    public static void main(String[] args)
    {
                new SwingApplication(); //tao GUI
    }
}
/*HIDE_ON_CLOSE:che, DO_NOTHING_ON_CLOSE:khong lam gi
  EXIT_ON_CLOSE:dong ung dung,DISPOSE_ON_CLOSE:huy,dong cua so
 */
```

Dang 2: có xử lý sự kiện

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class SwingApplication implements ActionListener
{
```

```
    JFrame Frm;
        public SwingApplication() //phương thức khoi tao
        {
        Frm = new JFrame("Tieu de cua so");
                    Frm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                    //Frm.setSize(new Dimension (120, 40));
            …
            Frm.pack();
                Frm.setVisible(true);
        }
        public void actionPerformed(ActionEvent event) //phương thức xu ly ActionEvent
        {
            //cac lenh xu ly su kien action
        }
        public static void main(String[] args)
        {
            new SwingApplication();
        }
}
```

II. DATABASE

1. Truy xuất CSDL: gồm các bước sau

STT	Thao tác	Lệnh (ví dụ)
1	Nạp driver truy xuất CSDL	Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
2	Kết nối CSDL	Connection con = DriverManager.getConnection("jdbc:odbc:student”,”sa”,””);
3	Tạo đối tượng câu lệnh	Statement stmt=con.createStatement();

4	Thực thi câu sql	ResultSet rs=stmt.executeQuery(sql);//select
		int n= stmt. executeUpdate (sql);// create, alter,drop, update, delete, insert

5	Lấy thông tin bảng kết quả rs	ResultSetMetaData rsmd=rs.getMetaData();
6	Lấy số cột trong bảng	int col=rsmd.getColumnCount();
7	Lấy tên cột thứ i (i=1,…)	String label=rsmd.getColumnLabel(i)
8	Lấy giá trị ở cột i hàng hiện tại	String value=rs.getString(i)//rs.getFloat(i),.. Tham số có thể là tên cột
9	Dời con trỏ mẫu tin tới hàng kế	rs.next(): false là cuối bảng, con trỏ bắt đầu trên dòng 1, lệnh next() đầu tiên sẽ dời con trỏ tới dòng 1

Ví dụ: in bangdiem

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
Connection con = DriverManager.getConnection("jdbc:odbc:qld", "sa", "");
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery("SELECT masv,diemthi FROM bangdiem");
while (rs.next()) {
    String masv = srs.getString("masv");//srs.getString(1);
    int diemthi = srs.getFloat("diemthi");//srs.getFloat(2);
    System.out.println(masv + "     " + diemthi);
}
```

*** Cách truy xuất CSDL nhanh hơn**

Nếu muốn thực thi câu lệnh sql nhiều lần với các giá trị tham số khác nhau nên dùng đối tượng PrepareStatement thay cho đối tượng Statement thì việc thực hiện sẽ nhanh hơn nhiều. Với đối tượng Statement mỗi khi gọi stmt.executeQuery(sql); hoặc stmt.executeUpdate(sql); thì câu lệnh sql được gởi lên cho hệ quản trị CSDL, hệ quản trị sẽ biên dịch và thực thi câu sql, nếu việc biên dịch thực hiện nhiều lần sẽ làm chậm việc thực thi. Với đối tượng PrepareStatement thì việc biên dịch câu lệnh sql chỉ thực hiện một lần do đó sẽ nhanh hơn nhiều.

Ví dụ:

```
Statement stmt=con.createStatement();
for(int i = 0; i < 10; i++)
        stmt. executeUpdate ( "insert into bangdiem values( ‘dh01”+i+”’, ”+i+”)" );
```

Có thể thay bằng đoạn mã sau nhanh hơn

```
PreparedStatement ps = con.prepareStatement( "insert into bangdiem values( ?, ?)");
for(int i = 0; i < 10; i++)
{
        ps.setString(1, “dh01”+i);
        ps.setInt(2, i);
        ps.executeUpdate();
}
```

2. Sử dụng Stored Procedure của SQL server

- Tạo Stored Procedure:

Stored Procedure có thể tạo sẵn trong SQL server hoặc dùng đoạn mã Java sau:

String CreateProc = "create procedure ProcName (d/s ts nếu có) as …"

Statement stmt = con.createStatement();

stmt.executeUpdate(CreateProc);

Khi thực thi, đoạn mã này sẽ tạo một Stored Procedure lưu vào CSDL.

- Gọi Stored Procedure:

//tạo đt chứa lời gọi SP

CallableStatement cs = con.prepareCall("{call ProcName (?,?,…)}");

//gởi tham số cho SP

cs.setString(int parameterIndex, String x);// parameterIndex=1,2… là số thứ tự của tham số

hoặc

cs.setInt(int parameterIndex, int x); cs.setFloat(int parameterIndex, float x); …

//đăng kí tham số là tham số trả về

cs.registerOutParameter(int parameterIndex, int sqlType)

sqlType có thể là các hằng sau: java.sql.Types.INTEGER, java.sql.Types.FLOAT, java.sql.Types.VARCHAR

//thực thi SP

ResultSet rs = cs.executeQuery(); //SP la select, rs là bảng kết qủa

hoặc

int rec = cs.executeUpdate(); //SP la insert, delete, update,…rec là số mẫu tin được xử lý

hoặc

boolean flag = cs.execute(); //SP phối hợp nhiều select, insert,…flag=true là thành công

//lấy kết quả do SP trả về

ResultSet rs=cs.getResultSet();//lấy kq ResultSet

int rec=cs.getUpdateCount();//lấy kq là số mẫu tin được xử lý

boolean flag= cs.getMoreResult();// (cs.getMoreResults() == false) && (cs.getUpdateCount==-1): là hết kết quả.

3. Các chức năng khác

3.1 Di chuyển cursor

*** Tạo đối tượng ResultSet có thể cuộn**

Statement stmt = con.createStatement(int resultSetType, int resultSetConcurrency);

resultSetType có thể là: ResultSet.TYPE_FORWARD_ONLY, ResultSet.TYPE_SCROLL_INSENSITIVE, hoặc ResultSet.TYPE_SCROLL_SENSITIVE (không hay có phản ánh những thay đổi khi ResultSet đang mở)

resultSetConcurrency có thể là: ResultSet.CONCUR_READ_ONLY hay ResultSet.CONCUR_UPDATABLE (ResultSet chỉ đọc hay có thể cập nhật)

Mặc định resultSetType=ResultSet.TYPE_FORWARD_ONLY, resultSetConcurrency = ResultSet.CONCUR_READ_ONLY

ResultSet srs = stmt.executeQuery("SELECT COF_NAME, PRICE FROM COFFEES");

Ví dụ: in bangdiem theo thứ tự ngược

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
Connection con = DriverManager.getConnection("jdbc:odbc:qld", "sa", "");
Statement stmt = con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
                                     ResultSet.CONCUR_READ_ONLY);
ResultSet rs = stmt.executeQuery("SELECT masv,diemthi FROM bangdiem");
rs.afterLast();
while (rs.previous()) {
    String masv = srs.getString("masv");
    int diemthi = srs.getFloat("diemthi");
    System.out.println(masv + "     " + diemthi);
}
```

*** Di chuyển cursor**

boolean rs.first();

boolean rs.last();

void rs.beforeFirst();

void rs.afterLast();

int absolute(int) :âm di chuyển ngược tính từ cuối: -1:cuối, -2 kế cuối,…

boolean rs.relative(int): di chuyển tính từ hàng hiện hành.

int getRow(): trả về vị trí cursor

vv…

Ví dụ:

```
rs.absolute(4); //tới hàng 4
int rowNum = rs.getRow(); // rowNum = 4
rs.relative(-3);
int rowNum = rs.getRow(); // rowNum = 1

rs.relative(2);
int rowNum = rs.getRow(); // rowNum = 3
```

isFirst, isLast, isBeforeFirst, isAfterLast: kiểm tra vị trí cursor

ví dụ:

```
if (rs.isAfterLast() == false) {
    rs.afterLast();
}
while (rs.previous()) {
    String masv = rs.getString("masv");
    int diemthi = rs.getFloat("diemthi");
    System.out.println(masv + "     " + diemthi);
}
```

3.2 Sử dụng ResultSet cập nhật CSDL

Ví dụ:

```
Connection con = DriverManager.getConnection("jdbc:odbc:qld", "sa", "");
Statement stmt = con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
                                     ResultSet.CONCUR_UPDATABLE);
ResultSet rs = stmt.executeQuery("SELECT masv,diemthi FROM bangdiem");
rs.last();
rs.updateInt("diemthi", 9); //cập nhật trong ResultSet
rs.cancelRowUpdates();          //bỏ cập nhật trong ResultSet
rs.updateFloat("diemthi", 8);
rs.updateRow(); //cập nhật trong CSDL
```

```
rs.moveToInsertRow(); //InsertRow là buffer chứa hàng cần thêm
rs.updateString("masv", "dh0101");
rs.updateInt("diemthi", 5);
rs.insertRow(); //thêm vào rs và CSDL
rs.updateString("masv", "dh0102");
rs.updateInt("diemthi", 7);
rs.insertRow(); //thêm vào rs và CSDL
rs.moveToInsertRow()//di chuyển tới hàng mới thêm
rs.deleteRow(); //xóa hàng mới thêm khỏi rs và CSDL
rs.first();
rs.refreshRow();//làm mới hàng hiện tại
```

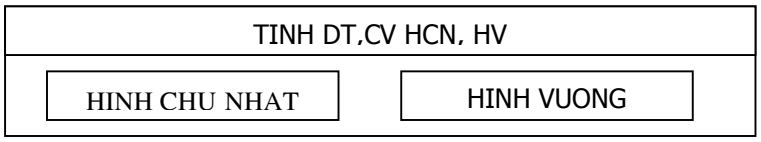
***Chú ý**:* mặc dù rs là SENSITIVE nhưng có khi rs vẫn chưa cập nhật record mới do phụ thuộc vào Driver và DBMS nên để chắc chắn có rs mới, thực hiện lại lệnh : rs.close();

```
rs = stmt.executeQuery("SELECT masv,diemthi FROM bangdiem");
```

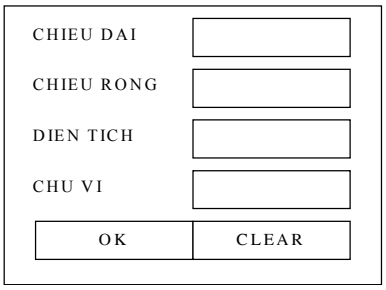
BÀI TẬP CHƯƠNG II:
GUI – DATABASE

*** Bài 1:**

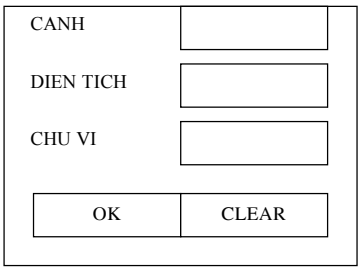
Viết chương trình tình diên tích, chu vi hcn, hv có giao diện như sau:



Khi click nút "HINH CHU NHAT" sẽ hiện:



Khi click nút "HINH VUONG" sẽ hiện:



Nhập cd, cr, click OK, hiện dt, cv. Click CLEAR xoá dữ liệu

*** Bài 2:**

Viết chương trình xử lý file văn bản có các chức năng sau: tạo file, xem file, sao chép file. Chương trình viết bằng hai cách: không có giao diện và có giao diện.

*** Bài 3:**

Viết chương trình quản lý sinh viên có các chức năng sau: nhập mã sv, họ tên, đlt, đth, tính đtb và ghi thông tin này vào file truy xuất ngẫu nhiên; tìm/xóa/sửa thông tin sinh viên khi biết masv; xem danh sách sinh viên. Chương trình viết bằng hai cách: không có giao diện và có giao diện

*** Bài 4:**

Tương tự bài 3 nhưng thông tin lưu trong CSDL SQL server.

- Hết -

CHƯƠNG III

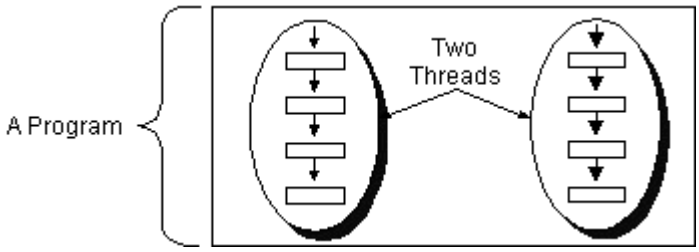
MULTI THREADS & APPLET &URL

I. LẬP TRÌNH MULTI THREADS

1. Khái niệm tiểu trình (thread):

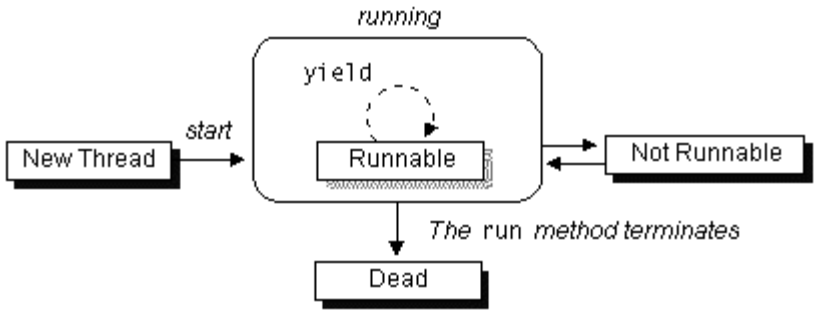
- Tiểu trình là một dòng điều khiển tuần tự bên trong một tiến trình.
- Một tiến trình có thể tạo nhiều tiểu trình, mỗi tiểu trình thực hiện một công việc nào đó và thực thi độc lập, đồng thời bằng cách chia sẻ CPU.
- Các tiểu trình trong cùng một tiến trình dùng chung không gian địa chỉ tiến trình nhưng có con trỏ lệnh, tập các thanh ghi và ngăn xếp riêng.
- Các tiến trình chỉ có thể liên lạc với nhau thông qua các cơ chế do hệ điều hành cung cấp. Các tiểu trình liên lạc với nhau dễ dàng thông qua các biến toàn cục của tiến trình.

Ví dụ: Trình duyệt là một chương trình có nhiều tiểu trình: tiểu trình tải hình, download file, tiểu trình thực hiện hoạt hình, …



***** Khi nào cần viết chương trình có sử dụng tiểu trình?

2. Chu kỳ sống của tiểu trình:



Các trạng thái mà tiểu trình có thể trải qua trong quá trình sống là:

Khi thread được tạo bởi lệnh new, thread chưa được cấp phát tài nguyên và ở trạng thái "New Thread". Pt start() sẽ cấp tài nguyên cho thread, sắp lịch thực thi cho thread và gọi pt run(), lúc này thread ở trạng thái "Runnable" (đợi CPU). Khi CPU rỗi hoặc đến lượt thread, JVM sẽ chuyển thread sang trạng thái "running". Thread sẽ chuyển sang trạng thái "Not Runnable' khi gọi pt sleep() hoặc wait() hoặc thread đang chờ nhập/xuất. Khi pt run() thực thi xong, thread sẽ chuyển sang trạng thái " Dead " và kết thúc, có thể dùng pt isAlive() để kiểm tra thread còn sống hay chết.

3. Độ ưu tiên của tiểu trình

Do có một CPU, nên tại một thời điểm chỉ có thể có một thread thực thi. Java dùng giải thuật "độ ưu tiên cố định " để điều phối hoạt động của các thread. Thread được chọn sẽ thực hiện cho tới khi một trong các trường hợp sau xảy ra:

- Thread có độ ưu tiên cao hơn ở trạng thái Runnable
- Thread đang thực thi nhường quyền thực thi hay pt run() của nó đã kết thúc
- Hết thời lượng quantum dành cho nó.

Vì để tránh trường hợp thread đói CPU, bộ lập lịch của Java không đảm bảo thread có độ ưu tiên cao nhất đang thực thi, nên khi lập trình không nên dựa vào độ ưu tiên. Ngoài ra cũng không nên dựa vào quantum vì có thể hệ thống thread đang thực thi không hỗ trợ quantum.

* Cú pháp lệnh:

set Priority (n): dùng để thay đổi độ ưu tiên, n là độ ưu tiên trong khoảng (MIN_PRIORITY, MAX_PRIORITY), MIN_PRIORITY, MAX_PRIORITY là hằng số định nghĩa trong lớp Thread, n lớn là độ ưu tiên cao. Khi một thread được tạo ra sẽ có cùng độ ưu tiên với thread tạo ra nó

4. Viết chương trình sử dụng tiểu trình

4.1 Sử dụng lớp Timer và TimerTask

Java cung cấp các lớp sau: java.util.Timer, java.util.TimerTask, javax.swing.Timer, javax.swing.SwingWorker để viết ứng dụng có nhiều công việc thực hiện đồng thời và mỗi công việc được thực hiện lặp lại sau một khoảng thời gian nào đó.

* Cú pháp lệnh

- **public void schedule(TimerTask task, long delay):**

  task sẽ được thực hiện sau khoảng thời gian delay (milliseconds)

- **public void schedule(TimerTask task, Date time)**

  task sẽ được thực hiện vào thời điểm xác định

Ví dụ: thực hiện task lúc 11:01 p.m

```
Calendar calendar = Calendar.getInstance();
calendar.set(Calendar.HOUR_OF_DAY, 23);
calendar.set(Calendar.MINUTE, 1);
calendar.set(Calendar.SECOND, 0);
Date time = calendar.getTime();
timer = new Timer();
timer.schedule(new Task(), time);
```

- **public void schedule(TimerTask task, long delay, long period)**
  public void schedule(TimerTask task, Date time, long period)

  Lặp lại task sau khoảng thời gian period (milliseconds)

- **public void cancel()**

  Kết thúc timer, loại bỏ các công việc do timer sắp đặt thực hiện. Nếu timer chưa kết thúc, chương trình vẫn thực thi, nếu muốn kết thúc ct có thể sử dụng lệnh System.exit(0)

4.2 Sử dụng lớp Thread

Java cung cấp lớp java.lang.Thread để viết tiểu trình dạng tổng quát hơn. Pt start() sẽ gọi pt run() thực thi công việc của thread.

4.3 Sử dụng giao diện Runnable

Nếu lớp đã thừa kế một lớp khác, vì do java không cho phép đa thừa kế, ta cần dùng giao diện Runnable

5. Đồng bộ các tiểu trình

Thông thường các tiểu trình thực thi độc lập, không cần quan tâm tới hoạt động của tiểu trình khác nhưng nếu các tiểu trình dùng chung tài nguyên, hoặc cần phải phối hợp để hoàn thành công việc thì ta cần phải đồng bộ việc thực thi của các tiểu trình.

Đoạn mã truy xuất tài nguyên dùng chung mà có khả năng xảy ra lỗi gọi là miền găng(Critical Section). Có thể giải quyết lỗi nếu bảo đảm tại một thời điểm chỉ có một tiểu trình truy xuất tài nguyên dùng chung. Trong Java miền găng là một phương thức được

khai báo bằng từ khoá synchronized, một đối tượng có miền găng sẽ là độc quyền truy xuất (tại một thời điểm chỉ có một tiểu trình truy xuất miền găng của đối tượng)

II. VIẾT ỨNG DỤNG APPLET

1. Giới thiệu

Applet là một ứng dụng java, thừa kế lớp java.applet.Applet hoặc lớp javax.swing.JApplet. Trong Applet không có hàm main nên không thể tự thực thi, muốn thực thi Applet cần "nhúng" Applet vào trang web và trình duyệt web sẽ thực thi Applet.

Applet có thể để ở máy Web Server, client dùng browser yêu cầu Web Server gởi cho client trang web có nhúng Applet, trang web cùng với Applet được download về máy client, trình duyệt ở máy client chịu trách nhiệm thông dịch trang web và thực thi Applet.

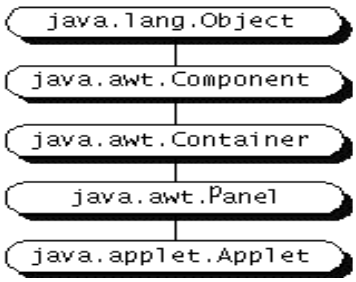
Lập trình Applet chính là một dạng lập trình mạng dạng đơn giản: viết applet để ở máy server, client dùng browser lấy applet về máy client và cho thực thi. Applet thường chậm vì phải download về máy client, ngoài ra Applet còn tìm ẩn sự không an toàn do Applet thực thi trên máy client.

*** Tính an toàn**

Vì lý do an toàn nên Applet bị hạn chế sau:

- Không thể đọc/ghi file hoặc thực thi ct trên máy mà Applet đang thực thi
- Không thể kết nối với máy khác ngoại trừ máy cung cấp Applet.
- Không thể thực thi ct trên

*** Sơ đồ thừa kế của lớp Applet:**



Tất cả các lớp đều thừa kế lớp java.lang.Object và lớp này tự động import vào tất cả các lớp khác.

2. Các pt đặc biệt:

```
public void init() { . . . }
public void start() { . . . }
public void stop() { . . . }
public void destroy() { . . . }
 public void paint(Graphics g) {…}
```

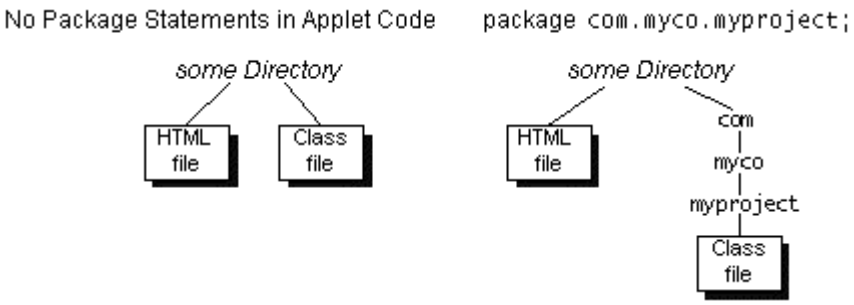
- Khi Applet được download về máy client thì pt init() và start(), paint() được thực thi. Pt init(), start() thường có các lệnh thực hiện các công việc khởi tạo ban đầu như kết nối CSDL, mở socket,…Pt paint() vẽ Applet.
- Khi user chuyển qua trang khác thì pt stop() được gọi và khi user trở về trang chứa Applet thì pt start(), paint() lại được gọi. Pt stop() thường có các lệnh để ngừng thực hiện Applet.
- Khi Applet kết thúc việc thực hiện hoặc khi đóng trình duyệt thì pt destroy() được gọi. Pt destroy() thường có các lệnh thực hiện các công việc dọn dẹp như đóng kết nối CSDL, đóng socket,…

3. Thực thi Applet

- **Tạo trang Web**
- **Nhúng Applet vào trang web**

<APPLET CODE=AppletSubclass.class WIDTH=anInt HEIGHT=anInt>

</APPLET>



- **Dùng trình duyệt, xem trang web có nhúng applet**

4. Sử dụng thẻ <APPLET>

- **Trình duyệt gởi tham số cho Applet:**

<APPLET CODE=AppletSubclass.class WIDTH=anInt HEIGHT=anInt>

<PARAM NAME=parameter1Name VALUE=aValue>

<PARAM NAME=parameter2Name VALUE=anotherValue>

</APPLET>

- **Applet lấy tham số do trình duyệt gởi**

public String getParameter(String name)

- **Xác định thư mục chứa Applet**

Mặc định browser tìm Applet trong thư mục chứa file html, nếu Applet trong gói, browser dùng tên gói để xây dựng đường dẫn bên dưới tm chứa file html. Nếu applet để ở nơi khác, dùng thuộc tính CODEBASE để xác định vị trí applet.

<APPLET CODE=AppletSubclass.class CODEBASE=aURL

WIDTH=anInt HEIGHT=anInt>

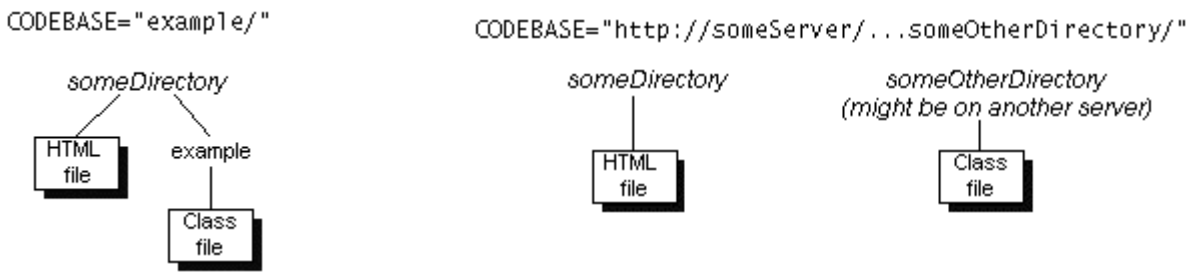
</APPLET>

Nếu aURL là một URL tương đối thì xét từ vị trí của file html.

Ví dụ:

<APPLET CODE=Simple.class CODEBASE="example/"

WIDTH=500 HEIGHT=20>



</APPLET>

5. Một số phương thức thông dụng

- **Xác định thư mục chứa file dữ liệu**

Ví dụ:

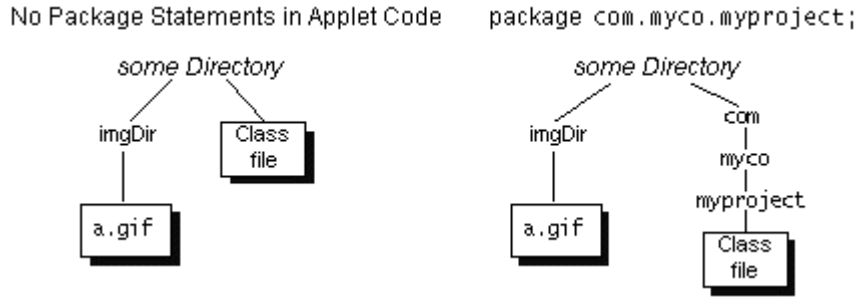


Image image = getImage(getCodeBase(), "imgDir/a.gif");

- **Hiện một thông báo ở thanh trạng thái (status bar)**

Ví dụ:

showStatus("MyApplet: Loading image file " + file);

- Hiện tài liệu trong browser

public void showDocument(java.net.URL url)

public void showDocument(java.net.URL url, String targetWindow)

targetWindow có thể có các giá trị sau:

"_blank"	: hiện tài liệu trong một cửa sổ mới, không có tên
"<windowName>"	: hiện tài liệu trong cửa sổ tên là <windowName>
"_self"	:hiện tài liệu trong cửa sổ và trong cùng khung chứa applet
"_parent"	:hiện tài liệu trong cửa sổ chứa applet nhưng trong khung cha của khung chứa applet . Nếu không có khung cha thì hiện giống "_self"
"_top"	:hiện tài liệu trong cửa sổ chứa applet nhưng trong khung mức cao nhất

Ví dụ:

```
AppletContext appletContext=getAppletContext();
appletContext.showDocument(url);
```

- **Gởi thông điệp tới applet khác**

Vì lý do an toàn nên một số browser yêu cầu các applet phải cùng một tm trên cùng một server và nhúng trong cùng trang. Mỗi applet phải được đặt tên dùng thuộc tính NAME:

```
<APPLET CODEBASE=example/ CODE=Sender.class
WIDTH=450
HEIGHT=200
NAME="buddy">
</applet>
```

hoặc gởi tên như là tham số:

```
<APPLET CODEBASE=example/ CODE=Receiver.class
  WIDTH=450
```

```
 HEIGHT=35>
<PARAM NAME="name" value="old pal">
</applet>
```

+ Tìm 1 applet

```
String receiverName = …;
Applet receiver = getAppletContext().getApplet(receiverName);
((Receiver)receiver).processRequestFrom(myName);//gọi pt của receiver
```

+ Tìm tất cả applet

```
Enumeration e = getAppletContext().getApplets();
```

- **Thực thi file âm thanh *.au**

getAudioClip(URL), getAudioClip(URL, String) : Trả về đối tượng sử dụng giao diện AudioClip, giao diện AudioClip có các pt sau:loop,play,stop.

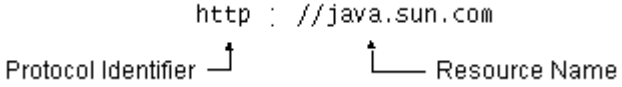
play(URL), play(URL, String) : thực thi AudioClip xác định bởi URL

III. SỬ DỤNG URL(Uniform Resource Locator)

1. Khái niệm:

- URL Là một tham chiếu đến một tài nguyên trên mạng (địa chỉ tài nguyên trên mạng), Web Browser hoặc các ứng dụng khác dùng URL để tìm tài nguyên trên mạng. Gói java.net có lớp URL dùng để biểu diễn địa chỉ URL
- URL có dạng chuỗi gồm: giao thức dùng để truy xuất tài nguyên và tên tài nguyên.

  Ví dụ:



  Có thể dùng các giao thức khác như: File Transfer Protocol (FTP), Gopher, File, News…
- Tên tài nguyên là thông tin đầy đủ để xác định tài nguyên, thông tin này phụ thuộc vào giao thức sử dụng, nhưng thường có các thông tin sau:

Host Name	Tên máy chứa tài nguyên.
Filename	Đường dẫn tới tài nguyên
Port Number	Số hiệu cổng dùng khi kết nối

Đối với http, filename mặc định là index.html

2. Tạo và sử dụng đối tượng URL

Phương thức	Ý nghĩa
URL(String spec)	Tạo đối tượng URL từ một chuỗi URL
URL(String protocol, String host, int port, String file)	Tạo đối tượng URL từ protocol, host, port và tên file
URL(String protocol, String host, String file)	Tạo đối tượng URL từ protocol, host, port và tên file, dùng port mặc định
Object getContent()	Lấy nội dung của URL
String getFile()	Lấy tên file cùng đường dẫn của URL
String getHost()	Lấy tên máy của URL
int getPort()	Lấy số hiệu cổng của URL
String getProtocol()	Lấy tên protocol của URL
InputStream openStream()	Kết nối tới url và mở luồng nhập để đọc thông tin từ url
URLConnection openConnection()	Mở kết nối tới URL

- Hết -

BÀI TẬP CHƯƠNG III

LẬP TRÌNH MULTITHREAD , APPLET , URL

*** Bài 1:**

Viết chương trình thực hiện n bài toán cộng ngẫu nhiên. Khi gần hết giờ sẽ kêu ba tiếng, mỗi tiếng cách nhau 2 giây, sau đó thông báo hết giờ, và đóng chương trình. Thời gian làm bài qui định là n giây (không tính thời gian thông báo hết giờ).

*** Bài 2:**

a) Viết chương trình gồm hai tiểu trình thực thi đồng thời, một tiểu trình in các kí tự từ 'a' đến 'z', một tiểu trình in số từ 1 đến 26.

b) Viết lại chương trình trên sao cho hai tiểu trình in luân phiên, kết quả có dạng sau: 1 a 2 b 3 c 4 d …

*** Bài 3:**

Viết chương trình mô phỏng bài toán "nhà sản xuất - người tiêu thụ". Các Nsx sản xuất ra các số nguyên, đặt vào kho, các Ntt lấy các số này ra sử dụng. Yêu cầu đặt ra:

- Nsx phải tạm ngừng sx khi kho đầy
- Ntt phải tạm ngừng tt khi kho rỗng

Khi Nsx và Ntt không được truy xuất kho đồng thời.

*** Bài 4:**

Viết Applet giải phương trình bậc hai, tham số a,b,c gởi cho Applet bằng hai cách: dùng thẻ PARAM hoặc cung cấp giao diện nhập.

*** Bài 5:**

Viết Applet hoạt hình "người đánh trống", khi qua trang khác "người đánh trống" phải ngừng đánh, khi trở lại trang có applet thì "người đánh trống" lại đánh tiếp.

*** Bài 6:**

Viết Applet đồng hồ đa năng có các chức năng sau: hiện đồng hồ điện tử, hiện đồng hồ số có kim quay, hiện quả lắc đang lắc, hiện dòng chữ quảng cáo đang chạy.

*** Bài 7:**

Viết chương trình xem file văn bản từ xa thông qua web server

*** Bài 8:**

Viết chương trình download file thông qua web server

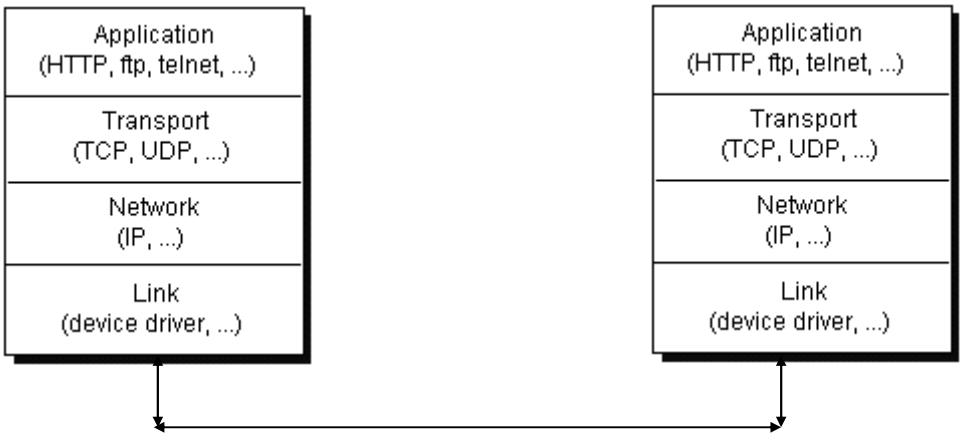
- Hết -

CHƯƠNG IV
LẬP TRÌNH CLIENT\SERVER

I. Tổng quan về lập trình mạng:

1. Giới thiệu

- Máy tính trên Internet liên lạc với nhau dùng giao thức TCP (Transmission Control Protocol) hoặc UDP (User Datagram Protocol).
- Khi viết chương trình Java liên lạc trên mạng là viết ở lớp ứng dụng (lớp application) và sử dụng những lớp trong gói java.net để truy xuất lớp TCP/UDP (lớp transport).



- **Lập trình client/server**: Là viết ứng dụng trên mạng gồm hai chương trình: chương trình client và chương trình server. Chương trình client gởi yêu cầu tới chương trình server, ct server xử lý yêu cầu và trả kết qủa về cho ct client. Ct server có thể phục vụ đồng thời nhiều yêu cầu của các ct client.
- **Lập trình WEB**: là trường hợp đặc biệt của lập trình client/server. Ct client là ct Browser (trình duyệt web), ct server là Web Server nhận yêu cầu trang web từ Browser, Web Server tìm trang web gởi về cho Browser, brower thực thi trang web hiện kết qủa trên màn hình client. Browser và web server liên lạc qua giao thức HTTP thông qua cổng mặc định là 80.

Trang Web là file .html viết bằng ngôn ngữ HTML (HyperText Markup Language), Browser sẽ thông dịch trang web. Khi client muốn yêu cầu trang web, dùng browser gõ vaò chuỗi có dạng sau:

http://NameServer:port/path/file.html

http là giao thức liên lạc giữa Browser và Web server, NameServer là tên máy web server đang thực thi, port là số hiệu cổng web server sử dụng, path/file.html là trang web được yêu cầu.

ví dụ: http://www.microsoft.com/index.html (không có port thì mặc định là 80)

Chuỗi này gọi là chuỗi định vị tài nguyên URL (Uniform Resource Locator) dùng để xác định tài nguyên trên mạng Internet. Ngoài giao thức HTTP còn có thể sử dụng các giao thức khác như là FTP, Gopher, File, và News. Hiện có các Browser thông dụng như: Internet Explorer, Netscape Navigator, và các Web Server như:IIS (Internet Information Server), PWS (Personal Web Server), JRUN, Tomcat,…

2. Giao thức TCP/UDP

a) Giao thức TCP:

- Thiết lập kết nối và duy trì kết nối.
- Đảm bảo dữ liệu gởi, được nhận chính xác và đúng thứ tự, ngược lại sẽ báo lỗi.

Các giao thức Hypertext Transfer Protocol (HTTP), File Transfer Protocol (FTP), và Telnet là những ứng dụng dùng kết nối TCP. Máy tính liên lạc dùng giao thức TCP giống như con người liên lạc bằng điện thoại.

b) Giao thức UDP:

- Không duy trì kết nối
- Không đảm bảo dữ liệu gởi, được nhận chính xác và đúng thứ tự,
- Gởi/nhận dữ liệu dạng gói (datagram), các gói gởi/nhận độc lập với nhau.

UDP nhanh hơn TCP vì không kiểm tra dữ liệu, không cần duy tri kết nối. Những ứng dụng như hỏi giờ, nhắn tin, lệnh ping nên dùng UDP. Máy tính liên lạc dùng giao thức UDP giống như con người liên lạc bằng thư tín.

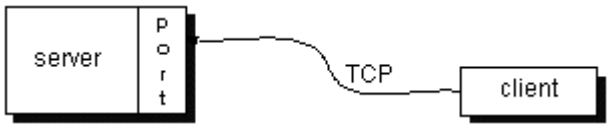
Lưu ý: có những firewalls và routers không cho phép gởi/nhận gói UDP do admin đã đặt cấu hình cấm gói UDP.

3. Địa chỉ IP, cổng (Port), socket:

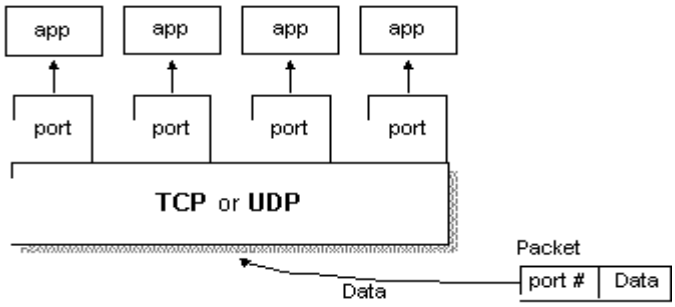
- **Địa chỉ IP**: là số 32 bit mà IP dùng để xác định máy tính.
- **Cổng**: là số 16 bit mà TCP/UDP dùng để xác định ứng dụng trên máy tính sẽ nhận dữ liệu. Dữ liệu khi gởi đi, được gởi kèm theo địa chỉ IP của máy nhận và cổng mà ứng dụng trên máy nhận sử dụng. Số hiệu cổng trong phạm vi từ 0 … 65,535, những số hiệu cổng từ 0 đến 1023 nên hạn chế sử dụng vì chúng đã được dùng cho những dịch vụ thông dụng như HTTP, FTP.

Dịch vụ	**Cổng**
FTP (truyền /nhận file)	21
HTTP (web)	80
TELNET (truy xuất máy tính từ xa)	23
SMTP (gửi mail)	25
POP3 (lấy mail)	110

- **Socket:** là cấu trúc dữ liệu lưu trữ các thông tin dùng để kết nối, dữ liệu được gởi/nhận thông qua socket. Trong liên lạc TCP, ứng dụng server kết buộc một socket với một cổng cụ thể, nghĩa là ứng dụng server đăng ký nhận tất cả dữ liệu gởi cho cổng đó.



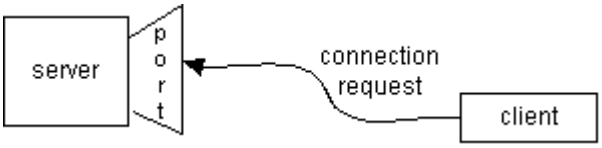
Trong liên lạc UDP, dữ liệu gởi/nhận dạng gói. Gói chứa số hiệu cổng, UDP sẽ gởi gói cho ứng dụng tương ứng.



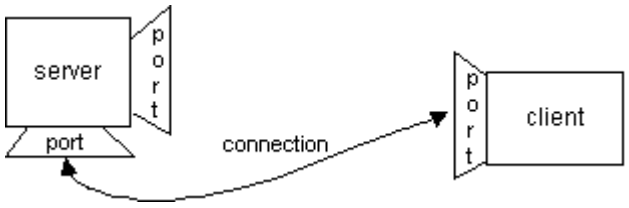
II. Lập trình TCP - Sử dụng socket

1. Khái niệm:

- Socket là một đầu trong kết nối liên lạc hai chiều giữa hai chương trình trên mạng. Gói java.net cung cấp lớp Socket để cài đặt kết nối phía client và ServerSocket để cài đặt kết nối phía server.
- SocketServer đợi, lắng nghe yêu cầu kết nối từ SocketClient



Khi chấp nhận kết nối, SocketServer tạo một socket mới kết buộc với một cổng khác để phục vụ cho client đã kết nối, trong khi SocketServer vẫn tiếp tục lắng nghe yêu cầu kết nối từ các client khác.



Sau đó server và client sẽ liên lạc với nhau thông qua socket của chúng.

Lưu ý:

+ Client dùng cổng cục bộ trên máy của client.

+ Nếu kết nối tới Web Server thì lớp URL thích hợp hơn lớp Socket, thực ra lớp URL cũng sẽ sử dụng lớp socket

2. Cấu trúc chương trình client

STT	Thao tác	Sử dụng lệnh
1	Mở socket (dùng để kết nối với server)	Socket cs = new Socket("ServerName", port);
2	Mở luồng đọc (dùng để đọc dl do server gởi)	BufferedReader in = new BufferedReader (new InputStreamReader(cs.getInputStream()));
3	Mở luồng ghi (dùng để gởi dl cho server)	PrintWriter out = new PrintWriter (cs.getOutputStream(), true);
4	Mở luồng đọc dl từ bàn phím (nếu cần)	BufferedReader stdIn = new BufferedReader (new InputStreamReader(System.in));
5	Gởi dl cho server	out.println(String str);
6	Lấy dl do server gởi	String str=in.readLine();
7	Đóng các luồng đọc, ghi, đóng socket	in.close();out.close(); sdtIn.close(); cs.close();

Ghi chú:

- Reader, writer: có thể đọc ghi kí tự Unicode qua socket

- Đóng luồng đọc/ghi, sau đó mới đóng socket

*** Chương trình client mẫu**

```
import java.io.*;
import java.net.*;
public class Client
{
   public static void main(String[] args) throws IOException
  {
       Socket cs = new Socket("ServerName", 1234);
       PrintWriter out = new PrintWriter(cs.getOutputStream(), true);
       BufferedReader in = new BufferedReader(new InputStreamReader(
       cs.getInputStream() ));
      //các lệnh gởi, nhận dữ liệu với server
      out.close();      in.close();      cs.close();
   }
}
```

3. Cấu trúc chương trình server

STT	Thao tác	Sử dụng lệnh
1	Mở socket (dùng để kết nối với client)	ServerSocket ss = new ServerSocket(port);
2	Chờ client kết nối và chấp nhận kết nối	Socket cs= ss.accept();
3	Mở luồng đọc (dùng để đọc dl do server gởi)	BufferedReader in = new BufferedReader (new InputStreamReader(cs.getInputStream()));
4	Mở luồng ghi (dùng để gởi dl cho client)	PrintWrite out = new PrintWriter (cs.getOutputStream(), true);
5	Tạo luồng đọc dl từ bàn phím (nếu cần)	BufferedReader stdIn = new BufferedReader (new InputStreamReader(System.in));
6	Gởi dl cho client	out.println(String str);
7	Lấy dl do client gởi	String str=in.readLine();
8	Đóng các luồng đọc/ghi, đóng các socket	in.close();out.close(); sdtIn.close(); cs.close(); ss.close();

*** <u>Chương trình server mẫu (1 client)</u>**

```
import java.net.*;
import java.io.*;
public class Server {
   public static void main(String[] args) throws IOException
  {
      ServerSocket ss = new ServerSocket(1234);
      Socket cs = ss.accept();
      PrintWriter out = new PrintWriter(cs.getOutputStream(), true);
      BufferedReader in = new BufferedReader(new InputStreamReader(
                              cs.getInputStream()));
       //các lệnh gởi/nhận dữ liệu với một client
      out.close();      in.close();      cs.close();      ss.close();
   }
}
```

Để phục vụ nhiều client kết nối đồng thời thì server cần tạo ra các tiểu trình, mỗi tiểu trình sẽ gởi/nhận dữ liệu với một client.

*** <u>Chương trình server mẫu (nhiều client)</u>**

```
import java.net.*;
import java.io.*;
public class Server
{
   public static void main(String[] args) throws IOException
  {
       ServerSocket ss = new ServerSocket(1234);    //có thể thay port khác
       boolean listening = true;
       while (listening)
       {
              Socket cs=ss.accept();
              new ServiceThread(cs).start();
       }
      ss.close();
```

```
    }
}

//lớp ServiceThread : tiểu trình phục vụ cho một client
import java.net.*;
import java.io.*;
public class ServiceThread extends Thread
{
   private Socket socket = null;
   public ServiceThread(Socket socket)
  {
        super("ServiceThread");
        this.socket = socket;
    }
   public void run()
  {
        try
        {
           PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
           BufferedReader in = new BufferedReader( new InputStreamReader(
        socket.getInputStream()));
        //các lệnh gởi, nhận dữ liệu với một client
           out.close(); in.close(); socket.close();
        } catch (IOException e) {   e.printStackTrace(); }
    }
}
```

*** <u>Chương trình server mẫu (nhiều client) (cách khác)</u>**

```
import java.net.*;
import java.io.*;
public class Server extends Thread
{
   private Socket socket = null;
   public Server(Socket socket)
```

```
  {
        super("ServiceThread");
        this.socket = socket;
   }
   public static void main(String[] args) throws IOException
  {
        ServerSocket ss = new ServerSocket(1234);    //có thể thay port khác
        boolean listening = true;
        while (listening)
        {
              Socket cs=ss.accept();
              new Server(cs).start();
        }
       ss.close();
   }
  public void run()
  {
        try
        {
           PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
           BufferedReader in = new BufferedReader( new InputStreamReader(
        socket.getInputStream()));
        //các lệnh gởi, nhận dữ liệu với một client
           out.close(); in.close(); socket.close();
        } catch (IOException e) {   e.printStackTrace(); }
   }
}
```

III. Lập trình UDP- sử dụng datagram

1. Khái niệm

Datagram là gói chứa dữ liệu được UDP sử dùng để gởi dữ liệu qua mạng. Thứ tự nhận, nội dung các gói không đảm bảo giống như khi gởi. Gói java.net có các lớp DatagramSocket và DatagramPacket, MulticastSocket dùng để gởi/nhận gói.

2. Cấu trúc chương trình client

STT	Thao tác	Sử dụng lệnh
1	Mở một datagram socket dùng để liên lạc với máy server	DatagramSocket socket = new DatagramSocket();

2	Tạo gói gởi và gởi gói chứa dữ liệu	byte[] buf = new byte[256]; //mảng dùng để chứa dl
		String str=…; //dữ liệu cần gởi
		buf=str.getBytes(); //cất dl vào mảng
		InetAddress address = InetAddress.getByName(NameServer); /lấy IP của máy server
		DatagramPacket packet = new DatagramPacket(buf, buf.length, address,1234); //tạo gói gởi
		socket.send(packet); //gởi gói

3	Tạo gói nhận và nhận gói trả lời	packet = new DatagramPacket(buf, buf.length); socket.receive(packet);
4	Lấy dữ liệu trong gói	String received = new String(packet.getData()).trim();
5	Đóng socket	socket.close();

*** Chương trình client mẫu**

```
import java.io.*;
import java.net.*;
public class Client {
   public static void main(String[] args) throws IOException {
      DatagramSocket socket = new DatagramSocket();
      byte[] buf = new byte[256];
      InetAddress address = InetAddress.getByName(NameServer);
      DatagramPacket packet = new DatagramPacket(buf, buf.length, address,1234);
      socket.send(packet);
```

```
        packet = new DatagramPacket(buf, buf.length);
        socket.receive(packet);
        String received = new String(packet.getData()).trim();
         //các lệnh xử lý dữ liệu
        socket.close();
    }
}
```

3. Cấu trúc chương trình server

STT	Thao tác	Sử dụng lệnh
1	Tạo một datagram socket dùng để liên lạc với máy client	DatagramSocket socket = new DatagramSocket(1234);

2	Tạo gói nhận và nhận gói do client gởi	byte[] buf = new byte[256];
		packet = new DatagramPacket(buf, buf.length);
		socket.receive(packet);

3	Lấy dữ liệu trong gói	String received = new String(packet.getData()).trim();

4	Tạo gói gởi và gởi gói trả lời cho client	String sendStr =”chuoi dl goi”;
		buf = sendStr.getBytes();
		InetAddress address = packet.getAddress();
		int port = packet.getPort();
		packet = new DatagramPacket(buf, buf.length, address, port);
		socket.send(packet);

5	Đóng socket	socket.close();

Ct server không cần tạo tiểu trình vì không có kết nối nào cần duy trì giữa client và server. Để phục vụ nhiều client, chương trình server chỉ cần dùng một vòng lặp lần lượt nhận các gói của các client và trả lời. Nếu công việc trả lời thực hiện tốn nhiều thời gian thì khi đó nên tạo tiểu trình thực hiện công việc trả lời.

* Chương trình server mẫu

```
import java.io.*;
import java.net.*;
public class Server {
   public static void main(String[] args) throws IOException {
      DatagramSocket socket = new DatagramSocket(1234);
      while (true) {
            byte[] buf = new byte[256];
            DatagramPacket packet = new DatagramPacket(buf, buf.length);
            socket.receive(packet);
            String received = new String(packet.getData()).trim();
               //xử lý dl nhận
            String sendStr =”chuoi dl goi”;
            buf = sendStr.getBytes();
            InetAddress address = packet.getAddress();
            int port = packet.getPort();
            packet = new DatagramPacket(buf, buf.length, address, port);
            socket.send(packet);
      }
      socket.close();
   }
```

- Hết -

BÀI TẬP CHƯƠNG IV
LẬP TRÌNH CLIENT-SERVER

*** Bài 1:**
Viết chương trình giải phương thức bậc nhất, dạng client-server dùng giao thức TCP, đáp ứng yêu cầu một client. Cải tiến chương trình để chương trình có thể đáp ứng yêu cầu của nhiều client đồng thời và client có thể yêu cầu server nhiều lần.

*** Bài 2:**
Tương tự bài 1 nhưng client viết ở dạng Applet.

*** Bài 3:**
Viết chương trình download, upload dạng client-server dùng giao thức TCP. Server đáp ứng yêu cầu download hay upload của nhiều client đồng thời.

*** Bài 4:**
Viết chương trình quản lý sinh viên có các chức năng sau: nhập mã sv, họ tên, đlt, đth, tính đtb và ghi thông tin này vào CSDL SQL SERVER; tìm/xóa/sửa thông tin sinh viên khi biết masv; xem danh sách sinh viên. Chương trình viết dạng client-server, dùng giao thức TCP, server có thể đáp ứng yêu cầu của nhiều client đồng thời và client có thể yêu cầu server nhiều lần. Hãy viết chương trình bằng hai cách: không có giao diện và có giao diện

*** Bài 5:**
Viết chương trình chat dạng client-server, dùng giao thức TCP. Server nhận một thông điệp từ một client, server sẽ gởi thông điệp này cho các client khác.

*** Bài 6:**
Viết chương trình hỏi thời tiết dạng client-server, dùng giao thức UDP. Biết rằng thông tin về thời tiết lưu trong tập tin thoitiet.txt dạng sau:

Tinh/tp	nhietdo	doam
Hanoi	20-30	10%
Hcm	30-40	5%
vv…		

*** Bài 7:**
Viết chương trình hỏi tỉ giá, biết rằng thông tin tỉ giá lưu trong CSDL SQL SERVER.

*** Bài 8:**
Viết hàm ping dùng giao thức UDP.

- Hết -

CHƯƠNG V
LẬP TRÌNH WEB

I. Khái niệm

1.1 Lập trình Web

- Khi lập trình mạng dạng client/server, nếu không có nhu cầu đặc biệt, thường ta không phải viết ct client và ct server mà sử dụng ct client có sẵn là Browser (web client), server có sẵn là Web Server, kỹ thuật này gọi là lập trình web.

- Người lập trình web viết những trang web bằng ngôn ngữ HTML, các trang web này để ở Web server. Web server và Web client liên lạc với nhau theo giao thức HTTP. Khi Web client gởi yêu cầu trang web tới Web Server tại cổng 80 (là cổng mặc định của Web Server), Web Server tìm trang web và trả về trang web cho Web Client. Web client sẽ thông dịch trang Web và hiện trên màn hình client.

- Các Web Server thông dụng như IIS (Internet Information Service), PWS (PersonalWeb Server), JSP (Java Web Server), Jrun, Tomcat,... Các Web Client thông dụng như là Netscape Navigator, Internet Explore,...

- Client được server trả về cho những trang web đã có sẵn trên máy server, trang web này có nội dung không thay đổi , không thể tương tác với người dùng, không có chức năng xử lý, không thể truy vấn CSDL,... gọi là trang web tĩnh (đây là hạn chế của ngôn ngữ HTML). Trang web động là trang web chưa có sãn trên server mà do một chương trình khác chạy trên server sinh ra khi nhận được yêu cầu từ client, trang web động sẽ được gởi về cho client. Trang web động có thể thay đổi nội dung tùy theo các tham số do client gởi hoặc tuỳ theo nội dung CSDL,...

- Các kỹ thuật lập trình web động thông dụng là CGI, ASP, APPLET, SERVLET, JSP, J2EE, EJB,...

1.2 Giao thức HTTP

- Giao thức mà Web Client dùng để yêu cầu trang Web từ Web Server. Mỗi khi yêu cầu được đáp ứng thì kết nối sẽ kết thúc (kết nối phi trạng thái).

- Giao thức có hai lệnh cơ bản là GET, POST. Khi NSD nhập URLvào hộp address của Browser thì Browser dùng lệnh GET để liên lạc với web server. Nếu trang web dùng thẻ lệnh <FORM METHOD=POST/GET> thì web client dùng lệnh POST/GET để gởi dữ liệu trong form cho web server. Method có thể là GET khi đó dữ liệu trong form sẽ hiện trên URL của client.

1.3 Lập trình CGI (Common Gateway Interface)

- Khi nhận yêu cầu từ client, server gọi một chương trình CGI xử lý yêu cầu này. Ct CGI sinh ra trang web và gởi về cho client. Ct CGI là một ct thực thi được (.exe, .com) có thể viết bằng C, Pascal,.. dùng để sinh ra trang web động.

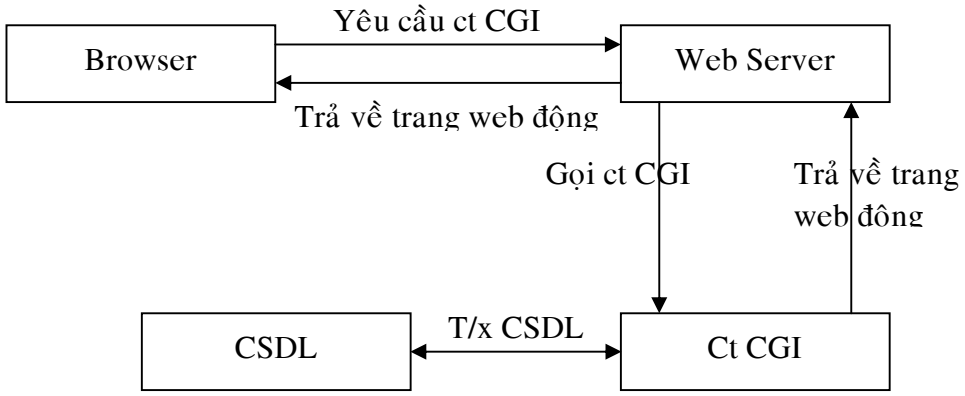
- **Cơ chế hoạt động của ct CGI**: Mỗi khi client yêu cầu ct CGI, server sẽ thực hiện các bước sau:

  . Nạp ct CGI vào bộ nhớ

  . Thực thi ct CGI, ct CGI tạo ra trang web động.

  . Trả kết quả là trang web động về cho client

  . Thu hồi bộ nhớ đã cấp phát cho ct CGI.



Nhận xét: kỹ thuật CGI thường chậm, tốn tài nguyên vì mỗi lần nhận được yêu cầu, server phải nạp ct CGI, sau đó phải giải phóng.

Ví dụ:

-viết file hello.cpp

```
#include <stdio.h>
void main()
{
 printf("<html>");
 printf("<h1> Hello World!</h1>");
 printf("</html>);
}
```

dịch ra hello.exe, chép vào web root của web server

- ở client gõ: http://localhost/hello.exe

II. Lập trình web bằng SERVLET

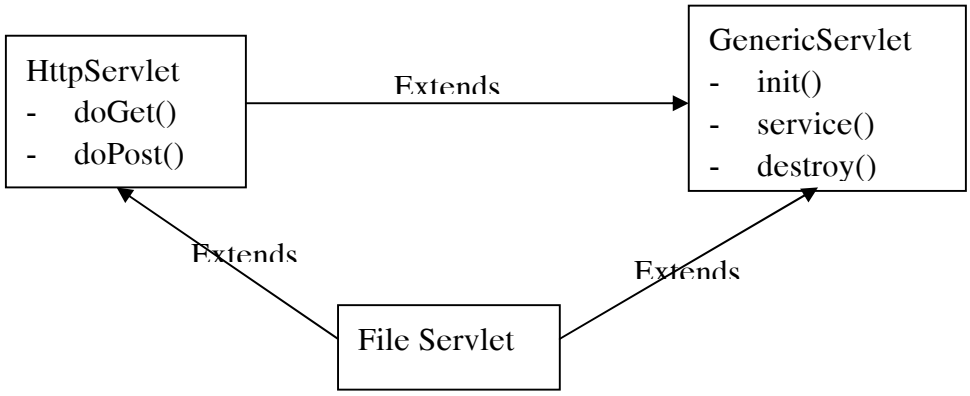
2.1 Khái niệm

- **Servlet là gì?**

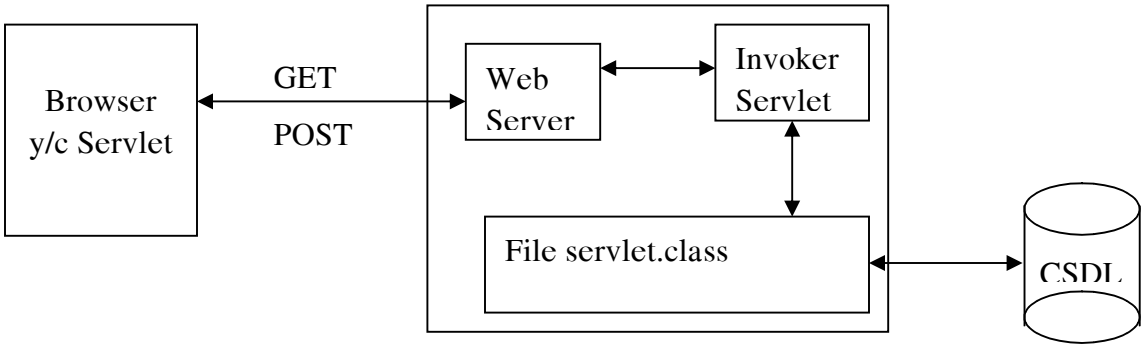
Servlet là file Java, thừa kế lớp ***javax.servlet.http.HttpServlet*** hoặc lớp ***javax.servlet.GenericServlet*** dùng để sinh ra trang web động. Khi client yêu cầu một servlet, bằng lệnh GET hoặc POST, web server nhận yêu cầu và gọi ct Invoker Servlet thực thi file sevlet và sẽ sinh ra một trang web động gởi về cho client.

Invoker Servlet là chương trình được nhúng sẵn vào các web server hiểu Java (JRUN, Tomcat, ...) dùng để thực thi file sevlet. Một số web server như là IIS, PWS không có trình Invoker Servlet nên không thể thực thi file servlet.

- Sơ đồ thừa kế



- **Mô hình hoạt động của servlet**



- **So sánh CGI, Servlet**
- **Chu kỳ sống của servlet:** Nạp servlet , khởi tạo servlet , thực thi servlet, dọn dẹp servlet
- **So sánh kỹ thuật servlet và applet.**

KỸ THUẬT APPLET	**KỸ THUẬT SERVLET**
Là lớp java thừa kế lớp Applet hoặc Japplet, có các phương thức đặc biệt như là: init, start, stop, run, destroy,... các phương thức này được thực thi bởi browser	Là lớp java thừa kế lớp HttpServlet hoặc GenericServlet, có các phương thức đặc biệt như là init, service, doGet, doPost,..., các phương thức này được Invoker Servlet thực thi.
Applet được download về máy client và thực thi nhờ trình duyệt. Lập trình Applet còn gọi là lập trình phía client)	File servlet thực thi trên máy server, tạo trang web động, trả về cho client (Lập trình servlet còn gọi là lập trình phía server)
Applet thường chậm, máy client phải có cấu hình cao vì phải download, và việc thực thi ở phía client	Servlet thực thi nhanh hơn, máy client không cần có cấu hình cao vì việc thực thi ở phía server.
Phụ thuộc vào Browser có hỗ trợ Java	Phụ thuộc vào Web Server có hỗ trợ Java, nhưng không phụ thuộc vào Browser.

2.2 Các phương thức đặc biệt

- ***public void init (ServletConfig config) throws ServletException***

{

/*các lệnh khởi tạo, lệnh chỉ thực hiện một lần như là: gán trị ban đầu cho biến, kết nối server, kết nối CSDL, ...*/

}

Phương thức này được Invoker Servlet gọi khi thực thi servlet lần đầu tiên.

- ***public void destroy (ServletConfig config) throws ServletException***

{

/*các lệnh dọn dẹp như là: đóng kết nối server, đóng kết nối CSDL,... */

}

Phương thức này được Invoker Servlet gọi khi servlet hết thời gian qui định lưu trong bộ nhớ. Thông thường servlet khi được nạp vào bộ nhớ để thực thi, thực thi xong vẫn lưu trong bộ nhớ trong một khoảng thời gian do web server qui định, nếu các client yêu cầu servlet lần nữa thì Invoker Servlet không phải nạp servelt vào bộ nhớ nữa mà chỉ việc thực thi servlet, tốc độ sẽ nhanh hơn nhiều và đây cũng là ưu điểm của servlet so với CGI.

- ***public void service (ServletRequest req, ServletResponse res) throws ServletException, IOException***

{

/*Xử lý yêu cầu GET và yêu cầu POST: Lấy dl do client gởi, xử lý dl, truy xuất CSDL, gởi trang web động về cho client */

}

Phương thức này được Invoker Servlet gọi khi client gởi yêu cầu bằng lệnh POST hoặc GET. req là đối tượng tiếp nhận dl do client gởi, res là đối tượng chứa dl trả về cho client.

- ***public void doGet (HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException***

{

/*Xử lý yêu cầu GET: lấy dl do client gởi, xử lý dl, truy xuất CSDL, gởi trang web động về cho client */

}

Phương thức này được Invoker Servlet gọi khi client gởi yêu cầu bằng lệnh GET. req là đối tượng tiếp nhận dl do client gởi, res là đối tượng chứa dl trả về cho client.

- ***public void doPost(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException***

{

/*Xử lý yêu cầu POST: lấy dl do client gởi, xử lý dl, truy xuất CSDL, gởi trang web động về cho client */

}

Phương thức này được Invoker Servlet gọi khi client gởi yêu cầu bằng lệnh POST. req là đối tượng tiếp nhận dl do client gởi, res là đối tượng chứa dl trả về cho client.

Lưu ý:

- Do doPost(), doGet() là các phương thức trong lớp HttpServlet nên muốn sử dụng các phương thức này, servlet phải thừa kế lớp HttpServlet.
- Nếu client gởi yêu cầu bằng lệnh GET hoặc POST thì servlet phải có phương thức tương ứng là doGET hoặc doPOST hoặc service để xử lý yêu cầu, nếu không sẽ báo lỗi.

2.3 Dạng file servlet

- **Dạng 1: thừa kế lớp GenericServlet**

```
import javax.servlet.*;

import java.io.*;

public class File_Name_Servlet extends GenericServlet

{

    public void service (ServletRequest req, ServletResponse res) throws ServletException, IOException

    {

        res.setContentType("text/html");// đặt kiểu dl trả về là text hoặc html

        PrintWriter out=res.getWriter();// tạo luồng xuất, dùng luồng này để xuất dl về cho client

        //xử lý lệnh GET, POST

    }

}
```

- **Dạng 2: thừa kế lớp HttpServlet**

```
import javax.servlet.*;

import javax.servlet.http.*;

import java.io.*;

public class File_Name_Servlet extends HttpServlet

{

    public void doPost (ServletRequest req, ServletResponse res) throws ServletException, IOException

    {

        res.setContentType("text/html");// đặt kiểu dl trả về là text hoặc html

        PrintWriter out=res.getWriter();// tạo luồng xuất, dùng luồng này để xuất dl về cho client

        //xử lý lệnh POST hoặc gọi doGET(req,res)

    }

    public void doGet (ServletRequest req, ServletResponse res) throws ServletException, IOException

    {

        res.setContentType("text/html");// đặt kiểu dl trả về là text hoặc html

        PrintWriter out=res.getWriter();// tạo luồng xuất, dùng luồng này để xuất dl về cho client

        //xử lý lệnh GET hoặc gọi doPost(req,res)

    }

}
```

2.4 Sử dụng servlet trong Jrun: Tạo file servlet, dịch ra .class

- **Cách 1:**

- Chép file servlet.class vào thư mục [JRUN_HOME]\servers\default\default-app\web-inf\classes.

(JRUN_HOME là thư mục cài Jrun)

- Ở client để gọi file servlet.class gõ URL

  http://< ServerName>:8100/<ServletName> (không có .class)

Hoặc gọi servlet.class thông qua một trang web khác: tạo trang web, trong trang web dùng thẻ

<form action = ServletName method=post/get>

...

<input type=submit value=submit>

</form>

rồi ở client thay vì gọi trực tiếp file servlet, ta gọi trang web có thẻ lệnh <form action = ServletName>, khi click nút submit sẽ gọi file servlet

- **Cách 2:**

- Chép file servlet.class vào thư mục [JRUN_HOME]\servlets.

- Ở client để gọi file servlet.class gõ

  http://< ServerName>:8100/servlet/<ServletName> (không có .class)

Hoặc dùng trang web trung gian có thẻ lệnh form

<form action = servlet/ServletName method=post/get>

- Chú ý: các file .class thông thường (không phải là servlet) chép vào tm [JRUN_HOME]\servers\default\default-app\web-inf\classes

2.5 Gởi/lấy tham số

- **Gởi tham số cho servlet: có hai cách**

- Gởi trực tiếp qua URL:

  URL?< tênthamsố1>=<giátrị1>&< tênthamsố2>=<giátrị2> [...]

- Gởi qua trung gian trang web có sử dụng thẻ form và nút submit

- **Lấy giá trị tham số**: nếu giá trị là null thì tham số không được gởi hoặc lấy sai tên ts

- **Lấy tham số có một giá trị**

```
String value= req.getParameter("tênthamsố");
```

- **Lấy tham số có nhiều giá trị**

```
String values[]= req.getParameterValues("tênthamsố");
```

- **Lấy tất cả các tham số**

```
Enumeration names[]= req.getParameterNames(); //lấy tất cả các tên tham số
While (names.hasMoreElements()) // trong khi còn phần tử (còn tên)
{
    String name= (String) names.nextElement();// lấy tên một tham số
    String value= req.getParameter(name);// lấy giá trị của tham số đó
    // xử lý giá trị này
}
```

2.6 Các ví dụ

- VD1: Xem giờ trên web
- VD2: Tính diện tích hình tròn
- VD3: Thi trắc nghiệm có nhiều dạng

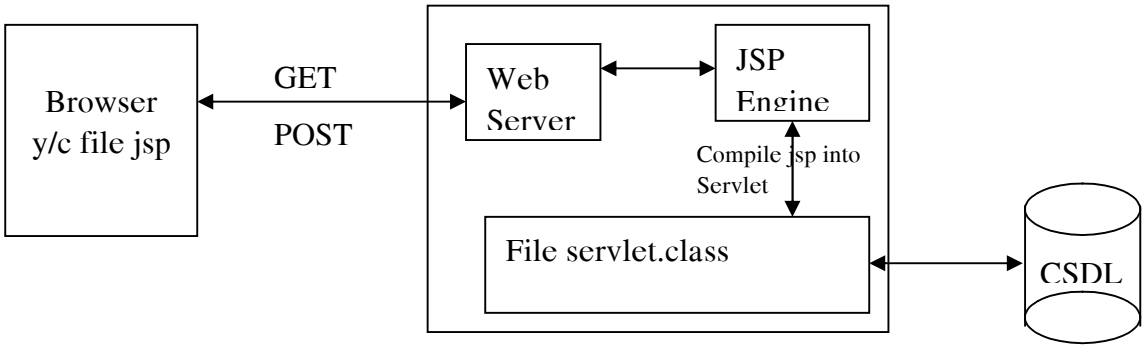
III. Lập trình web bằng JSP

3.1 Khái niệm

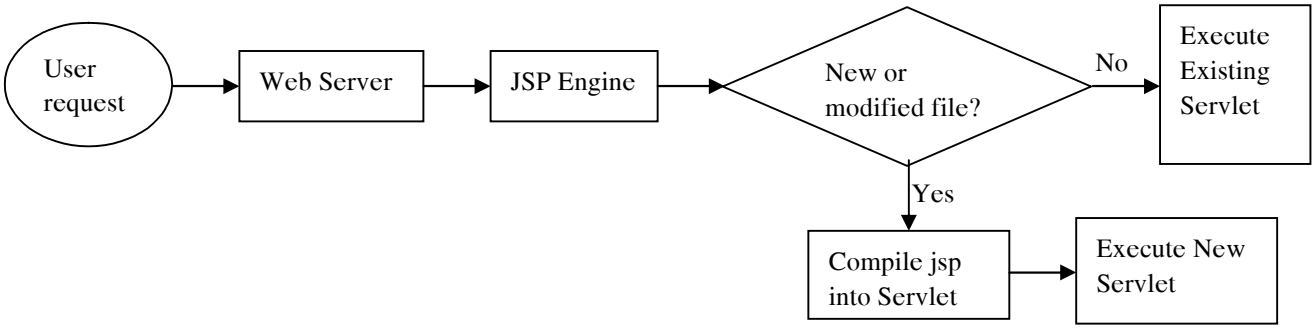
- **File jsp là gì?**

File Jsp là file văn bản mà tên có phần mở rộng là jsp. Trong file trộn lẫn các lệnh java và lệnh html. Khi client yêu cầu trang jsp, web server gọi JSP Engine, JSP Engine biên dịch file jsp thành file servlet và nạp servlet vào bộ nhớ cho thực thi, sinh ra trang web động và trả về cho client. Nếu client yêu cầu trang jsp đã được biên dịch thành servlet.class thì JSP Engine sẽ không biên dịch nữa nên file jsp chỉ thực hiện lần đầu tiên là chậm, những lần sau sẽ nhanh hơn.

- **Mô hình hoạt động của công nghệ JSP**



- Lưu đồ hoạt động của mô hình JSP



3.2 Các đối tượng do web server tạo sẵn

- out: là đối tượng dùng để gởi dữ liệu cho client (browser)
- request: là đối tượng dùng để lấy dữ liệu do client gởi
- response:là đối tượng dùng goi dữ liệu cho client
- session: là đối tượng dùng để lưu các biến session. Biến session có hiệu lực trong một phiên làm việc.
- application: là đối tượng dùng để lưu các biến application. Biến application có hiệu lực trong tất cả các phiên làm việc.

3.3 Các thẻ lệnh của JSP

- **Chú thích:**

  <%-- chú thích các lệnh html, java --%>

  Chú thích java dùng // hoặc /* */

- **Thông báo khối lệnh Java**

  <% các lệnh Java %>

- **Hiện giá trị biến (không có dấu chấm phẩy sau tên biến)**

  <%=tên biến%>

  giống lệnh out.print(tên_biến);

- **Đn hàm hoặc khai báo biến**

  <%! Đn hàm hoặc khai báo biến có phạm vi ứng dụng %>

  Trong các hàm muốn sử dụng đối tượng out, cần phải gởi như tham số.

- **Nhúng file (file nhúng thường là html hoặc jsp hoặc class)**

  a) Nhúng nội dung

  <%@ include file="tenfile"%>

b) Nhúng kết qủa

<jsp:include page=”tenfile” />

hoặc

<jsp:include page=”tenfile” >

<jsp:param name=”tenbien” value=”giatri”/>

</jsp:include>

c) Nhúng lớp

<%@ page import="tengoi.tenlop"%>

- **Chuyển trang**

a) Tự động chuyển tham số

<jsp:forward page=”tenfile”/>

b) Không tự chuyển tham số, để chuyển tham số dùng URL

response.sendRedirect(“tenfile?tents=gt&tents=gt”);

3.4 Biến session, cookies, application

Ngay sau khi yêu cầu trang web của client được đáp ứng, kết nối http giữa client và server sẽ bị đóng, nên giao thức http còn gọi là giao thức kết nối phi trạng thái. Do kết nối bị đóng nên lần kết nối sau để yêu cầu một trang web khác thì thông tin liên quan đến kết nối trước, kết nối sau sẽ không thể biết được. Để khắc phục khuyết điểm này Java cung cấp ba loại biến là: session, cookies, application.

3.5 Session

- **Phiên làm việc (session):** là khoảng thời gian được tính từ thời điểm mở trình duyệt liên lạc với web server đến khi đóng trình duyệt. Nếu mở trình duyệt mới là qua phiên làm việc khác

- **Đối tượng session:** Mỗi phiên làm việc, web server tạo ra một đối tượng session dùng để lưu những biến session. Đối tượng session chỉ tồn tại ở máy server trong một phiên làm việc hoặc trong khoảng thời gian qui định bởi web server , thường web server có option để thay đổi thời gian này.

- **Biến session:** là biến lưu trong đối tượng session nên biến session cũng tồn tại trong một phiên làm việc, mọi trang cùng phiên làm việc đều truy cập được biến session.

- **Các phương thức:** Đối tượng sesion có các phương thức thông dụng sau:

a) public void setAttribute(String name, Object value)

Đặt biến sesion tên là name có giá trị là value vào trong đối tượng session

b) public Object getAttribute(String name)

Lấy giá trị biến sesion trong đối tượng session có tên là name. Trả về null nếu không tìm thấy biến này. Các giá trị lưu trữ hay lấy ra có kiểu là Object.

c) public void removeAttribute(String name)

Hủy biến sesion tên là name trong đối tượng session

d) public void invalidate()

Hủy đối tượng session, khi đó tất cả các biến sesion trong đối tượng session cũng bị hủy.

3.6 Application

- **Đối tượng application :** web server tạo ra một đối tượng application dùng để lưu những biến application. Đối tượng application tồn tại ở máy server trong khoảng thời gian qui định bởi web server , thường web server có option để thay đổi thời gian này, nếu hết thời gian qui định sẽ bị hủy và tạo mới. Nếu khởi động lại web server thì đối tượng application cũng sẽ bị hủy và sẽ tạo mới.

- **Biến application:** là biến trong đối tượng application, mọi trang cùng phiên làm việc hoặc khác phiên đều truy cập được biến application.

- **Các phương thức:** Đối tượng application cũng có các phương thức giống như đối tượng session.

3.7 Cookies

- **Biến cookies**

Là biến do người dùng tạo ra và lưu ở máy client trong khi đó biến session, application lưu ở server. Các biến cookie mặc định được browser gởi cho web server khi gởi yêu cầu trang web. Biến cookie sẽ bị hủy khi đóng trình duyệt hoặc hết thời gian sống được qui định vào lúc tạo biến cookie.

- **Các phương thức:**

a) Tạo biến cookie

Cookie c= new Cookie("tenbien",giatri);

b) Đặt thời gian sống cho biến cookie

c.setMaxAge(time); time tính bằng giây, time=0 là hủy cookie

c) Ghi biến cookie vào client

response.addCookie(c)

d) Lấy tất cả các biến cookie

Cookie[] cs=request.getCookies();

e) Lấy tên biến cookie thứ i

cs[i].getName();

f) Lấy giá trị biến cookie thứ i

cs[i].getValue();

3.8 Truyền dữ liệu qua nhiều trang

- Dùng biến ẩn
- Dùng URL
- Dùng session, application, cookie

IV. Java bean

- **Java bean**

Là tập hợp của một hay nhiều lớp java dùng giao diện Serializable , bean độc lập nền và có thể dùng lại được. Trong file bean phải có các thuộc tính, phương thức set thuộctính, get thuộctính.. Bean có thể dùng trong application, applet, jsp,servlet. Trong file jsp, Bean sẽ che dấu và để truy xuất đối tượng bean dùng các thẻ lệnh của JSP.

- **Các thẻ lệnh jsp để truy xuất bean**

a) Khai báo

<jsp:useBean id="tênđtbean" scope="page/session/application" class="tênlớpbean"/>

b) Gán thuộc tính cho bean

<jsp:setProperty name="tênđtbean" property="tênthuộctính" value="giátrị"/>

c) Trả về giá trị thuộc tính của bean dạng chuỗi

<jsp:getProperty name="tênđtbean" property="tênthuộctính"/>

d) Đóng khai báo bean: nếu không dùng các thẻ setProperty, getProperty thì không cần đóng

</jsp:useBean>

- **Nhận xét**

Để đặt thuộc tính, lấy thuộc tính có thể dùng thẻ hoặc tạo đối tượng bean và gọi trực tiếp.

BÀI TẬP

* SERVLET

1. Viết servlet xem cửu chương, nhập số hiệu cưu chương và hiện cửu chương tương ứng.

2. Viết servlet giải pt bn.

3. Viết ct thi trắc nghiệm có nhiều loại chọn

- Hết -

CHƯƠNG 1
LẬP TRÌNH ĐỐI TƯỢNG PHÂN TÁN
(DISTRIBUTED OBJECT PROGRAMMING)

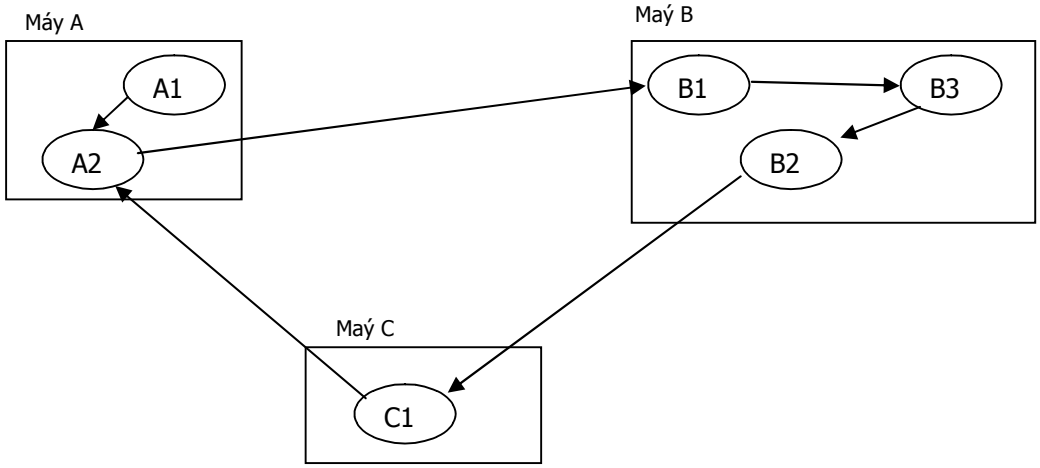
I. TỔNG QUAN

Công nghệ lập trình đối tượng phân tán là công nghệ cho phép các đối tượng viết bằng những ngôn ngữ khác nhau, chạy trên các máy khác nhau, có thể gọi các phương thức của nhau. Thông thường ứng dụng phân tán có dạng client-server, chương trình server tạo những đối tượng có thể được gọi từ xa (remote objects), chương trình client sẽ triệu gọi phương thức của đối tượng trên máy server.

* Ưu điểm của lập trình phân tán

- Các chương trình đã viết sẵn bằng những ngôn ngữ khác nhau không cần phải viết lại mà vẫn giao tiếp được với nhau.
- Các đối tượng trên các máy khác nhau, do các chương trình khác nhau sinh ra, giao tiếp được với nhau như trên cùng một máy.
- Một ứng dụng lớn có thể được xử lý phân tán ở nhiều máy khác nhau trên mạng nên công nghệ đối tượng phân tán còn gọi là công nghệ xử lý phân tán (Distributed Computing).
- Công nghệ này đặc biệt hữu ích khi một công việc chỉ có thể thực thi trên một máy chuyên dụng. Ví dụ client có một công việc mà công việc này chỉ có thể thực hiện ở một máy server chuyên dụng, client có thể gởi công việc (gởi đối tượng) cho một phương thức của đối tượng trên server thực hiện sau đó trả kết quả về cho client.

* Mô hình lập trình phân tán



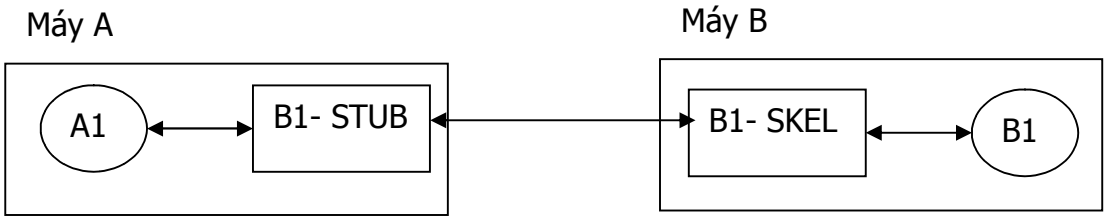
*** Các vấn đề phát sinh khi gọi đối tượng từ xa**

- Các đối tượng trên hai máy khác nhau hoạt động trên hai không gian địa chỉ khác nhau nên việc tham chiếu biến, địa chỉ của đối tượng trên mỗi máy là khác nhau. Ví dụ khi gởi một biến con trỏ (biến chứa địa chỉ) cho phương thức của đối tượng ở xa thì có thể địa chỉ này không tồn tại ở máy chứa đối tượng.
- Lời gọi phương thức ở xa phải thông qua kết nối mạng nên có thể không thực hiện hoặc nhận được kết quả sai khi mạng hay server có vấn đề.

*** Các lớp trung gian**

Để giải quyết hai vấn đề trên các đối tượng trên hai máy khác nhau không gọi trực tiếp lẫn nhau mà thông qua hai lớp trung gian là lớp STUB (lớp móc) và lớp SKEL (lớp nối)

*** Mô hình lập trình phân tán thông qua lớp trung gian Stub, Skel**



Để A1 có thể gọi phương thức của B1, B1 cần cung cấp hai lớp B1-STUB và B1-SKEL. B1-STUB cài đặt ở máy A (máy client), B1-SKEL cài đặt ở máy B (máy server).

Khi A1 gọi B1, lời gọi sẽ chuyển tới B1-STUB. B1-Stub đóng gói lời gọi cùng tham số, chuyển gói đến B1-SKEL. B1-SKEL lấy tham số trong gói lưu vào vùng địa chỉ phù hợp của máy B, sau đó gọi phương thức của B1 cùng với tham số đã nhận. Kết quả nếu có sẽ được B1-SKEL đóng gói và trả về cho B1-STUB, B1-STUB lấy kết qủa trong gói và chuyển kết quả cho A1. Nếu mạng gặp sự cố thì B1-STUB và B1-SKEL sẽ gởi lại gói hoặc báo lỗi.

II. CÔNG NGHỆ RMI (Remote Method Invocation)

*** Giới thiệu**

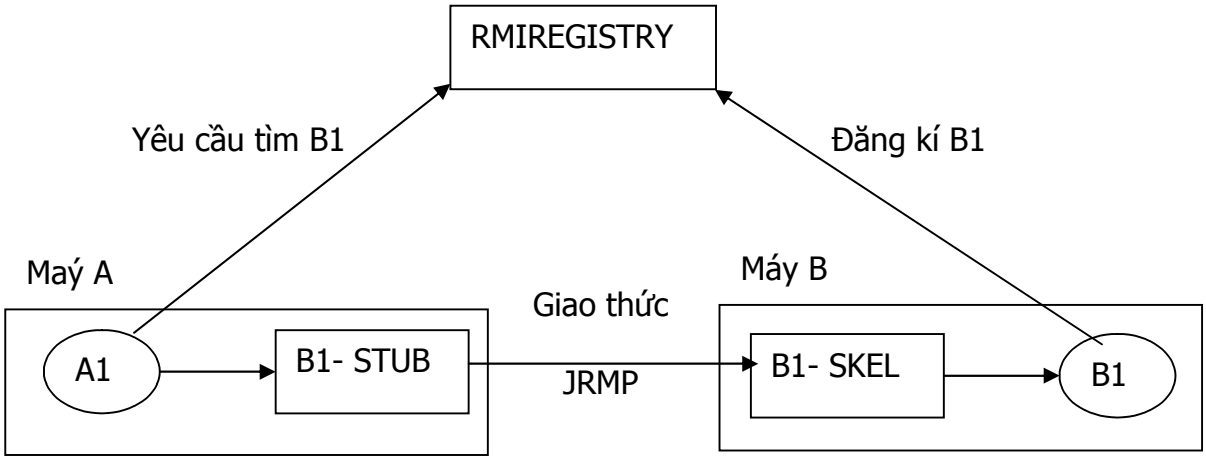
Công nghệ RMI là công nghệ lập trình đối tượng phân tán, trong đó các đối tượng viết bằng cùng ngôn ngữ Java, liên lạc với nhau theo giao thức Java Remote Method Protocol (JRMP).

- Ct sever tạo các đối tượng có thể được gọi từ xa (đối tượng remote), đăng kí các đối tượng này với trình quản lý đối tượng (rmiregistry), sau đó đợi client gọi những pt trên các đối tượng remote.
- Ct client yêu cầu rmiregistry tìm đối tượng remote, rmiregistry trả về tham chiếu tới đối tượng remote, client dùng tham chiếu này gọi pt của đt remote.

RMI cung cấp môi trường để client và server liên lạc với nhau, ngoài ra RMI còn cho phép client gởi một đt cho đt remote, điều này làm mở rộng khả năng của ứng dụng ở xa.

*** Mô hình RMI:**

Server đăng kí đt remote với rmiregistry. Client yêu cầu rmiregistry tìm đt remote, nếu tìm thấy sẽ gọi pt của đt remote.



*** Đối tượng remote**

Một đt trở thành đt remote bằng cách dùng giao diện remote. Giao diện remote có những đặc tính sau:

. Khai báo public

. Thừa kế giao diện java.rmi.Remote

. Mỗi pt của giao diện khai báo throws java.rmi.RemoteException

*** Cài đặt ứng dụng RMI**

. Định nghĩa giao diện remote: giao diện remote khai báo các pt mà có thể gọi từ xa

. Xây dựng lớp cài đặt đối tượng remote, lớp này sử dụng giao diện remote

. Xây dựng lớp server

. Xây dựng lớp client

. Biên dịch mã nguồn và sinh ra lớp stub, skel : dùng javac và rmic

. Khởi động ứng dụng: chạy rmiregistry, server, client

Lưu ý:

- Máy server cần có 5 lớp sau: giao diện remote, lớp cài đặt, lớp Stub, lớp Skel, lớp server. Nếu dùng JDK 1.3 trở lên thì lớp Skel được tích hợp trong lớp cài đặt.
- Máy client cần có 3 lớp sau: giao diện remote, lớp Stub, lớp client. Lớp giao tiếp rất nhỏ nên hầu như khi sử dụng sẽ không tốn tài nguyên của client. Ngoài ra Java cũng cung cấp cơ chế để tự động nạp Lớp Stub xuống máy client.

*** Truyền tham số**

Khi gọi phương thức ở xa, biến kiểu cơ bản như int, float ,… được truyền theo tham trị. Biến kiểu đối tượng muốn truyền qua mạng phải dùng giao diện Remote (truyền theo tham chiếu) hay Serializable (truyền theo tham trị).

III. CÔNG NGHỆ CORBA

1. Khái niệm:

- CORBA (Common Object Request Brokerage Architecture): Công nghệ lập trình đối tượng phân tán do tổ chức OMG (Object Management Group) đề xuất.
- IDL(Interface Definition Language), là ngôn ngữ đặc tả giao diện, dùng IDL để có thể cài đặt client/server bằng các ngôn ngữ khác nhau có hỗ trợ CORBA như là C, C++, Smalltalk, COBOL, Ada, Java, mỗi ngôn ngữ hỗ trợ CORBA đều có trình chuyển đổi IDL sang ngôn ngữ đó. IDL không có lệnh, chỉ có các mô tả hàm, kiểu dữ liệu, các khai báo để đặc tả đối tượng. Các giao diện đầu tiên được đặc tả bằng ngôn ngữ IDL, sau đó dùng trình biên dịch tương ứng để dịch sang ngôn ngữ cụ thể, đồng thời tạo lớp stub, skel và một số lớp hỗ trợ CORBA.

  ví dụ:

  idlj <tengiaodien.idl>: dịch sang Java

  idl2cpp <tengiaodien.idl>: dịch sang C++

  idl2pas <tengiaodien.idl>: dịch sang Pascal

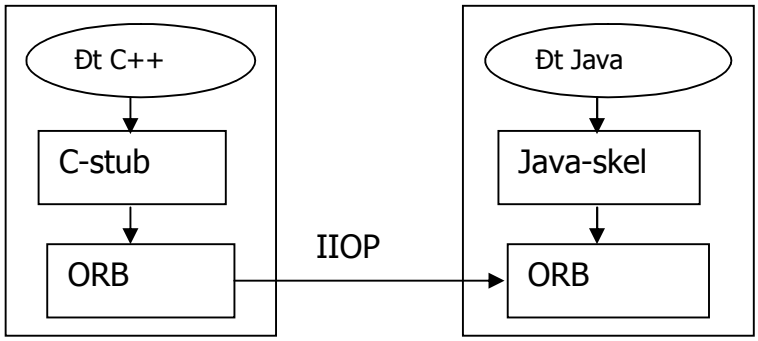
- ORB(Object Request Broker): Các đối tượng viết bằng những ngôn ngữ khác nhau muốn gọi lẫn nhau phải thông qua chương trình môi giới ORB. Trong Jdk 1.3 có sẵn ORB, trong Jbuilder có ORB tên là VisiBroker và OrbixWeb. IIOP(the Internet Inter-ORB Protocol) là giao thức liên lạc giữa các ORB, ORB có thể cung cấp một số dịch vụ khác như là dịch vụ tìm kiếm đối tượng theo tên, dịch vụ duy trì đối tượng lân dài,…

2. Cơ chế hoạt động của CORBA

Khi client gọi phương thức của đối tượng trên server, lời gọi được chuyển qua lớp Stub, rồi qua ORB trên client. ORB ở client kết nối với ORB ở server theo giao thức IIOP. ORB trên server chuyển lời gọi cho Skel, rồi phương thức của đối tượng trên server được gọi. Kết quả nếu có sẽ được đối tượng trên server trả lại qua Skel, ORB trên server, ORB trên client, rồi đến đối tượng trên client

Ví dụ:

Giả sử muốn cài đặt đối tượng ở client bằng C++, đối tượng ở server là Java: ở client dùng idl2cpp dịch giao diện sang C++, đồng thời tạo lớp Stub, sau đó viết đt client bằng C++. Ở server dùng idlj dịch giao diện sang Java, đồng thời tạo lớp Skeleton, sau đó viết đt server bằng Java.



3. So sánh RMI, CORBA

Công nghệ RMI	**Công nghệ CORBA**
Các đt viết bằng Java	Các đt có thể viết bằng ngôn ngữ khác nhau
Không phụ thuộc hđh, phần cứng nơi đt thực thi	Phụ thuộc hđh, phần cứng nơi đt thực thi
Không cần biết nn IDL, không cần trình môi giới ORB	Cần biết nn IDL, cần có trình môi giới ORB
Cho phép gởi đt theo tham chiếu hoặc tham trị	Chỉ cho phép gởi đt theo tham chiếu

4. Viết ứng dụng CORBA

- Định nghĩa giao diện remote: dùng ngôn ngữ IDL viết giao diện remote.
- Biên dịch giao diện remote sang ngôn ngữ mong muốn: ví dụ dùng trình biên dịch idlj.exe để biên dịch giao diện remote từ IDL sang Java và sinh ra stub, skel cùng với mã dùng để kết nối với ORB
- Cài đặt Server
- Cài đặt Client
- Thực thi ứng dụng

Ví dụ: Viết ứng dụng hiện câu chào "Hello World" dùng ORB của Jdk 1.3

B1: Viết lớp giao diện bằng IDL (Hello.idl). file này để ở trong thư mục src

```
module HelloCorba
{
        interface Hello
        {
                string sayHello();
        };
};
```

B2: Trên client dịch Hello.idl sang Java:

idlj –fclient Hello.idl

Sẽ tạo ra 5 file sau: Hello.java, HelloHelper.java, HelloHolder.java, _HelloStub.java, HelloOperations.java. Có thể

B3: Trên server dịch Hello.idl sang Java:

idlj –fserver Hello.idl

Sẽ tạo ra 3 file sau: HelloOperations.java (Hello.idl dạng Java), Hello.java, _HelloImpBase.java.

Trong vd này B2, B3 có thể thay bằng một lệnh: idlj –fall Hello.idl và sẽ sinh ra 6 file: Hello.java, HelloHelper.java, HelloHolder.java, _HelloStub.java, HelloOperations.java, _HelloImpBase.java.

- HelloOperations.java: chính là bản dịch sang Java của giao diện Hello.idl
- Hello.java: thừa kế HelloOperations, org.omg.CORBA.Object, org.omg.CORBA.portable.IDLEntity để trở thành đt CORBA
- _HelloStub.java (lớp stub): cung cấp chức năng CORBA cho client, dùng giao diện Hello.java
- _HelloImpBase.java: dùng để tạo lớp cài đặt đối tượng trên server
- HelloHelper.java, HelloHolder.java: cung cấp các pt truy xuất đt CORBA

B4: Xây dựng lớp dùng để tạo đối tượng CORBA (file HelloServant.java)

```
package HelloCorba;
class HelloServant extends _HelloImplBase
{
   public String sayHello()
   {
         return "\nHello world !!\n";
   }
}
```

B5: Viết ct server dùng để tạo đối tượng và đăng ký đối tượng với trình quản lý đối tượng

```
package HelloCorba;
import org.omg.CosNaming.*; // HelloServer se dung dich vu ql ten
import org.omg.CORBA.*;     // tat ca ung dung CORBA can lop nay
import org.omg.CosNaming.NamingContextPackage.*;
public class HelloServer
{
  public static void main(String args[])
  {
    try{

      //khoi dong trinh  ORB
      ORB orb = ORB.init(args, null);
      // tao dt corba
      HelloServant helloRef = new HelloServant();
      // ket noi ORB va thong bao dt voi ORB, giong exportObject o RMI
      orb.connect(helloRef);
      // lay tham chieu den dich vu quan ly ten tnameserv, giong rmiregistry
      org.omg.CORBA.Object objRef = orb.resolve_initial_references("NameService");
```

```
    //chuyen dt objRef la dt corba tong quat sang dt kieu NamingContext (ep kieu)
    //dt nay dung de goi dich vu ql ten
    NamingContext ncRef = NamingContextHelper.narrow(objRef);
    // tao ten dt
    NameComponent nc = new NameComponent("Hello", "");
    //tao duong dan de luu ten dt
    NameComponent path[] = {nc};
    //dang ky dt helloRef ten la "Hello"
     ncRef.rebind(path, helloRef);
    // tao dt Object
    java.lang.Object sync = new java.lang.Object();
    //vong lap vo han cho nhan yeu cau tu may khach
    synchronized(sync){
      sync.wait();
    }
   } catch(Exception e) {
      System.err.println("ERROR: " + e);
      e.printStackTrace(System.out);
    }
  }
}
```

B6: Viết ct client triệu gọi phương thức của đối tượng CORBA

```
package HelloCorba;
import org.omg.CosNaming.*; // HelloClient se dung dich vu ql ten
import org.omg.CORBA.*;     // tat ca ung dung CORBA can lop nay
public class HelloClient
{
  public static void main(String args[])
  {
    try{
      // khoi dong trinh moi gioi ORB
```

```
    ORB orb = ORB.init(args, null);
     // lay tham chieu den dich vu quan ly ten (tnameserv.exe)
    org.omg.CORBA.Object objRef = orb.resolve_initial_references("NameService");
    //chuyen dt objRef la dt corba tong quat sang dt kieu NamingContext (ep kieu)
    //dt nay dung de goi dich vu ql ten
    NamingContext ncRef = NamingContextHelper.narrow(objRef);
    // tao ten dt
    NameComponent nc = new NameComponent("Hello", "");
    //tao duong dan de luu ten dt
    NameComponent path[] = {nc};
    //dung duong dan de lay tham chieu den dt tren server
    Hello helloRef = HelloHelper.narrow(ncRef.resolve(path));
    String hello = helloRef.sayHello();
    System.out.println(hello);
   } catch(Exception e) {
      System.out.println("ERROR : " + e);
      e.printStackTrace(System.out);
    }
  }
}
```

B7: Thử nghiệm ct:

- *Dich tat ca *.java sang *.class : click chột phải trên tên gói corbaHello, chọn make*
- *Nạp trình quản lý tên đối tượng (tnameserv.exe): có thể thực thi ct này ở bất cứ vị trí nào.*
- *Chạy ct server/ct client*

1. Viết ứng dụng CORBA dùng ORB VisiBroker (tich hop trong Jbuilder)

- Thiết lập các tùy chọn cho Jbuilder để viết các ứng dụng CORBA (chỉ cần làm 1 lần)

Tools/Enterprise Setup/Corba, chọn"Configuration" là VisiBroker, chọn ba check box. Chọn Edit, nhập vào hộp "Path for ORB tool": D:/Borland/AppServer/bin (đường dẫn tới osagent.exe). Thêm thư viện Borland/AppServer/lib/vbjorb.jar

- Chọn Project/ Project Properties/Build/IDL, chọn "IDL Complier" là VisiBroker

- *B1: Tạo project, nhớ chọn thư viện VisiBroker*
- *B2: Định nghĩa giao tiếp bằng IDL: File/New/Sample IDL*
- *B3: Sinh ra file Stub và Servant (Skeleton): click chuột phải trên file idl, chọn make*
- *B4: Cài đặt client:*

- *Tạo ứng dụng: file/new/application*
- *Tạo giao diện giao tiếp với server: Wizards/Use CORBA Interface.*
- *Đăng ký giao diện với ORB: Thêm đt OrbConnect vào ứng dụng, ràng buộc giao diện với ORB*

- *B6: Cài đặt server:File/New/corba/ CORBA Server Application*
- *B7: Viết cài đặt cho đt server*
- *B8: Chạy thử nghiệm:*

- *Thực thi Smart Agent*
- *Thực thi server*
- *Thực thi client*

BÀI TẬP

Dùng công nghệ RMI và CORBA làm các bài tập sau:

1. Giải phương thức bậc nhất, cho client hỏi nhiều lần

2. Viết ct hỏi thời tiết, cho client hỏi nhiều lần. Biết rằng thông tin thời tiết lưu trong tập tin thoitiet.txt có dạng sau

TP	Nhiệt Độ
HCM	25-30
HN	20-25
vv...	

3. Tương tự bài 2 nhưng thông tin thời tiết lưu trong table thoitiet, dùng hệ qủan trị CSDL SQL

4. Viết ct hỏi điểm, cho client hỏi nhiều lần, thông tin về điểm lưu trong table bangdiem, dùng hệ qủan trị CSDL SQL. Biet rang dtb=(dlt+dth)/2

masv	dlt	dth	dtb
1	10	6	8
2	6	8	7
vv...			

- Hết -

ON TAP THI

I. APPLET

- Goi tham so cho applet
- su dung thread trong applet
- Su dung giao dien trong applet

II. LAP TRINH CLIENT/SERVER

Dang 1: dung giao thuc TCP

- Nhieu client y/c dong thoi (dung thread)
- 1 client co the y/c nhieu lan
- du lieu truy xuat o dang file van ban hoac csdl
- Co su dung giao dien

Dang 2: dung giao thuc UDP

- Cac y/c tuong tu dang 1

BAI TAP ON

I. Applet

1. viet applet cong hai so, co su dung giao dien, co goi tham so a, b.

a=	2X
b=	3
a+b=	5
OK	CLEAR
hien thong bao	Du lieu sai

(xmoi,ymoi),(xmoi+cd/4,ymoi+cr/4)
(xmoi+3cd/4,ymoi+cr/4,xmoi+cd,ymoi)

2. Viet applet cho mot dia bay, bay ngau nhien, khi qua trang khac phai ngung bay, kich thuoc dia bay duoc goi nhu tham so.

II. CLIENT/SERVER

1. Quan ly diem, dung TCP, co giao dien, co CSDL SQL.
CSDL QLD.MDF, TABLE BANGDIEM(masv vachar(10), diemthi int)

- ct chinh

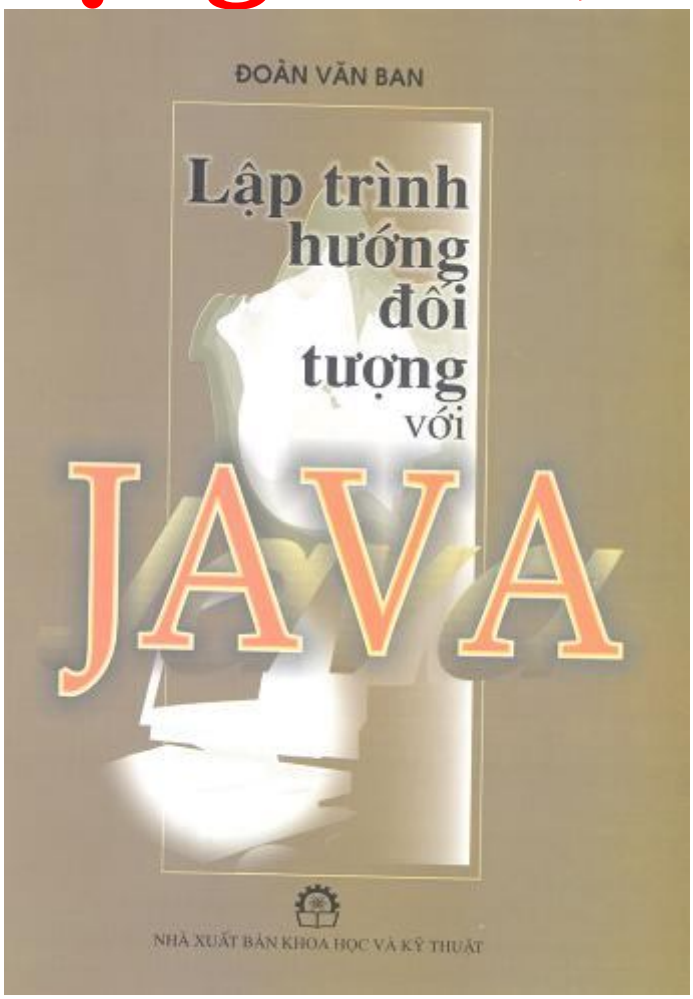
CHUONG TRINH QUAN LY DIEM				
NHAP DIEM	XEM DIEM	SUA DIEM	XOA SV	THONG KE

- ct nhap

MASV	1
DIEM THI	9
OK	CLEAR
THONG BAO	DU LIEU SAI/TRUNG KHOA/DA NHAP

2) Lam lai bai 1 bang giao thuc UDP

Giáo trình lập trình mạng Java, C



Lời mở đầu

Lập trình mạng là một trong những nhiệm vụ căn bản để phát triển các ứng dụng doanh nghiệp. Một chương trình mạng được viết ra để các chương trình trên các máy tính khác nhau có thể truyền tin với nhau một cách hiệu quả và an toàn cho dù chúng được cài đặt trên mạng LAN, WAN hay mạng toàn cầu Internet, đây là điều căn bản đối với sự thành công của nhiều hệ thống.

Java là ngôn ngữ lập trình hướng đối tượng thuần túy với nhiều đặc trưng ưu việt so với các ngôn ngữ lập trình hướng đối tượng khác như tính độc lập với nên, tính bảo mật,…Java là ngôn ngữ ngay từ khi ra đời đã hướng đến lập trình mạng nên việc viết một chương trình lập trình mạng bằng Java dễ dàng hơn nhiều so với các ngôn ngữ khác.

Giáo trình này bao gồm 10 chương:

Chương 1*: Giới thiệu những khái niệm căn bản về mạng máy tính* để người đọc có thể tiếp cận với các chương tiếp theo. Trong chương này chúng ta sẽ đi vào xem xét mạng vật lý, phần cứng được sử dụng trong các mạng LAN. Tiếp theo chúng ta sẽ tìm hiểu mô hình phân tầng OSI bảy tầng, và sự tương ứng của họ giao thức TCP/IP với các tầng trong mô hình OSI. Sau đó chúng ta sẽ đi vào tìm hiểu các giao thức mạng, giao thức Internet, và giao thức e-mail.

Chương 2: *Giới thiệu ngôn ngữ lập trình Java*. Chương này trình bày các khái niệm căn bản về ngôn ngữ lập trình Java. Giới thiệu lịch sử phát triển và cấu trúc của máy ảo Java. Những đặc trưng đã tạo nên sức mạnh của ngôn ngữ Java cũng được giới thiệu trong chương này. Cũng trong chương này chúng ta sẽ đi vào tìm hiểu cách cấu hình và cài đặt môi trường biên dịch, chạy và soạn thảo ngôn ngữ Java. Tiếp đến ta sẽ đi vào tìm hiểu các thành phần cơ bản của Java như kiểu dữ liệu, cấu trúc lệnh tuần tự rẽ nhánh, lặp, và nhảy. Tiếp theo chúng ta sẽ đi vào tìm hiểu các khái niệm liên quan đến lập trình hướng đối tượng trong Java như lớp, phương thức, thuộc tính, các từ khóa bổ trợ như static, final, abstract, thừa kế và tính đa hình trong Java. Một trong những khái niệm mới mà các ngôn ngữ truyền thống trước đây không có là ngoại lệ và đón bắt ngoại lệ trong Java cũng được giới thiệu.

Chương 3*: Các luồng vào ra*. Chương này giới thiệu khái niệm vào ra bằng các luồng dữ liệu. Trước tiên ta sẽ tìm hiểu về các luồng và ý nghĩa của luồng trong chương trình Java. Tiếp đến chúng ta sẽ lần lượt tìm hiểu các luồng vào ra chuẩn trong gói làm việc với console. Các luồng trừu tượng java.io.InputStream, java.io.OutputStream là các luồng cơ bản để từ đó xây dựng nên các luồng cụ thể. Luồng được chia thành các nhóm như luồng byte và luồng ký tự. Từ phiên bản Java 1.4 một đặc trưng vào ra mới trong Java được đưa vào cũng được giới thiệu trong chương này. Việc nắm vững kiến thức ở chương này cũng giúp cho việc lập trình ứng dụng mạng trở nên đơn giản hơn vì thực chất của việc truyền và nhận dữ liệu giữa các ứng dụng mạng là việc đọc và ghi các luồng.

Chương 4: *Lập trình đa tuyến đoạn*. Trong các ngôn ngữ lập trình trước đây các ứng dụng hầu hết là các ứng dụng đơn tuyến đoạn. Để tăng tốc độ xử lý và giải quyết vấn đề tương tranh của các ứng dụng nói chung và ứng dụng mạng nói riêng ta cần sử dụng khái niệm đa tuyến đoạn. Phần đầu của chương này trình bày các khái niệm căn bản về tiến trình, tuyến đoạn. Tiếp đến chúng ta sẽ xem xét các cách cài đặt một ứng dụng tuyến đoạn trong Java bằng lớp Thread và thực thi giao tiếp Runnable. Sau đó ta sẽ đi vào tìm hiểu các phương thức của lớp Thread. Sự đồng bộ hóa và cách cài đặt một chương trình đồng bộ hóa cũng được giới thiệu trong chương này.

Chương 5: *Lập trình mạng với các lớp InetAddress, URL và URLConnection.* Lớp InetAddress là lớp căn bản đầu tiên trong lập trình mạng mà ta cần tìm hiểu. Nó chỉ ra cách một chương trình Java tương tác với hệ thống tên miền. Tiếp đến ta sẽ đi vào tìm hiểu các khái niệm về URI, URL,URN và lớp biểu diễn URL trong Java. Cách sử dụng URL để tải về thông tin và tệp tin từ các server. Sau đó ta đi vào tìm hiểu lớp URLConnection, lớp này đóng vai trò như một động cơ cho lớp URL.

Chương 6*: Lập trình Socket cho giao thức TCP*. Trong chương này chúng ta sẽ tìm hiểu cách lập trình cho mô hình client/server và các kiểu kiến trúc client/server. Các lớp Socket và ServerSocket được trình bày chi tiết trong chương này để lập các chương trình cho giao thức TCP.

Chương 7: *Lập trình ứng dụng cho giao thức UDP*. Chương này giới thiệu giao thức UDP và các đặc trưng của giao thức này. Tiếp đến ta đi vào tìm hiểu các lớp DatagramPacket và DatagramSocket để viết các chương trình ứng dụng mạng cho giao thức UDP.

Chương 8: *Tuần tự hóa đối tượng và ứng dụng trong lập trình mạng*. Trình bày các vấn đề về tuần tự hóa và ứng dụng của tuần tự hóa trong lập trình mạng.

Chương 9: *Phân tán đối tượng bằng Java RMI*. Chương này tìm hiểu chủ đề về lập trình phân tán đối tượng bằng kỹ thuật gọi phương thức RMI (Remote Method Invocation).

Chương 10:*Xử lý cơ sở dữ liệu trong Java.* Trình bày cách thức kết nối các cơ sở dữ liệu và xử lý cơ sở dữ liệu bằng Java thông qua giao diện lập trình ứng dụng JDBC.

Tìm hiểu về lập trình mạng tốt nhất là trên các hệ thống mạng thực sự với nhiều máy tính được kết nối vật lý. Tuy nhiên trong giáo trình này hầu hết các ví dụ được trình bày để bạn đọc có thể lập trình và thử nghiệm các ứng dụng mạng trên các máy đơn.

Mặc dù đã hết sức cố gắng để trình bày giáo trình một cách dễ hiểu với các ví dụ minh họa giúp bạn đọc có thể thử nghiệm ngay sau khi tìm hiểu các vấn đề lý thuyết, nhưng chắc chắn giáo trình này không thể tránh khỏi những thiếu sót nhất định. Rất mong sự góp ý và phê bình của các bạn độc giả. Mọi thắc mắc và góp ý các bạn có thể gửi về theo địa chỉ e-mail sau:lequocdinh@vnn.vn hoặc hoan_td2001@yahoo.com

Để hoàn thành giáo trình này các tác giả đã nhận được sự giúp đỡ rất nhiều từ bạn bè, đồng nghiệp và những người thân.

Xin chân thành cảm ơn tới tất cả mọi người.

Nhóm tác giả

MỤC LỤC

Chương 1

Các khái niệm căn bản về mạng và giao thức mạng

1. Mạng máy tính

Mạng máy tính Là tập hợp các máy tính hoặc các thiết bị được nối với nhau bởi các đường truyền vật lý và theo một kiến trúc nào đó.

Chúng ta có thể phân loại mạng theo qui mô của nó:

- Mạng LAN (Local Area Network)-mạng cục bộ: kết nối các nút trên một phạm vi giới hạn. Phạm vi này có thể là một công ty, hay một tòa nhà.
- Mạng WAN (Wide Area Network): nhiều mạng LAN kết nối với nhau tạo thành mạng WAN.
- MAN (Metropolitan Area Network), tương tự như WAN, nó cũng kết nối nhiều mạng LAN. Tuy nhiên, một mạng MAN có phạm vi là một thành phố hay một đô thị nhỏ. MAN sử dụng các mạng tốc độ cao để kết nối các mạng LAN của trường học, chính phủ, công ty, ..., bằng cách sử dụng các liên kết nhanh tới từng điểm như cáp quang.

Khi nói đến các mạng máy tính, người ta thường đề cập tới mạng xương sống (backbone). Backbone là một mạng tốc độ cao kết nối các mạng có tốc độ thấp hơn. Một công ty sử dụng mạng backbone để kết nối các mạng LAN có tốc độ thấp hơn. Mạng backbone Internet được xây dựng bởi các mạng tốc độ cao kết nối các mạng tốc độ cao. Nhà cung cấp Internet hoặc kết nối trực tiếp với mạng backbone Internet, hoặc một nhà cung cấp lớn hơn.

1.1. Các đường kết nối trong mạng WAN

Để kết nối tới một mạng WAN, có một số tùy chọn như sau:

- Khi một khách hàng cụ thể yêu cầu sử dụng mạng với thông lượng xác định, chúng ta có thể sử dụng các đường thuê bao (leased line).
- Các đường chuyển mạch (switched lines) được sử dụng bởi dịch vụ điện thoại thông thường. Một mạch được thiết lập giữa phía nhận và phát trong khoảng thời gian thực hiện cuộc gọi hoặc trao đổi dữ liệu. Khi không còn cần dùng đường truyền nữa, thì cần phải giải phóng đường truyền cho khách hàng khác sử dụng.

Các ví dụ về các đường chuyển mạch là các đường POTS , ISDN, và DSL.

- Mạng chuyển mạch gói là mạng mà trong đó nhà cung cấp dịch vụ cung cấp công nghệ chuyển mạch để giao tiếp với mạng xương sống. Giải pháp này cung cấp hiệu năng cao và khả năng chia sẻ tài nguyên giữa các khách hàng.

  Các giao thức được sử dụng cho các mạng chuyển mạch bao gồm X.25 (64Kbps), Frame Relay (44.736Mbps), và ATM (9.953 Gbps).

  Kiến trúc mạng: Một trong những vấn đề cần quan tâm đối với một mạng máy tính là kiến trúc mạng. Nó cập tới hai khía cạnh là Hình trạng mạng và Giao thức mạng.

- *Hình trạng mạng:* Là cách nối các máy tính với nhau. Người ta phân loại mạng theo hình trạng mạng như mạng sao, mạng bus, mạng ring…
- *Giao thức mạng:* Là tập hợp các qui tắc, qui ước truyền thông của mạng mà tất cả các thực thể tham gia truyền thông phải tuân theo.

1.2. Giao thức Ethernet

Để có được sự hiểu biết tốt hơn về các mạng vật lý hoạt động như thế nào, chúng ta sẽ xem xét một số giao thức LAN phổ biến: giao thức Ethernet. Chín mươi phần trăm các thiết bị gắn với một mạng LAN sử dụng giao thức Ethernet, ban đầu được phát triển bởi Xerox, Digital Equipement, và Intel năm 1972.

Ngày nay, Ethernet có thể hỗ trợ các đường truyền 100Mbps và 1Gbps. Rất nhiều công nghệ đường truyền có thể được sử dụng với một Ethernet. Người ta sử dụng một số qui ước để đặt tên giao thức Enternet. Tên này chỉ ra tốc độ của mạng Ethernet và các thuộc tính của công nghệ đường truyền. Các tên như vậy được bắt đầu bằng một số để chỉ ra tốc độ truyền tối đa, tiếp theo là một từ được sử dụng để xác định công nghệ truyền dẫn, và cuối cùng là một số để chỉ ra khoảng cách giữa hai nút. Ví dụ, 10Base2 ký hiệu một Ethernet hoạt động với tốc độ 10Mbps sử dụng kỹ thuật truyền trên băng tần cơ sở, với các cáp có chiều dài tối đa là 200m. Một số cấu hình thông dụng khác như sau:

Chuẩn Ethernet	Tốc độ	Kiểu cáp	Mô tả
10Base5	10Mbps	Cáp đồng trục	Đây là chuẩn ban đầu cho Ethernet
10BaseT	10Mbps	Cáp đồng	10BaseT là một mạng 10Mbps với cáp xoắn.
100BaseTX	100Mbs	Cáp đồng	100Mbps công nghệ cáp xoắn và khả năng truyền song công
1000BaseSX	1000Mbps	Cáp đa chế độ	1000Mbps với cáp sợi quang. S :Short wavelength (850nm)

Bảng 1.1

- **CSMA/CD** (Carrier Sense Multiple Access/Collision Detect).

Nhiều thiết bị được kết nối vào cùng một mạng và tất cả đều cùng có quyền truy xuất đồng thời. Khi một thông điệp được gửi đi, nó được truyền thông qua một mạng . Phía nhận được định danh bởi một địa chỉ duy nhất, và chỉ có nút này đọc thông điệp, còn các nút khác thì bỏ qua.

Một vấn đề đặt ra là khi có nhiều nút cùng cố gắng gửi thông điệp tại cùng một thời điểm, điều này có thể phá hỏng các gói tin. Giải pháp cho vấn đề này là mỗi nút mạng giám sát mạng và có thể phát hiện mạng đang rảnh hay bận. Một nút chỉ có thể bắt đầu gửi dữ liệu khi không có dữ liệu nào được gửi đi trên mạng trước đó. CSMA là một bộ phận của CSMA/CD.

Tuy nhiên vẫn có khả năng là hai nút, sau khi kiểm tra thấy mạng không bận, bắt đầu gửi gói tin cùng một thời điểm trên cùng cáp mạng. Điều này có thể gây lên xung đột giữa hai gói tin, kết quả là phá hỏng dữ liệu. Cả hai phía gửi đều nhận thức được gói tin bị hỏng bởi vì nó vẫn lắng nghe mạng khi gửi dữ liệu, và vì thế có thể phát hiện xung đột. Đây là CD (Collision Dection) trong CSMA/CD. Cả hai nút dừng việc truyền dữ liệu ngay tức thời, và chờ một thời điểm nhất định trước khi kiểm tra mạng trở lại để xem mạng có rỗi hay không và truyền lại.

Mỗi nút trên mạng sử dụng một địa chỉ MAC (Media Access Control) để định danh duy nhất. Địa chỉ này được định nghĩa bởi thiết bị giao tiếp mạng. Một gói tin được gửi đi trên mạng, nhưng nếu thiết bị mạng không nhận diện host của nó như một host nhận, nó sẽ bỏ qua gói tin và chuyển tiếp nó.

- ***Các giao thức khác***

IBM đã phát triển giao thức Token Ring (IEEE802.5), trong đó các nút mạng được kết nối theo một vòng. Với Ethernet, bất kỳ một nút nào cũng có thể gửi một thông điệp khi không có gói tin nào trên mạng. Với Token Ring mỗi nút có một quyền truy xuất tới mạng theo một thứ tự định trước. Một token lưu chuyển vòng quanh vòng, và chỉ nút lệnh nào có thẻ bài mới có thể gửi thông điệp. Ngày nay, Ethernet đang thay thế dần các mạng Token Ring bởi vì các mạng này tốn kém và khó cài đặt.

AppleTalk là một giao thức mạng LAN được phát triển bởi Apple tương đối phổ biến trong các trường học, các nhà máy,...

ATM là một giao thức khác có thể tìm thấy trong mạng LAN. Nó hỗ trợ các mạng tốc độ cao sử dụng kỹ thuật chuyển mạch và có đảm bảo chất lượng dịch vụ.

1.3. Các thành phần vật lý

Một vấn đề quan trọng để biết về mạng là biết về phần cứng. Chúng ta sẽ xem xét các thành phần chủ yếu của một mạng LAN sau:

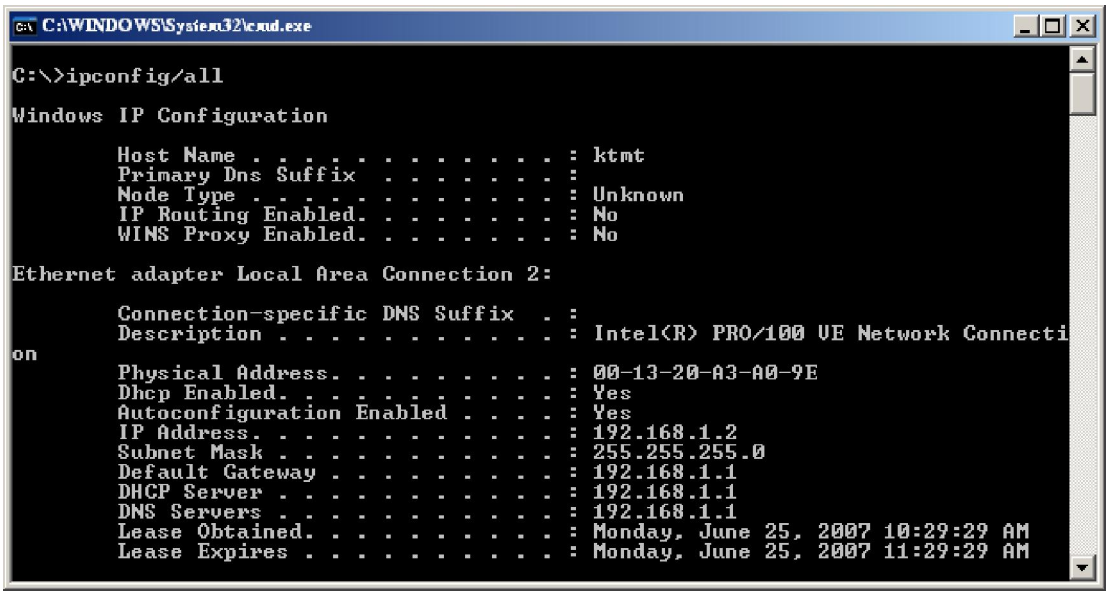
- Thiết bị giao tiếp mạng
- Hub
- Switch
- Router

- **Thiết bị giao tiếp mạng (Network Interface Thiết bị)**

NIC là thiết bị giao tiếp được sử dụng để kết nối một thiết bị với mạng LAN. Nó cho phép chúng ta gửi và nhận các thông điệp từ mạng. Một NIC có một địa chỉ MAC duy nhất mà cung cấp định danh duy nhất cho từng thiết bị.

Địa chỉ MAC là một số 12 byte-hệ 16 được gán cho thiết bị mạng. Địa chỉ này có thể được thay đổi bởi một trình điều khiển mạng một cách linh hoạt (như trong trường hợp của hệ thống DECnet, mạng được phát triển bởi Digital Equipment), nhưng thông thường địa chỉ MAC không thay đổi.

Ta có thể tìm địa chỉ MAC của một máy sử dụng hệ điều hành Windows bằng cách dùng tiện ích dòng lệnh ipconfig trong DOS với tham số switch



Hình 1.1

- ***Hub***

Nhiều thiết bị có thể được kết nối một cách dễ dàng với sự giúp đỡ của một hub. Hub là một thiết bị kết nối gắn nhiều thiết bị vào LAN. Mỗi thiết bị thường kết nối thông qua một cáp tới một cổng trên hub.

Hub hoạt động như một bộ chuyển tiếp. Khi nó chuyển từng thông điệp từ cổng này tới cổng khác, và chuyển tới mạng. Hub là một thành phần tương đối đơn giản của một mạng, hoạt động ở tầng vật lý để truyền dữ liệu mà không cần thao tác xử lý nào. Điều này làm cho các hub dễ cài đặt và quản lý, vì chúng không đòi hỏi cấu hình đặc biệt nào.

- ***Switch***

Các chuyển mạch (switch) phân chia mạng thành các đoạn (segment). So với hub, switch là một thiết bị thông minh hơn nhiều. Switch lưu trữ các địa chỉ MAC của các thiết bị được kết nối tới các cổng của nó trong bảng lookup. Các bảng lookup cho phép switch lọc các thông điệp mạng và không giống với hub, nó không chuyển tiếp các thông điệp tới từng cổng. Điều này loại bỏ các xung đột có thể xảy ra và mạng có thể đạt được hiệu năng tốt hơn. Chức năng chuyển mạch được thực hiện bằng cách sử dụng phần cứng.

- ***Router***

Router là một thiết bị trung gian mạng, kết nối nhiều mạng vật lý. Một mạng có nhiều host có thể được phân chia thành các phần riêng, hay còn gọi là subnet. Ưu điểm của các subnet là:

Hiệu năng được cải thiện bằng cách giảm broadcast, broadcast là 1 thông điệp được gửi tới tất cả các nút của mạng.

Khả năng hạn chế người dùng trong từng mạng con xác định đưa ra những ưu điểm về bảo mật.

Các subnet nhỏ hơn sẽ dễ quản lý hơn so với một mạng lớn.

Các router không chỉ được sử dụng trong LAN, chúng có một vai trò quan trọng trong WAN. Router nhận một thông điệp và chuyển tiếp nó tới đích bằng cách sử dụng đường đi tốt nhất tới đích đó.

Một Router lưu giữ một bảng định tuyến liệt kê tất cả các cách mà các mạng có thể đạt tới. Thông thường sẽ có một số đường đi từ mạng này tới mạng khác, nhưng chỉ có một trong số đó là tốt nhất, và nó là con đường được mô tả trong bảng định tuyến. Các router truyền tin bằng cách sử dụng các giao thức định tuyến để phát hiện các router khác trên mạng, và hỗ trợ cho việc trao đổi thông tin về các mạng được gắn với từng bộ định tuyến.

Thông tin mà một bộ định tuyến thu thập về các đường đi giữa các mạng được gọi là độ đo router, và có thể bao gồm những thông tin như sự mất mát gói tin và thời gian truyền tin. Thông tin được sử dụng để tạo ra độ đo tùy thuộc vào giao thức định tuyến:

Giao thức định tuyến vectơ khoảng cách

Các giao thức RIP(Routing Information Protocol) và IGRP(Interior Gateway Routing Protocol) sử dụng một biến đếm để chỉ ra số router mà gói tin phải đi qua để đến đích. Các giao thức này thường lựa chọn các đường đi với ít router, mà không quan tâm đến tốc độ và độ tin cậy.

Các giao thức định tuyến trạng thái liên kết

Việc tính toán đường đi tốt nhất của các giao thức định tuyến OSPF và BGP quan tâm đến nhiều yếu tố như tốc độ, độ tin cậy, và thậm chí là chi phí của đường đi.

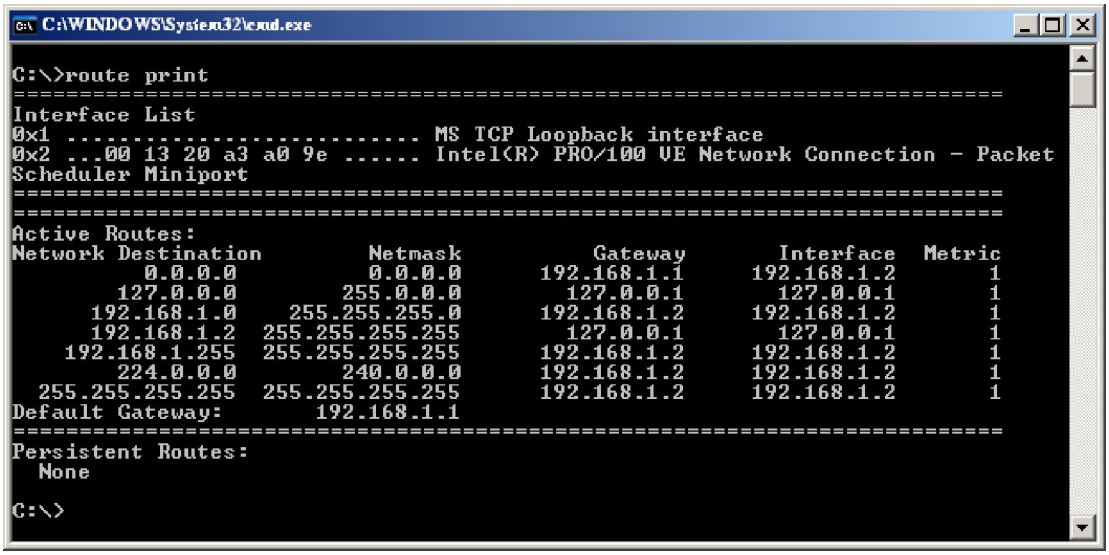
Các giao thức định tuyến lai

Các giao thức này sử dụng sự kết hợp việc tính toán trạng thái liên kết và vectơ khoảng cách.

- ***Vấn đề tìm đường đi***

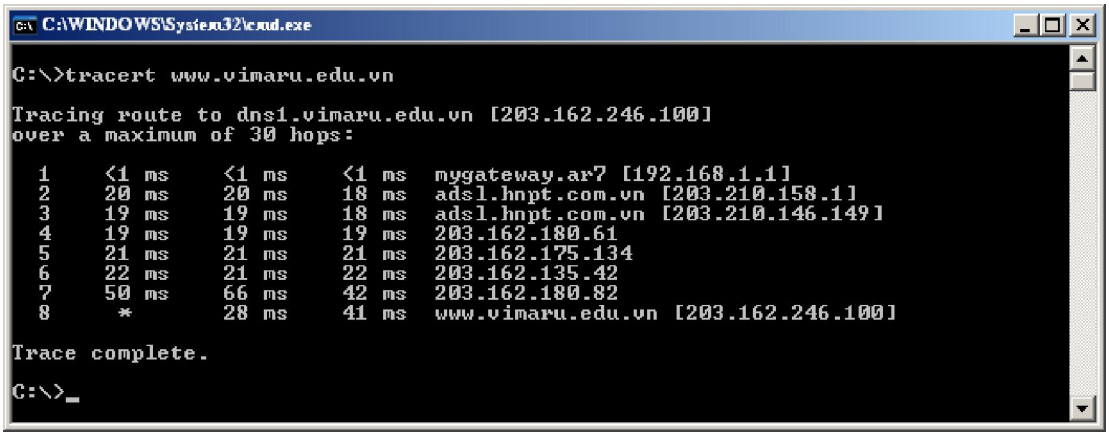
Với cấu hình TCP/IP, một gateway mặc định được thiết lập. Đây là một địa chỉ IP của cổng bộ định tuyến mà subnet kết nối tới. Bộ định tuyến này được sử dụng khi một host ở bên ngoài subnet cần được liên lạc.

Ta có thể thấy bảng định tuyến cục bộ trên hệ điều hành Windows bằng cách sử dụng lệnh ROUTE PRINT trên dòng lệnh.. Lệnh này hiển thị các gateway sẽ được sử dụng cho mỗi liên kết mạng.



Hình 1.2

Một lệnh hữu ích khác là lệnh TRACERT. Lệnh này cho phép chúng ta kiểm tra đường đi được sử dụng để đi tới đích.



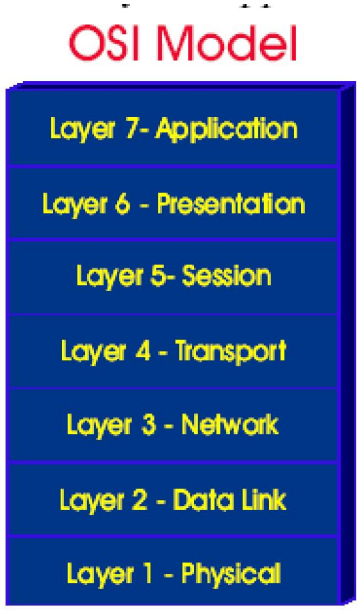
Hình 1.3

2. Mô hình phân tầng

ISO đã định nghĩa một mô hình cho một mạng đã được chuẩn hóa sẽ thay thế cho TCP/IP,DECNet và các giao thức khác như là một giao thức mạng cơ bản được sử dụng cho Internet. Tuy nhiên, do sự phức tạp của OSI, mô hình này không được cài đặt và sử dụng nhiều trong thực tế. TCP/IP đơn giản hơn nhiều và vì vậy có thể tìm thấy ở nhiều nơi. Nhưng có rất nhiều ý tưởng mới từ giao thức OSI có thể tìm thấy trong phiên bản tiếp theo của IP, IPv6.

Trong khi giao thức OSI không được xây dựng đầy đủ trong thực tế, nhưng mô hình bảy tầng đã rất thành công và nó hiện đang được sử dụng như là một mô hình tham chiếu để mô tả các giao thức mạng khác nhau và chức năng của chúng.

Các tầng của mô hình OSI phân chia các nhiệm vụ cơ bản mà các giao thức mạng phải thực hiện, và mô tả các ứng dụng mạng có thể truyền tin như thế nào. Mỗi tầng có một mục đích cụ thể và được kết nối với các tầng ở ngay dưới và trên nó. Bảy tầng của mô hình OSI.



Hình 1.4

- Tầng ứng dụng (Application): định nghĩa một giao diện lập trình giao tiếp với mạng cho các ứng dụng người dùng.
- Tầng trình diễn (Presentation): có trách nhiệm mã hóa dữ liệu từ tầng ứng dụng để truyền đi trên mạng và ngược lại.
- Tầng phiên (Session): tạo ra một liên kết ảo giữa các ứng dụng.
- Tầng giao vận (Transport): cho phép truyền dữ liệu với độ tin cậy cao.
- Tầng mạng (Network): cho phép truy xuất tới các nút trong mạng LAN bằng cách sử dụng địa chỉ logic
- Tầng liên kết dữ liệu (Data Link): truy xuất tới một mạng vật lý bằng các địa chỉ vật lý.
- Cuối cùng, tầng vật lý (Physical): có thể bao gồm các thiết bị kết nối, cáp nối.

Bây giờ chúng ta tìm hiểu khái niệm của các tầng này bằng cách xem xét chức năng của từng tầng chi tiết hơn.

2.1. Tầng 1:Tầng vật lý

Tầng vật lý bao gồm môi trường vật lý như yêu cầu về cáp nối, các thiết bị kết nối, các đặc tả giao tiếp, hub và các repeater,...

2.2. Tầng 2: Tầng liên kết dữ liệu

Địa chỉ MAC mà chúng ta đã đề cập là địa chỉ của tầng 2. Các nút trên LAN gửi thông điệp cho nhau bằng cách sử dụng các địa chỉ IP, và các địa chỉ này phải được chuyển đổi sang các địa MAC tương ứng.

Giao thức phân giải địa chỉ (ARP: Address Resolution Protocol) chuyển đổi địa chỉ IP thành địa chỉ MAC.Một vùng nhớ cache lưu trữ các địa chỉ MAC tăng tốc độ xử lý này, và có thể kiểm tra bằng tiện ích arp -a,

2.3. Tầng 3: Tầng mạng

Tầng mạng là tầng nằm phía trên tầng liên kết. Trong tầng 3, địa chỉ logic được sử dụng để kết nối tới các nút khác. Các địa chỉ MAC của tầng 2 chỉ có thể được sử dụng trong một mạng LAN, và chúng ta phải sử dụng cách đánh địa chỉ của tầng 3 khi truy xuất tới các nút trong mạng WAN.

Internet Protocol là giao thức tầng 3; nó sử dụng các địa chỉ IP để định danh các nút trên mạng. Các router ở tầng 3 được sử dụng để định đường đi trong mạng.

2.4.Tầng 4:Tầng giao vận

Tầng mạng định danh các host bởi các địa chỉ logic. Tầng ứng dụng nhận biết một ứng dụng thông qua cái gọi là điểm cuối (endpoint). Với giao thức TCP, endpoint được nhận biết bởi một số hiệu cổng và địa chỉ IP.

Tầng giao vận được phân loại theo cách truyền tin với độ tin cậy hay không. Truyền tin với độ tin cậy là khi có một lỗi được tạo ra nếu thông điệp được gửi đi nhưng không nhận được một cách đúng đắn. Trong khi truyền tin có độ tin cậy không cao sẽ không kiểm tra xem liệu thông điệp được gửi đi đã nhận được hay chưa. Trong truyền tin với độ tin cậy, tầng giao vận có nhiệm vụ gửi đi các gói tin xác thực hay các thông điệp truyền lại nếu dữ liệu bị hỏng hay bị thất lạc, hay dữ liệu bị trùng lặp.

Một cách khác để phân loại các mạng truyền tin là phân loại mạng theo hướng liên kết hay phi liên kết

- Với truyền tin hướng liên kết, một liên kết phải được thiết lập trước khi các thông điệp được gửi hoặc được nhận.
- Với truyền tin phi liên kết thì không cần giai đoạn thiết lập liên kết.

2.5. Tầng 5: Tầng phiên

Với mô hình OSI, tầng phiên xác định cá dịch vụ cho một ứng dụng, như đăng nhập và đăng xuất một ứng dụng. Tầng phiên biểu diễn một liên kết ảo giữa các ứng dụng. Liên kết tầng phiên độc lập với liên kết vật lý ở tầng giao vận, và các liên kết tầng giao vận được yêu cầu cho một liên kết ở tầng phiên.

2.6.Tầng 6:Tầng trình diễn

Tầng trình diễn được sử dụng để định dạng dữ liệu theo các yêu cầu của ứng dụng. Mã hóa, giải mã, và nén dữ liệu thường diễn ra ở tầng này.

2.7. Tầng 7:Tầng ứng dụng

Tầng ứng dụng là tầng cao nhất của mô hình OSI. Tầng này bao gồm các ứng dụng sử dụng các tiện ích mạng. Các ứng dụng này có thể thực hiện các tác vụ như truyền tệp tin, in ấn, e-mail, duyệt web,…

3. Các giao thức mạng

Các tầng OSI định nghĩa một mô hình các tầng giao thức, và cách mà chúng hoạt động cùng với nhau. Chúng ta so sánh các tầng OSI với một cài đặt cụ thể:. Chồng giao thức TCP/IP là một dạng cụ thể của mô hình OSI, nó bao gồm 4 tầng. Giao thức IP tương ứng với tầng 3 của mô hình OSI; TCP và UDP tương ứng với tầng 4 của mô hình OSI, và chúng thực hiện các nhiệm vụ của tầng phiên, tầng trình diễn, và tầng ứng dụng

Trong mục tiếp theo, chúng ta sẽ xem xét chức năng và mục đích của các giao thức của họ giao thức TCP/IP theo trình tự sau:

- Các giao thức cơ bản
- Các giao thức Internet
- Các giao thức E-mail
- Các giao thức khác

3.1. Các giao thức cơ bản

Như chúng ta có thể thấy, họ giao thức TCP/IP có cấu trúc phân tầng đơn giản hơn nhiều so với mô hình 7 tầng của mô hình OSI. TCP và UDP là các giao thức tầng giao vận tương ứng với tầng 4 của mô hình 7 tầng OSI. Cả hai giao thức này đều sử dụng giao thức IP, một giao thức tương ứng với tầng 3 của mô hình OSI (tầng mạng). Cũng như ba giao thức này có hai giao thức cơ bản trong họ giao thức TCP/IP mở rộng tính năng của giao thức IP: ICMP và IGMP.

3.1.1. IP-Internet Protocol

Giao thức Internet kết nối hai nút. Mỗi nút được định danh bởi một địa chỉ IP 32bit, được gọi là địa chỉ IP của host. Khi gửi một thông điệp, giao thức IP nhận thông điệp từ các giao thức tầng trên như TCP hay UDP và đưa vào trường header chứa thông tin của host đích.

Cách tốt nhất để hiểu giao thức IP là bằng cách xem các trường thông tin header IP chi tiết. Thông tin này được liệt kê trong bảng sau.

Trường	Độ dài	Mô tả
IP Version (Phiên bản IP)	4 bits	Phiên bản IP. (Phiên bản giao thức hiện nay là IPv4)
IP Header Length (Chiều dài Header)	4 bits	Chiều dài của header.
Type of Service (Kiểu dịch vụ)	1 byte	Kiểu dịch vụ cho phép một thông điệp được đặt ở chế độ thông lượng cao hay bình thường, thời gian trễ là bình thường hay lâu, độ tin cậy bình thường hay cao. Điều này có lợi cho các gói được gửi đi trên mạng. Một số kiểu mạng sử dụng thông tin này để xác định độ ưu tiên
Total Length (Tổng chiều dài)	2 bytes	Hai byte xác định tổng chiều dài của thông điệp-header và dữ liệu. Kích thước tối đa của một gói tin IP là 65,535, nhưng điều này là không thực tế đối với các mạng hiện nay. Kích thước lớn nhất được chấp nhận bởi các host là 576 bytes. Các thông điệp lớn có thể phân thành các đoạn-quá trình này được gọi là quá trình phân đoạn

Identification (Định danh)	2 bytes	Nếu thông điệp được phân đoạn, trường định danh trợ giúp cho việc lắp ráp các đoạn thành một thông điệp. Nếu một thông điệp được phân thành nhiều đoạn, tất cả các đoạn của một thông điệp có cùng một số định danh.
Flags	3 bits	Các cờ này chỉ ra rằng thông điệp có được phân đoạn hay không, và liệu gói tin hiện thời có phải là đoạn cuối cùng của thông điệp hay không.
Fragment Offset	13 bits	13 bit này xác định offset của một thông điệp. Các đoạn có thể đến theo một thứ tự khác với khi gửi, vì vậy trường offset là cần thiết để xây dựng lại dữ liệu ban đầu. Đoạn đầu tiên của một thông điệp có offset là 0
Time to Live	1 byte	Xác định số giây mà một thông điệp tồn tại trước khi nó bị loại bỏ.
Protocol	1 byte	Byte này chỉ ra giao thức được sử dụng ở mức tiếp theo cho thông điệp này. Các số giao th ức
Header Checksum	2 bytes	Đây là chỉ là checksum của header. Bởi vì header thay đổi với từng thông điệp mà nó chuyển tới, checksum cũng thay đổi.
Source Address	4 bytes	Cho biết địa chỉ IP 32 bit của phía gửi
Destination Address	4 bytes	Địa chỉ IP 32 bit của phía nhận
Options	variable	
Padding	variabe	

Bảng 1.2

- ***Các địa chỉ IP***

Mỗi nút trên mạng TCP/IP có thể được định danh bởi một địa chỉ IP 32-bit. Thông thường một địa chỉ IP được biểu diễn bởi bộ bốn x.x.x.x, chẳng hạn 192.168.0.1 . Mỗi số trong bốn số này biểu diễn một byte của địa chỉ IP.

Một địa chỉ IP gồm hai phần: phần mạng và phần host. Tùy thuộc vào lớp mạng, phần mạng bao gồm một, hoặc hai hoặc ba byte đầu tiên.

Lớp	Byte 1	Byte 2	Byte 3	Byte 4
A	Networks (1-126)	Host (0-255)	Host (0-255)	Host (0-255)
B	Networks (128-191)	Networks (0-255)	Host (0-255)	Host (0-255)
C	Networks (192-223)	Networks (0-255)	Networks (0-255)	Host (0-255)

Bảng 1.3

Bit đầu tiên của địa chỉ mạng lớp A là 0,vì vậy byte đầu tiên của địa chỉ lớp A nằm trong dải từ 00000001 (1) đến 01111110 (126). Ba byte còn lại phục vụ cho việc định danh các nút trên mạng, cho phép ta kết nối hơn 16 triệu thiết bị vào mạng lớp A. Chú ý rằng các mạng trong bảng trên không đề cập tới các địa chỉ có byte đầu là 127-đây là khoảng địa chỉ dự phòng. Địa chỉ 127.0.0.1 là địa chỉ của localhost, và địa chỉ 127.0.0.0 là địa chỉ loopback.

Các địa chỉ IP của các mạng thuộc lớp B luôn luôn có hai bit đầu tiên của byte đầu là 10, đưa ra khoảng địa chỉ là 10000000 (128) đến 10111111 (191). Byte thứ hai dùng để định danh mạng có giá trị từ 0 đến 255, hai byte còn lại để định danh các nút trên một mạng; tổng cộng là 65534 thiết bị.

Các địa chỉ IP của các mạng thuộc lớp C luôn luôn có ba bit đầu tiên của byte đầu là 110, khoảng giá trị của byte đầu là từ 11000000 (192) đến 11011111 (223). Mạng này chỉ có một byte được thiết lập để định danh host, vì vậy chỉ có 254 thiết bị được kết nối vào mạng lớp C.

- ***Các địa chỉ IP riêng***

Để tránh cạn kiệt các địa chỉ IP, các host không được kết nối trực tiếp với Internet có thể sử dụng một địa chỉ trong các khoảng địa chỉ riêng. Các địa chỉ IP riêng không duy nhất về tổng thể, mà chỉ duy nhất về mặt cục bộ trong phạm vi mạng đó. Tất cả các lớp mạng dự trữ các khoảng nhất định để sử dụng như là các địa chỉ riêng cho các host không cần truy cập trực tiếp tới Internet. Các host như vậy vẫn có thể truy cập Internet thông qua một gateway mà không cần chuyển tiếp các địa chỉ IP riêng.

Lớp	Khoảng địa chỉ riêng
A	10
B	172.16-172.31
C	192.168.0-192.168.255

Bảng 1.4

- ***Các subnet***

Việc kết nối hai nút của hai mạng khác nhau cần có một router. Định danh host của mạng lớp A cần có 24 bit; trong khi mạng lớp C, chỉ có 8 bit. Router phân chia định danh host thành hai phần một phần được gọi là subnet và phần còn lại là phần host

3.1.2. IPv6

Tiền thân của giao thức IP được phát triển bởi Bộ Quốc Phòng Mỹ năm 1960 và cho tới năm 1980 họ giao thức TCP/IP mới ra đời. Bởi IP được xây dựng dựa trên các giao thức mạng DARPA hiện có, nó trở thành phiên bản 4, gọi là IPv4. Lúc đó ý tưởng về các máy di động chưa được kết nối vào Internet nên số host được hỗ trợ bởi IP là tạm đủ. Nhưng hiện nay có rất nhiều thiết bị được kết nối vào Internet, nhu cầu về số địa chỉ IP tăng cao. Một phiên bản mới của địa chỉ IP được phát triển bởi IETF: IPv6. Sự thay đổi quan trọng nhất so với IPv4 là việc sử dụng 128bit để đánh địa chỉ các nút chứ không phải là 32bit nữa.

3.1.3. -Số hiệu cổng

Giao thức IP sử dụng các địa chỉ IP để định danh các nút trên mạng, trong khi tầng giao vận sử dụng các điểm cuối (endpoint) để định danh các ứng dụng. Các giao thức TCP và UDP sử dụng một số hiệu cổng cùng với một địa chỉ IP để xác định điểm cuối của một ứng dụng.

Các số hiệu cổng của TCP và UDP được phân thành ba loại

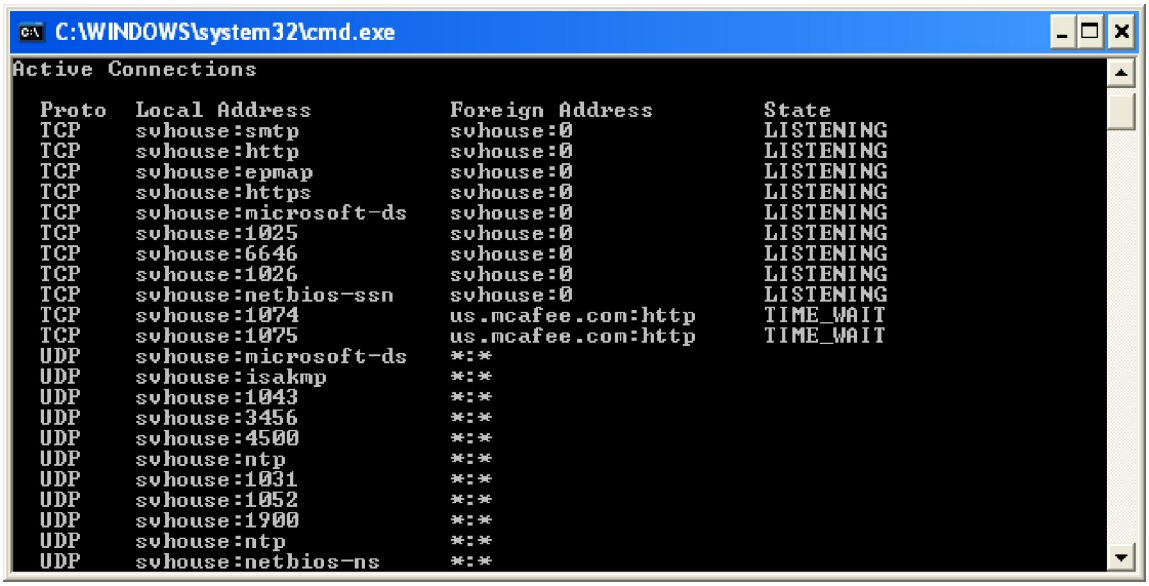
- Các số hiệu cổng hệ thống
- Các số hiệu cổng người dùng
- Các số hiệu cổng riêng và động

Các số hiệu cổng hệ thống nằm trong khoảng từ 0 đến 1023. Các cổng hệ thống chỉ được sử dụng bởi các tiến trình được quyền ưu tiên của hệ thống. Các giao thức nổi tiếng có các số hiệu cổng nằm trong khoảng này.

Các số hiệu cổng người dùng nằm trong khoảng từ 1024 đến 49151. Các ứng dụng server của bạn sẽ nhận một trong các số này làm cổng, hoặc bạn có thể đăng ký số hiệu cổng với IANA .

Các cổng động nằm trong khoảng từ 49152 đến 65535. Khi không cần thiết phải biết số hiệu cổng trước khi khởi động một ứng dụng, một số hiệu cổng trong khoảng này sẽ là thích hợp. Các ứng dụng client kết nối tới server có thể sử dụng một cổng như vậy.

Nếu chúng ta sử dụng tiện ích netstat với tùy chọn –a, chúng ta sẽ thấy một danh sách tất cả các cổng hiện đang được sử dụng, nó cũng chỉ ra trạng thái của liên kết-nó đang nằm trong trạng thái lắng nghe hay liên kết đã được thiết lập.



Hình 1.5

3.1.4. TCP (Transmission Control Protocol)

Giao thức TCP là giao thức truyền tin hướng liên kết có thể sử dụng truyền tin với độ tin cậy cao. Trong đó giao thức tầng 4 có thể gửi các xác thực rằng đã nhận dữ liệu và yêu cầu truyền lại dữ liệu nếu chưa nhận được dữ liệu hoặc dữ liệu bị hỏng.

Các trường header được liệt kê trong bảng sau:

Trường	Độ dài	Mô tả
Cổng nguồn (source port)	2 bytes	Số hiệu cổng của nguồn
Cổng đích (destination port)	2 bytes	Số hiệu cổng đích
Số thứ tự (Sequence Number)	4 bytes	Số thứ tự được tạo ra bởi nguồn và được sử dụng bởi đích để sắp xếp lại các gói tin để tạo ra thông điệp ban đầu, và gửi xác thực tới nguồn.
Acknowledge Number	4 bytes	
Data offset	4 bits	Các chi tiết về nơi dữ liệu gói tin bắt đầu
Reserved	6 bit	Dự phòng
Control		
Window Size	2	Trường này chỉ ra kích thước của vùng đệm nhận.

	bytes	Phía nhận có thể thông báo cho phía gửi kích thước dữ liệu tối đa mà có thể được gửi đi bằng cách sử dụng các thông điệp xác thực
Checksum	2 bytes	Checksum cho header và dữ liệu để xác định xem gói tin có bị hỏng không
Urgent Pointer	2 bytes	Trường này thông báo cho phía nhận biết có dữ liệu khẩn
Options		
Padding		

Bảng 1.5

Giao thức TCP là một giao thức phức tạp và mất thời gian do cơ chế bắt tay, nhưng giao thức này đảm bảo các gói tin đến đúng đích.

Một số giao thức ứng dụng sử dụng TCP như HTTP, FTP, SMTP, và Telnet. TCP yêu cầu một liên kết phải được thiết lập trước khi dữ liệu được gửi đi. Ứng dụng server phải thực hiện một thao tác mở thụ động để tạo một liên kết với một số hiệu cổng cho trước.

3.1.5. UDP-User Datagram Protocol

Ngược với giao thức TCP, UDP là một giao thức có tốc độ truyền tin nhanh vì nó chỉ xác định cơ chế tối thiểu để truyền dữ liệu. Tất nhiên điều này có một số nhược điểm. Các thông điệp có thể được nhận theo bất kỳ thứ tự nào. Thông điệp được gửi đầu tiên có thể được nhận sau cùng. Không có gì đảm bảo là các gói tin sẽ đến đích, và các thông điệp có thể bị thất lạc, hoặc thậm chí có thể nhận được hai bản sao của cùng một thông điệp.

UDP không cần giai đoạn thiết lập liên kết, dữ liệu được gửi đi ngay khi cần. UDP không gửi các thông điệp xác thực, vì vậy dữ liệu có thể nhận được hoặc bị thất lạc. Nếu cần truyền dữ liệu có độ tin cậy nó phải được thực hiện trong một giao thức mức cao hơn.

Vậy đâu là ưu điểm của giao thức UDP, tại sao chúng ta lại cần sử dụng một giao thức có độ tin cậy thấp như vậy? Để hiểu được lý do tại sao ta lại phải sử dụng giao thức UDP ta cần phân biệt giữa truyền unicast, broadcast và multicast.

Một thông điệp unicast được gửi từ nút này tới nút khác. Kiểu truyền tin là truyền tin điểm-điểm. Giao thức TCP chỉ hỗ trợ truyền tin unicast. Nếu một server muốn truyền tin với nhiều client bằng cách sử dụng giao thức UDP, mỗi client phải thiết lập một liên kết, vì các thông điệp chỉ có thể gửi tới một nút. Truyền tin broadcast nghĩa là một thông điệp có thể được gửi tới tất cả các nút trong một mạng. Multicast cho phép các thông điệp được truyền tới một nhóm các nút được lựa chọn.

UDP có thể được sử dụng cho truyền tin unicast nếu cần tới tốc độ truyền tin nhanh, như truyền tin đa phương tiện, nhưng ưu điểm chính của UDP là truyền tin broadcast và truyền tin multicast. Thông thường chúng ta không muốn tất cả các nút gửi về các xác thực cho server vì như vậy sẽ làm cho server quá tải.

Header UDP ngắn và đơn giản hơn rất nhiều so với TCP

Trường thông tin	Độ dài	Mô tả
Source port (Cổng nguồn)	2 byte	Xác định cổng nguồn là một tùy chọn với UDP. Nếu trường này được sử dụng, phía nhận thông điệp có thể gửi một phúc đáp tới cổng này
Destination Port	2 byte	Số hiệu cổng đích
Length	2 byte	Chiều dài của thông điệp bao gồm header và

		dữ liệu
Checksum	2 byte	Để kiểm tra tính đúng đắn

Bảng 1.5

3.1.6. ICMP-Internet Control Message Protocol

ICMP là một giao thức được phát triển từ giao thức IP, điểm khác biệt của ICMP so với giao thức IP là các thông tin phản hồi về trạng thái của hệ thống được ICMP phản hồi bởi các thông điệp.

Các lỗi được phát hiện có thể được thông báo bằng các thông điệp ICMP. Các thông điệp ICMP được sử dụng để gửi các thông tin phản hồi về tình trạng của mạng. Ví dụ, một router gửi thông điệp ICMP "destination unreachable" nếu không tìm thấy một điểm vào cho mạng trong bảng định tuyến. Một router cũng có thể gửi thông điệp ICMP "redirect" nếu tìm thấy đường đi tốt hơn.

ICMP không có trên giao thức IP mà được gửi đi trong các header IP.

Trường thông tin	Độ dài	Mô tả
Type	1 byte	Trường này xác định kiểu thông điệp ICMP. Ví dụ, type có giá trị 3 nghĩa là không đến được đích, 11 nghĩa là quá thời gian, và 12 nghĩa là các tham số header không đúng
Code	1 byte	Code cung cấp thông tin về kiểu thông điệp. Nếu kiểu type là 3, "destination unreachable", thì code xác định là mạng (0), host (1), hay protocol (2), hoặc port (3) là không thể đến được
Checksum	2 byte	Checksum của thông điệp ICMP
	4 byte	Bốn byte cuối cùng của header ICMP có thể cung cấp thông tin bổ trợ tùy thuộc vào kiểu thông điệp
Header IP thông thường	…	

Bảng 1.6

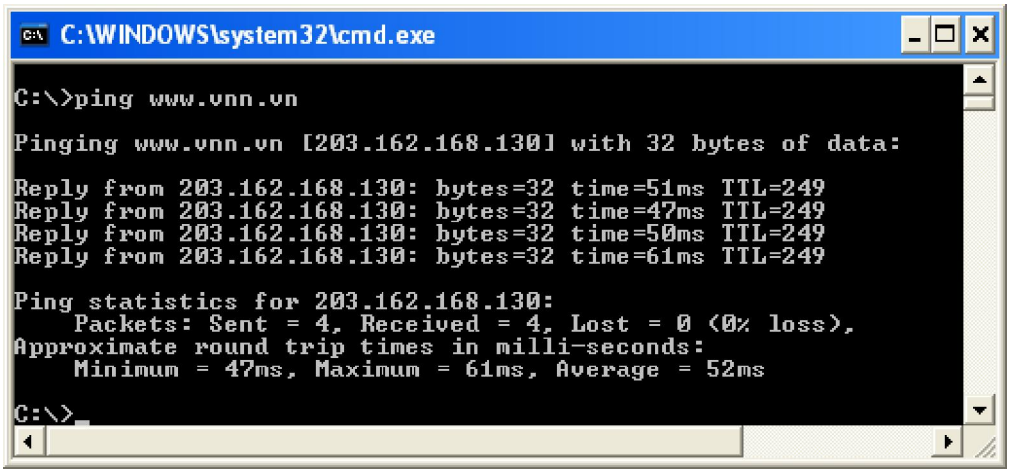
Một số kiểu có thể được gửi bằng cách sử dụng các thông điệp ICMP:

- Echo, Echo Reply
- Lệnh ping gửi lệnh ICMP tới thiết bị đích, xem thiết bị hoạt động tốt hay không và có kết quả trả lời lại.
- Destination unreachable (Không đến được đích), Redirect
  Một router trả về thông điệp ICMP "destination unreachable" nếu không thể liên lạc được với thiết bị đích, hoặc "redirect" (định hướng lại) nếu tìm thấy một đường đi tốt hơn tới đích.
- TTL (Time To Live):Vượt quá thời gian cho phép

- ***Lệnh Ping***

Tiện ích dòng lệnh ping gửi một thông điệp ICMP tới thiết bị đích được xác định bởi hostname và địa chỉ IP trong lệnh ping. Nếu thiết bị là đến được thì ICMP Echo Reply được gửi trở lại.

Lệnh này rất hữu ích khi muốn kiểm tra xem có liên lạc được với thiết bị hay không, hay là có các vấn đề lỗi trung gian



Hình 1.6

3.1.7. IGMP-Internet Group Management Protocol

Tương tự với ICMP, IGMP là sự mở rộng của giao thức IP và phải được cài đặt trong module IP. IGMP được sử dụng bởi các ứng dụng multicast. Khi gửi các thông điệp broadcast tới một LAN, mỗi nút trong LAN phân tích thông điệp và gửi lên cho tầng giao vận để kiểm tra xem có ứng dụng nào muốn nhận các thông điệp từ cổng broadcast. Nếu không ứng dụng nào lắng nghe, thông điệp bị phá hủy và không vượt qua được tầng giao vận. Điều này nghĩa là mỗi host cần một số chu kỳ của CPU cho dù thông điệp broadcast có cần hay không.

Multicast giải quyết vấn đề này, bằng cách gửi các thông điệp tới một nhóm các nút chứ không phải là tất cả các nút trong LAN. Thiết bị giao tiếp mạng có thể phát hiện xem hệ thống có cần quan tâm đến thông điệp hay không bằng cách phân tích địa chỉ broadcast mà không cần sự trợ giúp của CPU.

3.2. Các giao thức Internet

3.2.1. Giao thức truyền tệp tin –FTP (File Transfer Protocol)

FTP được sử dụng để tải các tệp lên server, và tải về các tệp từ server. Nó là một giao thức mức ứng dụng, dựa trên nền tảng của giao thức TCP. Ứng dụng client cung cấp một giao diện người dùng và tạo ra một yêu cầu FTP tương ứng với yêu cầu của người dùng cùng với đặc tả của FTP. Lệnh FTP được gửi tới ứng dụng server thông qua giao thức TCP/IP, trình thông dịch trên FTP phải thông dịch lệnh FTP tương ứng. Tùy thuộc vào lệnh FTP, một danh sách các tệp hoặc một tệp từ hệ thống tệp của server được trả về cho client trong đáp ứng của FTP.

Giao thức FTP có các đặc trưng sau:

- Truyền dữ liệu với độ tin cậy cao thông qua giao thức TCP
- Cho phép truy xuất vô danh hoặc xác thực người dùng với username và password
- Các tệp tin được truyền đi dưới dạng mã ASCII hoặc dữ liệu nhị phân

Các lệnh FTP có thể được nhóm thành các loại sau:

- Các lệnh điều khiển việc truy xuất

Các lệnh điều khiển việc truy nhập xác định tên người dùng và mật khẩu, các chế độ thiết lập có thể được thiết lập lại (REIN), và liên kết có thể được kết thúc (QUIT).

- Các lệnh truyền tham số

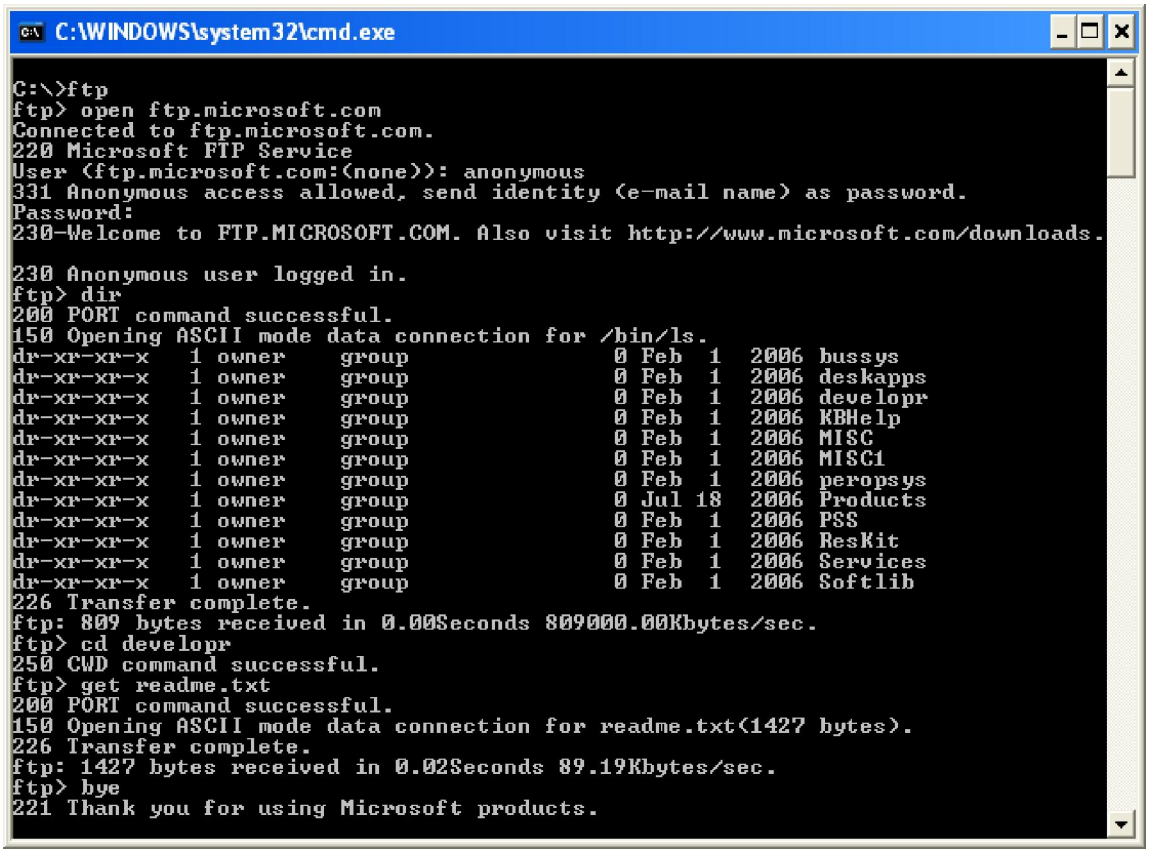
Truyền FTP có thể được cấu hình với các lệnh tham số truyền. Thay đổi việc truyền dữ liệu từ ASCII thành nhị phân, nén dữ liệu, thay đổi các cổng để gửi dữ liệu được hỗ trợ bởi các lệnh này.

- Các lệnh dịch vụ FTP

Sao chép các tệp tin từ server (RETR), sao chép các tệp tin lên server (STOR), xóa các tệp (DELE), thay đổi các tên tệp tin (RNTO), tạo các thư mục (MKD), và yêu cầu liệt kê một danh sách các tệp.

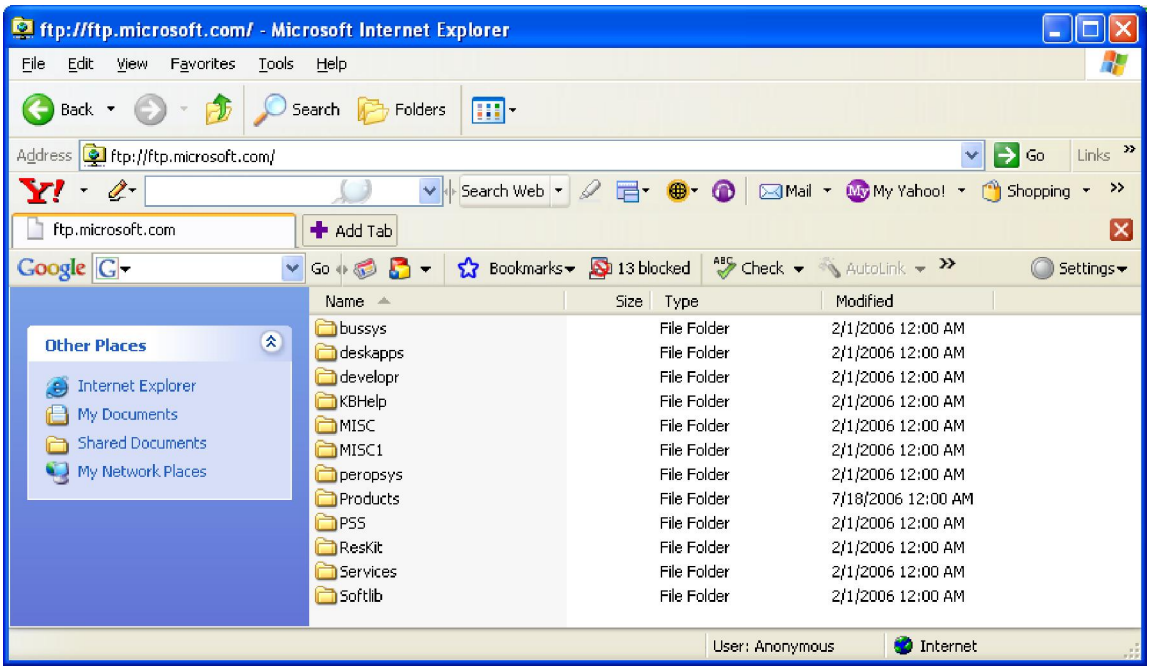
FTP Client

Cách tốt nhất để làm quen với giao thức FTP là bằng cách sử dụng tiện ích ftp như dưới đây. Chương trình ftp hoạt động thông qua dấu nhắc lệnh ftp> cho phép chúng ta nhập vào các lệnh. Các lệnh này khác với các lệnh của giao thức FTP-ta có thể thấy được chúng bằng cách nhập vào ?. Dưới đây, ta sẽ nhập vào lệnh open ftp.microsoft.com để tạo ra một liên kết tới host ftp.microsoft.com. Nhập vào username là anonymous. Đáp ứng 230 chỉ ra liên kết đã được thiết lập.



Hình 1.7

Một client FTP khác có trong trình duyệt Microsoft Internet Explorer



Hình 1.8

3.2.2. HTTP-Giao thức truyền siêu văn bản (Hypertext Transfer Protocol)

HTTP là một giao thức được sử dụng bởi các ứng dụng web. HTTP là một giao thức có độ tin cậy cao, được cài đặt dựa trên nền giao thức TCP. Tương tự như FTP, HTTP cũng được sử dụng để truyền các tệp tin qua mạng. Tuy nhiên, không giống với FTP, nó có các đặc trưng như đệm dữ liệu, định danh các ứng dụng client, hỗ trợ cho các định dạng kèm theo khác, như MIME,…Những đặc trưng này có trong header HTTP.

3.2.3. HTTPS-HTTP over SSL (Secure Socket Layer)

Nếu có yêu cầu trao đổi dữ liệu mật với một webserver, người ta sử dụng giao thức HTTPS. HTTPS là một sự mở rộng của giao thức HTTP và các nguyên tắc đã được thảo luận ở mục trước vẫn được áp dụng ở đây. Tuy nhiên cơ chế thì hoàn toàn khác, HTTPS sử dụng lớp Socket bảo mật SSL(Secure Socket Layer) được phát triển bởi Netscape. SSL ở tầng trên của giao thức TCP và bảo mật thông tin được truyềntrên mạng bằng cách sử dụng nguyên tắc mã hóa công khai.

3.3. Các giao thức E-mail

Có một số giao thức sử dụng cho e-mail phổ biến như sau

- ***SMTP-Simple Mail Transfer Protocol***

SMTP là một giao thức để gửi và nhận các e-mail. Nó có thể được sử dụng để gửi e-mail giữa client và server sử dụng cùng giao thức giao vận, hoặc để gửi e-mail giữa các server sử dụng các giao thức giao vận khác nhau. SMTP có khả năng chuyển tiếp các thông điệp thông qua các môi trường dịch vụ giao vận. SMTP không cho phép chúng ta đọc các thông điệp từ một mail server.

- ***POP3-Post Office Protocol***

POP3 được thiết kế cho các môi trường không được liên kết. Trong các môi trường không duy trì liên kết thường trực với mail server, ví dụ, trong các môi trường trong đó thời gian liên kết lâu. Với POP3, client có thể truy xuất tới server và tìm kiếm các thông điệp mà server hiện đang nắm giữ. Khi các thông điệp được tìm kiếm từ client, chúng thường bị xóa khỏi server, mặc dù điều này là không cần thiết.

- ***IMAP-Inernet Message Access Protocol***

Giống như POP3, IMAP được thiết kế để truy xuất tới các mail trên một mail server. Tương tự như các client POP3, một client IMAP có thể có chế độ offline. Không giống như các client POP3, các client IMAP có các khả năng lớn hơn trên chế độ online, như tìm kiếm các header, các đoạn mail, tìm kiếm các thông điệp cụ thể trên các server, và thiết lập các cờ như cờ trả lời. Về căn bản, IMAP cho phép các client làm việc trên các hộp thư ở xa như là các hộp thư cục bộ.

- ***NNTP-Network News Transfer Protocol***

NNTP là giao thức tầng ứng dụng để gửi, chuyển tiếp, và tìm kiếm các thông điệp tạo nên một phần của các cuộc thảo luận nhóm tin. Giao thức này cung cấp khả năng truy cập tới một server tin tức để tìm kiếm các thông điệp có chọn lọc và hỗ trợ cho việc truyền thông điệp từ server tới server.

3.4. Một số giao thức ứng dụng khác

Có hai giao thức ứng dụng thú vị khác là: SNMP và Telnet

SNMP (Simple Network Management Protocol) cho phép quản lý các thiết bị trên mạng. Có các thông tin như các biến đếm hiệu năng từ các thiết bị SNMP quản lý các thiết bị một cách hiệu quả bằng cách sử dụng các báo chuông báo hiệu được kích hoạt bởi các vấn đề về hiệu năng và lỗi, và cho phép cấu hình các thiết bị.

Một tác tử SNMP được gắn với một thiết bị mạng cụ thể sẽ có một cơ sở dữ liệu MIB (Management Information Base) bao gồm tất cả các thông tin có thể kiểm soát về thiết bị đó theo phương pháp hướng đối tượng. Một client SNMP truy xuất thông tin trong cơ sở dữ liệu bằng cách gửi các yêu cầu GET. Ngược lại, yêu cầu SET được sử dụng để cấu hình cơ sở dữ liệu MIB.

Trong những trường hợp có lỗi hoặc có các vấn đề về hiệu năng, tác tử SNMP gửi các thông điệp tới SNMP client.

4. Soket

Socket là một phương pháp để thiết lập kết nối truyền thông giữa một chương trình yêu cầu dịch vụ và một chương trình cung cấp dịch vụ trên mạng LAN, WAN, hay Internet và đôi khi là giữa các tiến trình trong cùng một máy tính. Thông tin của một Socket bao gồm địa chỉ IP và số hiệu cổng.

5. Dịch vụ tên miền

Các địa chỉ IP viết dưới dạng 4 nhóm bit không dễ nhớ một chút nào, vì vậy có người ta đã đưa ra một hệ thống tương đương dễ nhớ hơn đối với người sử dụng. Do các tên miền này là duy nhất, nên hệ thống tên miền được sử dụng để hỗ trợ hệ thống tên có phân cấp

Như ta đã biết, tiền thân của mạng Internet là mạng Arpanet của Bộ quốc phòng Mỹ. Thời kỳ đó số máy tính ở mức đủ để liệt kê chúng trong một tệp tin văn bản và lưu trên từng máy kết nối vào mạng. Thông tin trong tập tin này bao gồm địa chỉ IP và hostname. Tuy nhiên do quy mô của mạng ngày càng mở rộng người ta cần có các máy tính chuyên dụng để lưu trữ và phân giải tên miền. Các máy tính có chức năng như vậy được gọi là Máy chủ DNS. Ví dụ www.microsoft.com, www.bbc.co.uk. Các tên này không bắt buộc phải có ba phần, nhưng việc đọc bắt đầu từ phải sang trái, tên bắt đầu với miền mức cao. Các miền mức cao là các tên nước cụ thể hoặc tên các tổ chức và được định nghĩa bởi tổ chức IANA. Các tên miền cấp cao được liệt kê trong bảng sau. Trong những năm gần đây, một số tên miền cấp cao mới được đưa vào.

Tên miền	Mô tả
.aero	Công nghiệp hàng không
.biz	Doanh nghiệp
.com	Các tổ chức thương mại

.coop	Các quan hệ hợp tác
.info	Không ràng buộc về sử dụng
.museum	Các viện bảo tàng
.name	Các tên cá nhân

Bảng 1.7

Tên miền	Mô tả
.net	Các mạng
.org	Các tổ chức phi chính phủ
.pro	Các chuyên gia
.gov	Chính phủ Hoa Kỳ
.edu	Các tổ chức giáo dục
.mil	Quân đội Mỹ
.int	Các tổ chức được thành lập bởi các hiệp ước quốc tế giữa các chính phủ.

Bảng 1.8

Ngoài ra, còn có các tên miền cho các quốc gia

Tên miền	Mô tả
.at	Autralia
.de	Germany
.fr	France
.uk	United Kingdom
.vn	Vietnam

Bảng 1.9

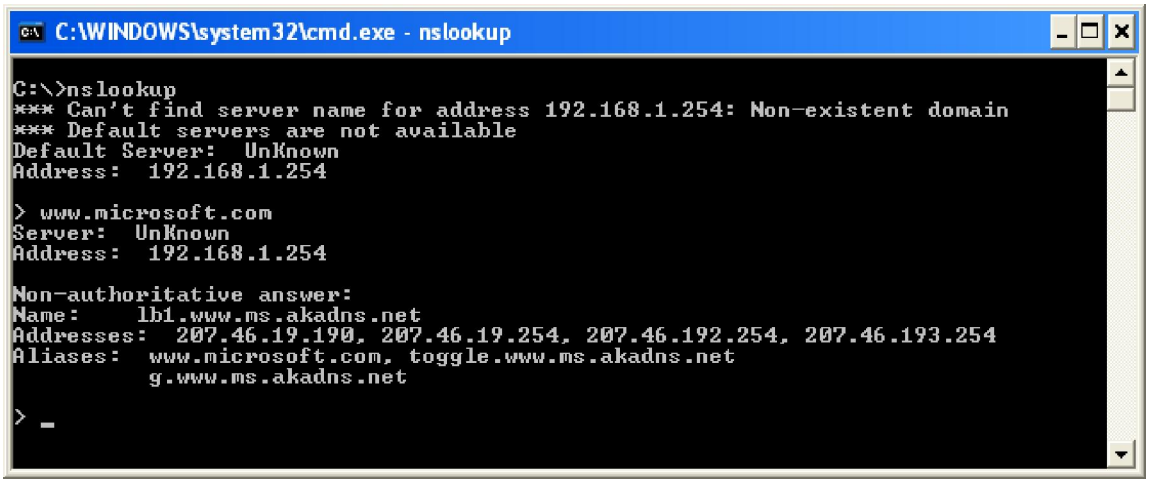
5.1. Các server tên miền

Các hostname được phân giải bằng cách sử dụng các server DNS (Domain Name Service). Các server này có một cơ sở dữ liệu các hostname và các bí danh ánh xạ các tên thành địa chỉ IP. Ngoài ra, các DNS cũng đăng ký thông tin cho các Mail Server, các số ISDN, các tên hòm thư, và các dịch vụ.

Trong Windows, chính các thiết lập TCP/IP xác định server DNS được sử dụng để truy vấn. Lệnh ipconfig/all chỉ ra các server DNS đã được thiết lập và các thiết lập cấu hình khác. Khi kết nối với một hệ thống ở xa sử dụng hostname, trước tiên server DNS được truy vấn để tìm địa chỉ IP. Trước tiên, DNS kiểm tra trong bộ cơ sở dữ liệu của riêng nó và bộ nhớ cache. Nếu thất bại trong việc phân giải tên, server DNS truy vấn server DNS gốc.

5.2. Nslookup

Dịch vụ tên miền (Domain Name Service) Là tập hợp nhiều máy tính được liên kết với nhau và phân bố rộng trên mạng Internet. Các máy tính này được gọi là name server. Chúng cung cấp cho người dùng tên, địa chỉ IP của bất kỳ máy tính nào nối vào mạng Internet hoặc tìm ra những name server có khả năng cung cấp thông tin này.



Hình 1.9

Cơ chế truy tìm địa chỉ IP thông qua dịch vụ DNS

Giả sử trình duyệt cần tìm tập tin hay trang Web của một máy chủ nào đó, khi đó cơ chế truy tìm địa chỉ sẽ diễn ra như sau:

1. Trình duyệt yêu cầu hệ điều hành trên client chuyển hostname thành địa chỉ IP.
2. Client truy tìm xem hostname có được ánh xạ trong tập tin localhost, hosts hay không?

-Nếu có client chuyển đổi hostname thành địa chỉ IP và gửi về cho trình duyệt.

-Nếu không client sẽ tìm cách liên lạc với máy chủ DNS.

3. Nếu tìm thấy địa chỉ IP của hostname máy chủ DNS sẽ gửi địa chỉ IP cho client.
4. Client gửi địa chỉ IP cho trình duyệt.
5. Trình duyệt sử dụng địa chỉ IP để liên lạc với Server.
6. Quá trình kết nối thành công. Máy chủ gửi thông tin cho client.

6. Internet

Trong chương này chúng ta đã đề cập tới nhiều công nghệ cơ sở: phần cứng, các giao thức, và các hệ thống tên miền. Trong phần này chúng ta sẽ thảo luận các vấn đề thú vị khác như:

- Intranet và Extranet
- Firewall và Web Proxy
- Các dịch vụ Web

6.1. Intranet và Extranet

Một intranet có thể sử dụng các công nghệ TCP/IP tương tự như với Internet. Sự khác biệt là intranet là một mạng riêng, trong đó tất cả mọi người đều biết nhau. Intranet

không phục vụ cho việc truy xuất chung, và một số dữ liệu cần phải được bảo vệ khỏi những truy xuất từ bên ngoài.

Một extranet là một mạng riêng giống như intranet nhưng các extranet kết nối nhiều Intranet thuộc cùng một công ty hoặc các công ty đối tác thông qua Internet bằng cách sử dụng một tunnel. Việc tạo ra một mạng riêng ảo trên Internet tiết kiệm chi phí nhiều cho công ty so với việc thuê riêng một đường truyền để thiết lập mạng.

6.2. Firewall

Có những kẻ phá hoại trên mạng Internet!. Để ngăn chặn chúng, người ta thường thiết lập các điểm truy cập tới một mạng cục bộ và kiểm tra tất cả các luồng truyền tin vào và ra khỏi điểm truy nhập đó. Phần cứng và phần mềm giữa mạng Internet và mạng cục bộ, kiểm tra tất cả dữ liệu vào và ra, được gọi là firewall.

Firewall đơn giản nhất là một bộ lọc gói tin kiểm tra từng gói tin vào và ra khỏi mạng, và sử dụng một tập hợp các quy tắc để kiểm tra xem luồng truyền tin có được phép vào ra khỏi mạng hay không. Kỹ thuật lọc gói tin thường dựa trên các địa chỉ mạng và các số hiệu cổng.

6.3. Proxy Server

Khái niệm proxy có liên quan đến firewall. Nếu mốt một firewall ngăn chặn các host trên mạng liên kết trực tiếp với thế giới bên ngoài. Một máy bị ngăn kết nối với thế giới bên ngoài bởi một firewall sẽ yêu cầu truy xuất tới một trang web từ một proxy server cục bộ, thay vì yêu cầu một trang web trực tiếp từ web server ở xa. Proxy server sau đó sẽ yêu cầu trang web từ một web server, và sau đó chuyển kết quả trở lại cho bên yêu cầu ban đầu. Các proxies cũng được sử dụng cho FTP và các dịch vụ khác. Một trong những ưu điểm bảo mật của việc sử dụng proxy server là các host bên ngoài chỉ nhìn thấy proxy server. Chúng không biết được các tên và các địa chỉ IP của các máy bên trong, vì vậy khó có thể đột nhập vào các hệ thống bên trong.

Trong khi các firewall hoạt động ở tầng giao vận và tầng internet, các proxy server hoạt động ở tầng ứng dụng. Một proxy server có những hiểu biết chi tiết về một số giao thức mức ứng dụng, như HTTP và FTP. Các gói tin đi qua proxy server có thể được kiểm tra để đảm bảo rằng chúng chứa các dữ liệu thích hợp cho kiểu gói tin. Ví dụ, các gói tin FTP chứa các dữ liệu của dịch vụ telnet sẽ bị loại bỏ.

Vì tất cả các truy nhập tới Internet được chuyển hướng thông qua proxy server, vì thế việc truy xuất có thể được kiểm soát chặt chẽ. Ví dụ, một công ty có thể chọn giải pháp phong tỏa việc truy xuất tới www.playboy.com nhưng cho phép truy xuất tới www.microsoft.com

Chương 2

Giới thiệu ngôn ngữ lập trình Java

1. Giới thiệu công nghệ Java

1.1 Lịch sử phát triển

Lịch sử phát triển của Java bắt đầu năm 1991 khi SUN tiến hành các dự án lập trình cho vi xử lý dùng trong các thiết bị điện tử khác nhau. C++ không đáp ứng được các yêu cầu này vì C++ cho đem mã nguồn từ máy này sang máy khác nhưng sau khi biên dịch lại hoàn toàn phụ thuộc vào từng bộ vi xử lý cụ thể. Trong khi đó bộ vi xử lý dùng trong các thiết bị điện tử rất đa dạng và có vòng đời khá ngắn ngủi. Nếu ta thay đổi bộ xử lý dẫn đến cần phải thay đổi trình biên dịch C++, điều này gây lên tốn kém. SUN đã thiết kế một ngôn ngữ lập trình mới có tính khả chuyển cao hơn đó chính là Java. Java là tên địa phương nơi xuất xứ của một loại cà phê ngon nổi tiếng.

Java được chính thức công bố năm 1995 và ngay lập tức đã tạo lên một trào lưu mới trên toàn thế giới và từ đó đến nay vẫn tạo được sức cuốn hút mạnh mẽ. Bởi vì Java không chỉ đơn thuần là một ngôn ngữ lập trình mà nó là giải pháp cho nhiều vấn đề.

1.2. Cấu trúc của máy ảo Java (Java Virtual Machine)

Chương trình ứng dụng hoạt động bằng cách sử dụng các đối tượng của Java (Java Object). Máy ảo Java tạo thành một cầu nối giữa trình ứng dụng viết bằng Java và hệ điều hành.

Chương trình Java: tập hợp các đối tượng
Máy ảo Java
Hệ điều hành

Máy ảo Java bao gồm các thành phần sau :

- Trình nạp lớp (Class Loader): đọc bytecode từ đĩa hoặc từ kết nối mạng.
- Trình kiểm tra lớp (Class Verifier): Kiểm tra các lớp sẽ không sinh ra các lỗi ảnh hưởng tới hệ thống khi thực thi.
- Trình thực thi (Execution Unit): sẽ thực hiện các lệnh được quy định trong từng bytecode.

Trong bộ công cụ Java, tệp tin java.exe chính là máy ảo Java.

1.3 Các đặc trưng của Java

- ***Java là một môi trường độc lập (Independent Platform)***

Do cấu trúc của Java nên ta có thể soạn thảo chương trình trên bất kỳ hệ thống nào. Sau khi đã được biên dịch thành tệp tin lớp (*.class) ứng dụng có thể thực thi ở bất kỳ hệ thống nào. Đó là đặc tính mà các ngôn ngữ khác không có.

- ***Java là một ngôn ngữ lập trình hướng đối tượng thuần túy (Pure Object Oriented Programming)***

Java là một ngôn ngữ lập trình hướng đối tượng thuần túy, mọi thứ trong Java đều là đối tượng.

- ***Java là một ngôn ngữ có tính khả chuyển (Portibility)***

Java có tính khả chuyển đối với cả mã nguồn và bản thân mã biên dịch (bytecode)

- ***Java là môi trường xử lý phân tán (Distributed Enviroments)***

Bytecode không phụ thuộc vào hệ thống vì vậy bytecode có thể nằm phân tán trên mạng. Việc liên kết với thư viện chỉ được thực hiện vào lúc chạy chương trình do vậy mã byte thường gọn nhẹ. Chương trình Java được nạp dần một cách linh hoạt nên không gây quá tải cho mạng. Ngoài ra, Java còn cho phép xử lý đa tuyến đoạn. Cơ chế truyền thông điệp thuận tiện cho việc tổ chức mạng.

- ***Java là môi trường an toàn***

Khi phát triển các ứng dụng phân tán thì một trong những vấn đề được quan tâm hàng đầu là an toàn hệ thống. Java được thiết kế để đảm bảo an toàn cho người dùng Java trên mạng. Java có bốn tầng bảo an:

Tầng 1: Mức ngôn ngữ và trình biên dịch. Java không có kiểu con trỏ. Trình biên dịch kiểm tra kiểu rất chặt chẽ. Mọi chuyển đổi kiểu đều phải được thực hiện một cách tường minh. Trình biên dịch Java từ chối sinh ra mã byte nếu mã nguồn không tuân thủ nghiêm ngặt các quy tắc an toàn.

Tầng 2: Trình nạp lớp (Class Loader) Có khả năng phân biệt những lớp đến từ mạng và những lớp nạp từ hệ thống. Nhờ khả năng phân biệt như vậy lớp được nạp qua mạng được khống chế chặt chẽ, không được phép thực hiện các thao tác mức thấp.

Tầng 3: Trình kiểm tra mã byte. Trình này sẽ kiểm tra mã byte vào lúc chạy chương trình bảo đảm chương trình Java đã được biên dịch một cách đúng đắn. Khi thực hiện sẽ không gây lỗi ảnh hưởng tới hệ thống cũng như không đụng chạm tới dữ liệu riêng tư trên máy khách.

Tầng 4*:* Trình bảo an. Kiểm tra mã byte vào lúc chạy nhằm bảo đảm mã đang xét không vi phạm qui tắc an toàn đã được thiết lập. Các thao tác của ứng dụng được xem là có khả năng gây nguy hiểm như đọc, xóa tệp đều phải được Trình bảo an cho phép.

- ***Java cung cấp cho người lập trình một thư viện khủng lồ***

Java cung cấp cho người lập trình một thư viện khổng lồ các hàm chuẩn, gọi là core API. Các hàm chuẩn này được đặt trong các gói.

- ***Java có cơ chế quản lý bộ nhớ tự động***

Quản lý bộ nhớ là một trong những vấn đề phức tạp đối với C và C^{++}. Khi thực hiện chương trình người lập trình chịu trách nhiệm khởi tạo các vùng nhớ, sau khi dùng phải giải phóng các vùng nhớ này. Chỉ cần một lỗi nhỏ có thể làm cạn kiệt tài nguyên dẫn đến treo hệ thống.

Java đã loại bỏ gánh nặng này cho người lập trình. Các vùng nhớ được tự động giải phóng nếu như nó không tham chiếu đến bất kỳ đối tượng nào đang hoạt động

- ***Chi phí phát triển ứng dụng bằng Java thấp***

Khi phát triển ứng dụng dựa trên công nghệ Java thì sẽ có rất nhiều công cụ phát triển và dịch vụ được cung cấp miễn phí.

1.4. Các ấn bản Java

- ***J2SE (Java 2 Platform, Second Edition)***

Đây là ấn bản chuẩn, bao gồm một môi trường thời gian chạy và một tập hợp các API để xây dựng một loạt các ứng dụng khác nhau từ applet, cho tới các ứng dụng độc lập chạy trên các nền khác nhau, ứng dụng cho client cho các ứng dụng doanh nghiệp khác nhau.

- ***J2EE (Java 2 Platform, Enterprise Edition (J2EE)***

J2EE là nền tảng để xây dựng các ứng dụng phía server.

- ***J2ME (Java 2 Platform, Micro Edition)***

Ấn bản này cho phép xây dựng các ứng dụng Java cho các "vi thiết bị" (các thiết bị có màn hình hiển thị và hỗ trợ bộ nhớ tối thiểu, như điện thoại di động và các thiết bị trợ giúp cá nhân).

1.5. Công cụ phát triển

SUN cung cấp một số tiện ích cho phép biên dịch bắt lỗi và tạo tài liệu cho một ứng dụng Java. JDK bao gồm:

- javac: Bộ biên dịch để chuyển mã nguồn thành bytecode.
- java: Bộ thông dịch để thực thi các ứng dụng Java trực tiếp từ tập tin lớp.
- appletviewer: Thực thi Java Applet từ tài liệu html.

1.6. Các kiểu ứng dụng trong Java

Có hai kiểu ứng dụng

- Ứng dụng độc lập (Standalone Applicaiton) Cho phép lập trình như các ngôn ngữ lập trình khác như Pascal, C.
- Ứng dụng ký sinh (Applet) Cho phép tạo ra chương trình liên kết với các văn bản Web và được khởi động bởi trình duyệt hỗ trợ Java.

Để thấy được sự khác biệt giữa hai kiểu ứng dụng nói trên chúng ta có thể xem những sự khác biệt về đặc trưng của hai kiểu ứng dụng ở bảng dưới đây:

	Ứng dụng độc lập	**Java Applet**
Khai báo	Là lớp con của bất kỳ lớp nào trong các gói thư viện các lớp	Phải là lớp con của Applet
Giao diện đồ họa	Tùy chọn	Do trình duyệt Web quyết định
Yêu cầu bộ nhớ	Bộ nhớ tối thiểu	Bộ nhớ dành cho cả trình duyệt và applet đó
Cách nạp chương trình	Nạp bằng dòng lệnh	Thông qua trang Web
Dữ liệu vào	Thông qua các tham số trên	Các tham số đặt trong tệp

	dòng lệnh	HTML gồm địa chỉ, kích th- ớc của trình duyệt
Cách thức thực hiện	Mọi hoạt động được bắt đầu và kết thúc ở main() nh- trong C/C++	Gọi các hàm: init(), start(), stop(), destroy(), paint()
Kiểu ứng dụng	Ứng dụng trên các máy chủ Server. Công cụ phát triển phần mềm, - ứng dụng trên các máy khách	Các ứng dụng trên Web

Bảng 2.1

1.7. Cài đặt chương trình dịch Java và các công cụ

Để cài đặt Java J2SDK ta cần tải về bộ J2SDK tại trang http://www.sun.com , trang này cập nhật những phiên bản mới nhất của Java. Sau đó ta tải về bộ công cụ soạn thảo Edit Plus tại địa chỉ http://www.editplus.com .

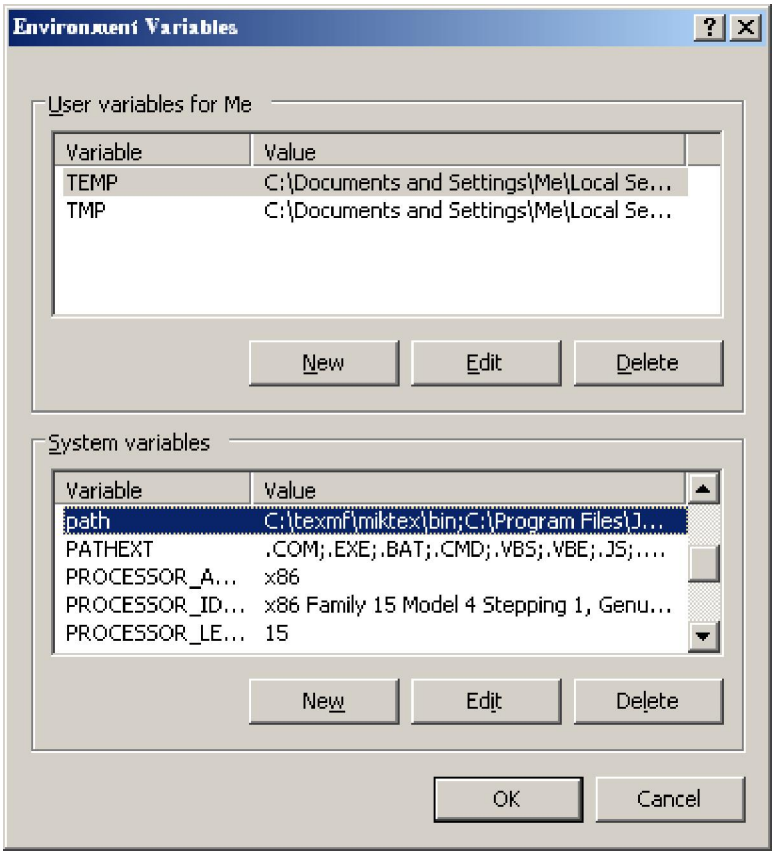
Các bước cài đặt Java

Bước 1: Cài đặt J2SDK. Sau khi cài đặt xong, Java được cài đặt tại thư mục: C:\Program Files\Java. Trong thư mục này có hai thư mục con là: jdk1.5.0_01 và jre1.5.0_01.

- Công cụ phát triển nằm trong thư mục:
- *C:\Program Files\Java\jdk1.5.0_01\bin*: Bao gồm các công cụ cho phép ta phát triển, thực thi, gỡ rối và soạn thảo các chương trình được viết bằng ngôn ngữ lập trình Java.
- Môi trường thời gian chạy (*Runtime Environment*)
- Nằm trong thư mục con JRE. Nó bao gồm một máy ảo Java, các thư viện lớp, và các tệp tin hỗ trợ việc xử lý các chương trình được viết bằng ngôn ngữ lập trình Java.
- Các thư viện
- Nằm trong thư mục con lib. Các thư viện lớp và các tệp tin bổ trợ cần cho các công cụ phát triển...
- Các ứng dụng và applet demo
- Nằm trong thư mục demo. Thư mục này bao gồm các ví dụ, mã nguồn lập trình cho Java/

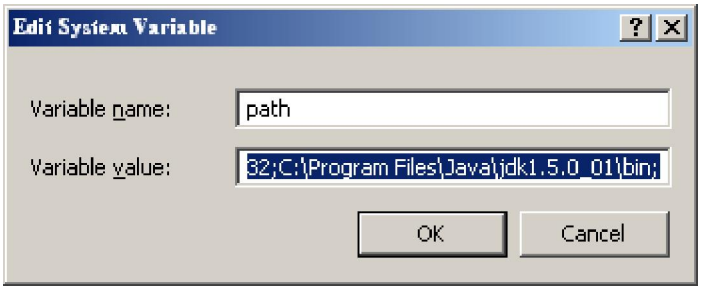
Bước 2: Thiết lập các biến môi trường
Biến môi trường path

Trên desktop ta kích chuột phải vào biểu tượng My Computer, sau đó chọn thẻ Advanced, nháy chuột vào nút lệnh Enviroment Variable, xuất hiện cửa sổ cho phép ta thiết lập biến môi trường như sau:



Hình 2.1

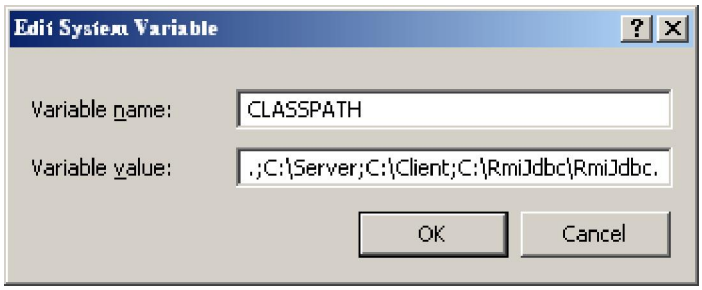
Nháy vào nút lệnh New để nhập vào thông tin về biến môi trường



Hình 2.2

Sau đó nháy chuột vào nút lệnh OK để hoàn thành

Thiết lập biến môi trường classpath cũng tiến hành tương tự



Hình 2.3

Bước 3:

Cài đặt chương trình soạn thảo Edit Plus.

Bước 4:

Tạo lập các công cụ biên dịch và thực thi chương trình trong Edit Plus. Giao diện của chương trình soạn thảo như sau:

1.8. Một số ví dụ mở đầu

Ví dụ 1: Tạo chương trình Java cho phép nhập một dòng ký tự từ đối dòng lệnh và hiển thị xâu ký tự đó lên trên màn hình:

```
class  ViDu1
{
        public static void main(String[] args)
        {
                System.out.println(args[0]);
        }
}
```

Biên dịch chương trình

C:\>javac ViDu1.java

Thực hiện chương trình

C:\>java ViDu1 "Lam quen voi ngon ngu lap trinh Java"

Kết quả in ra là:

Lam quen voi ngon ngu lap trinh Java

Chương trình trên ta nhận thấy ngay đây là một chương trình ứng dụng độc lập đơn giản. Chương trình thực hiện một tác vụ hết sức đơn giản là nhận một xâu ký tự được truyền vào từ đối dòng lệnh sau đó in xâu ký tự lên màn hình. Đối dòng lệnh là một mảng xâu ký tự String[] args. Khi ta thực thi chương trình trên, args[0] được gán bằng xâu ký tự "Lam quen voi ngon ngu lap trinh Java".

Ví dụ 2: Tạo một applet hiển thị một xâu ký tự lên màn hình

```
import java.applet.*;
import java.awt.*;
```

```
public class  ViDu2 extends Applet
{
        public void paint(Graphics g)
        {
                g.drawString("Chuc mung cac ban da thanh cong voi vi du thu 2",30,30);
        }

}
```

Biên dịch applet

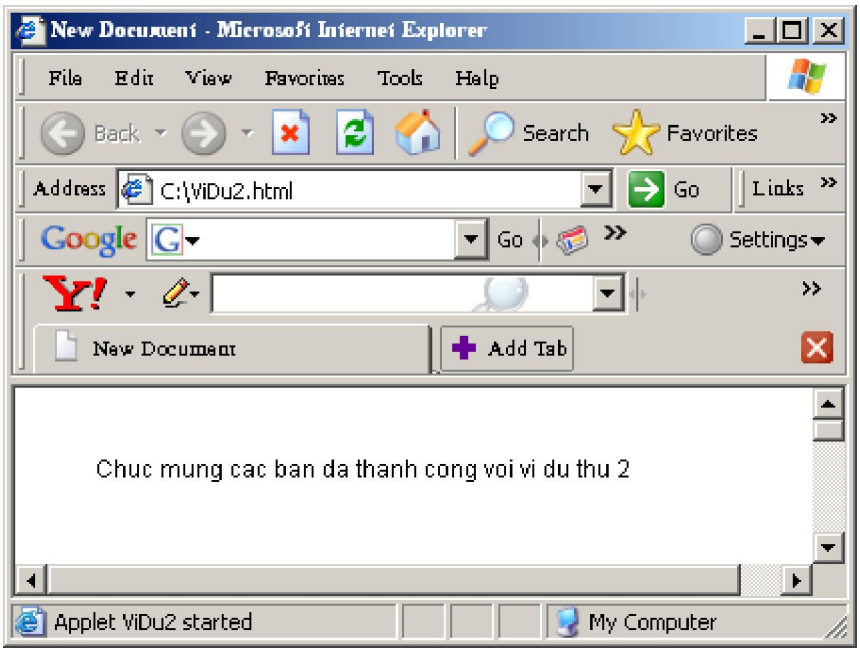
C:\>javac ViDu2.java

Tạo một trang html để nhúng applet

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE> New Document </TITLE>
<META NAME="Generator" CONTENT="EditPlus">
<META NAME="Author" CONTENT="">
<META NAME="Keywords" CONTENT="">
<META NAME="Description" CONTENT="">
</HEAD>

<BODY>
<applet code="ViDu2.class" width=400 height=400>
</BODY>
</HTML>
```

Trong tài liệu trên ta để ý thấy dòng <applet code="ViDu2.class" width=400 height=400> cho phép ta nhúng một applet trong một văn bản web.

Để thực thi applet ta chỉ cần hiển thị trang ViDu2.html trên trình duyệt. Kết quả được thể hiện ở hình dưới đây:



Hình 2.4

2. Ngôn ngữ lập trình Java

2.1 Cấu trúc tệp của một chương trình Java

Một tệp tin chương trình Java có thể có các phần được đặc tả như sau:

1. Định nghĩa một gói là tùy chọn thông qua định danh của gói (package).
2. Một số lệnh nhập import (0 hoặc nhiều).
3. Một số định nghĩa lớp và interface có thể định nghĩa theo thứ tự bất kỳ

Ví dụ giả sử có tệp nguồn Java như sau:

```
// Phần 1: Tùy chọn // Định nghĩa
gói package GoiNhaTruong;
```

```
// Phần 2: (0 hoặc nhiều hơn) //
các gói cần sử dụng import
java.io.*;
```

```
// Phần 3: (0 hoặc nhiều hơn) //
Định nghĩa các lớp và các
interface public class NewApp{
...} class C1{...} interface I1{...} //
class Cn {...} interface Im{...}
```

Hình 2.5

Lưu ý:

- Tệp chương trình Java luôn có tên trùng với tên của một lớp công khai (lớp chứa hàm main() nếu là ứng dụng độc lập) và có phần mở rộng là .java.
- Tệp NewApp.java nếu là chương trình ứng dụng độc lập thì phải có một lớp có tên là NewApp và lớp này phải có phương thức main(). Phương thức này luôn có dạng: public static void main(String args[]){ // Nội dung cần thực hiện của chương trình ứng dụng }
- Khi dịch (javac) thì mỗi lớp trong tệp chương trình sẽ được dịch thành byte code và được ghi thành tệp riêng có tên trùng với tên của lớp và có đuôi .class. Những lớp này sẽ được nạp vào chương trình lúc thông dịch và thực hiện theo yêu cầu.
- Trong chương trình Java không có các khai báo biến, hàm tách biệt khỏi lớp và chỉ có các khai báo và định nghĩa các lớp, interface. Như thế chương trình Java được xem như là tập các lớp, interface và các đối tượng của chúng trao đổi thông điệp với nhau (bằng các lời gọi hàm) để thực hiện các nhiệm vụ của ứng dụng.

2.2. Định danh, kiểu dữ liệu và khai báo biến

Định danh (Identifier) Tên gọi của các thành phần trong chương trình được gọi là định danh

Định danh thường được sử dụng để xác định biến, kiểu, phương thức, lớp.

Qui tắc cho định danh:

- Định danh là một dãy các ký tự gồm các chữ cái, chữ số và các ký tự khác: '_', $,...
- Định danh không bắt đầu bằng chữ số.
- Độ dài của định danh không giới hạn.
- Java phân biệt chữ hoa và chữ thường.

Qui ước đặt tên

- Định danh cho các lớp: chữ cái đầu của mỗi từ trong định danh đều viết hoa
- Ví dụ: MyClass, HocSinh, SocketServer, URLConnection,...
- Định danh cho các biến, phương thức, đối tượng: chữ cái đầu của mỗi từ trong định danh đều viết hoa trừ từ đầu tiên.

Ví dụ: hoTen (họ tên), namSinh (năm sinh), tinhTong (tính tổng).

Chú thích

Chú thích trong các chương trình để giải thích công việc hoặc cách thực hiện để người đọc dễ hiểu và tiện theo dõi.

- Chú thích trên một dòng

//Đây là chú thích dòng

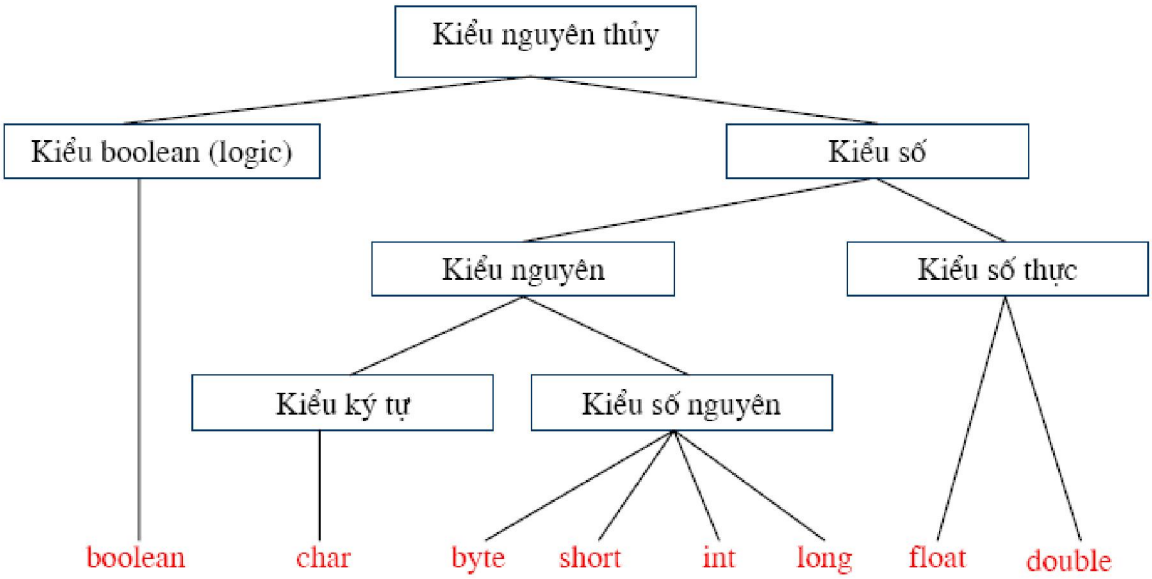
- Chú thích trên nhiều dòng

/* Đây là chú thích khối nhiều dòng */

2.3. Các kiểu dữ liệu nguyên thủy (primitive datatype)

Kiểu dữ liệu cơ bản định nghĩa sẵn được gọi là kiểu nguyên thủy

Kiểu nguyên thủy bao gồm các kiểu:



Hình 2.6

- Kiểu nguyên: char (ký tự), byte, short, int, long.
- Kiểu số thực: float, double.
- Kiểu logic: boolean.

Kiểu dữ liệu	Độ rộng
char	16
byte	8
short	16
int	32
long	64
float	32
doube	64

Bảng 2.2

Chú ý:

- Các giá trị kiểu nguyên thủy không phải là đối tượng.
- Mỗi kiểu dữ liệu có miền xác định và các phép toán xác định trên nó.
- Mỗi kiểu dữ liệu nguyên thủy có một lớp gói (wrapper class) để sử dụng các kiểu nguyên thủy như là các đối tượng.

Kiểu dữ liệu	Lớp gói
char	Char
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double

Bảng 2.3

2.4. Khai báo các biến

a. Các loại biến trong Java

- Các biến thành phần: là các thành phần của lớp và được khởi tạo giá trị mỗi khi một đối tượng của lớp được tạo ra.
- Ví dụ:

URL u ;

HocSinh hs = new HocSinh("Tuan Anh");

- Các biến tham chiếu đối tượng (Object Reference) là biến được sử dụng để xử lý các đối tượng.
- Các biễn tĩnh (static variable) là biến của lớp đại diện cho cả lớp.
- Các biến cục bộ: là biến được khai báo trong các phương thức và trong các khối.

b. Khai báo biến

kieu_du_lieu bien;

Ví dụ:

```
int i;
char b,c;
float giaBan, dienTich;
```

c. Khởi tạo giá trị cho các biến

Các giá trị mặc định cho các biến thành phần

Kiểu dữ liệu	Giá trị mặc định
boolean	false
char	'\u000'
byte, short, int, long	0
float, double	+0.0F,+0.0D
tham chiếu đối tượng	null

Bảng 2.4

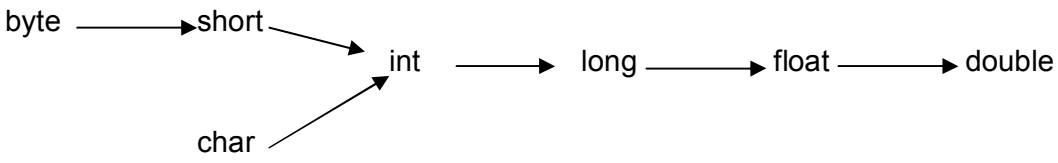
Qui tắc chuyển đổi kiểu trong Java

(<kieu>)<bieu_thuc>

Ví dụ

float f = (float)100.15D;

Mở rộng và thu hẹp kiểu



Ví dụ 1:Mở rộng kiểu

char c = 'A';

int k =c;

Ví dụ 2:Thu hẹp kiểu

int k =10;

char c=(char)k;

2.3. Các lệnh trong Java

Một lệnh có thể thay đổi trạng thái của máy tính: giá trị của các biến, các phần tử của mảng, nội dung của các tệp tin,…

2.3.1. Lệnh biểu thức

Một lệnh biểu thức là lệnh có dạng:

<Biểu thức>;

Nó được xử lý bằng cách đánh giá biểu thức mà không cần lưu lại giá trị tính toán.

2.3.2. Khối lệnh

Khối lệnh trong Java tương tự như khối lệnh trong C/C++, là những lệnh nằm trong cặp ngoặc mở { và đóng }.

Một khối lệnh là một dãy không hoặc nhiều lệnh hoặc các khai báo biến hoặc khai báo lớp theo bất kỳ thứ tự nào được đặt trong cặp ngoặc {}

{

S1;

S2;

…

Sn;

}

- Lệnh gán

  Ví dụ

  int a, b, c,d;

  d=b*b-4*a*c;

- Biểu thức điều kiện

Biểu thức điều kiện A?B:C trả về giá trị B nếu A có giá trị true, trả về giá trị C nếu A có giá trị false.

Ví dụ:

- *byte b;*
- *int i=b>=0?b:b+255;*

- Trong ví dụ trên thực hiện việc chuyển đổi các số nguyên kiểu byte có dấu về số nguyên kiểu int không có dấu. Nếu b lớn hơn hoặc bằng 0 thì I nhận giá trị là b, ngược lại I sẽ nhận giá trị là 255+b.

2.3.4. Các lệnh điều khiển rẽ nhánh chương trình

- *Lệnh if đơn giản*

  Cú pháp

  if<biểu_thức_đk> <câu_lệnh>

- *Lệnh if – else*

  Cú pháp

```
if <biểu_thức_đk>
        <câu_lệnh_1>;
else
        <câu_lệnh_2>;
```

Ví dụ: Viết chương trình nhập vào một dãy số nguyên từ đối dòng lệnh, sắp xếp dãy số đó và hiển thị dãy số sau khi sắp xếp lên màn hình.

```
class SapXep
{
    public static void main(String[] args)
    {
        int a[]=null;
        int i,j,tg;
        if(args.length>0)
        {
            a=new int[args.length];
        }


        for(i=0;i<args.length;i++)
        {
            a[i]=Integer.parseInt(args[i]);
            System.out.print(a[i]+" ");
        }
```

```
                System.out.println();

        for(i=0;i<a.length-1;i++)
                        for(j=i+1;j<a.length;j++)
                          if(a[i]>a[j])
                    {
                                    tg=a[i];
                                    a[i]=a[j];
                                    a[j]=tg;
                                }
                System.out.println("Day so sau khi sap xep la:");
                for(i=0;i<args.length;i++)  System.out.print(a[i]+" ");
          }
}
```

Biên dịch chương trình

```
C:\>javac SapXep.java
```

Thực hiện chương trình

```
C:\>java SapXep -2 3 1 4 -5 6 -10
-2 3 1 4 -5 6 -10
Day so sau khi sap xep la:
-10 -5 -2 1 3 4 6
```

Lệnh switch

Lệnh này cho phép rẽ nhánh theo nhiều nhánh tuyển chọn dựa trên các giá trị nguyên của biểu thức.

Cú pháp:

```
switch(<biểu_thức_nguyên>)
{
        case nhan_1:<câu_lệnh_1>;
        case nhan_2:<câu_lệnh_2>;
        case nhan_3:<câu_lệnh_3>;
        ...
        case nhan_n:<câu_lệnh_n>;
        default:<câu_lệnh>;
}
```

2.3.5. Lệnh lặp

Lệnh lặp cho phép một khối các câu lệnh thực hiện một số lần lặp lại.

- ***Lệnh while***

while (<Điều_kiện>) <Thân_chu_trình>

Chú ý:

Có thể <Thân_chu_trình> không được thực hiện lần nào nếu ngay từ đầu <Điều_kiện> có giá trị false.

<Điều_kiện> là biểu thức boolean.

Ví dụ: Lập chương trình in ra dãy số Fibonaci có các giá trị nhỏ hơn 50.

```
class Fibonaci
{
        public static void main(String[] args)
        {
                int lo=1;
                int hi =1;
                System.out.println(lo);
                while(hi<50)
                {
                        System.out.println(hi);
                        hi=lo+hi;
                        lo=hi-lo;
                }
        }
}
```

- ***Lệnh do – while***

Cú pháp

```
do
{
        //Các lệnh
}
while(<Điều_kiện>);
```

Chú ý:

Thân chu trình được thực hiện ít nhất một lần

- ***Lệnh for***

Cú pháp

```
for(<Biểu_thức_bắt_đầu>; <Điều_kiện_lặp>; <Biểu_thức_gia_tăng>)
                <Thân_chu_trình>
```

- <Biểu_thức_bắt_đầu>: Khai báo và gán các giá trị khởi đầu cho các biến điều khiển quá trình lặp.
- <Điều_kiện_lặp>:Biểu thức logic.

- ***Sự tương đương giữa vòng for và while***

```
{
        <Biểu_thức_bắt_đầu>;
        while(<Điều_kiện_lặp>)
    {
            <lệnh>;
            <Biểu_thức_gia_tăng>;
}
```

- ***Lệnh lặp vô hạn:***

Ta có thể sử dụng vòng for với cú pháp như sau làm vòng lặp vô hạn.

```
for(;;)<lệnh>
```

Vòng lặp vô hạn while:

```
while(true){
}
```

2.3.6. Các câu lệnh nhảy

Java hỗ trợ ba lệnh nhảy: break, continue, và return. Các lệnh này truyền điều khiển sang phần khác của chương trình.

- ***Lệnh break****:* Được dùng để kết thúc trình tự thực hiện lệnh trong lệnh switch hoặc được sử dụng để thoát ra khỏi một vòng lặp.

```
class BreakDemo
{
    public static void main(String[] args)
    {
        for(int i=0;i<20;i++){
            if(i==10)break;
            System.out.println("i="+i);
        }

    }
}
```

Chương trình này in ra kết quả là:

```
i=0
i=1
```

i=2

i=3

i=4

i=5

i=6

i=7

i=8

i=9

Như chúng ta có thể thấy, vòng lặp for được thiết kế để chạy với các giá trị từ 0 đến 9, lệnh break làm cho vòng lặp kết thúc sớm khi i bằng 10.

Lệnh break có thể được sử dụng trong bất kỳ vòng lặp Java nào, bao gồm cả vòng lặp vô hạn

- ***Lệnh continue***

Lệnh continue sẽ bỏ qua không xử lý một lệnh nào đó trong vòng lặp nhưng vẫn tiếp tục thực hiện phần còn lại của vòng lặp.

```
class ContinueDemo
{
        public static void main(String[] args)
        {
                for(int i=0;i<20;i++){
                        if(i==10)continue;
                        System.out.println("i="+i);
                }

        }
}
```

Kết quả thực hiện chương trình trên như sau:

i=0

i=1

i=2

i=3

i=4

i=6

i=7

i=8

i=9

Ta thấy rằng chương trình không in ra giá trị i=5 vì khi kiểm tra điều kiện chương trình bỏ qua giá trị i=5 và thực hiện lệnh với giá trị I còn lại của vòng lặp.

- ***Lệnh return***

Được sử dụng để kết thúc việc thực hiện của hàm hiện thời và chuyển điều khiển chương trình về cho chương trình đã gọi phương thức

Ví dụ

```
protected double giaTriKhongAm(double v)
{
        if(v<0)
 return 0;
        else
                return v;
}
```

2.4 Các lớp và các đối tượng trong Java

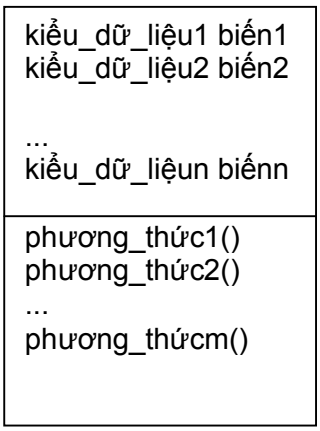
Đơn vị cơ bản trong lập trình Java là lớp. Các lớp bao gồm các phương thức thực hiện công việc tính toán. Các lớp cũng đưa ra cấu trúc của đối tượng và cơ chế tạo ra đối tượng từ các định nghĩa lớp.

2.4.1. Cấu trúc chung của một lớp

[<phạm vi hoặc kiểm soát truy nhập>] class <Tên lớp> [extends <Tên lớp cha>] [implements <Tên giao diện>]

```
{
  <Các thành phần lớp>
}
```

Khuôn mẫu của một lớp



Các kiểu lớp trong Java

- Lớp có sẵn (built-in).

- Lớp do người dùng định nghĩa (user-defined)

Định nghĩa phương thức

Hành vi của các đối tượng của một lớp được xác định bởi các phương thức của lớp đó.

Cú pháp định nghĩa phương thức

[<Phạm vi hoặc thuộc tính kiểm soát truy nhập>]<Kiểu trả về><Tên phương thức>

([<Danh sách tham biến hình thức>])[<Mệnh đề throws>]

{

<Nội dung phương thức>

}

Trong đó

<Kiểu trả về> có thể là kiểu nguyên thủy, kiểu lớp hoặc không có giá trị trả lại (kiểu void)

<Danh sách tham biến hình thức> bao gồm dãy các tham biến (kiểu và tên) phân cách với nhau bởi dấu phẩy.

Các kiểu phạm vi hay kiểm soát truy cập

- *public:* Các thành phần được khai báo là public có thể được truy cập ở bất kỳ nơi nào có thể truy cập được và chúng được thừa kế bởi các lớp con của nó.
- *private:* Các thành phần được khai báo là private chỉ có thể được truy cập trong chính lớp đó
- *protected:* Các thành phần được khai báo là protected có thể được truy cập và thừa kế bởi các lớp con và có thể truy xuất bởi mã lệnh của cùng gói đó.

Constructor

Constructor là một phương thức đặc biệt không có giá trị trả về và có tên trùng với tên lớp. Trường hợp không có constructor nào được đưa ra trình biên dịch cung cấp constructor mặc định cho lớp đó.

Một đối tượng được tạo mới và được cho trước một giá trị khởi đầu. Các trường có thể được khởi tạo bằng một giá trị khi chúng được khai báo, điều này đủ để đảm bảo một trạng thái khởi đầu đúng đắn.

Các phương thức

Các phương thức của một lớp thường chứa mã có thể hiểu và thao tác một trạng thái của đối tượng. Một số lớp có các trường public để người lập trình có thể thao tác trực tiếp, nhưng trong hầu hết các trường hợp điều này không phải là một ý tưởng tốt.

public class Point

```
{
        protected double x,y;
        public Point(double x,double y)
        {
                this.x=x;
                this.y=y;
        }
        public void move(double dx, double dy)
        {
                x=x+dx;
                y=y+dy;
        }
        public void print()
        {
                System.out.println("x="+x+", y="+y);
        }
        public static void main(String[] args)
        {
                Point p=new Point(3.0,6.0);
                System.out.println("Thong tin ve toa do diem ban dau:");
      p.print();
                p.move(-1.5,2.0);
                System.out.println("Thong tin ve toa do diem sau khi tinh tien theo vec to:");
                p.print();
        }
}
```

C:\>java Point

Thong tin ve toa do diem ban dau:

x=3.0, y=6.0

Thong tin ve toa do diem sau khi tinh tien theo vec to:

x=1.5, y=8.0

Tham chiếu this

Thông thường, đối tượng nhận phương thức cần phải biết tham chiếu của nó.

Trong lớp Point có constructor

```
public Point(double x,double y)
        {
                this.x=x;
                this.y=y;
        }
```

Giả sử nếu sửa đổi constructor bằng cách thay đổi như sau:

```
public Point(double x,double y)
        {
                x=x;
                y=y;
        }
```

Khi biên dịch chương trình sẽ báo lỗi vì có sự nhập nhằng trong các lệnh gán. Tham chiếu this đã khắc phục được điều này, this.x và this.y muốn đề cập tới trường thông tin của lớp Point còn x, y là tham biến truyền vào của constructor.

2.4.2. Thừa kế trong Java

```
class Point3C extends Point2C
{
        protected double z;
        public Point3C(double x, double y, double z)
   {
                super(x,y);
                this.z=z;
        }
        public void move(double dx, double dy, double dz)
        {
                super.move(dx,dy);
                z+=dz;
        }
        public void print()
        {
                super.print();
```

```
            System.out.print("  z="+z);
        }
        public static void main(String[] args)
        {
            Point3C p=new Point3C(3.0,4.5,5.0);
            System.out.println("Toa do ban dau:");
            p.print();
            System.out.println();
            p.move(-1.0,0.5,-1.0);
            System.out.println("Toa do sau khi tinh tien:");
            p.print();
            System.out.println();

        }
}
```

- ***Từ khóa super***

Từ khóa super được sử dụng để gọi constructor của lớp cha hoặc truy xuất tới các thành phần của lớp cha được che dấu bởi một thành phần của lớp con.

Ở trên ta đã xét hai đoạn mã, đoạn mã thứ nhất khai báo lớp Point2C biểu diễn một đối tượng điểm hai chiều, đoạn mã thứ hai khai báo lớp Point3C biểu diễn một đối tượng điểm ba chiều. Lớp Point3C được kế thừa lớp từ lớp Point2C. Lời gọi super(x,y) trong lớp Point3C gọi tới constructor Point2C hay super.move(dx,dy) gọi tới phương thức move(dx,dy) của lớp Point2C.

Biên dịch chương trình

C:\>javac Point3C.java

Thực thi chương trình

C:\>java Point3C

Kết quả chương trình

Toa do ban dau:

x=3.0, y=4.5 z=5.0

Toa do sau khi tinh tien:

x=2.0, y=5.0 z=4.0

2.4.3. Truyền tham số trong Java

Thông thường, trong một ngôn ngữ lập trình thường có hai cách truyền tham biến cho một thủ tục: truyền theo tham trị và truyền theo tham chiếu.

- ***Truyền theo tham trị***

Phương thức sẽ sao chép giá trị của một tham biến vào tham biến hình thức của thủ tục. Vì vậy, những thay đổi đối với tham số thủ tục sẽ không ảnh hưởng tới tham số thực sự.

```
class CallByValue
{       void test(int i,int j)
        {
                i*=2;
                j*=2;
        }
        public static void main(String[] args)
        {
                CallByValue cbl=new CallByValue();
                int a=10, b=30;
                System.out.println("Gia tri a va b truoc khi goi phuong thuc:"+a+" "+b);
                cbl.test(a,b);
                System.out.println("Gia tri a va b truoc sau goi phuong thuc:"+a+" "+b);

        }
}
```

```
C:\MyJava>javac CallByValue.java
C:\MyJava>java CallByValue
Gia tri a va b truoc khi goi phuong thuc:10 30
Gia tri a va b truoc sau goi phuong thuc:10 30
```

Tất cả các tham số đối với các phương thức Java là “gọi theo trị”. Nghĩa là, các giá trị của các biến tham số trong một phương thức là các bản sao của các giá trị do người gọi xác định.

Ví dụ:

```
class TruyenThamTri
{
        public static void main(String[] args)
        {
                double one =1.0;
                System.out.println("Truoc khi goi ham:one="+one);
```

```
            chia(one);
            System.out.println("Sau loi goi ham chia:one ="+one);

        }
        public static void chia(double x)
        {
            x/=2.0;
            System.out.println("Sau khi chia:x="+x);
        }
}
```

C:\MyJava\Baitap>java TruyenThamTri

Truoc khi goi ham:one=1.0

Sau khi chia:x=0.5

Sau loi goi ham chia:one =1.0

- ***Truyền theo tham chiếu***

Một tham chiếu tới tham biến được truyền cho tham số. Bên trong thủ tục, tham chiếu này được sử dụng để truy xuất tới tham số thực sự được xác định trong lời gọi. Điều này nghĩa là những thay đổi đối với tham biến sẽ ảnh hưởng tới tham số đã được sử dụng để gọi phương thức.

Mảng trong Java được xem là đối tượng. Các phần tử của mảng có thể có kiểu nguyên thủy hoặc kiểu tham chiếu (kiểu lớp)

Ví dụ: Lọc ra phần tử lớn nhất của một mảng và đưa về cuối mảng.

C:\MyJava\Baitap>java Loc

8 1 4 3 2 5

1 4 3 2 5 8

Vì trong Java các lời gọi hàm đều thực hiện theo tham trị (call by value) với kiểu nguyên tố và tham chiếu đối với kiểu lớp nên hàm doiCho() thay đổi được các phần tử của day.

- ***Các tham biến final***

Tham biến hình thức có thể được khai báo với từ khóa final đứng trước. Tham biến final là tham biến không được khởi tạo giá trị cho đến khi nó được gán một trị nào đó và khi đã được gán trị thì không thể thay đổi giá trị đó được.

Ví dụ

```
class BienFinal
{
        public static void main(String[] args)
        {
```

```
        HangSX banh = new HangSX();
        int     giaBan=20;
        double tien = banh.tinh(10,giaBan);
        System.out.println("Gia ban: "+giaBan);
        System.out.println("Tien ban duoc: "+tien);

    }
}
class HangSX
{
    double tinh(int so, final double gia)
    {
        gia = gia/2.0;
        return so*gia;
    }
};
```

Khi biên dịch hàm tinh(), chương trình dịch sẽ thông báo lỗi và không được phép thay đổi giá trị của biến final *gia.*

- ***Các đối trong chương trình***

Chúng ta có thể truyền các tham số cho chương trình trên dòng lệnh

Ví dụ

```
class TinhTong
{
    public static void main(String[] args)
    {
        float s=0.0f;
        for(int i=0; i<args.length;i++)
            s+=Float.valueOf(args[i]).floatValue();
        System.out.println("Tong cua day so la "+s);
    }
}
```

Kết quả thực hiện chương trình

C:\MyJava\Baitap>java TinhTong 12 34 56

Tong cua day so la 102.0

2.4.4. Đa hình trong Java

- ***Các phương thức nạp chồng (overloaded method)***
- ***Các phương thức nạp chồng là các phương thức nằm trong cùng một lớp có cùng tên nhưng khác nhau về danh sách tham số.***

Các phương thức nạp chồng là một dạng của tính đa hình thời gian biên dịch.

Ví dụ

```
class TinhToan
{
        public static void main(String[] args)
        {
                Tinh c = new Tinh();
                c.add(10,20);
                c.add(40.0f,35.65f);
                c.add("Good ","Morning");

        }
}
class Tinh
{
        public void add(int a, int b)
        {
                int c = a+b;
                System.out.println("Phep cong hai so nguyen :"+c);
        }
        public void add(float a, float b)
        {
                float c = a+b;
                System.out.println("Phep cong hai so dau phay dong :"+c);
        }
        public void add(String a, String b)
        {
                String c = a+b;
                System.out.println("Phep cong hai xau :"+c);
        }
};
```

Kết quả:

C:\MyJava\Baitap>java TinhToan

Phep cong hai so nguyen :30

Phep cong hai so dau phay dong :75.65

Phep cong hai xau :Good Morning

Giải thích: Trong chương trình trên phương thức add() là phương thức được nạp chồng. Có ba phương thức có cùng tên add() nhưng có các tham số khác nhau. Khi chúng ta gọi phương thức add()…????

2.4.5. Các thành phần static

Đôi khi các thành phần static ta cần phải định nghĩa các thành phần lớp được sử dụng độc lập với bất kỳ đối tượng nào của lớp. Thông thường một thành phần của lớp phải được truy xuất thông qua đối tượng của lớp. Tuy nhiên, ta có thể có thể tạo ra một thành phần mà được sử dụng độc lập. Để tạo ra một thành phần như vậy trước khai báo của mỗi thành phần ta đặt một từ khóa **static.** Khi một thành phần được khai báo static, nó có thể được truy xuất trước khi một đối tượng được tạo ra và không cần tham chiếu tới bất kỳ đối tượng nào. Các thành phần static bao gồm: biến static, phương thức static, khối static.

- ***Biến static***

Biến static về cơ bản là biến tổng thể. Khi các đối tượng của một lớp được khai báo, không có bản sao của biến đối tượng nào được tạo ra. Thay vào đó, tất cả các đối tượng cùng chung một biến static.

Ví dụ về biến static

```
class StaticVariable {
   static int count=20;
        StaticVariable(){
                count++;
        }
        public static void main(String[] args)
        {
                StaticVariable c1=new StaticVariable();
                System.out.println("Bien dem count="+count);
                StaticVariable c2=new StaticVariable();
                System.out.println("Bien dem count="+count);
                StaticVariable c3=new StaticVariable();
                System.out.println("Bien dem count="+count);
        }
}
```

Biến count được khai báo là static nên nó chỉ có một bản sao trong mọi đối tượng, vì vậy khi đối tượng được tạo ra thì các biến count được tăng lên 1 do trong hàm constructor biến count được tăng lên 1.

- ***Phương thức static***

Các phương thức được khai báo static có một số hạn chế sau:

  - Chúng chỉ có thể gọi các phương thức static khác.
  - Chúng chỉ truy xuất tới các dữ liệu static
  - Chúng không thể tham chiếu tới *this* và *super*

Ví dụ

```
class StaticMethod
{
        public static void main(){
                System.out.println("Hello");
        }
        public static void main(String[] args)
        {
                main();
                System.out.println("Hello World!");
        }
}
```

Trong ví dụ này ta thấy khai báo hai phương thức main đều là phương thức tĩnh. Phương thức main này gọi tới phương thức main khác.

- ***Khối static***

Nếu cần tính toán để khởi tạo các biến static, ta có thể khai báo khối static để nó có thể xử lý ngay tức thời khi lớp lần đầu tiên được tải vào bộ nhớ. Các khối static luôn được xử lý trước

Ví dụ

```
class StaticDemo
{
        static{
                System.out.println("Khoi static 1");
        }

        public static void main(String[] args)
        {
                System.out.println("Hello World!");
```

```
        }
        static {
                System.out.println("Khoi static 2");
        }
}
```

Vì khối static luôn được xử lý trước nên kết quả in ra của chương trình trên sẽ là:

Khoi static 1

Hello World!

Khoi static 2

2.4.6. Các thành phần final

- **Biến final**

Một biến được khai báo final. Làm như vậy sẽ ngăn ngừa nội dung của biến bị sửa đổi. Điều này nghĩa là ta phải khai báo một biến final khi nó được khai báo.

Ví dụ

```
final double pi=3.1416;
```

- **Sử dụng final với thừa kế**

Mặc dù nạp chồng phương thức là một trong các đặc trưng mạnh của Java, nhưng sẽ có những lúc ta cần ngăn ngừa điều này xảy ra. Để không cho phép một phương thức được nạp chồng, xác định từ khóa final như là một bổ từ tại đầu mỗi khai báo của nó. Các phương thức được khai báo là final không thể được nạp chồng.

Ví dụ

```
class A {
 final void method(){
 }
}
class B extends A{
final void method(){
 }
}
```

Khai báo lớp B có lỗi, vì ở lớp A, phương thức method đã được khai báo với từ khóa final nên nó không thể được nạp chồng trong lớp B.

Sử dụng từ khóa final để cấm thừa kế

Đôi khi ta cần cấm một số lớp không có lớp con. Ta có thể thực hiện điều này bằng cách khai báo lớp với từ khóa final.

Ví dụ

```
final class A {
}
```

Lúc này, các lớp khác không thể thừa kế từ lớp A.

2.5. Các lớp trừu tượng

Trong lập trình Java, có những lúc ta cần định nghĩa lớp cha và khai báo cấu trúc một cách khái quát mà không cài đặt cụ thể cho từng phương thức. Các lớp cha sẽ định nghĩa dạng tổng quát hóa được dùng chung bởi các lớp con của nó, việc cài đặt chi tiết các phương thức này sẽ được thực hiện ở trong từng lớp con cụ thể.

Ví dụ:

```
abstract class  Hinh2D
{
        double a,b,r;
        public abstract double dientich();
        public abstract double chuvi();
}
class HinhTron extends Hinh2D
{
        public HinhTron(double r)
        {
                this.r=r;
        }
        public double dientich()
        {
                return Math.PI*r*r;
        }
        public double chuvi()
        {
                return Math.PI*2*r;
        }
}
class HinhChuNhat extends Hinh2D
{
        public HinhChuNhat(double a,double b)
        {
                this.a=a;
                this.b=b;
```

```
        }
        public double dientich()
        {
                return a*b;
        }
        public double chuvi()
        {
                return (a+b)*2;
        }
}
class AbstractDemo
{
        public static void main(String args[])
        {
                Hinh2D ht=new HinhTron(1);
                System.out.println("Dien tich hinh tron ban kinh 1.0 la:"+ht.dientich());
                System.out.println("Chu vi hinh tron ban kinh 1.0 la:"+ht.chuvi());
                Hinh2D hcn=new HinhChuNhat(3,4);
                System.out.println("Dien tich hinh chu nhat la:"+hcn.dientich());
                System.out.println("Chu vi hinh chu nhat la "+hcn.chuvi());

        }
};
```

Kết quả thực hiện chương trình

```
C:\MyJava>java AbstractDemo
Dien tich hinh tron ban kinh 1.0 la:3.141592653589793
Chu vi hinh tron ban kinh 1.0 la:6.283185307179586
Dien tich hinh chu nhat la:12.0
Chu vi hinh chu nhat la 14.0
```

Trong chương trình trên ta khai báo lớp trừu tượng Hinh2D, lớp này có các phương thức trừu tượng là dientich() để tính diện tích của hình và lớp chuvi() để tính chu vi. Các lớp trừu tượng không được cài đặt mã lệnh.

Các lớp HinhTron và HinhChuNhat là các lớp con cụ thể của lớp trừu tượng Hinh2D. Các lớp này cài đặt các phương thức tính diện tích và chu vi cụ thể

2.6. Giao tiếp (Interface)

Thừa kế đóng một vai trò rất quan trọng trong việc tiết kiệm thời gian và công sức của người lập trình. Hầu hết các chương trình trong thực tế đều sử dụng đa thừa kế. Trong đa thừa kế, chúng ta có thể thừa kế các phương thức và thuộc tính từ một số lớp khác nhau. Java không hỗ trợ đa thừa kế. Tuy nhiên, nhận thấy tầm quan trọng của đa thừa kế trong Java, Java đã đưa ra khái niệm interface. Với giao tiếp ta có thể xác định một lớp phải làm gì nhưng không xác định cách làm thế nào.

- ***Định nghĩa***

Một giao tiếp là một tập hợp các định nghĩa phương thức (không có cài đặt). Một giao tiếp cũng có thể định nghĩa các hằng.

Ta cũng có thể đặt câu hỏi vậy giao tiếp khác gì so với các lớp trừu tượng? Dưới đây là những sự khác biệt giữa giao tiếp và các lớp trừu tượng:

  - Một giao tiếp không thể thực thi bất kỳ phương thức nào, ngược lại một lớp trừu tượng có thể thực thi một số phương thức nào đó.
  - Một lớp có thể thực thi nhiều giao tiếp nhưng một lớp chỉ có một lớp cha.
  - Một giao tiếp không phải là bộ phận của sơ đồ phân cấp lớp, các lớp có thể thực thi cùng một giao tiếp.

- **Khai báo một giao tiếp**

Cú pháp chung khi khai báo một giao tiếp là

public interface InterfaceName extends SuperInterfaces

{

//Thân giao tiếp

}

Hai thành phần bắt buộc trong một khai báo giao tiếp là-từ khóa interface và tên của giao tiếp. Từ khóa bổ trợ truy xuất là public chỉ ra rằng giao tiếp có thể được sử dụng bởi bất kỳ lớp nào bất kỳ gói nào. Nếu không xác định giao tiếp là public thì giao tiếp sẽ chỉ có thể truy xuất bởi các lớp được định nghĩa trong cùng gói với giao tiếp.

Một khai báo giao tiếp có thể có một thành phần khác: danh sách các giao tiếp cha. Một giao tiếp có thể thừa kế các giao tiếp khác, giống như một lớp có thể thừa kế hoặc là lớp của lớp khác. Danh sách các giao tiếp cha được phân cách bởi dấu phẩy.

Thân giao tiếp

Thân giao tiếp chứa các khai báo phương thức cho tất cả các phương thức có trong giao tiếp. Một khai báo phương thức trong một giao tiếp kết thúc bởi dấu chấm phẩy (;) vì một giao tiếp không cung cấp cách cài đặt cho các phương thức được khai báo trong nó.

Một giao tiếp có thể chứa các khai báo hằng ngoài các khai báo phương thức. Các khai báo thành phần trong một giao tiếp không được phép sử dụng một số từ khóa bổ trợ như private, protected transient, volatile, hoặc synchronized trong các khai báo thành phần của một giao tiếp.

Trong ví dụ sau ta sẽ tìm hiểu cách định nghĩa một giao tiếp và cách thực thi một giao tiếp trong.

public interface CalculatorInterface

{

```
        public double add(double x, double y);
        public double sub(double x, double y);
        public double mul(double x, double y);
        public double div(double x, double y);
}
```

- **Thực thi giao tiếp**

Một giao tiếp định nghĩa một tập hợp hợp các quy ước về hành vi. Một lớp thực thi một giao tiếp tuân theo những quy ước đã được khai báo trong giao tiếp đó. Để khai báo một lớp thực thi một giao tiếp, ta đưa vào mệnh đề implements trong khai báo lớp. Một lớp có thể thực thi nhiều giao tiếp (Java hỗ trợ đa thừa kế giao tiếp), vì vậy sau từ khóa implements là một danh sách các giao tiếp được thực thi bởi một lớp.

Chú ý: Mệnh đề implements đứng sau mệnh đề extends nếu tồn tại mệnh đề extends.

```
class CalculatorTest implements CalculatorInterface
{
        public double add(double x, double y)
        {
                return x+y;
        }
        public double sub(double x, double y)
        {
                return x-y;
        }
        public double mul(double x, double y)
        {
                return x*y;
        }
        public double div(double x, double y)
        {return x/y;

        }

        public static void main(String[] args) throws Exception
        {
                CalculatorInterface cal=new CalculatorTest();
                if(args.length!=2)
                {
```

```
                System.out.println("Cach chay chuong trinh: java CalculatorImpl so1 so2");
                return;
            }
            else
            {
                double x,y,z;
                x=Double.parseDouble(args[0]);
                y=Double.parseDouble(args[1]);

                System.out.println(x+"+"+y+"="+cal.add(x,y));
                System.out.println(x+"-"+y+"="+cal.sub(x,y));
                System.out.println(x+"*"+y+"="+cal.mul(x,y));
                System.out.println(x+"/"+y+"="+cal.div(x,y));
            }

        }
}
```

Kết quả thực hiện chương trình là

```
C:\MyJava>java CalculatorTest 12 3
12.0+3.0=15.0
12.0-3.0=9.0
12.0*3.0=36.0
12.0/3.0=4.0
```

Sử dụng giao tiếp như là một kiểu

Khi ta định nghĩa một giao tiếp mới, ta có thể định nghĩa kiểu dữ liệu tham chiếu mới. Giao tiếp có thể được sử dụng khi khai báo một biến tham chiếu. Giả sử MyInterface là một giao tiếp thì ta có thể khai báo như sau:

```
MyInterface mi;
```

2.7. Các gói và sử dụng gói trong Java

Các gói có các thành phần là các lớp, các interface, và các gói con có liên quan với nhau. Việc tổ chức thành các gói có một số lợi ích sau đây:

- Các gói cho phép ta tổ chức các lớp thành các đơn vị nhỏ hơn (như các thư mục), và giúp cho việc định vị và sử dụng các lớp tương ứng trở nên dễ dàng hơn.
- Tránh được các vấn đề về xung đột tên.
- Cho phép ta bảo vệ các lớp, dữ liệu và các phương thức theo một quy mô lớn hơn so với phạm vi lớp.

- Các tên gói có thể được sử dụng để định danh các lớp của bạn.

- ***Truy xuất tới các thành phần của gói trong Java***

Để truy xuất tới thành phần của gói trong Java ta có thể sử dụng cú pháp sau:

MyPackage.MyClass

MyPackage là tên gói, MyClass là tên lớp nằm trong gói MyPackage.

- **Khai báo các gói trong chương trình**

Để có thể sử dụng các thành phần của một gói trong chương trình Java, ta cần phải khai báo gói cụ thể chứa lớp đó:

import ten_goi.*;// ten_goi: tên gói

Với khai báo như trên, ta có thể truy xuất tới tất cả các lớp, các interface nằm trong gói đó. Để khai báo sử dụng một lớp cụ thể trong chương trình ta khai báo dòng lệnh sau:

import ten_goi.ten_lop; // ten_lop: tên lớp

Giả sử ta có gói MyPackge, bên trong gói MyPackage lại có một số gói con như SubPackage1, SubPackage2,..ta có thể khai báo sử dụng các thành phần trong gói con SubPackage1 như sau:

import MyPackage.SubPackage1.;*

- ***Cách tạo ra các gói trong Java***

Bước 1: Khai báo một gói trong Java

Giả sử ta khai báo một gói có tên là *mypackage*, bên trong gói này có lớp *Calculator.*

```
package mypackage;
public class  Calculator
{
        public double cong(double a,double b)
        {
              return a+b;
         }
         public double nhan(double a, double b)
         {
               return a*b;
         }
         public double tru(double a,double b)
         {
                  return a-b;
         }
        public double  chia(double a,double b) throws Exception
        {
```

```
            return a/b;
        }

}
```

Bước 2: Biên dịch

C:\>javac -d C:\MyJava Calculator.java

Một vài điều cần lưu ý khi khai báo các thành viên của gói.

Thứ nhất, các thành phần của gói cần được khai báo với thuộc tính public, nếu cần truy xuất chúng từ bên ngoài.

2.6. Quản lý ngoại lệ (Exception Handling)

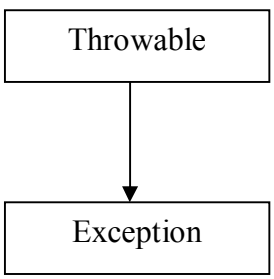
- ***Khái niệm***

Trong quá trình xử lý, các ứng dụng có thể bất ngờ gặp các lỗi với các mức độ nghiêm trọng khác nhau. Khi một phương thức tác động trên một đối tượng, đối tượng có thể phát hiện các vấn đề trạng thái bên trong (chẳng hạn các giá trị không nhất quán, như lỗi chia 0), phát hiện các lỗi với các đối tượng hay dữ liệu mà nó thao tác (như file hay địa chỉ mạng) xác định nó vi phạm các qui tắc cơ bản (như đọc dữ liệu từ một luồng đã bị đóng),...Rất nhiều người lập trình không thể kiểm tra tất cả các trạng thái lỗi có thể xảy ra.

Exception cung cấp một cách để kiểm tra các lỗi mà không chia cắt mã. Exception cũng đưa ra một cơ chế báo lỗi một cách trực tiếp chứ không sử dụng các cờ hay các hiệu ứng phụ.

- ***Các ngoại lệ trong Java***

Trong Java có một lớp Exception, mọi lớp ngoại lệ là lớp con của lớp này. Lớp Exception là lớp con của lớp Throwable



Hình 2.7

Lớp Throwable chứa một xâu được sử dụng để mô tả ngoại lệ. Ngoại lệ được phân thành hai loại: Ngoại lệ được kiểm tra (checked exception) và ngoại lệ không được kiểm tra (unchecked exception).

Ngoại lệ được kiểm tra là ngoại lệ mà trình biên dịch sẽ kiểm tra phương thức của người lập trình và chỉ đưa ra ngoại lệ khi chúng được thông báo để đưa ra.

Ngoại lệ không được kiểm tra là lớp con của các lớp Error; RuntimeException.

Java cung cấp một mô hình quản lý các ngoại lệ cho phép kiểm tra các lỗi ở các vị trí có liên quan.

- ***Khối try và catch***

Cú pháp

  - *Khối try*

Bao gồm một tập hợp các lệnh có thể phát sinh ngoại lệ trong khi xử lý. Một phương thức, có thể đưa ra một ngoại lệ cũng được đặt try

Các khối try lồng nhau

```
try{
        stmt1;
        stmt2;
        try{
                stmt3;
                stmt4;
        }
        catch(Exception e)
        {
        }
}
catch(Exception e)
{
}
```

Khi các khối try được lồng nhau, khối try bên trong được xử lý trước và một ngoại lệ được đưa ra trong khối đó được đón bắt trong các khối try catch con. Nếu khối catch bên trong không thỏa mãn thì các khối try bên ngoài được kiểm tra. Nếu một khối catch phù hợp được tìm thấy, thì ngoại lệ được quản lý trong khối đó ngược lại thì môi trường Java Runtime quản lý ngoại lệ.

```
try
                {
                        doFileProcessing();
```

```
                displayResults();
        }
        catch(Exception e)
        {
                System. er.println(e.getMessage());
        }
```

Bất kỳ lỗi nào xảy ra trong quá trình xử lý doFileProcessing() hay displayResult() thì sẽ được đón bắt bởi khối catch và được xử lý.

Nếu có lỗi xảy ra trong quá trình xử lý doFileProcessing(), phương thức displayResult() sẽ không bao giờ được gọi, và khối catch sẽ được xử lý.

Một khối try có thể có nhiều khối catch và xử lý các kiểu khác nhau

```
        try
        {
        }
        catch(Exception e)
        {
        }
        catch(Exception e)
        {
        }
        finally
        {
                //Thực hiện công việc thu dọn
        }
```

Ví dụ:

```
class TryCatch
{
    public static void main(String[] args)
    {int x,y;
        try{
                x=Integer.parseInt(args[0]);
                y=Integer.parseInt(args[1]);
                x=x/y;
                System.out.println("x="+x);
        }
```

```
            catch(ArithmeticException e)
            {
                  System.out.println("Khong the chia cho 0");
                  System.err.println(e);
            }
      }
}
```

Kết quả 1

C:\MyJava\Baitap>java TryCatch 18 9

x=2

Kết quả 2

C:\MyJava\Baitap>java TryCatch 9 0

Khong the chia cho 0

java.lang.ArithmeticException: / by zero

- ***Khối finally***

Khối finally là khối mà chúng ta thấy các lệnh trả về các tài nguyên cho hệ thống và các lệnh khác để in ra bất kỳ thông báo nào.

Các lệnh trong khối finanally có thể là:

- Đóng một file.
- Đóng một resultset (Lập trình cơ sở dữ liệu).
- Ngắt liên kết được thiết lập với cơ sở dữ liệu.

…

Khối finally luôn được xử lý dù ngoại lệ có xảy ra hay không

- ***Mệnh đề throw***

Các ngoại lệ được đưa ra bằng cách sử dụng lệnh throw, nó nhận một đối tượng làm tham số, đối tượng thuộc lớp là lớp con của lớp Throwable

Ví dụ:

```
class ArraySizeException extends NegativeArraySizeException
{
      ArraySizeException()
      {
            super("Nhap kich thuoc mang khong hop le");
      }
}
```

```
class ThrowDemo
{
        int size, a[];
        ThrowDemo(int s)
        {
                size =s;
                try{
                        checkSize();
                }
                catch(ArraySizeException e)
                {
                        System.out.println(e);
                }
        }
        void checkSize() throws ArraySizeException
        {
                 if(size<0) throw new ArraySizeException();
                 a= new int[3];
                 for(int i=0;i<3;i++)a[i]=i+1;
        }

        public static void main(String[] args)
        {
                new ThrowDemo(Integer.parseInt(args[0]));
        }
}
```

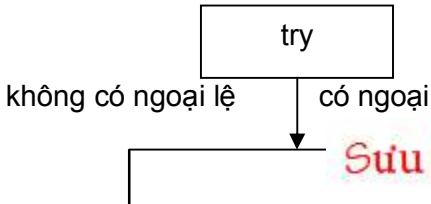
Kết quả thực hiện chương trình

C:\MyJava\Baitap>java ThrowDemo -1

ArraySizeException: Nhap kich thuoc mang khong hop le

Giải thích:

Chúng ta đã tạo ra một lớp có tên ArraySizeException, lớp này là lớp con của lớp NegativeArraySizeException. Bằng cách tạo ra một đối tượng của lớp này, chúng ta đã in ra thông báo ngoại lệ. Phương thức checkSize() có thể đưa ra ngoại lệ ArraySizeException.



Hình 2.8

Ví dụ:

```
class FinallyDemo
{
        String name;
        int x,y;
        FinallyDemo(String s[])
        {
                try{
                        name = new String("try catch finally demo");
                        x = Integer.parseInt(s[0]);
                        y=Integer.parseInt(s[1]);
                        System.out.println(name);
                        System.out.println("Ket qua "+x/y);
                }
                catch(ArithmeticException e)
                {
                         System.err.println("Khong the chia 0!");
                }
                finally
                {
                        name = null;
                        System.out.println("Xu ly khoi finally");
                }
```

```
        }

        public static void main(String[] args)
        {
                new FinallyDemo(args);
        }
}
```

Kết quả 1

```
C:\MyJava\Baitap>java FinallyDemo 16 0
try catch finally demo
Khong the chia 0!
Xu ly khoi finally
```

Kết quả 2

```
C:\MyJava\Baitap>java FinallyDemo 16 4
try catch finally demo
Ket qua 4
Xu ly khoi finally
```

- ***Một số ngoại lệ thường gặp***
    - RuntimeException
    - ArithmeticException
    - IllegalArgumentException
    - ArrayIndexOutOfBoundsException
    - NullPointerException
    - SecurityException
    - NoSuchElementException
    - ClassNotFoundException
    - AWTException
    - DataFormatException
    - SQLException
    - IOException
    - UnknownHostException
    - SocketException

- EOFException
- MalformedURLException
- FileNotFoundException
- IllegalAccessException
- NoSuchMethodException

Chương 3

Các luồng vào ra

1. Khái niệm về luồng trong Java

Khi lập bất kỳ chương trình nào trong một ngôn ngữ nào thì vấn đề vào ra dữ liệu giữa chương trình và nguồn dữ liệu cũng như đích dữ liệu là vấn đề mà người lập trình cần phải quan tâm. Làm thế nào để ta có thể truyền dữ liệu cho một chương trình Java. Có hai cách hiệu quả để thực hiện điều này:

- Thông qua một tài nguyên tuần tự nào đó như file hoặc qua một máy tính khác.
- Thông qua giao diện người máy.

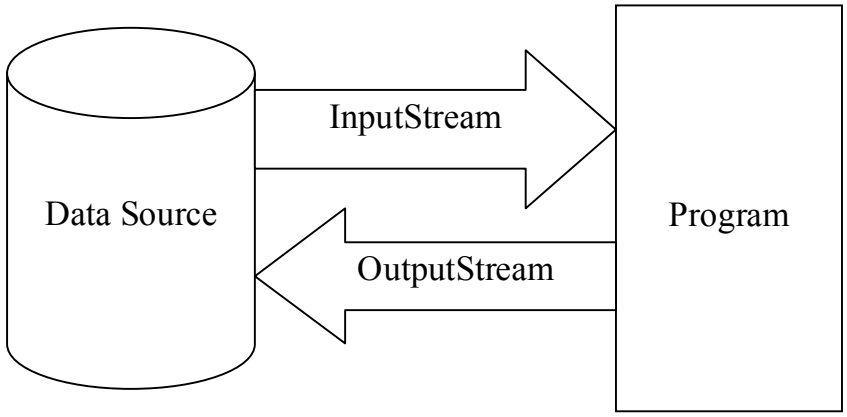
Mục đích của chương này là xem xét cách truyền dữ liệu cho một chương trình thông qua một máy tính khác hay tập tin.

1.1. Khái niệm luồng (stream)

Theo nghĩa đen luồng là một đường ống nước.

Về mặt thuật ngữ chuyên ngành ta có thể hiểu "Các luồng là các dãy dữ liệu có sắp thứ tự".

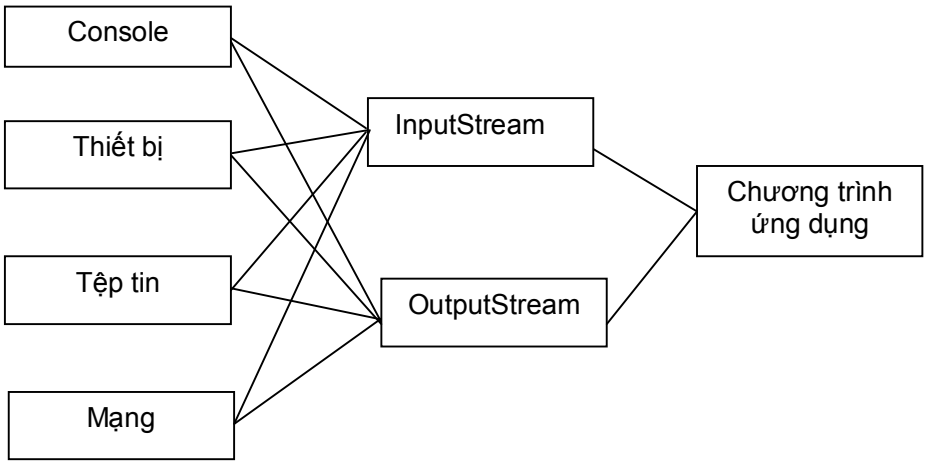
Xét trên quan điểm của chương trình và nguồn dữ liệu (Data Soure) ta có thể phân loại luồng thành hai loại: Luồng xuất (output stream) và luồng nhập (input stream). Để trực quan hơn chúng ta xem hình vẽ dưới đây:



Hình 3.1

Như vậy nếu chúng ta cần lấy dữ liệu từ nguồn vào chương trình thì cần phải sử dụng luồng nhập. Ngược lại, nếu ta cần ghi dữ liệu từ chương trình ra nguồn dữ liệu thì ta cần phải sử dụng luồng xuất.

Ta có thể thấy rằng có rất nhiều luồng dữ liệ, chẳng hạn như từ một tệp tin, từ các thiết bị xuất và nhập chuẩn, từ liên kết mạng. Như vậy một chương trình có thể truy xuất tới nhiều nguồn dữ liệu.



Hình 3.2

2. Luồng xuất nhập chuẩn

- System.out: Luồng xuất chuẩn thường được sử dụng để hiển thị kết quả đầu ra trên màn hình.
- System.in: Luồng nhập chuẩn thường đến từ bàn phím và được sử dụng để hiện các ký tự.
- System.err: Luồng lỗi chuẩn.

Các luồng trên còn được gọi là các luồng hệ thống. Mặc dù các luồng này rất có ích khi lập trình nhưng chúng không đủ mạnh khi giải quyết các vấn đề vào ra quan trọng khác. Trong các mục tiếp theo ta sẽ tìm hiểu sâu một số luồng trong gói *java.io*

3. Luồnng nhị phân

3.1. Lớp InputStream

Lớp trừu tượng *InputStream* khai báo các phương thức để đọc dữ liệu đầu vào từ một nguồn cụ thể. Lớp InputStream là lớp cơ sở của hầu hết các luồng nhập trong gói *java.io*, và nó hỗ trợ các phương thức sau:

Các phương thức:

- *public InpuStream()*

InputStream chỉ hỗ trợ constructor không tham số.

- *public abstract int read() throws IOException*

  Phương thức cơ bản của lớp *InputStream* là phương thức read(). Phương thức này đọc một byte dữ liệu từ luồng nhập và trả về một số kiểu nguyên int có giá trị nằm trong khoảng từ 0 đến 255. Giá trị trả về là -1 khi kết thúc luồng. Phương thức read() chờ và phong tỏa các đoạn mã sau nó cho tới khi một byte dữ liệu được đọc. Việc nhập và xuất diễn ra với tốc độ chậm, vì vậy nếu chương trình của ta thực hiện một công việc khác quan trọng thì tốt nhất là đặt các lệnh nhập xuất vào một tuyến đoạn riêng của nó. Phương thức read() là phương thức trừu tượng bởi vì các lớp con cần thay đổi để thích ích với môi trường cụ thể.

- *public int read(byte[] b) throws IOException*

Phương thức này đọc một dãy các byte dữ liệu liên tục từ một nguồn của luồng nhập và lưu vào mảng b.

- *public int read(byte[] b, int offs, int len) throws IOException*

  Phương thức này đọc một dãy các byte dữ liệu và lưu vào mảng b, vị trí bắt đầu lưu dữ liệu là offs và lưu *len* byte dữ liệu

- *public int available() throws IOException*

  Phương thức này cho biết còn bao nhiêu byte dữ liệu trong luồng.

- *public long skip(long count) throws IOException*

  Phương thức skip(long count) bỏ qua long byte dữ liệu

- *public synchronized void mark(int readLimit)*

  Phương thức này được sử dụng để dánh dấu vị trí hiện thời trong luồng

- *public void reset() throws IOException*

  Phương thức này xác định lại vị trí luồng là vị trí đánh dấu lần gần đây nhất.

- *public boolean markSupported()*

  Phương thức này trả về giá trị true nếu luồng này hỗ trợ đánh dấu và false nếu nó không hỗ trợ đánh dấu.

- *public void close() throws IOException*

Khi đã làm việc xong với một luồng, ta cần đóng lại luồng đó. Điều này cho phép hệ điều hành giải phóng các tài nguyên gắn với luồng.

3.2. Lớp OutputStream

Lớp trừu tượng *OutputStream* khai báo các phương thức để ghi dữ liệu ra luồng. Chúng bao gồm các phương thức sau đây:

- public OuputStream()

  Phương thức *OutputStream* hỗ trợ constructor không tham số

- *public abstract void write(int b)throws IOException*

  Phương thức này ghi một byte không dấu có giá trị trong khoảng từ 0 đến 255. Nếu ta truyền vào một số có giá trị lớn hơn 255 hoặc nhỏ hơn 0, nó sẽ thực hiện phép tính

  b =b mod 256 trước khi ghi giá trị vào luồng.

- *public void write(byte[] b)throws IOException*

  Phương thức này ghi dữ liệu từ luồng vào toàn bộ mảng b.

- *public void write(byte[] b, int off, int len) throws IOException*

  Phương thức này chỉ ghi một đoạn con của mảng dữ liệu bắt đầu từ vị trí offs và tiếp tục cho tới khi ghi hết *len* byte.

- *public void close()*

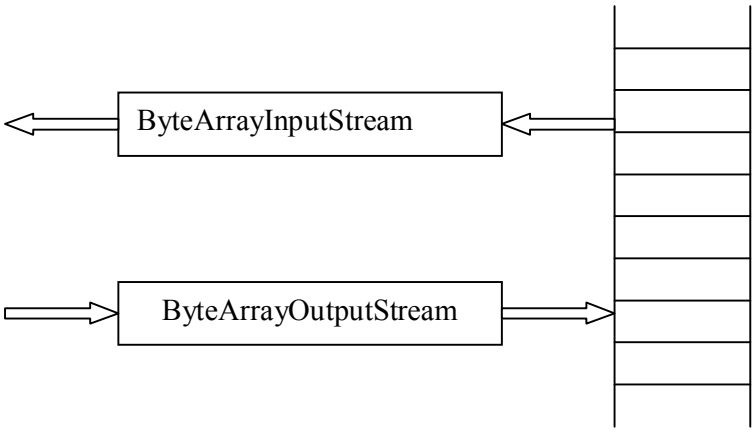
  Phương thức này đóng một luồng. Phương thức này được gọi để giải phóng các tài nguyên gắn với luồng.

- *public void flush()*

Các luồng xuất nhập khác được thừa kế từ các luồng trừu tượng *InputStream* và *OutputStream*. Đối với mỗi kiểu dữ liệu và nguồn dữ liệu chúng ta có thể có các kiểu luồng xuất và nhập riêng, chẳng hạn *DataInputStream*, *DataOutputStream*, *FileInputStream*, *FileOutputStream*,… Sau đây chúng ta sẽ lần lượt xem xét từng kiểu luồng cụ thể.

3.3. Các luồng xuất nhập mảng byte

Để xây dựng một xâu ký tự biểu diễn dữ liệu có thể đọc được hoặc giải mã dữ liệu, người ta xem các mảng byte như là nguồn của các luồng nhập hoặc đích của các luồng xuất. Các luồng byte cung cấp các khả năng này.



Hình 3.3

3.3.1. Luồng nhập mảng byte

Lớp ByteArrayInputStream sử dụng một mảng byte như là một nguồn dữ liệu đầu vào. Nó có hai constructor:

- *public ByteArrayInputStream(byte[] buf)*

Tạo ra một đối tượng ByteArrayInputStream từ một mảng xác định. Mảng đầu vào được sử dụng một cách trực tiếp. Khi kết thúc buf nghĩa là kết thúc nhập từ luồng.

- *public ByteArrayInputStream(byte[] buf, int offset, int length)*

Tạo ra một đối tượng ByteArrayInputStream từ một mảng xác định, chỉ sử dụng một phần của mảng buf từ buf[offset] đến buff[offset+length-1] hoặc kết thúc mảng.

ByteArrayInputStream tạo ra một luồng nhập từ một vùng đệm trong bộ nhớ được biểu diễn bằng một mảng byte. Lớp này không hỗ trợ bất kỳ phương thức mới nào, nó nạp chồng các phương thức read(), skip(), available(), và reset() của lớp cha InputStream.

Ví dụ:

Tạo một mảng gồm 100 byte rồi gắn vào mảng này một luồng ByteArrayInputStream để lấy dữ liệu ra.

```
import java.io.*;
public class LuongNhapMang
{
 public static void main(String[] args)
 {
                byte[] b = new byte[100];
                for(byte i=0;i<b.length;i++) b[i]=i;
                try{
                        InputStream is = new ByteArrayInputStream(b);
                        for(byte i=0;i<b.length;i++)
```

```
                    System.out.print(is.read()+" ");
                }
                catch(IOException e)
                {
                    System.err.println(e);
                }
        }
        }
```

Kết quả thực hiện chương trình

C:\MyJava\Baitap>java LuongNhapMang

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99

Chú ý: Mỗi lần đọc luồng bằng phương thức read(), một byte dữ liệu không còn trong luồng, nhưng vẫn tồn tại trong mảng.

3.3.1. *Luồng nhập mảng byte*

ByteArrayOutputStream tạo ra một luồng xuất trên một mảng byte. Nó cũng cung cấp các khả năng bổ sung.

Các constructor:

- *public ByteArrayOutputStream()*

Tạo ra một đối tượng ByteArrayOutputStream với kích thước mặc định

- *public ByteArrayOutputStream(int size)*

Tạo ra một đối tượng ByteArrayOutputStream với kích thước xác định ban đầu.

Các phương thức mới của lớp ByteArrayOutputStream:

- *public synchronized byte[] toByteArray()*:

Phương thức này trả về một bản sao dữ liệu của luồng và lưu dữ liệu vào một mảng và có thể sửa đổi dữ liệu trong mảng này mà không cần thay đổi các byte của luồng xuất.

- *public size()*

  Trả về kích thước hiện thời của vùng đệm

- *public String toString(int hiByte)*

  Tạo một đối tượng String mới từ nội dung của luồng xuất mảng byte

- *public String toString()*

  Phương thức chuyển đổi một luồng byte thành một đối tượng String

Ví dụ:

Viết chương trình tạo lập một luồng xuất mảng (*ByteArrayOutputStream*) 100 byte. Ghi vào luồng xuất mảng 100 phần tử từ 0 đến 99. Đổ dữ liệu từ luồng xuất mảng vào mảng b. In dữ liệu từ mảng b ra màn hình.

```
import java.io.*;
class LuongXuatMang
{
```

```
 public static void main(String[] args)
 {
        try{
        //Tao mot luong xuat mang 100 byte
ByteArrayOutputStream os = new ByteArrayOutputStream(100);
        //Ghi du lieu vao luong
          for(byte i=0;i<100;i++) os.write(i);
        //Doc du lieu tu luong vao mang
          byte[] b = os.toByteArray();
         for(byte i=0;i<100;i++) System.out.print(b[i]+" ");
                os.close();
        }
        catch(IOException e)
        {
                System.err.println(e);
        }

 }
}
```

Kết quả thực hiện chương trình:

C:\MyJava\Baitap>java LuongXuatMang

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99

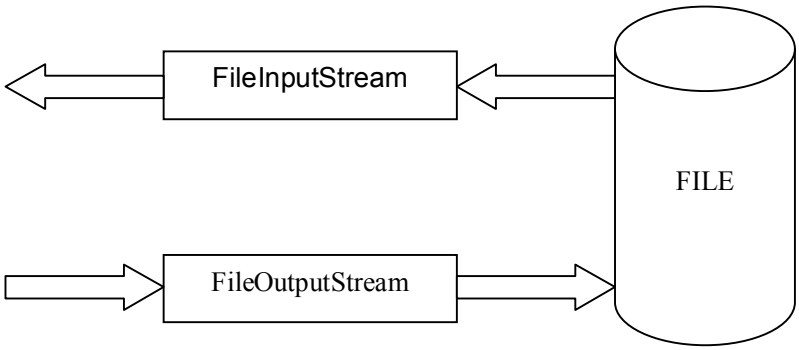
3.4. Luồng xuất nhập tập tin

Phần lớn việc nhập và xuất dữ liệu trong các ứng dụng là đọc và ghi dữ liệu từ các tệp tin và ghi vào dữ liệu vào tệp tin. Hai luồng trong java.io thực hiện việc xuất nhập tệp tin là *FileInputStream* và *FileOutputStream*. Mỗi kiểu luồng có ba constructor.

- Một constructor nhận một đối tượng *String* làm tên của tệp tin.
- Một constructor nhận một đối tượng *File* để tham chiếu đến tệp tin.
- Một constructor nhận đối tượng *FileDescriptor* làm tham số.

FileDescriptor biểu diễn một giá trị phụ thuộc vào hệ thống mô tả một tệp đang mở.

Đối với luồng xuất nhập tập tin ta hình dung như sau: chương trình Java là nơi tiêu thụ dữ liệu, tập tin là nơi cung cấp dữ liệu. Để đọc dữ liệu từ tập tin vào bộ nhớ ta sử dụng luồng nhập tập tin *FileInputStream*. Để ghi dữ liệu từ bộ nhớ vào tập tin ta sử dụng luồng xuất tập tin *FileOutputStream*.



Hình 4.5

Ví dụ

```
import java.io.*;
public class  FileIOExam
{
 public static void main(String[] args)
 {
        //Tao mot file de ghi
        try{
        OutputStream os = new FileOutputStream(args[0]);
        String s = "Thu nghiem voi luong xuat nhap tap tin";
        for(int i=0;i<s.length();i++) os.write(s.charAt(i));
        os.close();
        //Mo de doc
        InputStream is = new FileInputStream(args[0]);
        int len = is.available();
        System.out.println("Luong nhap co "+len+ " bytes");
        byte b[] = new byte[len];
        int sobyte = is.read(b,0,len);
        System.out.println(sobyte+ " la so bytes da doc");
        System.out.println(new String(b));
        is.close();
        }
        catch(IOException e)
        {
                System.err.println(e);
        }

 }
}
```

Kết quả thực hiện chương trình

C:\MyJava\Baitap>java FileIOExam abc.txt

Luong nhap co 38 bytes

38 la so bytes da doc

Thu nghiem voi luong xuat nhap tap tin

3.5. Truy nhập tệp ngẫu nhiên

RandomAccessFile cho phép ta truy nhập trực tiếp vào các tệp, nghĩa là có thể đọc, ghi các byte ở bất kỳ vị trí nào đó trong tệp.

Các phương thức tạo luồng truy nhập tệp ngẫu nhiên

- *RandomAccessFile(String name, String mode) throws IOException*
- *RandomAccessFile(File file, String mode) throws IOException*

Tệp được xác định bởi tên hoặc đối tượng File.

Tham số mode cho phép xác định mở file để đọc hay ghi.

-"r": Dùng để đọc.

-"rw": Dùng để ghi.

Các phương thức khác

- *long getFilePointer() throws IOException :* Trả về vị trí của con trỏ tệp.
- *long length() throws IOException:* cho biết số byte hay độ dài của tệp.
- *void seek(long offset) throws IOException:* Chuyển con trỏ tệp đi offset vị trí kể từ đầu tệp.
- *void close() throws IOException:* Khi không cần truy nhập tệp nữa thì đóng lại.

Ví dụ:

```
import java.io.*;
public class  RandomAccessDemo
{
 static String filename="dayso.dat";
 final static int INT_SIZE=4;
 //Tao file de ghi
 public void createFile() throws IOException
 {
        File datFile = new File(filename);
        RandomAccessFile out_file = new RandomAccessFile(datFile,"rw");
        for(int i=0;i<10;i++)out_file.writeInt(i*i);
        out_file.close();
 }
 //Mo file de doc
 public void readFile() throws IOException
 {
        File datFile = new File(filename);
```

```
        RandomAccessFile inp_file= new RandomAccessFile(datFile,"r");
        System.out.println("Cac so doc tu file:");
        long len = inp_file.length();
        for(int i=INT_SIZE;i<len;i+=2*INT_SIZE)
        {
                inp_file.seek(i);
                System.out.println(inp_file.readInt());
        }
        inp_file.close();
}
//Mo file de ghi
public void extendFile() throws IOException
{
        RandomAccessFile out_file = new RandomAccessFile(filename,"rw");
        for(int i=10;i<20;i++) out_file.writeInt(i*i);
        out_file.close();
}
public static void main(String[] args)
{
        try{
                RandomAccessDemo rnd = new RandomAccessDemo();
                rnd.createFile();
                rnd.readFile();
                rnd.extendFile();
                rnd.readFile();
        }
        catch(IOException e)
        {
                System.err.println(e);
        }

}
}
```

3.5. Luồng PrintStream

Luồng PrintStream được sử dụng mỗi khi cần sử dụng các phương thức print và println trong chương trình. Lớp PrintStream là lớp con của lớp InputStream, vì vậy ta có thể sử dụng luồng này để lọc các byte. Nó cung cấp các phương thức print và println cho các kiểu dữ liệu sau:

char int float Object boolean

char[ ] long double String

Ngoài ra phương thức println không tham số được sử dụng để kết thúc một dòng.

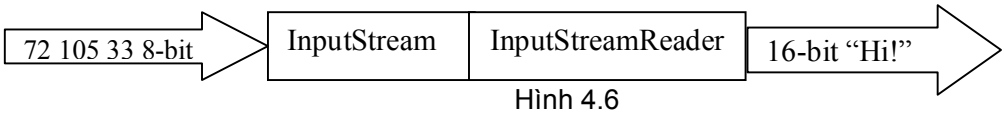
PrintStream hỗ trợ hai constuctor. Constructor thứ nhất nhận tham số là một luồng. Constructor thứ hai có thêm tham số điều khiển việc đẩy dữ liệu ra khỏi luồng.

Ngoài ra còn một số kiểu luồng xuất và nhập khác như DataInputStream, DataOutputStream,…tùy thuộc vào từng tình huống cụ thể mà chúng ta có những lựa chọn cho phù hợp.

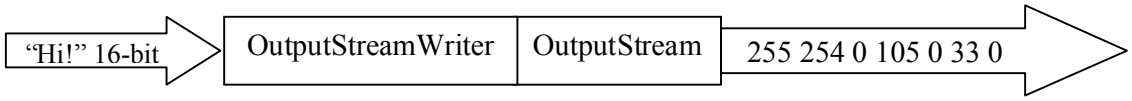
4. Luồng ký tự

Luồng ký tự cung cấp một cách thức để quản lý việc vào ra với các ký tự. Các luồng này sử dụng tập ký tự Unicode và vì thế có thể quốc tế hóa. Trong một số trường hợp làm việc với các luồng ký tự hiệu quả hơn luồng byte.

Các luồng ký tự chuẩn vay mượn từ rất nhiều các lớp luồng hướng byte, bao gồm luồng lọc, luồng đệm, và các luồng tệp tin, và tất cả các luồng được dẫn xuất từ các lớp cha Reader và Writer. Ngoài ra, có hai lớp đóng vai trò cầu nối giữa các luồng byte và các luồng ký tự. Hai lớp này kết hợp các hàm chuyển đổi các ký tự thành các byte và ngược lại theo một kiểu mã hóa đã được xác định. Điều này cho phép một nguồn dữ liệu ASCII được chuyển đổi dễ dàng thành một luồng ký tự Unicode và tương tự cho dữ liệu Unicode được ghi một cách dễ dàng vào một tệp tin theo chuẩn mã hóa cục bộ, cho dù nó là chuẩn 8-bit, UTF-8, hoặc 16 bit.



Hình 4.6



Hình 4.7

4.1. Sự tương ứng giữa luồng byte và luồng ký tự

Bảng dưới đây chỉ ra sự tương ứng giữa luồng byte và luồng ký tự

Luồng byte	Luồng ký tự
OuputStream	Writer
InputStream	Reader
FileOutputStream	FileWriter
FileInputStream	FileReader
ByteArrayInputStream	CharArrayReader

ByteArrayOutputStream	CharArrayWriter
-	StringWriter
StringBufferedInputStream	StringReader
PipedOuputStream	PipedWriter
PipedInputStream	PipedReader
FilterOutputStream	FilterWriter
FilterInputStream	FilterReader
BufferedOuputStream	BufferedWriter
BufferedInputStream	BufferedReader
PushbackInputStream	PushbackReader
LineNumberInputStream	LineNumberReader
PrintStream	PrintWriter
DataOutputStream	-
DataInputStream	-
ObjectInputStream	-
ObjectOuputStream	-
SequenceInputStream	-
-	OuputStreamWriter
-	OutputStreamReader

Bảng 3.1

4.2. Mã hóa ký tự

Khi tạo cầu nối giữa một luồng ký tự và một luồng byte, cần thiết phải xác định cách mã hóa được sử dụng bởi các luồng byte; nghĩa là, các ký tự được biểu diễn bởi từng byte hoặc từng nhóm các byte. Tên của cách mã hóa các byte được đặc tả bởi một xâu ký tự được truyền cho constructor tạo cầu nối OuputStreamReader và InputStreamReader.

Mã hóa	Char	Bytes
US-ASCII	!	33
IBM-EBCDIC	!	90
ISO Latin		
ISO Latin 2		
UTF-8		

Bảng 3.2

Các phương thức hướng ký tự của các luồng byte tương đương với latin 1, còn được biết đến như là ISO Latin 1 hoặc ISO 8859-1; nghĩa là cách mã hóa 8-bit tương ứng với 256 ký tự Unicode đầu tiên. Các tên kiểu mã hóa ký tự thường là phân biệt chữ hoa và chữ thường.

4.3 Lớp Writer

Writer là lớp cha của tất cả các luồng xuất ký tự. Nó cung cấp các phương thức tương tự như luồng OuputStream, nhưng chủ yếu là ghi các ký tự.

4.3.1. Các constructor

Có hai constructor được cung cấp bởi lớp này. Các constructor này là protected. Ta không thể tạo các đối tượng của lớp Writer nhưng ta có thể tạo ra các đối tượng thuộc lớp con của lớp này.

- protected Writer()
- protected Writer(Object obj)

4.3.1. Các phương thức

Lớp Writer cung cấp các phương thức để ghi các ký tự, từ một mảng, hoặc một phần của xâu ký tự.

- void write(int c) throws IOException

  Phương thức này ghi ký tự c vào kênh truyền tin được biểu diễn bởi luồng này.

- void write(char cbuff[]) throws IOException

  Phương thức này ghi mảng ký tự vào luồng

- abstract void write(char cbuff[], int off, int len) throws IOException

Phương thức này ghi *len* ký tự từ mảng cbuff ra luồng gắn với mảng, bắt đầu từ vị trí off. Đây là phương thức trừu tượng bởi vì nó phải được cài đặt bởi một lớp con gắn với kênh truyền tin thực sự, như tệp tin hoặc một luồng khác.

- void write(String str) throws IOException.

  Phương thức này ghi một xâu ký tự str ra luồng.

4.4. Lớp Reader

Reader là lớp cha của tất cả các luồng nhập ký tự. Nó cung cấp các phương thức tương tự như luồng InputStream, nhưng chủ yếu phục vụ cho việc đọc các ký tự.

- protetected Reader()
- protected Reader(Object lock)

4.4.1. Các phương thức

Các phương thức của lớp Reader giống như các phương thức của lớp InputStream ngoại trừ phương thức available() được thay thế bởi phương thức ready().

- int read() throws IOException

Phương thức này đọc một ký tự từ kênh truyền tin được biểu diễn bởi luồng này và trả về ký tự, hoặc giá trị -1 nếu kết thúc luồng.

int read(char cbuff[]) throws IOException

Phương thức này đọc các ký tự vào mảng cbuff

4.5. Lớp OutputStreamWriter

Lớp này cung cấp một cầu nối Writer hướng ký tự với một luồng OutputStream. Các ký tự được ghi vào lớp Writer được chuyển thành các byte tương ứng với một kiểu mã hóa được xác định trong constructor và sau đó được ghi vào luồng OutputStream gắn với nó. Lớp này cung cấp khả năng đệm dữ liệu cho các byte để ghi vào luồng.

Các constructor

- public OutputStreamWriter(OutputStream out)
- public OutputStreamWriter(OutputStream out, String encoding)
- String getEncoding()

Phương thức này trả về tên cách mã hóa các byte được sử dụng để chuyển đổi các ký tự thành các byte.

4.6. Lớp InputStreamReader

Lớp này biểu diễn một cầu nối hướng ký tự của một luồng nhập InputStream. Các byte được đọc từ luồng nhập và được chuyển thành các ký tự tương ứng với kiểu mã hóa được xác định trong constructor.

- InputStreamReader(InputStream in)
- InputStreamReader(InputStream in, String enc)
- String getEncoding()

Phương thức này trả về tên của cách mã hóa byte được sử dụng bởi luồng này để chuyển đổi từ các byte thành các ký tự.

Ví dụ: Chuyển đổi cách mã hóa

```
import java.io.*;
public class Convert
{
 public static void main(String[] args) throws Exception
 {
        if(args.length!=4)throw new IllegalArgumentException("Convert <srcEnc> <source> <dstEnc> <dst>");
        FileInputStream fis=new FileInputStream(args[1]);
        FileOutputStream fos=new FileOutputStream(args[3]);
        InputStreamReader isr=new InputStreamReader(fis,args[0]);
        OutputStreamWriter osw=new OutputStreamWriter (fos,args[2]);
        char b[]=new char[16];
        int num;
        while((num=isr.read(b))>-1)osw.write(b,0,num);
        osw.close();
        isr.close();



 }
}
```

4.7. *Lớp FileWriter*

Lớp Writer này cung cấp một interface luồng ký tự để ghi các tệp tin văn bản bằng cách sử dụng mã hóa mặc định.

Để xác định cách mã hóa được sử dụng để mã hóa một tệp tin, ta sử dụng một luồng OuputStreamWriter gắn với luồng FileOutputStream.

Các constructor

Tạo một đối tượng FileWriter hoàn toàn tương đương với việc tạo ra một đối tượng OuputStreamWriter sử dụng cách mã hóa mặc định và gắn nó với đối tượng FileOuputStream

- *FileWriter(String filename)throws IOException.*

  Constructor này tạo ra một đối tượng FileWriter để ghi thông tin vào một tệp tin cụ thể là fileName, sử dụng cách mã hóa ký tự mặc định. Bất kỳ tệp nào có cùng tên sẽ bị xóa.

- *FileWriter(File file)throws IOException.*

  Constructor này tạo ra một đối tượng *FileWriter* để ghi thông tin vào một tệp tin cụ thể, sử dụng cách mã hóa ký tự mặc định. Bất kỳ tệp nào có cùng tên sẽ bị xóa.

- *FileWriter(String fileName, boolean append) throws IOException.*

  Constructor này tạo ra một đối tượng FileWriter để ghi thông tin vào một tệp tin cụ thể, sử dụng cách mã hóa ký tự mặc định. Biến boolean append xác định cách thức ghi vào tệp tin: ghi mới hay bổ sung thêm nội dụng vào tệp hiện có.

Các phương thức

Lớp FileWriter cung cấp tất cả các phương thức thường dùng của lớp Writer. Việc ghi các ký tự vào một luồng FileWriter tạo ra các ký tự được chuyển thành các byte tương ứng với cách mã hóa cụ thể và các byte này được ghi vào tệp tin gắn với luồng này.

Ngoại lệ IOException sẽ được đưa ra bởi các phương thức của lớp FileWriter nếu gặp một lỗi trong quá trình ghi tệp tin, hoặc không tạo ra được đối tượng FileWriter thì nó đưa ra ngoại lệ IOException.

4.8. Lớp FileReader

Lớp Reader này cung cấp một interface luồng ký tự để đọc các tệp văn bản bằng cách sử dụng cách mã hóa ký tự mặc định. Lớp này cho phép ta đọc các tệp tin văn bản như đọc các luồng ký tự Unicode mà không cần quan tâm đến cách thức mã hóa ký tự .

Để xác định cách mã hóa được sử dụng để giải mã một tệp tin, ta sử dụng một đối tượng InputStreamReader gắn với đốit tượng InputStreamReader.

Việc tạo ra một đối tượng FileReader hoàn toàn tương đương với việc tạo ra một đối tượng InputStreamReader và sau đó gắn nó với luồng FileInputStream.

- FileReader(String fileName)throws FileNotFoundException.

Constructor này tạo ra một đối tượng FileReader đọc nội dung của một tệp tin cụ thể, được xác định bởi tham số fileName bằng cách sử dụng cách mã hóa mặc định.

- FileReader(File file)throws FileNotFoundException.

Constructor này tạo một đối tượng FileReader để đọc nội dung của một tệp tin cụ thể được xác định bởi tệp tin file sử dụng cách thức mã hóa mặc định.

Các phương thức

Lớp FileReader cung cấp các phương thức của lớp Reader. Đọc các ký tự từ một đối tượng FileReader.

Ví dụ minh họa

Ví dụ dưới đây minh họa cách sử dụng các luồng FileWriter và FileReader.

```
import java.io.*;
public class  TepKyTu
{
 public static void main(String[] args) throws IOException
 {
```

```
        FileReader fr=new FileReader(FileDescriptor.in);
        FileWriter fw=new FileWriter(FileDescriptor.out);
        char[] b=new char[256];
        int num;
        while((num=fr.read(b))>-1)
        {
                String upper=new String(b,0,num).toUpperCase();
                fw.write(upper);
                fw.flush();
        }
 }

}
```

Kết quả thực hiện chương trình là

```
C:\>java TepKyTu
Xin chao cac ban! Day la chuong trinh minh hoa FileReader va FileWriter
XIN CHAO CAC BAN! DAY LA CHUONG TRINH MINH HOA FILEREADER VA FILEWRITER
```

Trong ví dụ này, ta gắn FileReader với FileDescriptor.in, luồng này gắn với bàn phím. Ta cũng gắn FileWriter với FileDescriptor.out, luồng này gắn với màn hình. Trong vòng lặp, dữ liệu được đọc từ bàn phím vào vùng đệm b[], chuyển đổi các ký tự này thành chữ viết hoa sau đó ghi dữ liệu ra luồng xuất.

5. Luồng đệm

Các luồng InputStream và OuptutStream là các luồng thô. Chúng đọc và ghi các byte theo từng nhóm. Việc xác định các byte này có ý nghĩa như thế nào-chúng là số nguyên hay các số dấu phẩy động theo chuẩn IEEE 754 hay là các ký tự Unicode-điều này hoàn toàn phụ thuộc vào người lập trình. Tuy nhiên, có những khuôn dạng dữ liệu rất phổ biến đã được cài đặt trong các thư viện lớp. Java cung cấp một số lớp lọc để ta gắn các luồng dữ liệu thô với chúng nhằm mục đích chuyển đổi qua lại giữa các byte và các khuôn dạng dữ liệu khác.

Các luồng lọc cũng có hai loại là:luồng đọc (reader) và luồng ghi (writer). Kết nối các luồng lọc với nhau. Các luồng lọc được kết nối với các luồng thông qua các constructor của chúng.

Ví dụ

```
FileInputStream fis=new FileInputStream("data.txt");
BufferedInputStream bis=new BufferedInputStream(fis);
```

Trước tiên, ta thấy một đối tượng *FileInputStream* được tạo ra với tham số đầu vào là một tên tệp tin *data.txt*. Sau đó, ta tạo tiếp đối tượng *BufferedInputStream bis* với tham số đầu vào của constructor *BufferedInputStream* là *fis*. Từ thời điểm này trở đi, ta có thể sử dụng các phương thức *read()* để đọc cả đối tượng *fis* và *bis*.

Ta cũng có thể xây dựng trực tiếp một luồng bên trong một luồng khác.

```
DataOuputStream dos =new DataOutputStream(new BufferedOutputStream(new FileOutputStream("data.txt")));
```

Các luồng đệm

Lớp *BufferedOutputStream* lưu trữ dữ liệu được ghi vào trong một vùng đệm cho tới khi vùng đệm đầy hoặc là luồng bị *flush()*. Sau đó nó ghi dữ tất cả dữ liệu ra luồng xuất đúng một lần. Điều này làm tăng tốc độ trao đổi dữ liệu.

Lớp *BufferedInputStream* cũng có một mảng byte được sử dụng để làm vùng đệm. Khi một trong các phương thức đọc luồng read() được gọi, trước tiên nó nhận dữ liệu được yêu cầu từ vùng đệm. Chỉ khi vùng đệm hết dữ liệu thực sự nó mới đọc dữ liệu từ nguồn. Lúc này, nó đọc càng nhiều dữ liệu từ nguồn vào vùng đệm nếu có thể, cho dù dữ liệu có cần ngay tức khắc hay không.

- *public BufferedInputStream(InputStream in)*
- *public BufferedInputStream(InputStream in, int bufferSize)*
- *public BufferedOutputStream(OutputSream out)*
- *public BufferedOutputStream(OutputStream out, int bufferSize)*

BufferedInputStream không khai báo các bất kỳ phương thức mới nào của riêng nó. Nó chỉ nạp chồng các phương thức từ InputStream. Nó thực sự hỗ trợ việc đánh dấu và khởi tạo lại luồng.

- *public int read() throws IOException*
- *public int read(byte[] input,*

6. Luồng vào ra mới – New Input Output

Bắt đầu từ phiên bản Java 1.4 Sun đã bổ sung thêm một cách mớI để quản lý các thao tác vào ra.

6.1. Căn bản về NIO

Hệ thống vào ra mớI được xây dựng dựa trên hai hạng mục cơ bản là: buffer và channel. Một vùng đệm (buffer) lưu trữ dữ liệu. Một kênh (channel) biểu diễn một liên kết mở tới một thiết bị vào ra mới, như tệp tin hoặc một socket. Để sử dụng hệ thống vào ra mới, ta phải nhận một kênh truyền tới một thiết bị vào ra và một vùng đệm để lưu trữ dữ liệu. Sau đó ta có thể thực hiện thao tác trên vùng đệm để vào và ra dữ liệu.

6.2. Buffer (Các vùng đệm)

Các vùng đệm được định nghĩa trong gói java.io. Tất cả các vùng đệm là các lớp con của lớp con *Buffer*, lớp này định nghĩa chức năng chính dùng chung cho tất cả các vùng đệm: vị trí hiện thời, giới hạn, và dung lượng. Vị trí hiện tại là chỉ mục trong vùng đệm mà tại đó thao tác đọc và ghi tiếp theo sẽ diễn ra. Giới hạn là chỉ mục cuối cùng của vùng đệm. Dung lượng là số phần tử có trong vùng đệm. *Buffer* cũng hỗ trợ khả năng đánh dấu và khởi tạo. *Buffer* định nghĩa một số phương thức.

Phương thức	Mô tả
Final int capacity()	Trả về số phần tử có trong vùng đệm
Final Buffer clear()	Xóa vùng đệm
Final Buffer flip()	Thiết lập giới hạn của vùng đệm về vị trí hiện hành và thiết lập lại vị trí hiện hành về 0
Final boolean hasRemaining()	Phương thức này trả về giá trị true nếu còn các phần tử trong vùng đệm. Trả về giá trị false nếu ngược lại
Abstract boolean isReadOnly()	Trả về giá trị true nếu vùng đệm là chỉ đọc. Trả về giá trị false nếu ngược lại
Final int limit()	Thiết lập giới hạn của vùng đệm là n
Final Buffer limit(int n)	Thiết lập giới hạn của vùng đệm là n và trả về tham chiếu tới vùng đệm được gọi
Final Buffer mark()	Thiết lập vị trí đánh dấu và trả về tham chiếu tới vùng

	đệm được gọi
Final int Position()	Trả về vị trí hiện hành của vùng đệm
Final Buffer position(int n)	Thiết lập vị trí của Buffer là n. Trả về một tham chiếu tới vùng đệm
Final Buffer reset()	Thiết lập lại vị trí hiện hành của vùng đệm và vị trí đánh dấu được thiết lập trước đó. Trả về một tham chiếu tới vùng đệm
Final Buffer rewind()	Thiết lập vị trí hiện hành của vùng đệm về 0

Bảng 3.3

Lớp Buffer có các lớp con chứa các kiểu dữ liệu như sau:

- ByteBuffer
- IntBuffer
- CharBuffer
- LongBuffer
- DoubleBuffer
- MappedByteBuffer
- FloatBuffer
- ShortBuffer

Tất cả các phương thức get() và put() cho phép ta nhận dữ liệu từ một vùng đệm và đặt dữ liệu vào một buffer.

Phương thức	Mô tả
Abstract byte get()	Trả về byte dữ liệu tại vị trí hiện hành
ByteBuffer get(byte[] vals)	Sao chép dữ liệu từ vùng đệm vào một mảng được tham chiếu bởi mảng vals. Trả về một tham chiếu tới buffer
ByteBuffer get(byte vals[], int start, int num)	Sao chép num số phần tử từ buffer vào mảng được tham chiếu bởi vals, bắt đầu tại chỉ mục được xác định bởi tham số start. Trả về tham chiếu tới vùng đệm. Nếu không còn phần tử nào trong vùng đệm, ngoại lệ BufferUnderflowException
Abstract byte get(int idx)	Trả về byte dữ liệu tại vị trí được xác định bởi chỉ mục idx trong vùng đệm
Abstract ByteBuffer put(byte b)	Sao chép byte dữ liệu b vào tại vị trí hiện hành
final ByteBuffer put(byte b[])	Sao chép tất cả các phần tử của mảng b vào vùng đệm, bắt đầu từ vị trí hiện hành. Trả về tham chiếu tới vùng đệm
ByteBuffer put(byte b[], int start, int num)	Sao chép num phần tử từ mảng b bắt đầu tại vị trí start vào vùng đệm. Trả về tham chiếu tới vùng đệm. Nếu vùng đệm không chưa được tất cả các phần tử của vùng đệm thì ngoại lệ BufferOverflowException sẽ được đưa ra
BufferByte put(ByteBuffer bb)	Sao chép tất cả các phần tử của vùng đệm BufferByte gọi, bắt đầu từ vị trí hiện hành . Nếu vùng đệm không chưa được tất cả các phần tử của vùng đệm thì ngoại lệ BufferOverflowException sẽ được đưa ra
Abstract ByteBuffer put(int idx, byte b)	Sao chép byte dữ liệu b tại vị trí idx vào vùng đệm. Trả về tham chiếu tới vùng đệm

Bảng 3.4

Phương thức put()được định nghĩa trong ByteBuffer. Tất cả các lớp buffer còn hỗ trợ các phương thức thực hiện các thao tác khác nhau trên vùng đệm.

6.3. Các kênh (Channel)

Các kênh được định nghĩa trong gói java.io.channel. Một kênh biểu diễn một liên kết mở tới một nguồn hoặc đích vào ra. Ta có thể nhận được một kênh bằng cách gọi phương thức getChannel() trên một đối tượng hỗ trợ kênh. Java 2 phiên bản 1.4 đưa thêm vào phương thức getChannel() cho các lớp sau:

- FileInputStream
- FileOutputStream
- RandomAccessFile
- Socket
- ServerSocket
- DatagramSocket

Để nhận được một kênh, trước tiên ta phải nhận một đối tượng của các lớp này và sau đó gọi phương thức getChannel() trên đối tượng đó.

Kiểu kênh cụ thể được trả về phụ thuộc vào kiểu đối tượng chịu tác động của phương thưc getChannel(). Ví dụ khi gọi phương thức getChannel() trên đối tượng FileInputStream, FileOutputStream hoặc RandomFileAccess thì kênh trả về là FileChannel. Khi gọi phương thức getChannel() trên đối tượng Socket thì kiểu kênh trả về là SocketChannel().

Các kênh FileChannel và SocketChannel hỗ trợ các phương thức read() và write() cho phép ta thực hiện các thao tác vào ra thông qua kênh. Dưới đây là một số phương thức read() và write()được định nghĩa trong FileChannel.

Phương thức	Mô tả
abstract int read(ByteBuffer bb)	Đọc các byte từ kênh vào một vùng đệm bb cho tới khi đầy vùng đệm hoặc không còn dữ liệu trên kênh. Kiểu trả về là số byte thực sự đọc được
abstract int read(ByteBuffer bb, long start)	Đọc các byte từ kênh vào một vùng đệm, bắt đầu từ vị trí start cho tới khi đầy vùng đệm hoặc không còn dữ liệu đầu vào. Vị trí hiện thời không thay đổi. Trả về số byte đã đọc được hoặc –1 nếu kết thúc luồng
abstract int write(ByteBuffer bb)	Ghi nội dung của vùng đệm ra kênh, bắt đầu tại vị trí hiện hành. Trả về số byte đã được ghi.
abtsract int write(ByteBuffer bb, int start)	Ghi nội dung của vùng đệm ra kênh. Bắt đầu tại vị trí hiện start. Trả về số byte đã được ghi

Bảng 3.5

Tất cả các kênh đều hỗ trợ các phương thức bổ trợ cho phép ta truy xuất và điều khiển kênh. Ví dụ, FileChannel hỗ trợ các phương thức để nhận và thiết lập vị trí hiện hành, truyền thông tin qua lại giữa các kênh, nhận kích thước hiện thời của kênh, khóa kênh,..FileChannel cũng cung cấp phương thức map()để ánh xạ một tệp vào một buffer.

6.4. Charset và Selector

Hai thực thể khác được sử dụng bởi NIO là các CharSet và Selector.

CharSet xác định cách ánh xạ các byte thành các ký tự. Ta có thể mã hóa một xâu ký tự bằng cách sử dụng một bộ mã hóa và cũng có thể giải mã một dãy các byte thành các ký

tự bằng cách sử dụng bộ giải mã. Charset, encoder và decoder được hỗ trợ bởi gói java.nio.charset.

Selector hỗ trợ vào ra ghép kênh, không phong tỏa, dựa trên phím. Ngoài ra, selector còn cho phép ta làm việc với nhiều kênh. Selector được hỗ trợ bởi các lớp trong gói java.io.channels. Các selector ứng dụng nhiều nhất với các kênh dựa trên luồng.

6.5. Sử dụng hệ thống vào ra mới

Đơn vị dữ liệu vào ra phổ biến nhất là tệp tin, trong phần này ta sẽ xem cách thức để truy xuất tới các tệp tin trên đĩa bằng cách sử dụng hệ thống vào ra mới. Do hầu hết các thao tác trên tệp là mức byte nên kiểu vùng đệm được sử dụng sẽ là ByteBuffer.

6.5.1. Đọc tệp

Có một số cách để đọc dữ liệu từ một tệp tin bằng cách sử dụng hệ thống vào ra mới. Chúng ta sẽ xem xét hai cách. Cách thứ nhất đọc một tệp tin bằng cách ánh xạ nó vào một buffer và sau đó thực hiện một thao tác đọc. Cách thứ hai để đọc một tệp tin là tự động hóa quá trình đọc.

- ***Cách 1:***

Bước 1: Mở một tệp tin để đọc bằng cách sử dụng luồng FileInputStream.

Bước 2: Nhận một kênh từ đối tượng FileInputStream nhờ phương thức

FileChannel getChannel()

Bước 3: Xác định kích thước của tệp tin bằng cách gọi phương thức size()

Long size() throws IOException

Bước 4:

Gọi phương thức allocate()để phân bổ một vùng đệm đủ lớn để lưu giữ nội dung của tệp.

static ByteBuffer allocate(int cap)

Ví dụ

```
import java.io.*;
import java.nio.*;
import java.nio.channels.*;

public class  ChannelRead
{
 public static void main(String[] args)
 {
        FileInputStream fis;
        FileChannel fc;
        long fSize;
        ByteBuffer bb;
        try{
              //Mo mot ep
              fis=new FileInputStream(args[0]);
              //Mo mot kenh toi tep
              fc=fis.getChannel();
```

```
                //Nhan kich thuoc tep tin
                fSize=fc.size();
                //Phan bo mot vung dem co kich thuoc can thiet
                bb=ByteBuffer.allocate((int)fSize);
                //Doc tep tin vao vung dem
                fc.read(bb);
                //Mo tep de doc
                bb.rewind();
                for(int i=0;i<fSize; i++) System.out.print((char)bb.get());
                fc.close();
                fis.close();
            }
            catch(IOException e)
            {
                                System.out.println(e);
            }

        }
    }
```

Kết quả thực hiện chương trình

C:\MyJava>javac ChannelRead.java

C:\MyJava>java ChannelRead Bai3.java

```
class Bai3 {
  public static void main( String args[] ) {
        double x = 42 ;
      System.out.println( x = 42 % 3 + 3 * 3 - 3 / 3  );
  }
}
```

- ***Cách 2***

Một cách dễ hơn để đọc một tệp tin là ánh xạ vào một vùng đệm. Ưu điểm cho của cách tiếp cận này là vùng đệm tự động lưu nội dung của tệp tin. Không cần thao tác đọc cụ thể nào.

Các bước thực hiện

Bước 1: Mở một tệp tin bằng cách sử dụng luồng FileInputStream

Bước 2: Nhận một kênh tới tệp tin đó bằng cách gọi phương thức getChannel() trên đối tượng FileInputStream.

Bước 3: Ánh xạ kênh với một vùng đệm bằng cách gọi phương thức map() trên đối tượng FileChannel. Phương thức map có dạng như sau:

MappedByteBuffer map(FileChannel.MapMode how, long pos, long size) throws IOException

Phương thức *map()* làm cho dữ liệu trong tệp tin được ánh xạ vàơo vùng đệm trong bộ nhớ. Tham số how xác định kiểu thao tác được phép thực hiện trên tệp tin:

MapMode.READ

MapMode.READ_WRITE

MapMode.PRIVATE

Để đọc một tệp tin ta dùng chế đọ *MapMode.READ*. Để đọc và ghi tệp ta dùng chế độ *MapMode.READ_WRITE*. Chế độ *MapMode.PRIVATE* chỉ làm cho một bản sao riêng của một tệp bị thay đổi và những thay đổi này không ảnh hưởng tới tệp tin. Vị trí trong tệp tin bắt đầu ánh xạ được xác định bởi tham số *pos* và số byte ánh xạ được xác định bởi *size*. Phương thức trả về là một tham chiếu *MappedByteBuffer*, là một lớp con của *ByteBuffer*. Mỗi khi tệp tin được ánh xạ vào vùng đệm ta có thể đọc tệp từ vùng đệm.

```
import java.io.*;
import java.nio.*;
import java.nio.channels.*;
public class  MappedChannelRead
{
 public static void main(String[] args)
 {
        FileInputStream fis;
        FileChannel fc;
        MappedByteBuffer mbb;
        long fSize;
        try{
                //Mo tep de doc
                fis=new FileInputStream(args[0]);
                //Mo kenh
                fc=fis.getChannel();
                //Nhan kich thuoc tep
                fSize=fc.size();
                // Anh xa file vao vung dem
                mbb=fc.map(FileChannel.MapMode.READ_ONLY,0,fSize);
                //Doc cac byte tu vung dem
                for(int i=0; i<fSize;i++) System.out.print((char)mbb.get());
                fc.close();
                fis.close();

        }
        catch(IOException e)
        {
```

```
            System.out.println(e.getMessage());
            System.exit(1);
            }
 }
}
```

Kết quả thực hiện

```
C:\MyJava>java MappedChannelRead Bai3.java
class Bai3 {
  public static void main( String args[] ) {
        double x = 42 ;
      System.out.println( x = 42 % 3 + 3 * 3 - 3 / 3  );
  }
 }
```

6.5.2. Ghi tệp tin

Có một số cách để ghi tệp thông qua một kênh. Ở đây, chúng ta cũng tìm hiểu hai cách ghi tệp. Cách thứ nhất là ghi tệp thông qua một kênh bằng cách sử dụng các thao tác write. Cách thứ hai, nếu tệp tin được mở để thực hiện các thao tác đọc/ghi, ta có thể ánh xạ tệp vào một vùng đệm và sau đó ghi vào vùng đệm. Những thay đổi với vùng đệm sẽ được tự động ảnh hưởng đến tệp tin. Cả hai cách đều được mô tả trong mục này.

Để ghi một tệp thông qua kênh bằng cách sử dụng các lời gọi tới phương thức write(), ta thực hiện các bước sau đây.

Bước 1: Mở một tệp để ghi.

Bước 2: Xác định một vùng đệm byte để ghi dữ liệu vào vùng đệm đó, sau đó gọi phương thức write().

```
import java.io.*;
import java.nio.*;
import java.nio.channels.*;


public class  ChannelWrite
{
 public static void main(String[] args)
 {
            FileOutputStream fos;
            FileChannel fc;
            ByteBuffer bb;
            try
            {       String s="This is a test of NIO system";
                    fos=new FileOutputStream(args[0]);
                    fc=fos.getChannel();
                    bb=ByteBuffer.allocateDirect(s.length());
```

```
                //Ghi mot so byte vao vung dem

                byte[] b=s.getBytes();
                for(int i=0;i<b.length;i++)bb.put(b[i]);
                bb.rewind();
                fc.write(bb);
                fc.close();
                fos.close();
            }
            catch(Exception e)
            {
                System.err.println(e);
            }


     }
    }
```

Sao chép một tệp bằng cách sử dụng tiện ích vào ra mới

Hệ thống vào ra mới đơn giản hóa một số kiểu thao tác trên tệp tin. Ví dụ, chương trình dưới đây sao chép một tệp tin.

```
import java.io.*;
import java.nio.*;
import java.nio.channels.*;

public class NIOCopy
{
 public static void main(String[] args)
 {
        FileOutputStream fos;
        FileInputStream fis;
        FileChannel fco,fci;
        long fSize;
        MappedByteBuffer mbb;
        try{
            fis=new FileInputStream(args[0]);
            fos=new FileOutputStream(args[1]);

            fci=fis.getChannel();
            fco=fos.getChannel();
```

```
                fSize=fci.size();
                mbb=fci.map(FileChannel.MapMode.READ_ONLY,0,fSize);
                fco.write(mbb);

                fci.close();
                fco.close();
                fos.close();
                fis.close();
            }
            catch(Exception e)
            }

        }
    }
```

7. Kết luận

Chương này chúng ta đã tìm hiểu các khái niệm căn bản về vào ra bằng cách sử dụng các luồng trong Java. Cũng trong chương này các luồng hướng byte và các luồng hướng ký tự trong Java đã được giới thiệu. Khái niệm vào ra mới bằng cách sử dụng các kênh (channel) và vùng đệm (buffer) cũng được giới thiệu trong chương này. Ở các chương tiếp theo các bạn sẽ thấy hầu hết các chương trình lập trình mạng đều vào ra dữ liệu bằng cách sử dụng các luồng. Việc hiểu biết sâu về luồng vào ra sẽ là một lợi thế để bạn đọc tiếp cận với các chương tiếp theo.

Chương 4

Lập trình đa tuyến đoạn

1. Tổng quan

Khi thực hiện một công việc phức tạp người ta thường chia công việc ra thành nhiều phần và giao công việc cho nhiều người cùng thực hiện, điều này giúp cho công việc được tiến hành nhanh chóng. Các ứng dụng phần mềm sử dụng một chiến lược tương tự được gọi là đa tuyến đoạn để chia nhỏ các tác vụ thành các đơn vị dễ quản lý. Lập trình đa tuyến đoạn là một khái niệm quan trọng trong lập trình mạng bằng Java vì các client và server thường phải thực hiện một số tác vụ đồng thời tại cùng một thời điểm (ví dụ lắng nghe các yêu cầu và đáp ứng các yêu cầu, xử lý dữ liệu và cập nhật giao diện đồ họa người dùng). Trước khi đi vào tìm hiểu lập trình đa tuyến đoạn trong Java, ta cần hiểu rõ sự khác nhau giữa lập trình đơn tuyến đoạn, lập trình đa tiến trình và lập trình đa tuyến đoạn.

1.1. Lập trình đơn tuyến đoạn

Khái niệm đa tuyến đoạn là khái niệm khó đối với những người mới bắt đầu làm quen. Rất nhiều ngôn ngữ lập trình và hệ điều hành trước đây không hỗ trợ đa tuyến đoạn.

Phần mềm truyền thống được viết bằng các ngôn ngữ thủ tục được biên dịch thành một khuôn dạng mà máy có thể hiểu được gọi là mã máy. Bộ xử lý trung tâm đọc mã này và xử lý các lệnh theo cấu trúc tuần tự hết lệnh này đến lệnh tiếp theo. Thời gian thực hiện các lệnh có thể thay đổi tùy thuộc vào bản chất của các lệnh.

Ưu điểm chính của kiểu lập trình này là tính đơn giản của nó. Nếu một lệnh không hoàn thành thì lệnh tiếp theo sẽ không được xử lý. Điều này nghĩa là người lập trình có thể dự đoán trạng thái của máy tại bất kỳ thời điểm nào cho trước.

1.2. Lập trình đa tiến trình

Đa nhiệm là khả năng của một hệ điều hành máy tính chạy nhiều chương trình đồng thời trên một CPU. Điều này được thực hiện bằng cách chuyển hoạt động từ một chương trình này sang chương trình khác tương đối nhanh để tạo cho người sử dụng cảm giác tất cả các chương trình đang được xử lý đồng thời. Có hai kiểu đa nhiệm:

- Đa nhiệm ưu tiên. Trong đa nhiệm ưu tiên, hệ điều hành xác định cách phân bổ các thời gian của CPU cho từng chương trình. Cuối mỗi khoảng thời gian mà CPU phân bổ, chương trình hiện đang hoạt động buộc phải trả quyền điều khiển cho hệ điều hành, dù nó có muốn hay không. Các ví dụ về hệ điều hành hỗ đa nhiệm ưu tiên là Unix, Windows 95/98, Windows NT.
- Đa nhiệm hợp tác. Trong đa nhiệm hợp tác, mỗi chương trình kiểm soát một phần thời gian CPU mà nó cần. Điều này nghĩa là một chương trình phải hợp tác để trao quyền điều khiển cho các chương trình khác, nếu không nó sẽ chiếm dụng CPU. Các hệ điều hành đa nhiệm hợp tác là Windows 3.1 và Mac OS 8.5.

Những ai đã quen lập trình trên hệ thống Unix hẳn là đã quen với khái niệm lập trình đa tiến trình. Để hỗ trợ đa nhiệm, Unix sử dụng khái niệm các tiến trình. Mỗi ứng dụng đang chạy là một tiến trình, với bộ nhớ được phân bổ cho chương trình và dữ liệu. Có nhiều tiến trình chạy trên cùng một máy. Hệ điều hành sẽ phân bổ thời gian cho từng tiến trình, dừng tiến trình khi hết thời gian và cho phép tiến trình khác tiếp tục. Đôi khi, một tiến trình bị phong tỏa hoặc có thể tự chọn để giành thời gian CPU.

Lập trình đa tiến trình có các lợi ích khác. Các chương trình tự chúng có thể tạo ra các tiến trình mới , một phần chương trình thực hiện một tác vụ trong khi một phần khác thực hiện công việc khác. Ví dụ, khi đang kiểm tra email trên một máy ở xa, giao diện người dùng có thể hiển thị diễn tiến của thao tác và cho phép người dùng soạn thảo các thông điệp và đọc các thông điệp đã được tải về trước đó.

Mặc dù lập trình đa tiến trình hoạt động tốt, nhưng nó vẫn có những nhược điểm. Trước hết, khi một tiến trình phân nhánh thành hai tiến trình, sẽ dẫn đến sự chồng chéo giữa

việc lưu trữ dữ liệu của tiến trình này với tiến trình khác. Mỗi tiến trình cần có một bản sao dữ liệu của riêng nó, vì vậy nếu có nhiều tiến trình thì sẽ cần nhiều bộ nhớ. Thứ hai là không có cách nào để một tiến trình truy xuất và sửa đổi dữ liệu của một tiến trình khác.

1.3. Lập trình đa tuyến đoạn

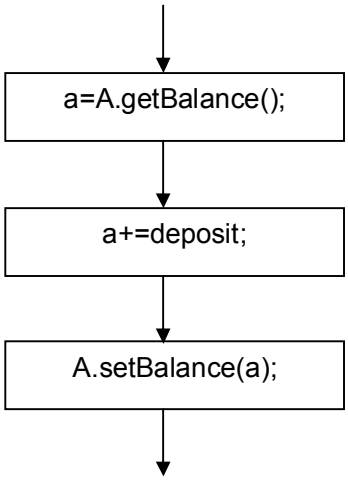
Đa tuyến đoạn mở rộng khái niệm đa nhiệm bằng cách cho phép một chương trình thực hiện một số tác vụ đồng thời. Mỗi tác vụ được xem như là một tuyến đoạn và nó có một luồng điều khiển riêng. Đa tuyến đoạn rất hữu ích trong các ứng dụng thực tế. Ví dụ, nếu quá trình nạp một trang web vào trình duyệt quá lâu, người sử dụng cần phải có khả năng ngắt việc nạp trang web đó bằng cách ấn nút lệnh stop. Giao diện người dùng có thể tiếp tục đáp ứng các yêu cầu của người dùng bằng cách sử dụng một tuyến đoạn riêng cho hoạt động nạp trang web.

Để lập trình đa tuyến đoạn ta cần có một cách nhìn nhận về phần mềm khác. Ngoài việc xử lý tuần tự, các tác vụ còn có thể được xử lý đồng thời-nghĩa là, nhiều tác vụ được thực hiện cùng một lúc mà không cần đợi tác vụ này hoàn thành mới tiến hành tác vụ khác. Đa tuyến đoạn còn có nghĩa là nhiều tuyến xử lý, cho phép một chương trình có nhiều thể hiện cùng hoạt động, cùng sử dụng chung bộ nhớ. Một ứng dụng có thể thực hiện nhiều tác vụ đồng thời và các tuyến đoạn có thể truy xuất tới các biến dữ liệu dùng chung để cùng làm việc hợp tác với nhau.

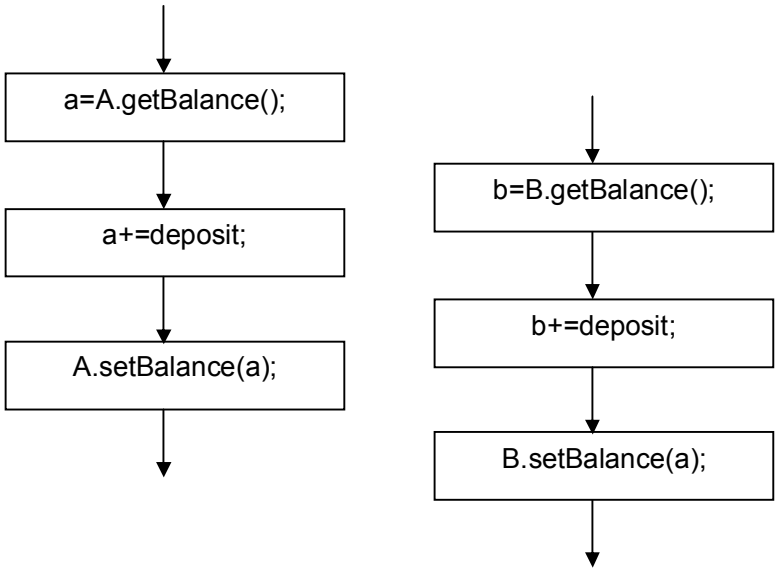
Nếu máy tính chỉ có một CPU, thì chỉ có một tuyến đoạn đang chạy tại một thời điểm cho trước. Hệ điều hành duy trì một hàng đợi các tuyến đoạn và phân bổ thời gian CPU cho chúng. Tuy nhiên, thời gian phân bổ thời gian cho một tuyến đoạn là công bằng trong nhiều tình huống, vì nó ngăn các tuyến đoạn khác thực hiện công việc. (tình trạng này còn được gọi là đói tuyến đoạn). Ta cần phải có một cách nào đó để phân bổ thời gian CPU giành cho mỗi tuyến đoạn là như nhau.

Mục đích của tiến trình và tuyến đoạn đều là cho phép nhiều máy tính thực hiện nhiều tác vụ đồng thời.

Ví dụ: A là một tài khoản tại ngân hàng. Phương thức getBalance(): lấy ra giá trị của tài khoản. Phương thức setBalance(): chép lại giá trị vào bản ghi tài khoản. Xét ví dụ tuyến đoạn dưới đây



Hình 4.1



Hình 4.2

Đa tuyến đoạn (Multi-thread)

Một ứng dụng có thể có nhiều tuyến đoạn. Khả năng làm việc với nhiều tuyến đoạn được gọi là đa tuyến đoạn. Đa tuyến đoạn cho phép bạn viết các chương trình hiệu quả tận dụng tối đa CPU bằng cách duy trì thời gian trễ là tối thiểu.

Tại cùng một thời điểm có hai khách hàng cùng ghi vào cùng một tài khoản. Nếu thực hiện như vậy thì chỉ có tuyến đoạn thứ hai mới thực sự ảnh hưởng tới tài khoản, kết quả của giao dịch thứ nhất sẽ bị mất. Để khắc phục điều này cần có một cơ chế để thông báo một đối tượng đang được sử dụng hay không.

Vòng đời của một tuyến đoạn

Một tuyến đoạn có thể ở một trong bốn trạng thái sau trong suốt vòng đời của nó:

- new-Một tuyến đoạn mới là một tuyến đoạn được tạo ra bằng cách sử dụng toán tử new nhưng vẫn chưa được khởi động.
- runnable-Một tuyến đoạn ở trạng thái runnable mỗi khi phương thức start() của nó được kích hoạt. Điều này nghĩa là mã lệnh trong phương thức run() có thể được xử lý bất kỳ khi nào giành được quyền xử lý từ hệ điều hành.
- blocked-(bị phong tỏa)- Một tuyến đoạn ở chuyển vào trạng thái blocked (bị phong tỏa) nếu một trong các sự kiện sau xảy ra:
    - Phương thức sleep() của tuyến đoạn được gọi. Trong trường hợp này, tuyến đoạn vẫn ở trạng thái blocked cho tới khi hết một số ms (mili giây) xác định.
    - Tuyến đoạn gọi phương thức wait() của một đối tượng. Trong trường hợp này tuyến đoạn vẫn ở trạng thái blocked cho tới khi phương thức notify() hoặc notifyAll() được gọi từ một tuyến đoạn khác. Các phương thức wait(), notify()và notifyAll() thường được tìm thấy trong các phương thức đồng bộ synchronized của đối tượng.
    - Tuyến đoạn phong tỏa một thao tác vào/ra. Trong trường hợp này, tuyến đoạn bị phong tỏa cho tới khi hoạt động vào ra hoàn thành.
- Dead (chết)-Một tuyến đoạn thường ở trạng dead khi phương thức run() hoàn thành việc xử lý.

2. Tạo các ứng dụng đa tuyến đoạn với lớp Thread

Lớp java.lang.Thread cung cấp các phương thức để khởi động (start()), tạm dừng (suspend()), phục hồi (resume()) và dừng hẳn (stop()) một tuyến đoạn, cũng như kiểm soát các khía cạnh khác như độ ưu tiên của tuyến đoạn hoặc tên của tuyến đoạn gắn với nó. Cách đơn giản nhất để sử dụng lớp Thread là thừa kế lớp này và nạp chồng phương thức run(), phương thức này được gọi khi tuyến đoạn được khởi động lần đầu. Bằng cách nạp chồng phương thức run(), một tuyến đoạn có thể thực hiện một số tác vụ hữu ích ở hậu trường.

Chú ý: Cần nhớ rằng các tuyến đoạn không chạy tự động tại thời điểm chạy. Thay vào đó, ta phải gọi phương thức Thread.start(), nếu không tuyến đoạn sẽ không chạy.

Khuông dạng chung để tạo một ứng dụng đa tuyến đoạn bằng cách sử dụng lớp Thread

```
class C1 extends Thread
{
  public C1(){this.start();}
  public void run(){...}
}
```

Ví dụ: Viết chương trình tạo lập một ứng dụng đa tuyến đoạn. Tạo lập ra hai tuyến đoạn mỗi tuyến đoạn in ra một từ với tốc độ khác nhau bằng cách thừa kế lớp Thread

```
class PingPong extends Thread {
    String word;
        int delay;
        PingPong(String s, int d)
        {
                word =s;
                delay=d;
        }
        public void run()
        {
                try{
                        for(;;)
                        {
                                System.out.print(word+" ");
                                sleep(delay);
                        }
                }
                catch(InterruptedException e)
                {
                        return;
                }
        }
        public static void main(String[] args)
        {
                new PingPong("ping",33).start();
                new PingPong("PONG",100).start();
        }
}
```

3. Tạo ứng dụng đa tuyến đoạn với giao tiếp Runnable

Thừa kế lớp Thread là một cách để tạo ra ứng dụng đa tuyến đoạn nhưng nó không phải là giải pháp tốt nhất. Chúng ta đều biết rằng Java chỉ hỗ trợ đơn thừa kế nên việc cài đặt các ứng dụng đa thừa kế là không thể được nếu tạo ứng dụng đa tuyến đoạn bằng cách thừa kế từ lớp Thread. Một giải pháp có thể khắc phục điều này là thực thi giao tiếp java.lang.Runnable.

Giao tiếp Runnable định nghĩa duy nhất một phương thức run(). Các lớp thực thi giao tiếp này chỉ ra rằng chúng có thể chạy độc lập như một tuyến đoạn riêng. Giao tiếp này không định nghĩa bất kỳ phương thức nào khác hoặc cung cấp bất kỳ chức năng tuyến đoạn cụ thể nào. Mục đích duy nhất của nó là báo hiệu các lớp thực thi giao tiếp này có thể chạy như các tuyến đoạn. Khi một đối tượng thực thi giao tiếp Runnale được truyền cho constructor của một tuyến đoạn, và các phương thức start() của tuyến đoạn được gọi, phương thức run() sẽ tự động được gọi bởi tuyến đoạn vừa được tạo ra. Khi phương thức run() kết thúc xử lý, tuyến đoạn sẽ dừng hoạt động.

Việc sử dụng giao tiếp Runnable có một số ưu điểm so với thừa kế lớp Thread. Trước hết, lớp thực thi giao tiếp Runnable có thể tự do thừa kế từ một lớp khác. Thứ hai, cùng một đối tượng Runnable có thể được truyền cho nhiều tuyến đoạn, vì vậy một số tuyến đoạn tương tranh có thể sử dụng chung mã và thao tác trên cùng dữ liệu.

Khuôn dạng chung để tạo một ứng dụng đa tuyến đoạn bằng cách thực thi giao tiếp Runnable

```
class C2() implements Runnable
{
  public C2(){Thread t = new Thread(this);}
  public void run(){...}
}
```

Để lập trình tuyến đoạn ta phải sử dụng gói java.lang, tuy nhiên do gói ngày được mặc định nên ta không cần phải khai báo.

Ví dụ: Viết chương trình tạo lập một ứng dụng đa tuyến đoạn. Tạo lập ra hai tuyến đoạn mỗi tuyến đoạn in ra một từ với tốc độ khác nhau bằng cách thực thi giao tiếp Runnable.

```
class RunPingPong implements Runnable
{
 String word;
 int delay;
 RunPingPong(String w, int d)
 {
        word =w;
        delay=d;
 }
 public void run(){
        try{
              for(;;){
                    System.out.println(word+" ");
                    Thread.sleep(delay);
              }
        }
```

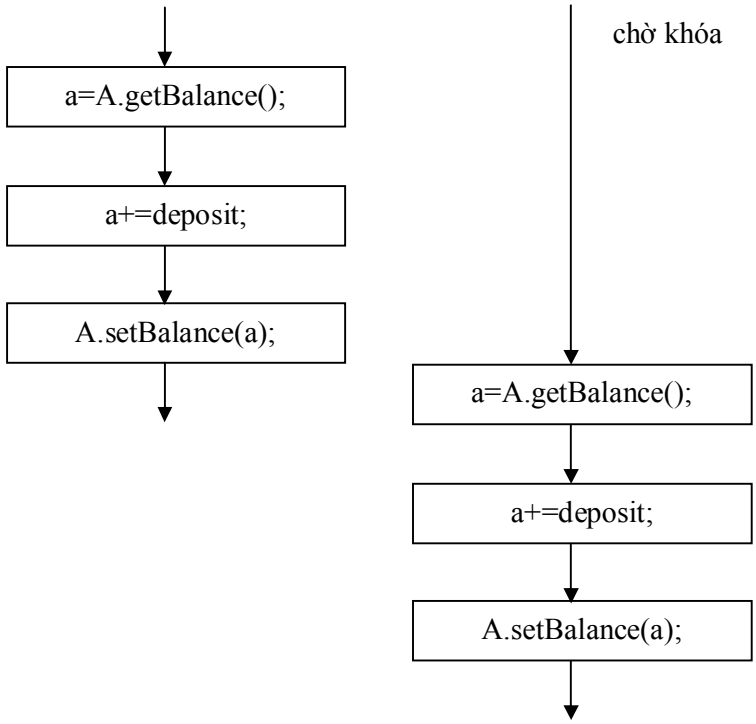
```
            catch(InterruptedException e)
            {
                  return;
            }
      }
      public static void main(String[] args)
      {
            Runnable ping = new RunPingPong("ping",33);
            Runnable pong = new RunPingPong("PONG",100);
            new Thread(ping).start();
            new Thread(pong).start();
      }
}
```

4. Sự đồng bộ hóa (Synchronization)

Khi hai tuyến đoạn cần sử dụng cùng một đối tượng, có một khả năng có các thao tác đan xen nhau làm phá hỏng dữ liệu. Đa tuyến đoạn có một cơ chế để ngăn ngừa điều đó, bằng cách sử dụng phương thức chiếm dụng đối tượng. Nếu một đối tượng bị phong tỏa bởi một tuyến đoạn nào đó thì chỉ có tuyến đoạn đó mới có thể truy cập tới đối tượng.

4.1. Các phương thức synchronized

- Để làm cho một lớp có thể sử dụng được trong môi trường đa tuyến đoạn, các phương thức tương ứng sẽ được khai báo là synchronized.
- Nếu một tuyến đoạn kích hoạt một phương thức synchronized trên một đối tượng, đối tượng đó sẽ bị chiếm dụng bởi tuyến đoạn đó. Một tuyến đoạn khác kích hoạt phương thức synchronized trên cùng đối tượng đó sẽ bị phong tỏa cho tới khi khóa trên đối tượng được giải phóng.



Hình 4.3

Ví dụ: Cài đặt lớp Account:

```
class Account{
  private double balance;
  public Account(double initialDeposit)
  {
        balance = initialDeposit;
  }
  public synchronized double getBalance()
  {
        return balance;
  }
  public synchronized void getBalance(double amount)
  {
     balance+=amount;
  }
 }
}
```

4.2.Lệnh synchronized

Lệnh synchronized cho phép đồng bộ hóa một đối tượng mà không cần yêu cầu bạn tác động một phương thức synchronized trên đối tượng đó.

- Cú pháp

```
synchronized (expr)
      statement
```

Khi có được khóa, statement được xử lý giống như nó là một phương thức synchronized trên đối tượng đó.

Ví dụ: Chuyển các phần tử trong mảng thành các số không âm

```
public static void abs(int[] v)
{
 synchronized(v)
 {
  for(int i=0;i<v.length;i++)
        {
            if(v[i]<0) v[i]=-v[i];
        }
 }
 }
```

Java có thể chạy trên cả máy đơn và máy đa xử lý, với nhiều tuyến đoạn hay đơn tuyến đoạn.

5. Phương thức wait và notify

Cơ chế chiếm dụng đồng bộ hóa ngăn cho các tuyến đoạn chồng chéo nhau. Nhưng trong một số trường hợp ta cũng cần phải cung cấp một cách nào đó để các tuyến đoạn truyền tin với nhau. Java cung cấp cho người sử dụng các phương thức cho phép các tuyến đoạn không bị xử lý chồng chéo mà vẫn có thể trao đổi thông tin với nhau bằng cách sử dụng các phương thức wait() và notify(). Để thực hiện điều này phương thức wait() được định nghĩa để cho phép một tuyến đoạn đợi cho tới khi một điều kiện nào đó xảy ra. Phương

thức notify() được định nghĩa để báo cho các tuyến đoạn biết sự kiện nào đó đã xảy ra. Các phương thức này được định nghĩa trong lớp Object và được thừa kế từ các lớp Object.

```
public class WaitNotify extends Thread
{
        public static void main(String args[]) throws Exception
        {
                Thread notificationThread = new WaitNotify();
                notificationThread.start();
                // Chờ tuyến đoạn cảnh báo kích hoạt sự kiện
                synchronized (notificationThread)
                {
                        notificationThread.wait();
                }
                // Báo cho người dùng biết phương thức wait đã hoàn thành
                System.out.println ("The wait is over");
        }
        public void run()
        {
                        System.out.println ("Hit enter to stop waiting thread");
                        try
                        {
                                System.in.read();
                        }
                        catch (java.io.IOException ioe)
                        {
                                // no code req'd
                        }
                        // Notify any threads waiting on this thread
                        synchronized (this)
                        {
                        this.notifyAll();
                        }
        }
}
```

Một số dạng của phương thức wait và notify

Tất cả các phương thức đều có trong lớp Object và hoạt động trên đối tượng hiện thời:

- *public final void wait(long timeout) throws InterruptedException*

  Tuyến đoạn hiện thời chờ cho tới khi được cảnh báo hoặc một khoảng thời gian timeout nhất định. Nếu timeout bằng 0 thì phương thức sẽ chỉ chờ cho tới khi có cảnh báo về sự kiện.

- public final void notify()

  Cảnh báo ít nhất một tuyến đoạn đang chờ một điều kiện nào đó thay đổi. Các tuyến đoạn phải chờ một điều kiện thay đổi trước khi có thể gọi phương thức wait nào đó.

- public final void notifyAll()

  Phương thức này cảnh báo tất cả các tuyến đoạn đang chờ một điều kiện thay đổi. Các tuyến đoạn đang chờ thường chờ một tuyến đoạn khác thay đổi điều kiện nào đó. Trong số các tuyến đoạn đã được cảnh báo, tuyến đoạn nào có độ ưu tiên cao nhất thì sẽ chạy trước tiên.

6. Lập lịch cho tuyến đoạn

Chương trình Java có thể chạy trên cả các máy đơn xử lý và đa xử lý với nhiều tuyến đoạn hoặc đơn tuyến đoạn. Java gán cho mỗi tuyến đoạn một độ ưu tiên để xác định cách tuyến đoạn đó được xử lý như thế nào so với các tuyến đoạn khác. Độ ưu tiên của các tuyến đoạn là các số nguyên. Các độ ưu tiên của tuyến đoạn chỉ có tính chất tương đối so với các tuyến đoạn khác. Các tuyến đoạn có độ ưu tiên cao nhất sẽ được xử lý trước tiên nếu không có gì đặc biệt. Các tuyến đoạn có độ ưu tiên thấp hơn sẽ chỉ chạy khi các tuyến đoạn có độ ưu tiên cao hơn bị phong tỏa.

Với một tuyến đoạn cho trước ta có thể xác định độ ưu tiên lớn nhất và độ ưu tiên nhỏ nhất nhờ các hằng số *Thread.MAX_PRIORITY*, *Thread.MIN_PRIORITY*. Độ ưu tiên chuẩn cho một tuyến đoạn mặc định là *Thread.NORM_THREAD*. Độ ưu tiên của tuyến đoạn hiện thời có thể bị thay đổi bất kỳ khi nào nhờ phương thức setPriority(). Để nhận về độ ưu tiên của một tuyến đoạn ta dùng phương thức getPriority().

Một số phương thức tĩnh của lớp Thread điều khiển lịch trình của tuyến đoạn hiện thời

- public static void sleep(long ms) throws InterruptedException

  Phương thức này đưa tiến đoạn hiện hành vào trạng thái nghỉ tối thiểu là ms (mili giây).

- public static void yeild()

  Phương này giành lấy quyền thực thi của tuyến đoạn hiện hành cho một trong các tuyến đoạn khác.

7. Bế tắc-Deadlock

Một kiểu lỗi đặc biệt mà ta cần phải tránh có liên quan đến đa nhiệm là bế tắc (deadlock), bế tắc xảy ra khi hai tuyến đoạn có một sự phụ thuộc xoay vòng trên một cặp đối tượng đồng bộ. Ví dụ, giả sử một tuyến đoạn chiếm dụng đối tượng X và một tuyến đoạn chiếm dụng đối tượng Y. Nếu tuyến đoạn chiếm dụng X cố gắng gọi bất kỳ phương thức đồng bộ trên Y, thì nó sẽ bị phong tỏa. Nếu tuyến đoạn chiếm dụng Y gọi phương thức đồng bộ trên X, tuyến đoạn sẽ chờ vô hạn.

Ví dụ:

```
class A
{
 synchronized void phuongthuc1(B b)
```

```
 {
        String tenTD=Thread.currentThread().getName();
        System.out.println(tenTD+" dang goi phuong thuc A.phuongthuc1()");
        try{
            Thread.sleep(1000);
        }
        catch(Exception e)
        {
            System.out.println("A bi ngat");
        }
        System.out.println(tenTD+" Dang thu goi B.phuongthuc4()");
        b.phuongthuc4();
 }
 synchronized void phuongthuc2()
 {
        System.out.println("Ben tron phuong thuc A.phuongthuc2()");
 }

}
class B
{
 synchronized void phuongthuc3(A a)
 {
        String tenTD=Thread.currentThread().getName();
        System.out.println(tenTD+" dang goi phuong thuc B.phuongthuc3");
        try{

        }
        catch(Exception e)
        {
            System.out.println("B bi ngat");
        }
        System.out.println(tenTD+" Dang thu goi phuon thuc A.phuongthuc2()");
        a.phuongthuc2();
 }
 synchronized void phuongthuc4(){
        System.out.println("Ben trong phuong thuc B.phuongthuc4()");
 }
}
```

```
class Deadlock implements Runnable
{

        A a=new A();
        B b=new B();
 Deadlock()
 {
        Thread.currentThread().setName("MainThread");
        Thread t=new Thread(this,"RacingThread");
        t.start();
        a.phuongthuc1(b);
        System.out.println("Trong tuyen doan main");
 }
 public void run()
 {
        b.phuongthuc3(a);
        System.out.println("Trong tuyen doan main");
 }
 public static void main(String[] args)
 {
        new Deadlock();
 }
};
```

Kết quả thực hiện chương trình

```
C:\MyJava>java Deadlock
MainThread dang goi phuong thuc A.phuongthuc1()
RacingThread dang goi phuong thuc B.phuongthuc3
RacingThread Dang thu goi phuon thuc A.phuongthuc2()
MainThread Dang thu goi B.phuongthuc4()
```

Chương trình bị treo hay nói cách khác hai tuyến đoạn A và B rơi vào tình huống bế tắc, tuyến đoạn nó chờ tuyến đoạn kia giải phóng đối tượng mà đối tượng kia đang chiếm dụng và ngược lại, điều này dẫn đến chờ đợi nhau mà ta có thể gọi là tình trạng hoài vọng.

8. Điều khiển tuyến đoạn

8.1. Ngắt một tuyến đoạn Thread

Khi gọi phương thức Thread.sleep(int) thì phương thức này sẽ đặt tuyến đoạn vào trạng thái nghỉ trong một khoảng thời gian xác định nào đó. Tuy nhiên để kích hoạt một tuyến đoạn sớm hơn ta phải sử dụng ngắt tuyến đoạn. Ta có ngắt một tuyến đoạn bằng cách gọi phương thức *interrupt().* Tất nhiên, điều này cần một tuyến đoạn khác tham chiếu tới tuyến đoạn hiện thời.

```
public class SleepyHead extends Thread
```

```
{
// Run method is executed when thread first started
        public void run()
        {
                System.out.println ("I feel sleepy. Wake me in eight hours");
            try
            {
                // Sleep for eight hours
                Thread.sleep( 1000*60 );
                System.out.println ("That was a nice nap");
            }
            catch (InterruptedException ie)
            {
                System.err.println ("Just five more minutes....");
            }
 }
// Main method to create and start threads
            public static void main(String args[]) throws java.io.IOException
            {
                // Create a 'sleepy' thread
                Thread sleepy = new SleepyHead();
                // Start thread sleeping
                sleepy.start();
                // Prompt user and wait for input
                System.out.println ("Press enter to interrupt the thread");
                System.in.read();
                //              Interrupt the thread
                sleepy.interrupt();
            }
}
```

8.2 Kết thúc việc thực thi một tuyến đoạn

Đôi khi cần thiết phải kết thúc một tuyến đoạn trước khi tác vụ của nó hoàn thành. Ví dụ, nếu một client đang gửi các thông điệp tới một mail server trong một tuyến đoạn thứ hai, và người sử dụng muốn hủy bỏ thao tác, tuyến đoạn phải được dừng lại ngay tức thời. Một tuyến đoạn có thể gửi một yêu cầu ngừng việc thực thi tuyến đoạn tới một tuyến đoạn khác nhờ phương thức *Thread.stop()*. Điều này đòi hỏi tuyến đoạn điều khiển duy trì một tham chiếu tới tuyến đoạn mà nó muốn dừng.

Ví dụ dưới đây minh họa cách sử dụng phương thức stop:

```
public class StopMe extends Thread
{
// Run method is executed when thread first started
        public void run()
        {
                int count = 1;
                System.out.println ("I can count. Watch me go!");
                        for (;;)
                        {
                                // Print count and increment it
                                System.out.print (count++ + " ");
                                // Sleep for half a second
                                try { Thread.sleep(500); }
                                catch(InterruptedException ie) {}
                                }
        }
// Main method to create and start threads
        public static void main(String args[]) throws java.io.IOException
        {
// Create and start counting thread
                Thread counter = new StopMe();
                counter.start();
// Prompt user and wait for input
                System.out.println ("Press any enter to stop the thread ounting");
                System.in.read();
                // Interrupt the thread
                counter.stop();
        }
}
```

C:\MyJava>java StopMe

Press any enter to stop the thread ounting

I can count. Watch me go!

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23

Chương trình trên sẽ tiếp tục biến đếm cho tới khi ta nhấn một phím bất kỳ để dừng việc xử lý của tuyến đoạn.

8.3. Tạm dừng và phục hồi việc xử lý các tuyến đoạn

Trước Java 2, ta có thể được phép tạm dừng việc xử lý của một tuyến đoạn. Điều này có thể thực hiện nhờ phương thức Thread.suspend() và phục hồi hoạt động của tuyến đoạn nhờ Thread.resume(). Tuy nhiên trong các phiên bản Java 2 người ta khuyến cáo không nên sử dụng các phương thức này vì chúng có dẫn đến tình trạng hoài vọng (deadlock).

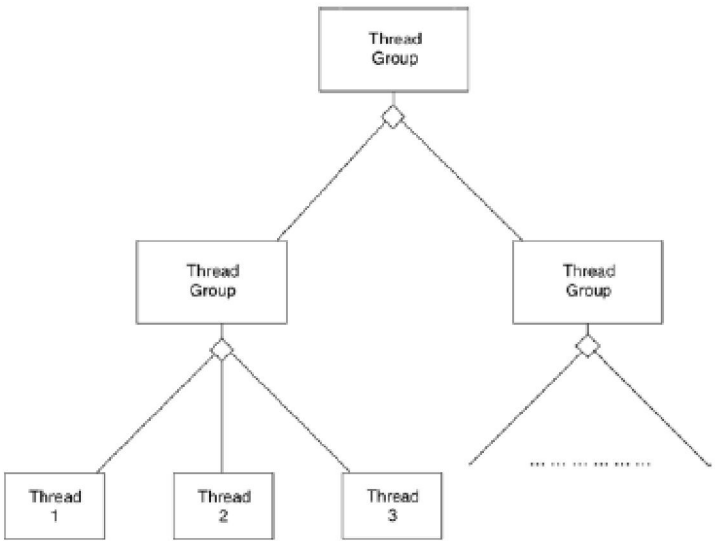
9. Nhóm các tuyến đoạn –ThreadGroup

Các tuyến đoạn có thể được tạo ra và được chạy riêng theo từng tuyến đoạn, song đôi khi làm việc với một nhóm các tuyến đoạn sẽ dễ dàng hơn so với làm việc với từng tuyến đoạn riêng rẽ tại từng thời điểm. Các thao tác ảnh hưởng tới các tuyến đoạn, như tạm dừng và phục hồi hoạt động của các tuyến đoạn, dừng hẳn hoặc ngắt các tuyến đoạn, có thể được thực hiện trên từng tuyến đoạn, nhưng điều này đòi hỏi các lập trình viên phải duy trì một danh sách các tuyến đoạn (bằng cách sử dụng các cấu trúc dữ liệu như vector hoặc mảng). Khi một thao tác được thực hiện, mỗi tuyến đoạn trong danh sách phải được duyệt và được xử lý riêng. Điều này tạo ra một khối lượng công việc khá lớn cho người lập trình vì cần phải viết nhiều đoạn mã phức tạp. Một giải pháp thay thế là nhóm các tuyến đoạn với nhau và áp dụng một thao tác trên nhóm mà không phải thực hiện thao tác trên từng tuyến đoạn riêng lẻ.

Java API hỗ trợ khả năng làm việc với các nhóm tuyến đoạn nhờ lớp ThreadGroup. Mục đích của lớp ThreadGroup là biểu diễn một tập hợp các tuyến đoạn, và cung cấp các phương thức tác động nhanh trên từng tuyến đoạn riêng trong nhóm. Nó còn cung cấp các cách thức để tập hợp các thông tin về các tuyến đoạn có liên quan nói chung. Nếu cần truy xuất tới từng tuyến đoạn riêng lẻ, ta có thể truy xuất thông qua ThreadGroup.

Khi JVM khởi động ứng dụng lần đầu tiên, tuyến đoạn chính sẽ chạy phương thức main(). Sau đó, ứng dụng tạo ra các nhóm tuyến đoạn một cách tự do, hoặc tạo ra các tuyến đoạn riêng không gắn với một nhóm nào cả. Một nhóm có thể bao gồm nhiều tuyến đoạn và ta có thể bổ sung thêm tuyến đoạn vào nhóm trong quá trình xử lý tuyến đoạn khi cần. Tuy nhiên không có cách nào để gỡ bỏ tuyến đoạn ra khỏi một nhóm.

Một nhóm các tuyến đoạn có thể chứa các nhóm tuyến đoạn khác được gọi là các nhóm tuyến con các tuyến đoạn. Các thao tác như dừng hoặc tạm dừng có thể được thực hiện trên các nhóm con hoặc nhóm cha. Khi một thao tác được áp dụng cho nhóm cha, thao tác sẽ được lan truyền tới nhóm con đó. Điều này có nghĩa là một thao tác có thể có tác động đến nhiều tuyến đoạn. Điều này khiến cho việc thiết kế ứng dụng trở nên đơn giản hơn.



Hình 4.4

9.1. Tạo một nhóm Thread

Các constructor

Lớp ThreadGroup cung cấp một số constructor sau:

- public ThreadGroup(String name) throws java.lang.SecurityException

Constructor này tạo ra một nhóm tuyến đoạn mới, nhóm này có tên được xác định bởi một xâu ký tự truyền vào như một tham số.

- ThreadGroup(ThreadGroup parentGroup,String name) throws java.lang.SecurityException

Tạo ra một nhóm tuyến đoạn mới được định danh bởi tên name. Nó cho phép một nhóm được lưu trữ như là một nhóm con.

Sử dụng một ThreadGroup

Mỗi khi được tạo ra, một nhóm tuyến đoạn có thể được sử dụng như một tuyến đoạn bình thường. Ta có thể tạm dừng, phục hồi, ngắt hoặc dừng nhóm tuyến đoạn bằng cách gọi phương thức thích hợp. Để sử dụng nhóm tuyến đoạn hiệu quả thì tuyến đoạn phải khác rỗng .

Các tuyến đoạn không được gắn với một nhóm cụ thể tại thời điểm tạo ra chúng. Vì thế, không thể gán một tuyến đoạn cho một nhóm sau đó, hoặc chuyển một tuyến đoạn từ nhóm này sang nhóm khác. Có ba constructor để thực hiện điều này

- Thread(ThreadGroup group, Runnable runnbale)
- Thread(ThreadGroup group, Runnable name)
- Thread(ThreadGroup group, Runnable runnbale, String name)

Mỗi khi tạo ra một nhóm các tuyến đoạn ta có thể tác động các phương thức trên nhóm.
Các phương thức

- int activeCount(): trả về số tuyến đoạn trong nhóm, và các nhóm con.
- int activeGroupCount(): trả về số nhóm con các tuyến đoạn
- boolean allowThreadSuspension(): chỉ ra tuyến đoạn bị tạm ngừng hay không.
- void checkAccess():
- void destroy(): hủy bỏ nhóm tuyến đoạn
- int enumerate(Thread[] threadList): đưa các tuyến đoạn trong nhóm vào một mảng các tuyến đoạn.
- int enumerate(Thread[] threadList, boolean subgroupFlag): đưa các tuyến đoạn trong nhóm vào một mảng các tuyến đoạn. Phương thức này đưa các tuyến đoạn trong nhóm bao gồm cả các tuyến đoạn nhóm con nếu biến logic boolean subgroupFlag được thiết lập là true.
- int enumerate(ThreadGroup[] groupList): đặt tất cả các nhóm con vào mảng

9.2. Minh họa về lớp ThreadGroup

```
public class GroupDemo implements Runnable{
 public static void main(String args[]) throws Exception
 {
        // Create a thread group
        ThreadGroup parent = new ThreadGroup("parent");
        // Create a group that is a child of another thread group
        ThreadGroup subgroup = new ThreadGroup(parent, "subgroup");
```

```
        // Create some threads in the parent, and subgroup class
        Thread t1 = new Thread ( parent, new GroupDemo() );
        t1.start();
        Thread t2 = new Thread ( parent, new GroupDemo() );
        t2.start();
        Thread t3 = new Thread ( subgroup, new GroupDemo() );
        t3.start();
        // Dump the contents of the group to System.out
        parent.list();
        // Wait for user, then terminate
        System.out.println ("Press enter to continue");
        System.in.read();
        System.exit(0);
    }

        public void run(){
        // Do nothing
        for(;;)
        {
        Thread.yield();
        }
     }
   }
```

Kết quả thực hiện chương trình

```
C:\MyJava>java GroupDemo
java.lang.ThreadGroup[name=parent,maxpri=10]
   Thread[Thread-0,5,parent]
   Thread[Thread-1,5,parent]
   java.lang.ThreadGroup[name=subgroup,maxpri=10]
      Thread[Thread-2,5,subgroup]
Press enter to continue
```

10. Một ví dụ minh họa việc sử dụng tuyến đoạn

Ví dụ

Chương trình vẽ đồng hồ số sử dụng tuyến đoạn và applet

```
    import java.awt.*;
    import java.util.Date;
    import java.applet.*;
    public class   Clock extends Applet implements Runnable
    {
     Font f = new Font("Times New Roman",Font.BOLD,24);
     Date d = new Date();
```

```
Thread t;
public void init()
{
        resize(400,400);
}
public void start()
{
        if(t==null)
        {
                t= new Thread(this);
                t.start();
        }
}
public void stop()
{
         if(t==null)
        {
                 t.stop();
                 t=null;
        }
}
public void run()
{
        while(true)
        {
                d= new Date();
                repaint();
                try{
                        Thread.sleep(1000);
                }
                catch(InterruptedException e)
                {
                        return;
                }
        }
}
public  void paint(Graphics g)
{
        g.setFont(f);
```

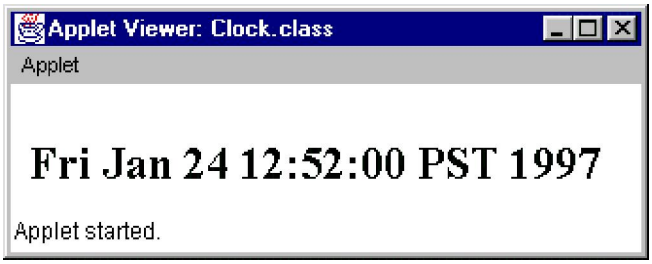
```
        g.drawString(d.toString(),10,50);
    }
}
```

```
<HTML>
<HEAD>
<TITLE> Clock </TITLE>
</HEAD>
<BODY>
<APPLET CODE ="Clock.class" WIDTH =200 HEIGHT = 100>
</APPLET>
</BODY>
</HTML>
```

Thực hiện chương trình

appletviewer Clock.html

Kết quả thực hiện



Hình 4.5

11. Kết luận

Những hiểu biết về lập trình đa tuyến đoạn có tầm quan trọng đối với các ứng dụng và các applet, đặc biệt là đối với môi trường mạng. Các mạng máy tính thường rất chậm và có độ tin cậy không cao, vì vậy chương trình mạng nên chạy trong một tuyến đoạn riêng biệt tách biệt với giao diện người dùng. Hơn thế nữa, phần mềm mạng tương tác với nhiều client hoặc server, ngoại trừ các thao tác đặc biệt nhanh (như nhận và gửi một gói tin). Chương trình cần có nhiều tuyến đoạn để các tương tác có thể xảy ra đồng thời. Trong các chương sau chúng ta sẽ xem xét cách ứng dụng tuyến đoạn trong việc xây dựng các chương trình mạng có xử lý tương tranh.

Chương 5

Lập trình mạng với các lớp InetAddress, URL và URLConnection

1. Lớp InetAddress

Các thiết bị được kết nối với mạng LAN có địa chỉ vật lý duy nhất. Điều này giúp cho các máy khác trên mạng trong việc truyền các gói tin đến đúng vị trí. Tuy nhiên, địa chỉ này chỉ có ích trong mạng LAN. Một máy không thể xác định được vị trí trên Internet bằng cách sử dụng các địa chỉ vật lý, vì các địa chỉ vật lý không chỉ ra vị trí của máy. Hơn nữa, các máy thường di chuyển từ vị trí này sang vị trí khác, trong trường hợp của máy xách tay hoặc máy palm chẳng hạn.

Những người lập trình mạng không cần phải quan tâm đến từng chi tiết dữ liệu được định tuyến như thế nào trong một mạng LAN. Hơn nữa, Java không cung cấp khả năng truy xuất tới các giao thức tầng liên kết dữ liệu mức thấp được sử dụng bởi LAN. Việc hỗ trợ như vậy là rất khó khăn. Vì mỗi kiểu giao thức sử dụng một kiểu địa chỉ khác nhau và có các đặc trưng khác nhau, chúng ta cần phải các chương trình khác nhau cho mỗi kiểu giao thức mạng khác nhau. Thay vào đó, Java hỗ trợ giao thức TCP/IP, giao thức này có nhiệu vụ liên kết các mạng với nhau.

Các thiết bị có một kết nối Internet trực tiếp được cung cấp một định danh duy nhất được gọi là địa chỉ IP. Các địa chỉ IP có thể là tĩnh hoặc động. Các địa chỉ IP được cấp phát động thường được sử dụng khi nhiều thiết bị cần truy cập Internet trong khoảng thời gian nhất định. Một địa chỉ IP chỉ có thể gắn với một máy, nó không thể dùng chung. Địa chỉ này được sử dụng bởi giao thức IP để định tuyến các datagram tới đúng vị trí. Không có địa chỉ, ta không thể liên lạc được với máy đó; vì thế tất cả các máy tính đều phải có một địa chỉ IP duy nhất.

Lớp java.net.InetAddress biểu diễn một địa chỉ Internet. Nó bao gồm hai trường thông tin: hostName (một đối tượng kiểu String) và address (một số kiểu int). Các trường này không phải là trường public, vì thế ta không thể truy xất chúng trực tiếp. Lớp này được sử dụng bởi hầu hết các lớp mạng, bao gồm Socket, ServerSocket, URL, DatagramSocket, DatagramPacket,…

1.1. Tạo các đối tượng InetAddress

Lớp InetAddress được sử dụng để biểu diễn các địa chỉ IP trong một ứng dụng mạng sử dụng Java. Không giống với các lớp khác, không có các constructor cho lớp InetAddress. Tuy nhiên, lớp InetAddress có ba phương thức tĩnh trả về các đối tượng InetAddress

Các phương thức trong lớp InetAddress

- public static InetAddress InetAddress.getByName(String hostname)
- public static InetAddress[] InetAddress.getAllByName(String hostname)
- public static InetAddress InetAddress.getLocalHost()

Tất cả các phương thức này đều thực hiện kết nối tới server DNS cục bộ để biết được các thông tin trong đối tượng InetAddress

Ta xét phương thức đầu tiên. Phương thức này nhận tên của hostname làm tham số và trả về đối tượng kiểu InetAddress

Ví dụ:

```
try{
InetAddress dc =InetAddress.getByName("www.microsoft.com");
System.out.println(dc);
```

```
}
catch(UnknownHostException e)
{
 System.err.println(e);
}
```

Ví dụ 1:Viết chương trình nhận hostname từ đối dòng lệnh và in ra địa chỉ IP tương ứng với hostname đó.

```
import java.net.*;
public class  TimDCIP
{
 public static void main(String[] args)
 {
        try{
                if(args.length!=1)
                {
                        System.out.println("Cach su dung: java TimDCIP <Hostname>");
                }
                InetAddress host = InetAddress.getByName(args[0]);
                String hostName = host.getHostName();
                System.out.println("Host name:"+hostName);
                System.out.println("Dia chi IP:"+host.getHostAddress());
        }
        catch(UnknownHostException e)
        {
                System.out.println("Khong tim thay dia chi");
                return;
        }

 }
}
```

1.2. Nhận các trường thông tin của một đối tượng InetAddress

Chỉ có các lớp trong gói java.net có quyền truy xuất tới các trường của lớp InetAddress. Các lớp trong gói này có thể đọc các trường của một đối tượng InetAddress bằng cách gọi phương thức getHostname và getAddress().

- public String getHostName(): Phương thức này trả về một xâu biểu diễn hostname của một đối tượng InetAddress. Nếu máy không có hostname, thì nó sẽ trả về địa chỉ IP của máy này dưới dạng một xâu ký tự.
- public byte[] getAddress() : Nếu bạn muốn biết địa chỉ IP của một máy, phương thức getAddress() trả về một địa chỉ IP dưới dạng một mảng các byte.

- Một số địa chỉ IP và một số mô hình địa chỉ có các ý nghĩa đặc biệt. Ví dụ, 127.0.0.1 là địa chỉ loopback. Các địa chỉ IPv4 trong khoảng 224.0.0.0 tới 239.255.255.255 là các địa chỉ multicast.

Java 1.5 thêm vào hai phương thức cho lớp InetAddress cho phép các ứng dụng kiểm tra liệu một nút cụ thể có đến được hay không với nút xuất phát là nút hiện hành; nghĩa là kiểm tra xem một liên kết mạng đã được thiết lập hay chưa. Các liên kết có thể bị phong tỏa vì nhiều nguyên nhân như firewall, các server ủy quyền, các router hoạt động sai chức năng, dây cáp bị đứt, hoặc host ở xa không bật.

- public boolean isReachable(int timeout) throws IOException
- public boolean isReachable(NetworkInterface interface, int ttl, int timeout) throws IOException

Các phương thức này cố gắng kết nối trên cổng echo trên host ở xa để tìm xem nó có thể đến được hay không. Nếu host đáp ứng trong khoảng thời gian timeout mili giây, các phương thức này trả về giá trị true nếu đến được, ngược lại nó trả về giá trị false.

1.3. Một số chương trình minh họa

Ví dụ 2 :Viết chương trình nhập một hostName từ đối dòng lệnh và in ra dòng thông báo cho biết địa chỉ IP tương ứng với địa chỉ IP đó thuộc lớp nào.

```
import java.net.*;
public class  PhanLoaiDCIP
{
 public static void main(String[] args)
 {
        try{
                if(args.length!=1)
                {
                        System.out.println("Cach su dung: java TimDCIP <Hostname>");
                }
                InetAddress host = InetAddress.getByName(args[0]);
                String hostName = host.getHostName();
                System.out.println("Host name:"+hostName);
                System.out.println("Dia chi IP:"+host.getHostAddress());
                byte[] b=host.getAddress();
                int i=b[0]>=0?b[0]:256+b[0];

                if((i>=1)&(i<=126)) System.out.println(host+" thuoc dia chi lop A");
                if((i<=191)&(i>=128)) System.out.println(host+" thuoc dia chi lop B");
                if((i<=223)&(i>=192)) System.out.println(host+" thuoc dia chi lop C");

        }
        catch(UnknownHostException e)
```

```
        {
                System.out.println("Khong tim thay dia chi");
                return;
        }

    }
}
```

Trong chương trình này ta chỉ cần lưu ý dòng lệnh int i=b[0]>=0?b[0]:256+b[0]. Vì ta biết trong Java kiểu byte là kiểu số nguyên có dấu có khoảng giá trị là từ -128 đến 127. Do vậy, dòng lệnh int i=b[0]>=0?b[0]:256+b[0] làm nhiệm vụ chuyển đổi số nguyên có dấu ở dạng bù 2 về dạng số nguyên không dấu

2. Lớp URL

Cách đơn giản nhất để một chương trình Java định vị và tìm kiếm dữ liệu là sử dụng một đối tượng URL. Bạn không cần phải lo lắng tới các chi tiết bên trong của giao thức đang được sử dụng, khuôn dạng dữ liệu được nhận, hay làm thế nào để truyền tin với server; bạn chỉ cần cho biết URL, Java sẽ lấy dữ liệu về cho bạn.

Lớp java.net.URL là một khái niệm về bộ định vị tài nguyên thống nhất. Nếu lưu trữ URL dưới dạng một đối tượng String sẽ không có lợi so với việc tổ chức URL như một đối tượng với các trường : giao thức (protocol), hosname, cổng (port), đường dẫn (path), tên tập tin (filename), mục tài liệu (document section), mỗi trường có thể được thiết lập một cách độc lập.

2.1. Tạo các URL

Có bốn constructor, khác nhau về thông tin mà nó cần. Constructor mà bạn sử dụng phụ thuộc vào thông tin mà bạn có, và khuôn dạng trong URL đó. Tất cả các constructor này sẽ đưa ra ngoại lệ MalformedURLException (URL không đúng khuôn dạng) nếu ta tạo ra một URL cho một giao thức mà nó không được hỗ trợ.URL cung cấp các hàm cấu tử sau:

- *public URL(String url) throws MalformedURLException*

Đây là constructor đơn giản nhất; tham số của nó chỉ là một URL ở dạng xâu.

Ví dụ

```
try{
        URL u = new URL("http://www.sun.com/index.html");
}
catch(MalformedURLException e)
{
   System.err.println(e);
}
```

- *public URL(String protocol, String host, String file) throws MalformedURLException*

Constructor này xây dựng một URL từ các xâu phân biệt xác định giao thức, hostname, và tệp tin. Port được thiết lập bằng -1 vì vậy cổng mặc định cho giao thức sẽ được sử dụng.

Ví dụ

```
try{
URL u = new URL("http","/www.sun.com","index.html");
}
catch(MalformedURLException e){
   System.err.println(e);
}
```

- public URL(String protocol, String host, int port, String file) throws MalformedURLException

Trong một số ít trường hợp khi cổng mặc định không còn đúng, constructor này cho phép bạn xác định cổng một cách rõ ràng, là một số kiểu int. Các tham số khác giống như trên.

Ví dụ

```
try{
URL u = new URL("http","/www.sun.com",80,"index.html");
}
catch(MalformedURLException e){
   System.err.println(e);
}
```

- public URL(URL u, String s) throws MalformedURLException

Hàm cấu tử này xây dựng một URL tuyệt đối từ URL tương đối; có thể là đây là constructor bạn sẽ sử dụng thường xuyên.

Ví dụ

```
URL u1,u2;

try{
  URL u1= new URL("http://www.macfaq.com/index.html");
  URL u2 = new URL(u1,"vendor.html");
}
catch(MalformedURLException e)
{
   System.err.println(e);
}
```

Tên file được loại khỏi đường dẫn của u1, và tên file mới vendor.html được gán vào để tạo lên u2. Constructor này đặc biệt hữu ích khi bạn muốn duyệt qua một danh sách các file mà tất cả cùng nằm trong một thư mục.

2.2. Phân tích một URL thành các thành phần

Có sáu trường thông tin trong lớp URL: giao thức, port, file, mục tham chiếu tài liệu.

- public String getProtocol()

  Phương thức getProtocol() trả về một xâu ký tự biểu diễn phần giao thức của URL

- public String getHost()

Phương thức getHost() trả về một xâu ký tự biểu diễn phần hostname của URL.

- public int getPort()

Phương thức getPort() trả về một số nguyên kiểu int biểu diễn số hiệu cổng có trong URL.

- public int getDefaultPort()

Phương thức getDefautlPort() trả về số hiệu cổng mặc định cho giao thức của URL

- public String getFile()

Phương thức getFile() trả về một xâu ký tự chứa phần đường dẫn của một URL; Java không phân chia một URL thành các phần đường dẫn và phần tệp tin riêng biệt.

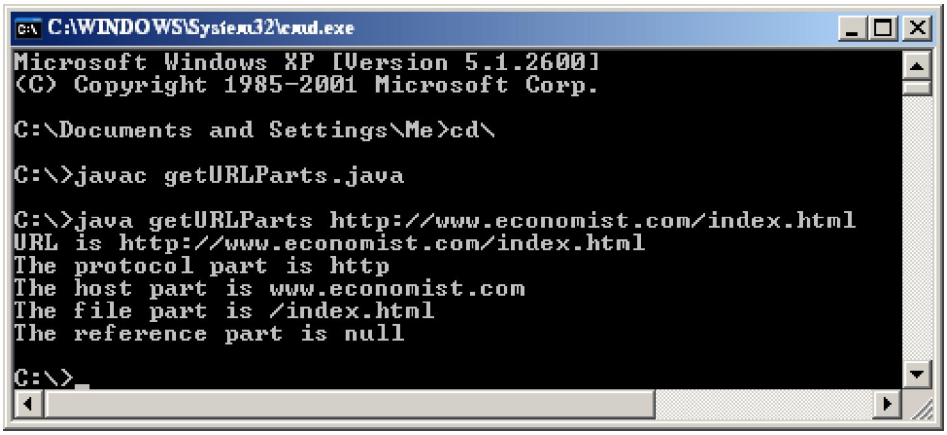
- public String getRef()

Phương thức này trả về phần định danh đoạn của URL

Ví dụ: Viết chương trình nhập vào một URL từ đối dòng lệnh và hiển thị từng thành phần tạo nên URL lên màn hình.

```
//Chương trình lấy thông tin của URL với các thông tin nhập từ bàn phím
    import java.net.*;

    class getURLParts{
        public static void main(String[] args)
        {
            try
            {
                URL u = new URL(args[0]);
                System.out.println("URL is "+u);
                System.out.println("The protocol part is "+u.getProtocol());
                System.out.println("The host part is "+u.getHost());
                System.out.println("The file part is "+u.getFile());
                System.out.println("The reference part is "+u.getRef());
            }
            catch(MalformedURLException e)
            {
                System.err.println(e);
            }
        }
    }
```

Kết quả thực hiện chương trình như sau:



Hình 5.1

2.3. Tìm kiếm dữ liệu từ một URL

Nếu chỉ có URL thuần túy thì không có gì thú vị. Điều thú vị là dữ liệu nằm trong các tệp tin mà nó trỏ tới. Lớp java.net.URL có ba phương thức để tìm kiếm dữ liệu từ một URL

- *public final InputStream openStream() throws java.io.IOException*

Phương thức này kết nối tới một tài nguyên được tham chiếu bởi một URL, thực hiện cơ chế bắt tay cần thiết giữa client và server, và trả về một luồng nhập InputStream. Ta sử dụng luồng này để đọc dữ liệu. Dữ liệu nhận từ luồng này là dữ liệu thô của một tệp tin mà URL tham chiếu (mã ASCII nếu đọc một tệp văn bản, mã HTML nếu đọc một tài liệu HTML, một ảnh nhị phân nếu ta đọc một file ảnh). Nó không có các thông tin header và các thông tin có liên quan đến giao thức

- *public URLConnection openConnection() throws java.io.IOException*

Phương thức opeConnection() mở một socket tới một URL xác định và trả về một đối tượng URL. Một đối tượng URLConnection biểu diễn một liên kết mở tới một tài nguyên mạng. Nếu lời gọi phương thức thất bại nó đưa ra ngoại lệ IOException.

- *public final Object getConetent() throws java.io.IOException*

Phương thức này cung cấp cách thứ ba để tải dữ liệu được tham chiếu bởi một URL. Phương thức getContent() tìm kiếm dữ liệu được tham chiếu bởi một URL và chuyển nó thành một kiểu đối tượng nào đó. Nếu đối tượng tham chiếu tới một kiểu đối tượng văn bản nào đó như tệp tin ASCII hoặc tệp HTML, đối tượng được trả về thông thường sẽ là một kiểu luồng nhập InputStream nào đó. Nếu URL tham chiếu tới một đối tượng ảnh như ảnh GIF hoặc JPEG thì phương thức getContent() trả về đối tượng java.awt.ImageProducer

Ví dụ:Viết chương trình nhập một URL từ bàn phím, kết nối với Internet và hiển thị mã nguồn của trang Web đó lên màn hình.

```
import java.net.*;
import java.io.*;
public class  ViewSource
{
 public static void main(String[] args)
 {
        URL u;
        String thisLine;
        if(args.length>0){
```

```
                try{
                        u =new URL(args[0]);
                        try{
                                DataInputStream dis = new
DataInputStream(u.openStream());
                                while((thisLine=dis.readLine())!=null)
System.out.println(thisLine);

                        }
                        catch(IOException e)
                        {
                                System.err.println(e);
                        }
                }
                catch(MalformedURLException e){
                        System.err.println(e);
                }
        }

    }
}
```

Kết quả thực hiện chương trình

2.4. Các phương thức tiện ích

Lớp URL cung cấp hai phương thức tiện ích để thực hiện các thao tác trên URL. Phương thức sameFile() xác định xem hai URL có cùng trỏ tới một tài liệu hay không. Phương thức toExternalForm() chuyển đổi một đối tượng URL thành một xâu ký tự được sử dụng trong một liên kết HTML hoặc hộp thoại của trình duyệt.

- public boolean sameFile(URL other)

3. Lớp URLConnection

URLConnection là một lớp trừu tượng biểu diễn một liên kết tích cực tới một tài nguyên được xác định bởi một URL.

Lớp URLConnection có hai mục đích khác nhau nhưng liên quan với nhau.

- Thứ nhất, nó cung cấp nhiều khả năng điều khiển hơn thông qua việc tương tác với một server chứ không phải lớp URL. Với URLConnection ta có thể kiểm tra các headerMIME được gửi bởi một Http Server phản ứng tương ứng. Ta cũng có thể sử dụng lớp URLConnection để tải về các tệp nhị phân. Ngoài ra URLConnection cho phép bạn gửi dữ liệu trở lại Web server bằng lệnh POST. Chúng ta sẽ xem tất cả các kỹ thuật đó trong chương này.
- Thứ hai, URLConnection là một phần của cơ chế quản trị giao thức, cơ chế này còn bao gồm cả lớp URLStreamHandler. Ý tưởng đằng sau các trình quản trị giao thức rất đơn giản: chúng cho phép bạn phân tách các chi tiết xử lý một giao thức với việc xử lý các kiểu dữ liệu cụ thể, cung cấp các giao diện người dùng, và thực

hiện các công việc khác mà một trình duyệt thường làm. Lớp cơ sở URLConnection là một lớp trừu tượng; để cài đặt một giao thức cụ thể bạn cần phải viết một lớp con. Các lớp con này có thể được tải bởi các ứng dụng của riêng bạn hay bởi các trình duyệt HotJava; trong tương lai, các ứng dụng Java có thể tải về các trình quản trị giao thức khi cần.

Mở các URLConnection

Một chương trình sử dụng lớp URLConnection trực tiếp theo một dãy các bước cơ bản sau:

- Xây dựng một đối tượng URL.
- Gọi phương thức openConnection() của đối tượng URL để tìm kiếm một đối tượng URLConnection cho URL đó.
- Cấu hình đối tượng URL.
- Đọc các trường header.
- Nhận một luồng nhập và đọc dữ liệu.
- Nhận một luồng xuất và ghi dữ liệu.
- Đóng liên kết.

Tuy nhiên, không phải lúc nào ta cũng phải thực hiện tất cả các bước này.

Ví dụ:Mở một URLConnection tới http:// www.microsoft.com

```
import java.net.*;
import java.io.*;
public class getURLConnection
{
 public static void main(String[] args)
 {
        URL u;
        URLConnection uc;
        try
          {
                u= new URL("http://www.microsoft.com");
                try
                        {
                         uc=u.openConnection();
                        }
                catch(IOException e)
                        {
                        System.err.println(e);
                        }
          }
        catch(MalformedURLException e)
                {
```

```
                System.err.println(e);
            }
        }
    }
```

Mặc dù lớp URLConnection là một lớp trừu tượng nhưng nó có một phương thức được cài đặt. Phương thức đó là connect(); phương thức này tạo một liên kết tới một server; vì vậy nó phụ thuộc vào kiểu dịch vụ mà ta cài đặt (HTTP, FTP,..). Tất nhiên, ta có thể cảm thấy tiện lợi hay cần thiết phải nạp chồng các phương thức này trong lớp.

Rất nhiều các phương thức và các trường trong lớp URLConnection có là phương thức *protected*. Mặt khác ta chỉ có thể truy cập tới chúng thông qua các thể hiện của lớp URLConnection hoặc các lớp con của nó. Rất ít khi chúng ta khởi tạo và truy cập trực tiếp các đối tượng; thay vào đó, môi trường thời gian chạy sẽ tạo ra các đối tượng khi cần tùy thuộc vào giao thức sử dụng. Sau đó lớp sẽ được khởi tạo bằng cách sử dụng các phương thức *forName()* và *newInstance()* của lớp java.lang.Class.

- *public abstract void connect() throws IOException*

  Phương thức connect() là một phương thức trừu tượng mở một liên kết tới một server. Tên của server được lấy ra từ một URL được lưu trữ như là một trường trong URLConnection, và được thiết lập bởi constructor của lớp. Các lớp con của lớp URLConnection nạp chồng các phương thức này để quản lý một kiểu liên kết cụ thể. Ví dụ, một phương thức connect() của lớp FileURLConnection chuyển đổi URL thành một filename trong thư mục tương ứng, tạo ra thông tin MIME cho file, và mở một luồng FileInputStream tới file. Phương thức connect() của lớp HttpURLConnection tạo ra một đối tượng HttpClient để kết nối với server. Phương thức openConnection() của đối tượng URL gọi phương thức connect() tương ứng, và trả về liên kết đã được mở. Vì vậy ta hiếm khi cần phải gọi phương thức connect() một cách trực tiếp.

Đọc dữ liệu từ một server

Dưới đây là các bước tối thiểu cần để tìm kiếm dữ liệu từ một URL bằng cách sử dụng đối tượng URLConnection:

- Bước 1: Xây dựng một đối tượng URL.
- Bước 2: Gọi phương thức openConnection() của lớp URL để tìm kiếm một đối tượng URL Connection cho đối tượng URL đó.
- Bước 3: Gọi phương thức getInputStream().
- Bước 4: Đọc từ luồng nhập bằng cách sử dụng API.

- public Object getContent() throws IOException

Phương thức về mặt ảo giác giống như phương thức getContent() của lớp URL. Thực tế, phương thức URL.getContent() chỉ việc gọi phương thức getContent() tải về đối tượng được chọn bởi URL của URLConnection này. Để phương thức getContent() hoạt động, môi trường cần nhận dạng và hiểu kiểu nội dung. Hiện nay chỉ có một số kiểu nội dung được hiểu là text/plain, image/gif, và image/jpeg. Bạn có thể cài đặt thêm các kiểu trình quản lý nội dung khác có thể hiểu các kiểu nội dung khác. Phương thức getContent() chỉ làm việc với các giao thức như HTTP mà có một sự hiểu biết rõ ràng về các kiểu nội dung MIME. Nếu kiểu nội dung không được biết trước, hoặc giao thức không hiểu các kiểu nội dung, thì ngoại lệ UnknownServicException được đưa ra.

- public InputStream getInputStream()

  Phương thức getContent() chỉ làm việc khi Java có một trình quản lý nội dung cho kiểu nội dung. Nếu không phải trường hợp này, bạn có lẽ sẽ cần một luồng InputStream chung, luồng này cho phép bạn tự đọc và phân tích dữ liệu. Để làm

như vậy, cần gọi phương thức getInputStream(). Phương thức này cũng hữu ích khi trình quản lý nội dung có sẵn không thực hiện chính xác cái mà bạn muốn.

Ví dụ: Tải về một trang web thông qua một URL.

```
import java.net.*;
import java.io.*;
public class  viewsource
{
        public static void main(String[] args)
        {
                String thisLine;
                URL u;
                URLConnection uc;
                if(args.length>0)
                {
                        try{
                                u =new URL(args[0]);
                                try{
                                        uc=u.openConnection();
                                        DataInputStream theHtml = new
DataInputStream(uc.getInputStream());
                                        try{

 while((thisLine=theHtml.readLine())!=null)
                                                {
                                                        System.out.println(thisLine);
                                                }
                                        }
                                        catch(Exception e)
                                        {
                                                System.err.println(e);
                                        }
                                }
                                catch(Exception e)
                                {
                                        System.err.println(e);
                                }
                        }
                        catch(MalformedURLException e)
                        {
                        System.err.println(args[0]+" is not a parseable URL");
```

```
                    System.err.println(e);
                }

            }
        }
    }
```

Phương thức openStream() của lớp URL trả về đối tượng InputStream từ đối tượng URLConnection.

- public OutputStream getOutputStream()

  Đôi khi bạn cần phải ghi dữ liệu vào một URLConnection-chẳng hạn khi bạn muốn gửi dữ liệu tới một web server sử dụng lệnh POST. Phương thức getOutputStream() trả về một luồng OutputStream trên đó bạn có thể ghi dữ liệu để truyền tới một server. Vì một URLConnection không cho phép xuất kết quả ra ở chế độ mặc định, bạn phải gọi phương thức setDoOutput() trước khi yêu cầu một luồng xuất. Mỗi khi bạn có một luồng OutputStream thông thường bạn phải gắn nó với luồng DataOutputStream hoặc một lớp con khác của lớp OutputStream mà đưa ra nhiều đặc trưng hơn. Ví dụ:

```
try{
  URL u = new URL("http:// www.somehost.com/cgi-bin/acgi");
  URLConnection uc = u.openConnection();
  uc.setDoOutput(true);
   DataOutputStream dos = new DataOutputStream(uc.getOutputStream());
 dos.writeByte("Herre is some data");
}
catch(Exception e)
{
        System.err.println(e);
}
```

Sự khác biệt giữa URL và URLConnection là:

- URLConnection cho phép truy xuất tới header HTTP.
- URLConnection có thể cấu hình các tham số yêu cầu được gửi cho server.
- URLConnection có thể ghi dữ liệu lên server cũng như đọc dữ liệu từ server.

3.1. Phân tích Header

HTTP Server cung cấp một số lượng thông tin đáng kể trong các header MIME trước mỗi đáp ứng. Thông tin trong các header MIME có thể bao gồm cơ chế mã hóa nội dung được sử dụng, ngày và giờ, chiều dài của nội dung được trả về bằng byte, ngày hết hạn của nội dung, ngày mà nội dung được sửa đổi lần cuối. Tuy nhiên, thông tin được gửi phụ thuộc vào server; một server nào đó gửi tất cả các thông tin này cho mỗi yêu cầu, các server khác gửi các thông tin nào đó, và một số ít server không gửi thông tin nào. Các phương thức của mục này cho phép ta truy vấn một URLConnection để tìm ra thông tin MIME nào mà server đã cung cấp.

Ngoài HTTP, rất ít giao thức sử dụng các header MIME. Khi viết lớp con của lớp URLConnection, thông thường cần phải nạp chồng các phương thức này sao cho chúng trả về các giá trị có ý nghĩa. Phần thông tin quan trọng nhất bạn có thể thiếu là kiểu nội dung MIME. URLConnection cung cấp một số phương thức tiện ích nào đó mà trợ giúp bạn đoán nhận ra kiểu nội dung, dựa trên tên file của nó hoặc một số byte đầu tiên của chính dữ liệu.

- public String getContentType()

  Phương thức trả về kiểu nội dung MIME của dữ liệu. Nó phụ thuộc vào web server gửi một header MIME tương ứng, bao gồm một kiểu nội dung xác thực. Nó không đưa ra ngoại lệ và trả về giá trị null nếu kiểu nội dung không có. text/htm sẽ là kiểu nội dung mà bạn thường xuyên gặp nhất khi kết nối với web server. Các kiểu nội dung phổ biến khác bao gồm: text/plain, image/gif, image/jpeg.

- public int getContentLength()

  Phương thức này cho ta biết nội dung có kích thước bao nhiêu byte. Rất nhiều server chỉ gửi các header độ dài nội dung khi chúng truyền một file nhị phân, chứ không phải khi truyền một file văn bản. Nếu không có chiều dài nội dung, phương thức getContentLength() trả về -1. Phương thức này không đưa ra ngoại lệ. Nó được sử dụng khi ta cần biết cần đọc bao nhiêu byte, hoặc khi ta cần tạo ra một buffer đủ lớn để lưu trữ dữ liệu.

- public String getContentEncoding()

  Phương thức này trả về String cho ta biết cách thức mã hóa. Nếu nội dung được gửi không được mã hóa (như trong trường hợp của HTTP server), phương thức này trả về giá trị null. Nó không đưa ra ngoại lệ.

- public long getDate()

  Phương thức getDate() trả về một số nguyên kiểu long cho bạn biết tài liệu đã được gửi khi nào. Ta có thể chuyển đổi nó sang một đối tượng kiểu java.util.Date. Ví dụ:

  Date docSent = new Date(uc.getDate());

  Đây là thời gian tài liệu được gửi trên server. Nếu header MIME không có một header Date.

- public long getExpiration()

- public long getLastModified()

  Phương thức date, getLastModified(), trả về ngày mà tài liệu được sửa đổi lần cuối

Ví dụ: Đọc các URL từ dòng lệnh, và sử dụng 6 phương thức để in ra kiểu nội dung, chiều dài nội dung, mã hóa nội dung, ngày sửa đổi cuối cùng, ngày hết hạn, và ngày hiện hành.

3.2. Tìm kiếm các trường Header MIME

Sáu phương thức cuối cùng đòi hỏi các trường nhất định từ header MIME, nhưng không có giới hạn nào về số các trường header mà một thông điệp MIME có thể có. Năm phương thức tiếp theo kiểm tra các trường nhất định trong header MIME.

FileURLConnection không có các header MIME thực sự , vì vậy tất cả các phương thức này trả về giá trị null khi bạn đang làm việc với một file: URL, hành vi mặc định của chúng. HttpURLConnections tìm ra một trường header để thỏa mãn yêu cầu của bạn. Nếu được tìm thấy, nó được trả về, ngược lại nó trả về giá trị null.

- public String getHeaderField(String name)

Phương thức getHeaderField() trả về giá trị của trường header MIME được đặt tên. Tên của header không phân biệt chữ hoa và chữ thường và không chứa dấu kết thúc.

Ví dụ, để tìm giá trị của các trường header Content-type, Content-encoding của một đối tượng *URLConnection uc* bạn có thể viết:

uc.getHeaderField("content-type");

uc.getHeaderField("content-encoding");

Để nhận thông tin về các trường Date, Content-length, hoặc Expiration bạn cũng thực hiện tương tự:

uc.getHeaderField("date");

uc.getHeaderField("expires");

uc.getHeaderField("Content-length");

Tất cả các phương thức này đều trả về các String, không phải int cũng không phải long như các phương thức getContentLength(); getExpirationDate(), getLastModified(), và getDate(). Nếu bạn quan tâm đến một giá trị số, bạn phải chuyển đổi String thành long hoặc int.

- public String getHeaderFieldKey(int n)

  Phương thức này trả về khóa (nghĩa là tên trường: ví dụ, Content-length hoặc Server) của trường header thứ n. Header đầu tiên là 0. Ví dụ, để nhận khóa thứ 6 của header MIME của URLConnection, bạn viết:

  String header5=uc.getHeaderFieldKey(5);

- public String getHeaderField(int n)

  Phương thức này trả về giá trị trường header MIME thứ n. Header MIME đầu tiên là một.

Ví dụ: Sử dụng phương thức kết hợp với phương thức getHeaderFieldKey()để in ra header MIME.

- public long getHeaderFieldDate(String name, long default)

  Phương thức này trước hết tìm kiếm trường header được xác định bởi tham số name và cố gắng chuyển đổi xâu này sang kiểu long.

3.3. Các phương thức RequestProperty

Bốn phương thức sau không thực hiện bất kỳ công việc gì trong lớp cơ sở URLConnection, cũng không được cài đặt trong các lớp FileURLConnection hoặc HttpConnection. Bạn có thể mong muốn nạp chồng chúng trong một lớp con để cài đặt phương thức tra cứu bảng băm, chẳng hạn để xây dựng một bảng băm chứa tất cả các header MIME của yêu cầu.

- public String getRequestProperty(String property_name)

  Phương thức này đưa ra một ngoại lệ IllegalAccesError nếu liên kết là mở, ngược lại phương thức trả về giá trị null. Nếu bạn nạp chồng nó, các phương thức của bạn cần trả về giá trị gắn với một thuộc tính cho trước như là một xâu.

- public static void setDefaultRequestProperty(String property_name, String property_value)

  Phương thức này không thực hiện công việc gì. Nếu bạn nạp chồng phương thức này, bạn sẽ sử dụng nó để lưu trữ một giá trị mặc định cho thuộc tính cho trước.

- public void setRequestProperty(String property_name, String property_value)

  Phương thức này trả về ngoại lệ IllegalAccessError nếu liên kết đang mở. Ngược lại nó không thực hiện gì. Nếu bạn nạp chồng nó, bạn sẽ sử dụng nó để lưu trữ giá trị của một thuộc tính cho trước.

- public String getDefaultRequest(String property_name)

Phương thức này luôn trả về giá trị null. Nếu bạn nạp chồng phương thức này, bạn cần trả về giá trị mặc định được gán cho một thuộc tính cho trước như là một String.

- protected URLConnection(URL u)

  Contructor trong URLConnection nhận một tham số là URL để thực hiện việc liên kết. Tất cả các tính chất khác của một URLConnection ban đầu được thiết lập là các giá trị mặc định của chúng và bạn có thể thay đổi chúng bằng tập các phương thức. Vì constructor có tính chất protected, chỉ có các đối tượng trong gói java.net mới có thể tạo ra một URLConnection. URLConnection là một lớp trừu tượng vì vậy bạn chỉ có thể gọi constructor của nó từ constructor của một trong các lớp con của nó.

Nếu bạn đang tạo ra lớp con của lớp URLConnection, bạn phải gọi constructor này trong dòng đầu của constructor của lớp con như sau:

```
myURLConnection(URL u)
{
super(u);
}
```

Nếu bạn không đưa vào một lời gọi cụ thể tới constructor trong lớp con của bạn, Java cố gắng tạo ra một constructor không tham số của lớp cha: ví dụ URLConnection(). Vì lớp URLConnection không cung cấp các constructor không tham số, loại bỏ lời gọi cụ thể sẽ gây ra lỗi biên dịch.

3.4. Các trường và các phương thức có liên quan

Có mười ba trường trong lớp java.net. URLConnection. Bảy trường là các biến tĩnh định nghĩa các giá trị mặc định cho các thể hiện của lớp URLConnection. Sáu phương thức khác định nghĩa trạng thái của một đối tượng URLConnection cụ thể. Một vài phương thức get và set thay đổi các giá trị của chúng.

Hầu hết các phương thức thiết lập các trường đưa ra ngoại lệ IllegalAccessExceptions nếu bạn gọi chúng trong khi liên kết đang mở. Nhìn chung, bạn chỉ có thể thiết lập các thuộc tính của một đối tượng URLConnection trong khi liên kết đóng.

- protected URL url
- public URL getURL()

Trường url xác định URL mà URLConnection liên kết tới nó. Nó được thiết lập bởi constructor khi bạn tạo ra một URLConnection, và không cần thay đổi. Bạn có thể tìm kiếm giá trị bằng cách gọi phương thức getURL().

```
import java.net.*;
import java.io.IOException;
public class   printURLConnection
{
 public static void main(String[] args)
 {
```

```
URL u;
URLConnection uc;
try{
        u = new URL("http://www.ora.com/");
        try
        {
                uc=u.openConnection();
                System.out.println(uc.getURL());
        }
        catch(IOException e)
        {
                 System.err.println(e);
        }
}
catch(MalformedURLException e)
{
        System.err.println(e);
}

 }
}
```

- protected boolean connected

Trường connected là đúng nếu liên kết là mở và là sai nếu liên kết đóng. Vì bạn không mở liên kết khi một đối tượng URLConnection được tạo ra, giá trị ban đầu của nó là false. Bạn chỉ có thể truy cập tới biến này thông qua các thể hiện của lớp URLConnection và các lớp con của nó. Không có các phương thức đọc hoặc thay đổi giá trị của nó. Khi viết một trình quản trị giao thức, bạn có nhiệm vụ thiết lập giá trị của biến này là true và thiết lập lại nó bằng giá trị false khi liên kết đóng. Rất nhiều phương thức trong gói java.net. URLConnection đọc biến này để xác định xem chúng có thể thực hiện việc gì. Nếu việc thiết lập không chính xác chương trình của bạn sẽ gặp các lỗi không dễ chuẩn đoán.

- protected boolean allowUserInteraction
- public void setAllowUserInteraction(boolean allowuserinteraction)
- public boolean getAllowUserInteraction()

Một số URLConnection cần tương tác với người dùng. Ví dụ, một trình duyệt web có thể yêu cầu username và password. Tuy nhiên, rất nhiều ứng dụng không thể khẳng định một người sử dụng đang có mặt để tương tác với nó. Như ý nghĩa của tên, allowUserInteraction xác định xem liệu tương tác người sử dụng có được phép hay không. Ở chế độ mặc định nó được thiết lập là false. Vì biến này là protected, ta có thể sử dụng phương thức getAllowUserInteraction()để đọc giá trị của nó và sử dụng phương thức setAllowUserInteraction()để thiết lập giá trị của nó. Giá trị true chỉ ra rằng tương tác với người sử dụng là được phép; giá trị false chỉ ra rằng không có tương tác với người dùng. Giá trị có thể được đọc ở bất kỳ thời điểm nào, nhưng nó chỉ có thể được thiết lập

khi liên kết bị đóng. Gọi phương thức setAllowUserInteraction() khi liên kết mở sẽ đưa ra ngoại lệ IllegalAccessError (chứ không phải là IllegalAccessException). Các chương trình thường không đón bắt các lỗi (không giống như các ngoại lệ); một lỗi không được đón bắt thường buộc chương trình phải kết thúc.

Chương 6

Lập trình Socket cho giao thức TCP

1. Mô hình client/server

Mô hình được phổ biến nhất và được chấp nhận rộng rãi trong các hệ thống phân tán là mô hình client/server. Trong mô hình này sẽ có một tập các tiến trình mà mỗi tiến trình đóng vai trò như là một trình quản lý tài nguyên cho một tập hợp các tài nguyên cho trước và một tập hợp các tiến trình client trong đó mỗi tiến trình thực hiện một tác vụ nào đó cần truy xuất tới tài nguyên phần cứng hoặc phần mềm dùng chung. Bản thân các trình quản lý tài nguyên cần phải truy xuất tới các tài nguyên dùng chung được quản lý bởi một tiến trình khác, vì vậy một số tiến trình vừa là tiến trình client vừa là tiến trình server. Các tiến trình phát ra các yêu cầu tới các server bất kỳ khi nào chúng cần truy xuất tới một trong các tài nguyên của các server. Nếu yêu cầu là đúng đắn thì server sẽ thực hiện hành động được yêu cầu và gửi một đáp ứng trả lời tới tiến trình client.

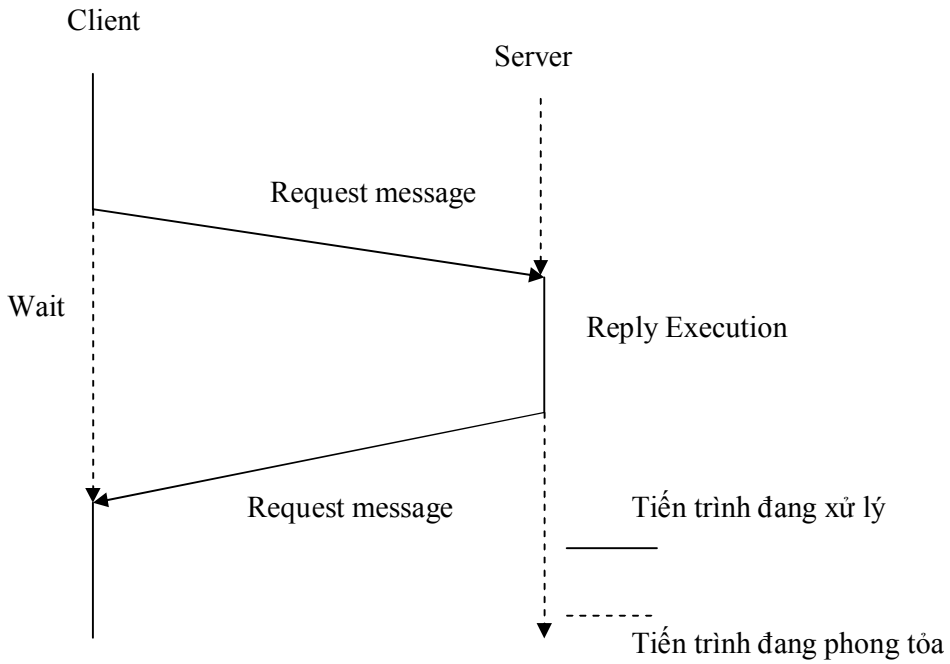
Mô hình client/server cung cấp một cách tiếp cận tổng quát để chia sẻ tài nguyên trong các hệ thống phân tán. Mô hình này có thể được cài đặt bằng rất nhiều môi trường phần cứng và phần mềm khác nhau. Các máy tính được sử dụng để chạy các tiến trình client/server có nhiều kiểu khác nhau và không cần thiết phải phân biệt giữa chúng; cả tiến trình client và tiến trình server đều có thể chạy trên cùng một máy tính. Một tiến trình server có thể sử dụng dịch vụ của một server khác.

Mô hình truyền tin client/server hướng tới việc cung cấp dịch vụ. Quá trình trao đổi dữ liệu bao gồm:

1. Truyền một yêu cầu từ tiến trình client tới tiến trình server
2. Yêu cầu được server xử lý
3. Truyền đáp ứng cho client

Mô hình truyền tin này liên quan đến việc truyền hai thông điệp và một dạng đồng bộ hóa cụ thể giữa client và server. Tiến trình server phải nhận thức được thông điệp được yêu cầu ở bước một ngay khi nó đến và hành động phát ra yêu cầu trong client phải được tạm dừng (bị phong tỏa) và buộc tiến trình client ở trạng thái chờ cho tớ khi nó nhận được đáp ứng do server gửi về ở bước ba.

Mô hình client/server thường được cài đặt dựa trên các thao tác cơ bản là gửi (send) và nhận (receive).



Hình 4.1

Quá trình giao tiếp client và server có thể diễn ra theo một t rong hai chế độ: bị phong tỏa (blocked) và không bị phong tỏa (non-blocked).

Chế độ bị phong tỏa (blocked):

Trong chế độ bị phong tỏa, khi tiến trình client hoặc server phát ra lệnh gửi dữ liệu (send), việc thực thi của tiến trình sẽ bị tạm ngừng cho tới khi tiến trình nhận phát ra lệnh nhận dữ liệu (receive).

Tương tự đối với tiến trình nhận dữ liệu, nếu tiến trình nào đó (client hoặc server) phát ra lệnh nhận dữ liệu, mà tại thời điểm đó chưa có dữ liệu gửi tới thì việc thực thi của tiến trình cũng sẽ bị tạm ngừng cho tới khi có dữ liệu gửi tới.

Chế độ không bị phong tỏa (non-blocked)

Trong chế độ này, khi tiến trình client hay server phát ra lệnh gửi dữ liệu thực sự, việc thực thi của tiến trình vẫn được tiến hành mà không quan tâm đến việc có tiến trình nào phát ra lệnh nhận dữ liệu đó hay không.

Tương tự cho trường hợp nhận dữ liệu, khi tiến trình phát ra lệnh nhận dữ liệu, nó sẽ nhận dữ liệu hiện có, việc thực thi của tiến trình vẫn được tiến hành mà không quan tâm đến việc có tiến trình nào phát ra lệnh gửi dữ liệu tiếp theo hay không.

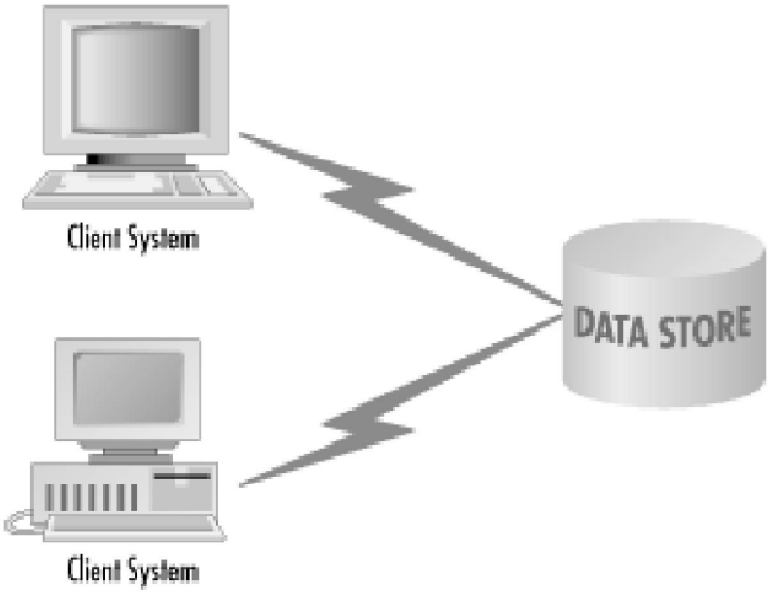
2. Các kiến trúc Client/Server

2.1. Client/Server hai tầng (two-tier client/server)

Kiến trúc client/server đơn giản nhất là kiến trúc hai tầng. Trong thực tế hầu hết các kiến trúc client/server là kiến trúc hai tầng. Một ứng dụng hai tầng cung cấp nhiều trạm làm việc với một tầng trình diễn thống nhất, tầng này truyền tin với tầng lưu trữ dữ liệu tập trung. Tầng trình diễn thông thường là client, và tầng lưu trữ dữ liệu là server.

Hầu hết các ứng dụng Internet như là email, telnet, ftp thậm chí là cả Web là các ứng dụng hai tầng. Phần lớn các lập trình viên trình ứng dụng viết các ứng dụng client/server có xu thế sử dụng kiến trúc này.

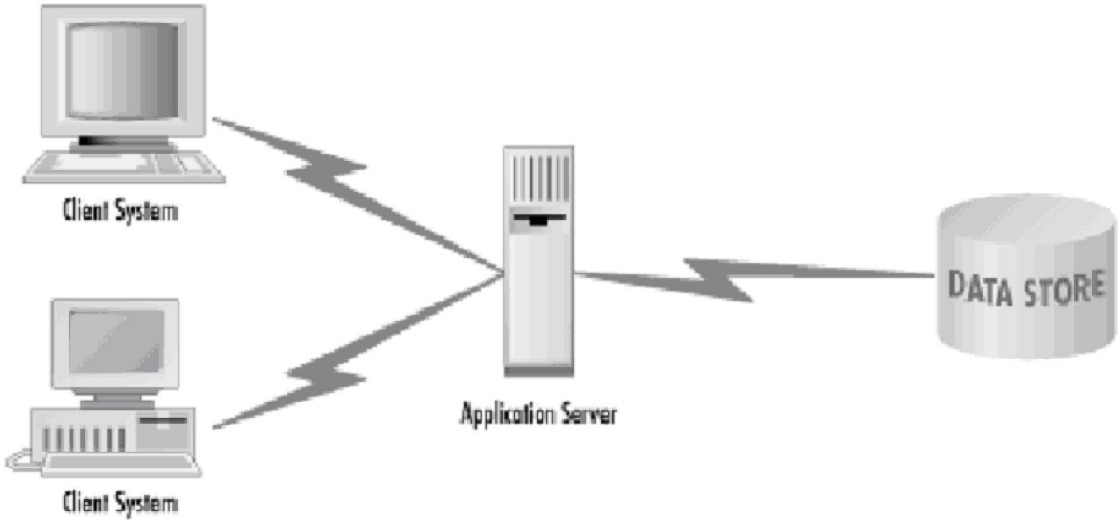
Trong ứng dụng hai tầng truyền thống, khối lượng công việc xử lý được dành cho phía client trong khi server chỉ đơn giản đóng vai trò như là chương trình kiểm soát luồng vào ra giữa ứng dụng và dữ liệu. Kết quả là không chỉ hiệu năng của ứng dụng bị giảm đi do tài nguyên hạn chế của PC, mà khối lượng dữ liệu truyền đi trên mạng cũng tăng theo. Khi toàn bộ ứng dụng được xử lý trên một PC, ứng dụng bắt buộc phải yêu cầu nhiều dữ liệu trước khi đưa ra bất kỳ kết quả xử lý nào cho người dùng. Nhiều yêu cầu dữ liệu cũng làm giảm hiệu năng của mạng. Một vấn đề thường gặp khác đối với ứng dụng hai tầng là vấn đề bảo trì. Chỉ cần một thay đổi nhỏ đối với ứng dụng cũng cần phải thay đổi lại toàn bộ ứng dụng client và server.



Hình 4.2

2.2. Client/Server ba tầng

Ta có thể tránh được các vấn đề của kiến trúc client/server hai tầng bằng cách mở rộng kiến trúc thành ba tầng. Một kiến trúc ba tầng có thêm một tầng mới tác biệt việc xử lý dữ liệu ở vị trí trung tâm.



Hình 4.3

Theo kiến trúc ba tầng, một ứng dụng được chia thành ba tầng tách biệt nhau về mặt logic. Tầng đầu tiên là tầng trình diễn thường bao gồm các giao diện đồ họa. Tầng thứ hai, còn được gọi là tầng trung gian hay tầng tác nghiệp. Tầng thứ ba chứa dữ liệu cần cho ứng dụng. Tầng thứ ba về cơ bản là chương trình thực hiện các lời gọi hàm để tìm kiếm dữ liệu cần thiết. Tầng trình diễn nhận dữ liệu và định dạng nó để hiển thị. Sự tách biệt giữa chức năng xử lý với giao diện đã tạo nên sự linh hoạt cho việc thiết kế ứng dụng. Nhiều giao diện người dùng được xây dựng và triển khai mà không làm thay đổi logic ứng dụng.

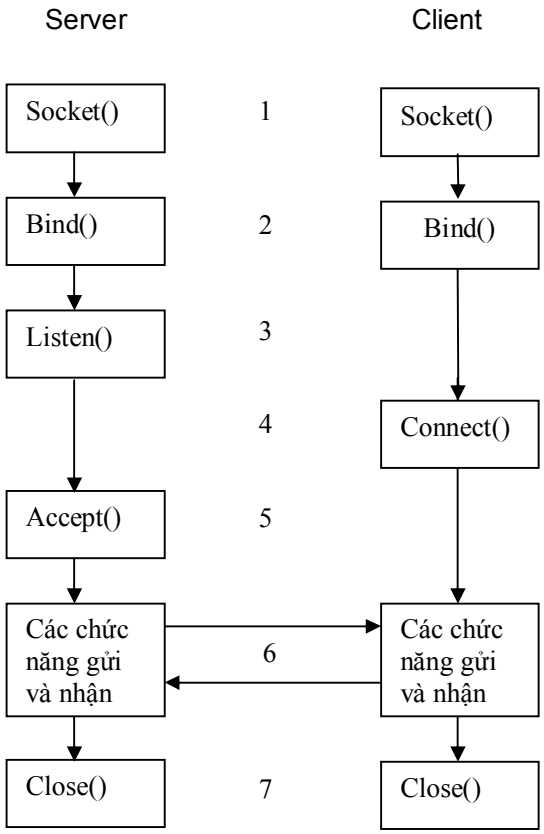
Tầng thứ ba chứa dữ liệu cần thiết cho ứng dụng. Dữ liệu này có thể bao gồm bất kỳ nguồn thông tin nào, bao gồm cơ sở dữ liệu như Oracale, SQL Server hoặc tài liệu XML.

2.3. Kiến trúc n-tầng

Kiến trúc n-tầng được chia thành các tầng như sau:

- Tầng giao diện người dùng: quản lý tương tác của người dùng với ứng dụng
- Tầng logic trình diễn: Xác định cách thức hiển thị giao diện người dùng và các yêu cầu của người dùng được quản lý như thế nào.
- Tầng logic tác nghiệp: Mô hình hóa các quy tắc tác nghiệp,
- Tầng các dịch vụ hạ tầng: Cung cấp một chức năng bổ trợ cần thiết cho ứng dụng như các thành phần (truyền thông điệp, hỗ trợ giao tác).

3. Mô hình truyền tin socket



Hình 4.4

Khi lập trình, ta cần quan tâm đến chế độ bị phong tỏa, vì nó có thể dẫn đến tình huống một tiến trình nào đó sẽ rơi vào vòng lặp vô hạn của quá trình gửi hoặc nhận.

Trong chương 1 chúng ta đã biết hai giao thức TCP và UDP là các giao thức tầng giao vận để truyền dữ liệu. Mỗi giao thức có những ưu và nhược điểm riêng. Chẳng hạn, giao thức TCP có độ tin cậy truyền tin cao, nhưng tốc độ truyền tin bị hạn chế do phải có giai đoạn thiết lập và giải phóng liên kết khi truyền tin, khi gói tin có lỗi hay bị thất lạc thì giao thức TCP phải có trách nhiệm truyền lại,…Ngược lại, giao thức UDP có tốc độ truyền tin rất nhanh vì nó chỉ có một cơ chế truyền tin rất đơn giản: không cần phải thiết lập và giải phóng liên kết. Khi lập trình cho TCP ta sử dụng các socket luồng, còn đối với giao thức UDP ta sẽ sử dụng lớp DatagramSocket và DatagramPacket.

Truyền tin hướng liên kết nghĩa là cần có giai đoạn thiết lập liên kết và giải phóng liên kết trước khi truyền tin. Dữ liệu được truyền trên mạng Internet dưới dạng các gói (packet) có kích thước hữu hạn được gọi là datagram. Mỗi datagram chứa một header và một payload. Header chứa địa chỉ và cổng cần truyền gói tin đến, cũng như địa chỉ và cổng xuất phát của gói tin, và các thông tin khác được sử dụng để đảm bảo độ tin cậy truyền tin, payload chứa dữ liệu. Tuy nhiên do các datagram có chiều dài hữu hạn nên thường phải phân chia dữ liệu thành nhiều gói và khôi phục lại dữ liệu ban đầu từ các gói ở nơi nhận. Trong quá trình truyền tin có thể có thể có một hay nhiều gói bị mất hay bị hỏng và cần phải truyền lại hoặc các gói tin đến không theo đúng trình tự. Để tránh những điều này, việc phân chia dữ liệu thành các gói, tạo các header, phân tích header của các gói đến, quản lý danh sách các gói đã nhận được và các gói chưa nhận được, ... rất nhiều công việc cần phải thực hiện, và đòi hỏi rất nhiều phần mềm phức tạp.

Thật may mắn, ta không cần phải tự thực hiện công việc này. Socket là một cuộc cách mạng của Berkeley UNIX. Chúng cho phép người lập trình xem một liên kết mạng như là một luồng mà có thể đọc dữ liệu ra hay ghi dữ liệu vào từ luồng này.

Về mặt lịch sử Socket là một sự mở rộng của một trong những ý tưởng quan trọng nhất của UNIX: tất cả các thao tác vào/ra giống như vào ra tệp tin đối với người lập trình, cho dù ta đang làm việc với bàn phím, màn hình đồ họa, một file thông thường, hay một liên kết mạng. Các Socket che dấu người lập trình khỏi các chi tiết mức thấp của mạng như môi kiểu đường truyền, các kích thước gói, yêu cầu truyền lại gói, các địa chỉ mạng...

Một socket có thể thực hiện bảy thao tác cơ bản:

- Kết nối với một máy ở xa (ví dụ, chuẩn bị để gửi và nhận dữ liệu)
- Gửi dữ liệu
- Nhận dữ liệu
- Ngắt liên kết
- Gán cổng
- Nghe dữ liệu đến
- Chấp nhận liên kết từ các máy ở xa trên cổng đã được gán

Lớp Socket của Java được sử dụng bởi cả client và server, có các phương thức tương ứng với bốn thao tác đầu tiên. Ba thao tác cuối chỉ cần cho server để chờ các client liên kết với chúng. Các thao tác này được cài đặt bởi lớp ServerSocket. Các socket cho client thường được sử dụng theo mô hình sau:

- Một socket mới được tạo ra bằng cách sử dụng hàm Socket().
- Socket cố gắng liên kết với một host ở xa.
- Mỗi khi liên kết được thiết lập, các host ở xa nhận các luồng vào và luồng ra từ socket, và sử dụng các luồng này để gửi dữ liệu cho nhau. Kiểu liên kết này được gọi

là song công (full-duplex)-các host có thể nhận và gửi dữ liệu đồng thời. Ý nghĩa của dữ liệu phụ thuộc vào giao thức.

- Khi việc truyền dữ liệu hoàn thành, một hoặc cả hai phía ngắt liên kết. Một số giao thức, như HTTP, đòi hỏi mỗi liên kết phải bị đóng sau mỗi khi yêu cầu được phục vụ. Các giao thức khác, chẳng hạn FTP, cho phép nhiều yêu cầu được xử lý trong một liên kết đơn.

4. Socket cho Client

4.1. Các constructor

- *public Socket(String host, int port) throws UnknownHostException, IOException*

Hàm này tạo một socket TCP với host và cổng xác định, và thực hiện liên kết với host ở xa.

Ví dụ:

```
try{
Socket s = new Socket( “www.vnn.vn”,80);
}
catch(UnknownHostException e){
  System.err.println(e);
}
catch(IOException e){
System.err.println(e);
}
```

Trong hàm này tham số host là hostname kiểu String, nếu host không xác định hoặc máy chủ tên miền không hoạt động thì constructor đưa ra ngoại lệ UnknownHostException. Vì một lý do nào đó mà không thể mở được socket thì constructor sẽ đưa ra ngoại lệ IOException. Có nhiều nguyên nhân khiến cho một liên kết thất bại: host mà ta đang cố gắng kết nối tới không chấp nhận liên kết, kết nối Internet có thể bị ngắt, hoặc vấn đề định tuyến có thể ngăn ngừa các gói tin của ta tới đích.

Ví dụ: Viết chương trình để kiểm tra trên 1024 cổng đầu tiên những cổng nào đang có server hoạt động

```
import java.net.*;
import java.io.*;
class PortScanner
{
 public static void main(String[] args)
 {
        String host="localhost";
        if(args.length>0){
              host=args[0];
        }
        for(int i=0;i<1024;i++){
        try{
                    Socket s=new Socket(host,i);
              System.out.println("Co mot server dang hoat dong  tren cong:"+i);
```

```
                }
                catch(UnknownHostException e){
                        System.err.println(e);
                }
                catch(IOException e){
                        System.err.println(e);
                }
        }

    }
}
```

- *public Socket(InetAddress host, int port)throws IOException*

  Tương tự như constructor trước, constructor này tạo một socket TCP với thông tin là địa chỉ của một host được xác định bởi một đối tượng InetAddres và số hiệu cổng port, sau đó nó thực hiện kết nối tới host. Nó đưa ra ngoại lệ IOException nhưng không đưa ra ngoại lệ UnknownHostException. Constructor đưa ra ngoại lệ trong trường hợp không kết nối được tới host.

- *public Socket (String host, int port, InetAddress interface, int localPort) throws IOException, UnknownHostException*

  Constructor này tạo ra một socket với thông tin là địa chỉ IP được biểu diễn bởi một đối tượng String và một số hiệu cổng và thực hiện kết nối tới host đó. Socket kết nối tới host ở xa thông qua một giao tiếp mạng và số hiệu cổng cục bộ được xác định bởi hai tham số sau. Nếu localPort bằng 0 thì Java sẽ lựa chọn một cổng ngẫu nhiên có sẵn nằm trong khoảng từ 1024 đến 65535.

- *public Socket (InetAddress host, int port, InetAddress interface, int localPort) throws IOException, UnknownHostException*

  Constructor chỉ khác constructor trên ở chỗ địa chỉ của host lúc này được biểu diễn bởi một đối tượng InetAddress.

4.2. Nhận các thông tin về Socket

Đối tượng Socket có một số trường thông tin riêng mà ta có thể truy nhập tới chúng thông qua các phương thức trả về các thông tin này.

- *public InetAddress getInetAddress()*

  Cho trước một đối tượng Socket, phương thức getInetAddress() cho ta biết host ở xa mà Socket kết nối tới, hoặc liên kết đã bị ngắt thì nó cho biết host ở xa mà Socket đã kết nối tới

- *public int getPort()*

  Phương thức này cho biết số hiệu cổng mà Socket kết nối tới trên host ở xa.

- *public int getLocalPort()*

  Thông thường một liên kết thường có hai đầu: host ở xa và host cục bộ. Để tìm ra số hiệu cổng ở phía host cục bộ ta gọi phương thức getLocalPort().

- *public InetAddress getLocalAddress()*

  Phương thức này cho ta biết giao tiếp mạng nào mà một socket gắn kết với nó.

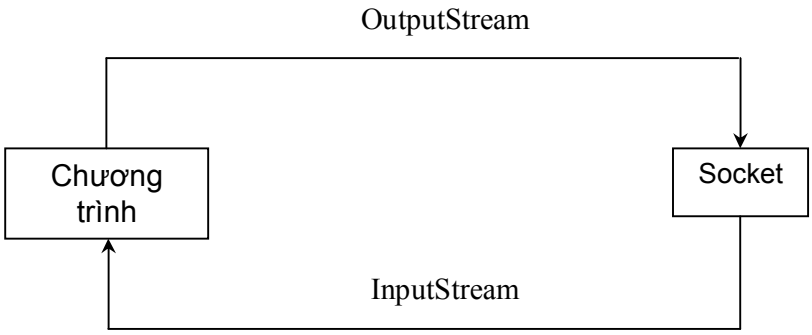
- *public InputStream getInputStream() throws IOException*

Phương thức geInputStream() trả về một luồng nhập để đọc dữ liệu từ một socket vào chương trình. Thông thường ta có thể gắn kết luồng nhập thô InputStream tới một luồng lọc hoặc một luồng ký tự nhằm đưa các chức năng tiện ích (chẳng hạn như các luồng InputStream, hoặc InputStreamReader). Để tăng cao hiệu năng, ta có thể đệm dữ liệu bằng cách gắn kết nó với luồng lọc BufferedInputStream hoặc BufferedReader.

- *public OutputStream getOutputStream() throws IOException*

  Phương thức getOutputStream() trả về một luồng xuất thô để ghi dữ liệu từ ứng dụng ra đầu cuối của một socket. Thông thường, ta sẽ gắn kết luồng này với một luồng tiện lợi hơn như lớp DataOuputStream hoặc OutputStreamWriter trước khi sử dụng nó. Để tăng hiệu quả ghi.

Hai phương thức getInputStream() và getOutputStream() là các phương thức cho phép ta lấy về các luồng dữ liệu nhập và xuất. Như đã đề cập ở chương 3 vào ra trong Java được tiến hành thông qua các luồng, việc làm việc với các socket cũng không phải là một ngoại lệ. Để nhận dữ liệu từ một máy ở xa ta nhận về một luồng nhập từ socket và đọc dữ liệu từ luồng đó. Để ghi dữ liệu lên một máy ở xa ta nhận về một luồng xuất từ socket và ghi dữ liệu lên luồng. Dưới đây là hình vẽ để ta hình dung trực quan hơn.



Hình 4.5

4.3. Đóng Socket

Đến thời điểm ta đã có đầy đủ các thông tin cần thiết để triển khai một ứng dụng phía client. Khi viết một chương trình ứng dụng phía client tất cả mọi công việc đều chuyển về việc quản lý luồng và chuyển đổi dữ liệu từ luồng thành dạng thức mà người sử dụng có thể hiểu được. Bản thân các socket rất đơn giản bởi vì các phần việc phức tạp đã được che dấu đi. Đây chính là lý do để socket trở thành một lựa chọn có tính chiến lược cho lập trình mạng.

- *public void close() throws IOException*

Các socket được đóng một cách tự động khi một trong hai luồng đóng lại, hoặc khi chương trình kết thúc, hoặc khi socket được thu hồi bởi gabbage collector. Tuy nhiên, thực tế cho thấy việc cho rằng hệ thống sẽ tự đóng socket là không tốt, đặc biệt là khi các chương trình chạy trong khoảng thời gian vô hạn. Để đóng một socket ta có thể dùng phương thức close().

Mỗi khi một Socket đã bị đóng lại, ta vẫn có thể truy xuất tới các trường thông tin InetAddress, địa chỉ cục bộ, và số hiệu cổng cục bộ thông qua các phương thức getInetAddress(), getPort(), getLocalHost(), và getLocalPort(). Tuy nhiên khi ta gọi các phương thức getInputStream() hoặc getOutputStream() để đọc dữ liệu từ luồng đọc InputStream hoặc ghi dữ liệu OuputStream thì ngoại lệ IOException được đưa ra.

Các socket đóng một nửa (Half-closed socket)

Phương thức close() đóng cả các luồng nhập và luồng xuất từ socket. Trong một số trường hợp ta chỉ muốn đóng một nửa kết nối, hoặc là luồng nhập hoặc là luồng xuất. Bắt đầu từ Java 1.3, các phương thức shutdownInput() và shutdownOutput() cho phép ta thực hiện điều này.

- *public void shutdownInput() throws IOException*
- *public void shutdownOutput() throws IOException*

Các phương thức này không thực sự ngắt liên kết. Tuy nhiên, nó chỉ điều chỉnh luồng kết nối tới nó sao cho.

Trong Java 1.4 đưa thêm vào hai phương thức các luồng nhập và luồng xuất mở hay đóng

- *public boolean isInputShutdown()*
- *public boolean isOutputShutdown()*

4.4. Thiết lập các tùy chọn cho Socket

4.4.1. TCP_NODELAY

- *public void setTcpNoDelay(boolean on) throws SocketException*
- *public boolean getTcpNoDelay() throws SocketException*

Thiết lập giá trị TCP_NODELAY là true để đảm bảo rằng các gói tin được gửi đi nhanh nhất có thể mà không quan tâm đến kích thước của chúng. Thông thường, các gói tin nhỏ được kết hợp lại thành các gói tin lớn hơn trước khi được gửi đi. Trước khi gửi đi một gói tin khác, host cục bộ đợi để nhận các xác thực của gói tin trước đó từ hệ thống ở xa.

4.4.2. SO_LINGER

- *public void setSoLinger(boolean on, int seconds) throws SocketException*
- *public int getSoLinger() throws SocketException*

Tùy chọn SO_LINGER xác định phải thực hiện công việc gì với datagram vẫn chưa được gửi đi khi một socket đã bị đóng lại. Ở chế độ mặc định, phương thức close() sẽ có hiệu lực ngay lập tức; nhưng hệ thống vẫn cố gắng để gửi phần dữ liệu còn lại. Nếu SO_LINGER được thiết lập bằng 0, các gói tin chưa được gửi đi bị phá hủy khi socket bị đóng lại. Nếu SO_LINGER lớn hơn 0, thì phương thức close() phong tỏa để chờ cho dữ liệu được gửi đi và nhận được xác thực từ phía nhận. Khi hết thời gian qui định, socket sẽ bị đóng lại và bất kỳ phần dữ liệu còn lại sẽ không được gửi đi.

4.4.3. SO_TIMEOUT

- *public void setSoTimeout(int milliseconds) throws SocketException*
- *public int getSoTimeout() throws SocketException*

Thông thường khi ta đọc dữ liệu từ mộ socket, lời gọi phương thức phong tỏa cho tới khi nhận đủ số byte. Bằng cách thiết lập phương thức SO_TIMEOUT, ta sẽ đảm bảo rằng lời gọi phương thức sẽ không phong tỏa trong khoảng thời gian quá số giây quy định.

4.5. Các phương thức của lớp Object

Lớp Socket nạp chồng phương thức chuẩn của lớp java.lang.Object, toString(). Vì các socket là các đối tượng tạm thời và thường chỉ tồn tại khi liên kết tồn tại.

- *public String toString()*

Phương thức toString() tạo ra một xâu ký tự như sau:

Socket[addr=www.oreilly.com/198.122.208.11,port=80,localport=50055]

Phương thức này thường hữu ích cho việc gỡ rối.

4.6. Các ngoại lệ Socket

Hầu hết các phương thức của lớp Socket được khai báo đưa ra ngoại lệ IOException, hoặc lớp con của lớp IOExcepton là lớp SocketException.

4.7. Các lớp SocketAddress

Lớp SocketAddress bắt đầu có từ phiên bản Java 1.4, biểu diễn một đầu cuối của liên kết. Lớp SocketAddress là một lớp trừu tượng mà không có phương thức nào ngoài construtor mặc định. Lớp này có thể được sử dụng cho cả các socket TCP và socket không phải là TCP. Các lớp con của lớp SocketAddress cung cấp thông tin chi tiết hơn thích hợp cho kiểu socket. Trong thực tế, chỉ hỗ trợ TCP/IP.

Mục đích chính của lớp SocketAddress là cung cấp một nơi lưu trữ các thông tin liên kết socket tạm thời (như địa chỉ IP và số hiệu cổng) có thể được sử dụng lại để tạo ra socket mới.

- *public SocketAddress getRemoteSocketAddress()*
- *public SocketAddress getLocalSocketAddress()*

Cả hai phương thức này trả về giá trị null nếu socket vẫn chưa kết nối tới.

5. Lớp ServerSocket

Lớp ServerSocket có đủ mọi thứ ta cần để viết các server bằng Java. Nó có các constructor để tạo các đối tượng ServerSocket mới, các phương thức để lắng nghe các liên kết trên một cổng xác định, và các phương thức trả về một Socket khi liên kết được thiết lập, vì vậy ta có thể gửi và nhận dữ liệu.

Vòng đời của một server

1. Một ServerSocket mới được tạo ra trên một cổng xác định bằng cách sử dụng một constructor ServerSocket.
2. ServerSocket lắng nghe liên kết đến trên cổng đó bằng cách sử dụng phương thức accept(). Phương thức accept() phong tỏa cho tới khi một client thực hiện một liên kết, phương thức accept() trả về một đối tượng Socket mà liên kết giữa client và server.
3. Tùy thuộc vào kiểu server, hoặc phương thức getInputStream(), getOutputStream() hoặc cả hai được gọi để nhận các luồng vào ra để truyền tin với client.
4. server và client tương tác theo một giao thức thỏa thuận sẵn cho tới khi ngắt liên kết.
5. Server, client hoặc cả hai ngắt liên kết
6. Server trở về bước hai và đợi liên kết tiếp theo.

5.1. Các constructor

- *public ServerSocket(int port) throws IOException, BindException*

Constructor này tạo một socket cho server trên cổng xác định. Nếu port bằng 0, hệ thống chọn một cổng ngẫu nhiên cho ta. Cổng do hệ thống chọn đôi khi được gọi là cổng vô danh vì ta không biết số hiệu cổng. Với các server, các cổng vô danh không hữu ích lắm vì các client cần phải biết trước cổng nào mà nó nối tới (giống như người gọi điện thoại ngoài việc xác định cần gọi cho ai cần phải biết số điện thoại để liên lạc với người đó).

Ví dụ: Để tạo một server socket cho cổng 80

```
try{
  ServerSocket httpd = new ServerSocket(80);
}
catch(IOException e)
{
  System. err.println(e);
}
```

Constructor đưa ra ngoại lệ IOException nếu ta không thể tạo và gán Socket cho cổng được yêu cầu. Ngoại lệ IOException phát sinh khi:

- Cổng đã được sử dụng
- Không có quyền hoặc cố liên kết với một cổng nằm giữa 0 và 1023.

Ví dụ;

```
import java.net.*;
import java.io.*;
public class congLocalHost
{
  public static void main(String[] args)
  {
        ServerSocket ss;
        for(int i=0;i<=1024;i++)
        {
              try{
                    ss= new ServerSocket(i);
                    ss.close();
              }
              catch(IOException e)
              {
                    System.out.println("Co mot server tren cong "+i);
              }
        }

  }
}
```

- *public ServerSocket(int port, int queuelength, InetAddress bindAddress)throws IOException*

Constructor này tạo một đối tượng ServerSocket trên cổng xác định với chiều dài hàng đợi xác định. ServerSocket chỉ gán cho địa chỉ IP cục bộ xác định. Constructor này hữu ích cho các server chạy trên các hệ thống có nhiều địa chỉ IP.

5.2. Chấp nhận và ngắt liên kết

Một đối tượng ServerSocket hoạt động trong một vòng lặp chấp nhận các liên kết. Mỗi lần lặp nó gọi phương thức accept(). Phương thức này trả về một đối tượng Socket biểu diễn liên kết giữa client và server. Tương tác giữ client và server được tiến hành thông qua socket này. Khi giao tác hoàn thành, server gọi phương thức close() của đối tượng socket. Nếu client ngắt liên kết trong khi server vẫn đang hoạt động, các luồng vào ra kết nối server với client sẽ đưa ra ngoại lệ InterruptedException trong lần lặp tiếp theo

- public Socket accept() throws IOException

Khi bước thiết lập liên kết hoàn thành, và ta sẵn sàng để chấp nhận liên kết, cần gọi phương thức accept() của lớp ServerSocket. Phương thức này phong tỏa; nó dừng quá trình xử lý và đợi cho tới khi client được kết nối. Khi client thực sự kết nối, phương thức accept() trả về đối tượng Socket. Ta sử dụng các phương thức getInputStream() và getOutputStream() để truyền tin với client.

Ví dụ:

```
try{
  ServerSocket theServer = new ServerSocket(5776);
while(true)
{
            Socket con = theServer.accept();
      PrintStream p = new PrintStream(con.getOutputStream());
      p.println("Ban da ket noi toi server nay. Bye-bye now.");
      con.close();
}
}
catch(IOException e)
{
      System.err.println(e);
}
```

- *public void close() throws IOException*

Nếu ta đã kết thúc làm việc với một đối tượng server socket thì cần phải đóng lại đối tượng này.

Ví dụ: Cài đặt một server daytime

```
import java.net.*;
import java.io.*;
import java.util.Date;
public class daytimeServer{
 public final static int daytimePort =13;

 public static void main(String[]args)
 {
      ServerSocket  theServer;
      Socket con;
```

```
        PrintStream p;
        try{
                theServer = new ServerSocket(daytimePort);
                try{
                        p= new PrintStream(con.getOutputStream());
                        p.println(new Date());
                        con.close();
}
catch(IOException e)
{
  theServer.close();
 System. err.println(e);
}
}
catch(IOException e)
{
     System. err.println(e);
}
}
}
```

- *public void close() throws IOException*

  Nếu đã hoàn thành công việc với một ServerSocket, ta cần phải đóng nó lại, đặc biệt nếu chương trình của ta tiếp tục chạy. Điều này nhằm tạo điều kiện cho các chương trình khác muốn sử dụng nó. Đóng một ServerSocket không đồng nhất với việc đóng một Socket.

Lớp ServerSocket cung cấp một số phương thức cho ta biết địa chỉ cục bộ và cổng mà trên đó đối tượng server đang hoạt động. Các phương thức này hữu ích khi ta đã mở một đối tượng server socket trên một cổng vô danh và trên một giao tiếp mạng không

- *public InetAddress getInetAddress()*

  Phương thức này trả về địa chỉ được sử dụng bởi server (localhost). Nếu localhost có địa chỉ IP, địa chỉ này được trả về bởi phương thức InetAddress.getLocalHost()

  Ví dụ:

```
try{
 ServerSocket httpd = new ServerSocket(80);
 InetAddress ia = httpd.getInetAddress();
}
catch(IOException                                    e)
{
}
```

- *public int getLocalHost()*

Các contructor ServerSocket cho phép ta nghe dữ liệu trên cổng không định trước bằng cách gán số 0 cho cổng. Phương thức này cho phép ta tìm ra cổng mà server đang nghe.

6. Các bước cài đặt chương trình phía Client bằng Java

Sau khi đã tìm hiểu các lớp và các phương thức cần thiết để cài đặt chương trình Socket. Ở mục 6 và mục 7 chúng ta sẽ đi vào các bước cụ thể để cài đặt các chương trình Client và Server.

Các bước để cài đặt Client

- Bước 1:Tạo một đối tượng Socket

  Socket client =new Socket("hostname",portName);

- Bước 2:Tạo một luồng xuất để có thể sử dụng để gửi thông tin tới Socket

  PrintWriter out=new PrintWriter(client.getOutputStream(),true);

- Bước 3:Tạo một luồng nhập để đọc thông tin đáp ứng từ server

BufferedReader in=new BufferedReader(new InputStreamReader(client.getInputStream()));

- Bước 4:Thực hiện các thao tác vào/ra với các luồng nhập và luồng xuất

  Đối với các luồng xuất, PrintWriter, ta sử dụng các phương thức print và println, tương tự như System.out.println.

  Đối với luồng nhập, BufferedReader, ta có thể sử dụng phương thức read() để đọc một ký tự, hoặc một mảng các ký tự, hoặc gọi phương thức readLine() để đọc vào một dòng ký tự. Cần chú ý rằng phương thức readLine() trả về null nếu kết thúc luồng.

- Bước 5: Đóng socket khi hoàn thành quá trình truyền tin

Ví dụ: Viết chương trình client liên kết với một server. Người sử dụng nhập vào một dòng ký tự từ bàn phím và gửi dữ liệu cho server.

```
import java.net.*;
import java.io.*;
public class  EchoClient1
{
 public static void main(String[] args)
 {
        String hostname="localhost";
        if(args.length>0)
        {
                hostname=args[0];
        }
        PrintWriter pw=null;
        BufferedReader br=null;
        try{
                Socket s=new Socket(hostname,2007);
                br=new BufferedReader(new InputStreamReader(s.getInputStream()));
```

```
                    BufferedReader user=new BufferedReader(new
InputStreamReader(System.in));
                    pw=new PrintWriter(s.getOutputStream());
                    System.out.println("Da ket noi duoc voi server...");
                    while(true)
                    {
                        String st=user.readLine();
                        if(st.equals("exit"))
                        {
                            break;
                        }
                        pw.println(st);
                        pw.flush();
                        System.out.println(br.readLine());
                    }
                }
                catch(IOException e)
                {
                    System.err.println(e);
                }
                finally{
                    try{
                        if(br!=null)br.close();
                        if(pw!=null)pw.close();
                    }
                    catch(IOException e)
                    {
                        System.err.println(e);
                    }
                }

        }
    }
```

Chương trình EchoClient đọc vào hostname từ đối dòng lệnh. Tiếp theo ta tạo một socket với hostname đã xác định trên cổng số 2007. Tất nhiên cổng này hoàn toàn do ta lựa chọn sao cho nó không trùng với cổng đã có dịch vụ hoạt động. Việc tạo socket thành công có nghĩa là ta đã liên kết được với server. Ta nhận luồng nhập từ socket thông qua phương thức getInputStream() và gắn kết nó với các luồng ký tự và luồng đệm nhờ lệnh:

```
br=new BufferedReader(new InputStreamReader(s.getInputStream());
```

Tương tự ta lấy về luồng xuất thông qua phương thức getOuputStream() của socket. Sau đó gắn kết luồng này với luồng PrintWriter để gửi dữ liệu tới server

pw=new PrintWriter(s.getOutputStream());

Để đọc dữ liệu từ bàn phím ta gắn bàn phím với các luồng nhập nhờ câu lệnh:

BufferedReader user=new BufferedReader(new InputStreamReader(System.in));

Sau đi đã tạo được các luồng thì vấn đề nhận và gửi dữ liệu trở thành vấn đề đơn giản là đọc dữ liệu từ các luồng nhập br, user và ghi dữ liệu lên luồng xuất pw.

7. Các bước để cài đặt chương trình Server bằng Java

Để cài đặt chương trình Server bằng ServerSocket ta thực hiện các bước sau:

- Bước 1

  Tạo một đối tượng ServerSocket

  ServerSocket ss=new ServerSocket(port)

- Bước 2:

  Tạo một đối tượng Socket bằng cách chấp nhận liên kết từ yêu cầu liên kết của client. Sau khi chấp nhận liên kết, phương thức accept() trả về đối tượng Socket thể hiện liên kết giữa Client và Server.

  ```
  while(condion)
  {
  Socket s=ss.accept();
  doSomething(s);
  }
  ```

  Người ta khuyến cáo rằng chúng ta nên giao công việc xử lý đối tượng s cho một tuyến đoạn nào đó.

- Bước 3: Tạo một luồng nhập để đọc dữ liệu từ client

BufferedReader in=new BufferedReader(new InputStreamReader(s.getInputStream()));

- Bước 4: Tạo một luồng xuất để gửi dữ liệu trở lại cho server

  PrintWriter pw=new PrintWriter(s.getOutputStream(),true);

  Trong đó tham số true được sử dụng để xác định rằng luồng sẽ được tự động đẩy ra.

- Bước 5: Thực hiện các thao tác vào ra với các luồng nhập và luồng xuất
- Bước 6: Đóng socket s khi đã truyền tin xong. Việc đóng socket cũng đồng nghĩa với việc đóng các luồng.

Ví dụ: Viết chương trình server EchoServer để phục vụ chương trình EchoClient1 đã viết ở bước 5

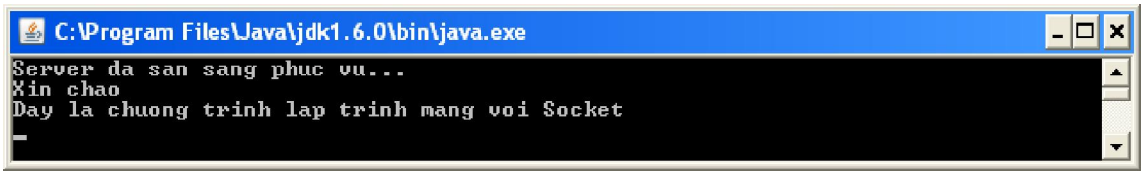
```
import java.net.*;
import java.io.*;


public class  EchoServer1
{
 public final static int DEFAULT_PORT=2007;
 public static void main(String[] args)
 {  int port=DEFAULT_PORT;
        try{
              ServerSocket ss=new ServerSocket(port);
```

```
                    Socket s=null;

                    while(true)
                    {
                        try{
                        s=ss.accept();
                        PrintWriter pw=new PrintWriter(new
OutputStreamWriter(s.getOutputStream()));
                      BufferedReader br=new BufferedReader(new
InputStreamReader(s.getInputStream()));
                        while(true){
                        String line=br.readLine();
                        if(line.equals("exit"))break;
                        String upper=line.toUpperCase();

                        pw.println(upper);
                        pw.flush();
                        }
                        }
                        catch(IOException e)
                        {
                        }
                        finally{
                            try{
                                if(s!=null){
                                    s.close();
                                }
                            }
                            catch(IOException e){}

                        }

                    }
                }
                catch(IOException e)
                {
                }

            }
        }
```

Chương trình bắt đầu bằng việc tạo ra một đối tượng ServerSocket trên cổng xác định. Server lắng nghe các liên kết trong một vòng lặp vô hạn. Nó chấp nhận liên kết bằng cách gọi phương thức accept(). Phương thức accept() trả về một đối tượng Socket thể hiện mối liên kết giữa client và server. Ta cũng nhận về các luồng nhập và luồng xuất từ đối tượng Socket nhờ các phương thức getInputStream() và getOuputStream(). Việc nhận yêu cầu từ client sẽ thông qua các luồng nhập và việc gửi đáp ứng tới server sẽ thông qua luồng xuất.

Khởi động chương trình server

start java EchoServer1



Hình 4.6

Khởi động client

C:\MyJava>start java EchoClient1

Hình 4.7

8. Ứng dụng đa tuyến đoạn trong lập trình Java

Các server như đã viết ở trên rất đơn giản nhưng nhược điểm của nó là bị hạn chế về mặt hiệu năng vì nó chỉ quản lý được một client tại một thời điểm. Khi khối lượng công việc mà server cần xử lý một yêu cầu của client là quá lớn và không biết trước được thời điểm hoàn thành công việc xử lý thì các server này là không thể chấp nhận được.

Để khắc phục điều này, người ta quản lý mỗi phiên của client bằng một tuyến đoạn riêng, cho phép các server làm việc với nhiều client đồng thời. Server này được gọi là server tương tranh (concurrent server)-server tạo ra một tuyến đoạn để quản lý từng yêu cầu, sau đó tiếp tục lắng nghe các client khác.

Chương trình client/server chúng ta đã xét mở mục 6 và mục 7 là chương trình client/server đơn tuyến đoạn. Các server đơn tuyến đoạn chỉ quản lý được một liên kết tại một thời điểm. Trong thực tế một server có thể phải quản lý nhiều liên kết cùng một lúc. Để thực hiện điều này server chấp nhận các liên kết và chuyển các liên kết này cho từng tuyến đoạn xử lý.

Trong phần dưới đây chúng ta sẽ xem xét cách tiến hành cài đặt một chương trình client/server đa tuyến đoạn.

Chương trình phía server

```
import java.io.*;
import java.net.*;


class EchoServe extends Thread
```

```
{
 private Socket socket;
 private BufferedReader in;
 private PrintWriter out;
 public EchoServe (Socket s) throws IOException
 {
  socket = s;
  System.out.println("Serving: "+socket);
  in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
  // Cho phép  auto-flush:
  out = new PrintWriter(new BufferedWriter(new OutputStreamWriter(
        socket.getOutputStream())), true);
  // Nếu bất kỳ lời gọi nào ở trên đưa ra ngoại lệ
  // thì chương trình gọi có trách nhiệm đóng socket. Ngược lại tuyến đoạn sẽ
  // sẽ đóng socket
  start();
 }
 public void run()
 {
 try
      {
 while (true)
 {
          System.out.println("....Server is waiting...");
     String str = in.readLine();
     if (str.equals("exit") ) break;
     System.out.println("Received: " + str);
     System.out.println("From: "+ socket);
      String upper=str.toUpperCase();
     // gửi lại cho client
     out.println(upper);
 }
 System.out.println("Disconnected with.."+socket);
 }
      catch (IOException e) {}
 finally
      {
    try
        {
```

```
            socket.close();
        }
        catch(IOException e) {}
    }
    }
}

public class TCPServer1
{
    static int PORT=0; .
    public static void main(String[] args) throws IOException
    {
      if (args.length == 1)
      {
    PORT=Integer.parseInt(args[0]); // Nhập số hiệu cổng từ đối dòng lệnh
      }
    // Tạo một đối tượng Server Socket
    ServerSocket s = new ServerSocket(PORT);
    InetAddress  addrs= InetAddress.getLocalHost();

    System.out.println("TCP/Server running on : "+ addrs +" ,Port "+s.getLocalPort());

    try
    {
  while(true)
        {
     // Phong tỏa cho tới khi có một liên kết đến
        Socket socket = s.accept();
        try
        {
     new EchoServe(socket);  // Tạo một tuyến đoạn quản lý riêng từng liên kết
        } catch(IOException e) {

          socket.close();
    }
  }
}
finally {
      s.close();
```

```
    }
   }
  }
```

Chương trình phía client

```
import java.net.*;
import java.io.*;

public class TCPClient1
{
 public static void main(String[] args) throws IOException
 {
    if (args.length != 2)
    {
            System.out.println("Sử dụng: java TCPClient  hostid port#");
            System.exit(0);
    }

    try
    {
          InetAddress addr = InetAddress.getByName(args[0]);
          Socket socket = new Socket(addr, Integer.parseInt(args[1]));

 try
 {
     System.out.println("socket = " + socket);
     BufferedReader in = new BufferedReader(new InputStreamReader(
          socket.getInputStream()));

          // Output is automatically flushed by PrintWriter:

      PrintWriter out =new PrintWriter(new BufferedWriter(
           new OutputStreamWriter(socket.getOutputStream())),true);

           // Đọc dòng ký tự từ bàn phím

          DataInputStream myinput = new DataInputStream(new
BufferedInputStream(System.in));
          try
    {
```

```
                    for(;;)
                {
                    System.out.println("Type anything followed by RETURN, or Exit to terminate the program.");
                    String strin=myinput.readLine();

                    // Quit if the user typed ctrl+D
                    if (strin.equals("exit")) break;
                    else
                        out.println(strin);         // Send the message
                    String strout = in.readLine();      // Recive it back
                    if ( strin.length()==strout.length())
                    { // Compare Both Strings

                        System.out.println("Received: "+strout);
                    }
                    else
                        System.out.println("Echo bad -- string unequal"+ strout);
                } // of for ;;

                }
            catch (IOException e)
            {
                    e.printStackTrace();
                }
                    // User is exiting
            }
        finally
        {
                    System.out.println("EOF...exit");
                    socket.close();
            }
            }
        catch(UnknownHostException e)
        {
                    System.err.println("Can't find host");
                    System.exit(1);
            }
        catch (SocketException e)
```

```
        {
                System.err.println("Can't open socket");
                e.printStackTrace();
                System.exit(1);
            }
        }
    }
```

9. Kết luận

Chúng ta đã tìm hiểu cách lập trình mạng cho giao thức TCP. Các Socket còn được gọi là socket luồng vì để gửi và nhận dữ liệu đều được tiến hành thông qua việc đọc ghi các luồng. Ta đọc cũng đã tìm hiểu cơ chế hoạt động của socket và cách thức lập các chương trình server và client. Ngoài ra, chương này cũng đã giải thích tạo sao cần có cài đặt server đa tuyến đoạn và tìm hiểu cách thức để lập các chương trình client/server đa tuyến đoạn. Trong chương tiếp theo chúng ta sẽ học cách xây dựng một chương trình client/server cho giao thức UDP, một giao thức gần với giao thức TCP.

Chương 7

Lập trình ứng dụng cho giao thức UDP

1. Tổng quan về giao thức UDP

TCP/IP là một họ các giao thức được gọi là họ giao thức IP, bao gồm bốn tầng. Cần nhớ rằng TCP/IP không phải là một giao thức mà thực sự là một họ các giao thức, và bao gồm các giao thức mức thấp khác như IP, TCP, và UDP. UDP nằm ở tầng giao vận, phía trên giao thức IP. Tầng giao vận cung cấp khả năng truyền tin giữa các mạng thông qua các gateway. Nó sử dụng các địa chỉ IP để gửi các gói tin trên Internet hoặc trên mạng thông qua các trình điều khiển thiết bị khác nhau. TCP và UDP là một phần của họ giao thức TCP/IP; mỗi giao thức có những ưu và nhược điểm riêng của nó.

Giao thức UDP là giao thức đơn giản, phi liên kết và cung cấp dịch vụ trên tầng giao vận với tốc độ nhanh. Nó hỗ trợ liên kết một-nhiều và thường được sử dụng thường xuyên trong liên kết một-nhiều bằng cách sử dụng các datagram multicast và unicast.

Giao thức IP là giao thức cơ bản của Internet. TCP và UDP đều là hai giao thức tầng giao thức vận trên cơ sở của giao thức IP. Hình dưới đây chỉ ra cách ánh xạ mô hình OSI ánh xạ vào kiến trúc TCP/IP và họ giao thức TCP/IP.

<table>
<tr><td></td><td>Các tầng OSI</td><td>Họ giao thức TCP</td><td colspan="5">TCP/IP Stack</td></tr>
<tr><td>7</td><td>Tầng ứng dụng</td><td rowspan="3">Tầng ứng dụng</td><td rowspan="3">HTTP</td><td rowspan="3">FTP</td><td rowspan="3">SMTP</td><td rowspan="3">RIP</td><td rowspan="3">DNS</td></tr>
<tr><td>6</td><td>Tầng trình diễn</td></tr>
<tr><td>5</td><td>Tầng phiên</td></tr>
<tr><td>4</td><td>Tầng giao vận</td><td>Tầng giao vận</td><td colspan="3">TCP</td><td colspan="2">UDP</td></tr>
<tr><td>3</td><td>Tầng mạng</td><td>Tầng Internet</td><td colspan="5">ICMP,IP, IGMP</td></tr>
<tr><td>2</td><td>Tầng liên kết dữ liệu</td><td rowspan="2">Tầng mạng</td><td colspan="5" rowspan="2">Ethernet, ATM, Frame Relay,..</td></tr>
<tr><td>1</td><td>Tầng vật lý</td></tr>
</table>

Bảng 7.1

1.1. Một số thuật ngữ UDP

Trước khi kiểm tra xem giao thức UDP hoạt động như thế nào, chúng ta cần làm quen với một số thuật ngữ. Trong phần dưới đây, chúng ta sẽ định nghĩa một số thuật ngữ cơ bản có liên quan đến giao thức UDP.

- Packet

Trong truyền số liệu, một packet là một dãy các số nhị phân, biểu diễn dữ liệu và các tín hiệu điều khiển, các gói tin này được chuyển đi và chuyển tới tới host. Trong gói tin, thông tin được sắp xếp theo một khuôn dạng cụ thể.

- Datagram

Một datagram là một gói tin độc lập, tự chứa, mang đầy đủ dữ liệu để định tuyến từ nguồn tới đích mà không cần thông tin thêm.

- MTU

MTU là viết tắt của Maximum Transmission Unit. MTU là một đặc trưng của tầng liên kết mô tả số byte dữ liệu tối đa có thể truyền trong một gói tin. Mặt khác, MTU là gói dữ liệu lớn nhất mà môi trường mạng cho trước có thể truyền. Ví dụ, Ethernet có MTU cố định là 1500 byte. Trong UDP, nếu kích thước của một datagram lớn hơn MTU, IP sẽ thực hiện phân đoạn, chia datagram thành các phần nhỏ hơn (các đoạn), vì vậy mỗi đoạn nhỏ có kích thước nhỏ hơn MTU.

- Port

UDP sử dụng các cổng để ánh xạ dữ liệu đến vào một tiến trình cụ thể đang chạy trên một máy tính. UDP định đường đi cho packet tại vị trí xác định bằng cách sử dụng số hiệu cổng được xác định trong header của datagram. Các cổng được biểu diễn bởi các số 16-bit, vì thế các cổng nằm trong dải từ 0 đến 65535. Các cổng cũng được xem như là các điểm cuối của các liên kết logic, và được chia thành ba loại sau:

  - Các cổng phổ biến: Từ 0 đến 1023
  - Các cổng đã đăng ký: 1024 đến 49151
  - Các cổng động/dành riêng 49152 đến 65535

Chú ý rằng các cổng UDP có thể nhận nhiều hơn một thông điệp ở một thời điểm. Trong một số trường hợp, các dịch vụ TCP và UDP có thể sử dụng cùng một số hiệu cổng, như 7 (Echo) hoặc trên cổng 23 (Telnet).

UDP có các cổng thông dụng sau:

Cổng UDP	Mô tả
15	Netstat- Network Status-Tình trạng mạng
53	DNS-Domain Name Server
69	TFTP-Trivial File Transfer Protocol Giao thức truyền tệp thông thường
137	NetBIOS Name Service
138	Dịch vụ Datagram NetBIOS
161	SNMP

Bảng 7.2

- TTL (Time To Live)

Giá trị TTL cho phép chúng ta thiết lập một giới hạn trên của các router mà một datagram có thể đi qua. Giá trị TTL ngăn ngừa các gói tin khỏi bị kẹt trong các vòng lặp định tuyến vô hạn. TTL được khởi tạo bởi phía gửi và giá trị được giảm đi bởi mỗi router quản lý datagram. Khi TTL bằng 0, datagram bị loại bỏ.

- Multicasting

Multicasting là phương pháp dựa trên chuẩn có tính chất mở để phân phối các thông tin giống nhau đến nhiều người dùng. Multicasting là một đặc trưng chính của giao thức UDP. Multicasting cho phép chúng ta truyền tin theo kiểu một nhiều, ví dụ gửi tin hoặc thư điện tử tới nhiều người nhận, đài phát thanh trên Internet, hoặc các chương trình demo trực tuyến.

1.2. Hoạt động của giao thức UDP

Khi một ứng dụng dựa trên giao thức UDP gửi dữ liệu tới một host khác trên mạng, UDP thêm vào một header có độ dài 8 byte chứa các số hiệu cổng nguồn và đích, cùng với

tổng chiều dài dữ liệu và thông tin checksum. IP thêm vào header của riêng nó vào đầu mỗi datagram UDP để tạo lên một datagram IP:

1.3. Các nhược điểm của giao thức UDP

So với giao thức TCP, UDP có những nhược điểm sau:

- Thiếu các tín hiệu bắt tay. Trước khi gửi một đoạn, UDP không gửi các tín hiệu bắt tay giữa bên gửi và bên nhận. Vì thế phía gửi không có cách nào để biết datagram đã đến đích hay chưa. Do vậy, UDP không đảm bảo việc dữ liệu đã đến đích hay chưa.
- Sử dụng các phiên. Để TCP là hướng liên kết, các phiên được duy trì giữa các host. TCP sử dụng các chỉ số phiên (session ID) để duy trì các liên kết giữa hai host. UDP không hỗ trợ bất kỳ phiên nào do bản chất phi liên kết của nó.
- Độ tin cậy. UDP không đảm bảo rằng chỉ có một bản sao dữ liệu tới đích. Để gửi dữ liệu tới các hệ thống cuối, UDP phân chia dữ liệu thành các đoạn nhỏ. UDP không đảm bảo rằng các đoạn này sẽ đến đích đúng thứ tự như chúng đã được tạo ra ở nguồn. Ngược lại, TCP sử dụng các số thứ tự cùng với số hiệu cổng và các gói tin xác thực thường xuyên, điều này đảm bảo rằng các gói tin đến đích đúng thứ tự mà nó đã được tạo ra.
- Bảo mật. TCP có tính bảo mật cao hơn UDP. Trong nhiều tổ chức, firewall và router cấm các gói tin UDP, điều này là vì các hacker thường sử dụng các cổng UDP.
- Kiểm soát luồng. UDP không có kiểm soát luồng; kết quả là, một ứng dụng UDP được thiết kế tồi có thể làm giảm băng thông của mạng.

1.4. Các ưu điểm của UDP

- Không cần thiết lập liên kết. UDP là giao thức phi liên kết, vì thế không cần phải thiết lập liên kết. Vì UDP không sử dụng các tín hiệu handshaking, nên có thể tránh được thời gian trễ. Đó chính là lý do tại sao DNS thường sử dụng giao thức UDP hơn là TCP-DNS sẽ chậm hơn rất nhiều khi dùng TCP.
- Tốc độ. UDP nhanh hơn so với TCP. Bởi vì điều này, nhiều ứng dụng thường được cài đặt trên giao thức UDP hơn so với giao thức TCP.
- Hỗ trợ hình trạng (Topology). UDP hỗ trợ các liên kết 1-1, 1-n, ngược lại TCP chỉ hỗ trợ liên kết 1-1.
- Kích thước header. UDP chỉ có 8 byte header cho mỗi đoạn, ngược lại TCP cần các header 20 byte, vì vậy sử dụng băng thông ít hơn.

Bảng dưới đây tổng kết những sự kác nhau giữa hai giao thức TCP và UDP:

Các đặc trưng	UDP	TCP
Hướng liên kết	Không	Có
Sử dụng phiên	Không	Có
Độ tin cậy	Không	Có
Xác thực	Không	Có
Đánh thứ tự	Không	Có
Điều khiển luồng	Không	Có
Bảo mật	Ít	Nhiều hơn

Bảng 7.3

1.5. Khi nào thì nên sử dụng UDP

Rất nhiều ứng dụng trên Internet sử dụng UDP. Dựa trên các ưu và nhược điểm của UDP chúng ta có thể kết luận UDP có ích khi:

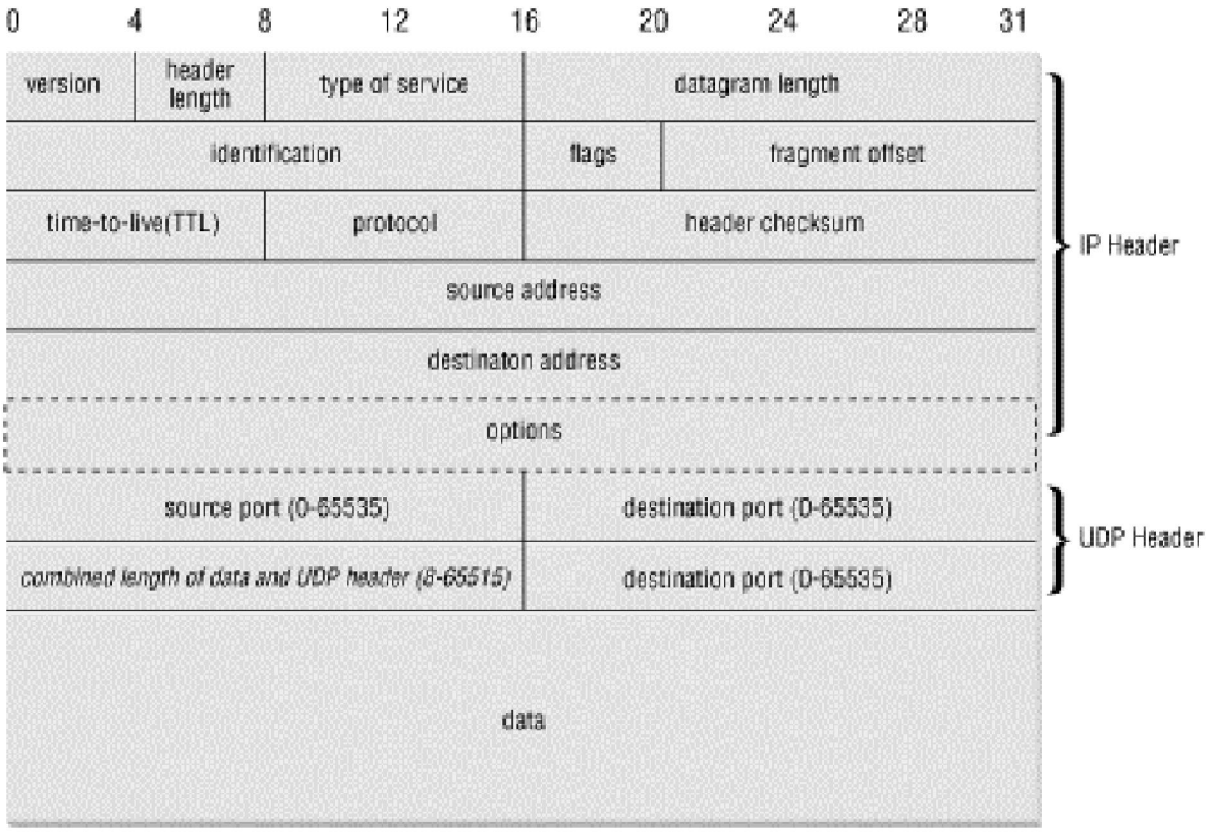
- Sử dụng cho các phương thức truyền broadcasting và multicasting khi chúng ta muốn truyền tin với nhiều host.
- Kích thước datagram nhỏ và trình tự đoạn là không quan trọng
- Không cần thiết lập liên kết
- Ứng dụng không gửi các dữ liệu quan trọng
- Không cần truyền lại các gói tin
- Băng thông của mạng đóng vai trò quan trọng

Việc cài đặt ứng dụng UDP trong Java cần có hai lớp là DatagramPacket và DatagramSocket. DatagramPacket đóng gói các byte dữ liệu vào các gói tin UDP được gọi là datagram và cho phép ta mở các datagram khi nhận được. Một DatagramSocket đồng thời thực hiện cả hai nhiệm vụ nhận và gửi gói tin. Để gửi dữ liệu, ta đặt dữ liệu trong một DatagramPacket và gửi gói tin bằng cách sử dụng DatagramSocket. Để nhận dữ liệu, ta nhận một đối tượng DatagramPacket từ DatagramSocket và sau đó đọc nội dung của gói tin.

UDP không có bất kỳ khái niệm nào về liên kết giữa hai host. Một socket gửi tất cả dữ liệu tới một cổng hoặc nhận tất cả dữ liệu từ một cổng mà không cần quan tâm host nào gửi. Một DatagramSocket có thể gửi dữ liệu tới nhiều host độc lập hoặc nhận dữ liệu từ nhiều host độc lập. Socket không dành riêng cho một liên kết cụ thể thể nào cả như trong giao thức TCP. Các socket TCP xem liên kết mạng như là một luồng: ta gửi và nhận dữ liệu với các luồng nhập và luồng xuất nhận được từ socket. UDP không cho phép điều này; ta phải làm việc với từng gói tin. Tất cả dữ liệu được đặt trong datagram được gửi đi dưới dạng một gói tin. Gói tin này cũng có thể nhận được bởi một nhóm hoặc cũng có thể bị mất. Một gói tin không nhất thiết phải liên quan đến gói tin tiếp theo. Cho trước hai gói tin, không có cách nào để biết được gói tin nào được gửi trước và gói tin nào được gửi sau.

2. Lớp DatagramPacket

Các datagram UDP đưa rất ít thông tin vào datagram IP. Header UDP chỉ đưa tám byte vào header IP. Header UDP bao gồm số hiệu cổng nguồn và đích, chiều dài của dữ liệu và header UDP, tiếp đến là một checksum tùy chọn. Vì mỗi cổng được biểu diễn bằng hai byte nên tổng số cổng UDP trên một host sẽ là 65536. Chiều dài cũng được biểu diễn bằng hai byte nên số byte trong datagram tối đa sẽ là 65536 trừ đi tám 8 byte dành cho phần thông tin header.



Trong Java, một datagram UDP được biểu diễn bởi lớp DatagramPacket:

- public final class DatagramPacket extends Object

Lớp này cung cấp các phương thức để nhận và thiết lập các địa chỉ nguồn, đích từ header IP, nhận và thiết lập các thông tin về cổng nguồn và đích, nhận và thiết lập độ dài dữ liệu. Các trường thông tin còn lại không thể truy nhập được từ mã Java thuần túy.

DatagramPacket sử dụng các constructor khác nhau tùy thuộc vào gói tin được sử dụng để gửi hay nhận dữ liệu.

2.1. Các constructor để nhận datagram

Hai constructor tạo ra các đối tượng DatagramSocket mới để nhận dữ liệu từ mạng:

- public DatagramPacket(byte[] b, int length)
- public DatagramPacket(byte[] b, int offset, int length)

Khi một socket nhận một datagram, nó lưu trữ phần dữ liệu của datagram ở trong vùng đệm b bắt đầu tại vị trí b[0] và tiếp tục cho tới khi gói tin được lưu trữ hoàn toàn hoặc cho tới khi lưu trữ hết length byte. Nếu sử dụng constructor thứ hai, thì dữ liệu được lưu trữ bắt đầu từ vị trí b[offset]. Chiều dài của b phải nhỏ hơn hoặc bằng b.length-offset. Nếu ta xây dựng một DatagramPacket có chiều dài vượt quá chiều dài của vùng đệm thì constructor sẽ đưa ra ngoại lệ IllegalArgumentException. Đây là kiểu ngoại lệ RuntimeException nên chương trình của ta không cần thiết phải đón bắt ngoại lệ này.

Ví dụ, xây dựng một DatagramPacket để nhận dữ liệu có kích thước lên tới 8912 byte

byte b[]=new byte[8912];

DatagramPacket dp=new DatagramPacket(b,b.length);

2.2. Constructor để gửi các datagram

Bốn constructor tạo các đối tượng DatagramPacket mới để gửi dữ liệu trên mạng:

- public DatagramPacket(byte[] b, int length, InetAddress dc, int port)
- public DatagramPacket(byte[] b, int offset, int length, InetAddress dc, int port)
- public DatagramPacket(byte[] b, int length, SocketAddress dc, int port)
- public DatagramPacket(byte[] b, int offset, int length, SocketAddress dc, int port)

Mỗi constructor tạo ra một DatagramPacket mới để được gửi đi tới một host khác. Gói tin được điền đầy dữ liệu với chiều dài là length byte bắt đầu từ vị trí offset hoặc vị trí 0 nếu offset không được sử dụng.

Ví dụ để gửi đi một xâu ký tự đến một host khác như sau:

```
String s="This is an example of UDP Programming";
byte[] b= s.getBytes();
try{
  InetAddress dc=InetAddress.getByName("www.vnn.vn");
  int port =7;
   DatagramPacket dp=new DatagramPacket(b,b.length,dc,port);
//Gửi gói tin
}
catch(IOException e){
     System.err.println(e);
}
```

Công việc khó khăn nhất trong việc tạo ra một đối tượng DatagramPacket chính là việc chuyển đổi dữ liệu thành một mảng byte. Đoạn mã trên chuyển đổi một xâu ký tự thành một mảng byte để gửi dữ liệu đi

2.3. Các phương thức nhận các thông tin từ DatagramPacket

DatagramPacket có sáu phương thức để tìm các phần khác nhau của một datagram: dữ liệu thực sự cộng với một số trường header. Các phương thức này thường được sử dụng cho các datagram nhận được từ mạng.

- public InetAddress getAddress()

Phương thức getAddress() trả về một đối tượng InetAddress chứa địa chỉ IP của host ở xa. Nếu datagram được nhận từ Internet, địa chỉ trả về chính là địa chỉ của máy đã gửi datagram (địa chỉ nguồn). Mặt khác nếu datagram được tạo cục bộ để được gửi tới máy ở xa, phương thức này trả về địa chỉ của host mà datagram được đánh địa chỉ.

- public int getPort()

Phương thức getPort() trả về một số nguyên xác định cổng trên host ở xa. Nếu datagram được nhận từ Internet thì cổng này là cổng trên host đã gửi gói tin đi.

- public SocketAddress()

Phương thức này trả về một đối tượng SocketAddress chứa địa chỉ IP và số hiệu cổng của host ở xa.

- public byte[] getData()

Phương thức getData() trả về một mảng byte chứa dữ liệu từ datagram. Thông thường cần phải chuyển các byte này thành một dạng dữ liệu khác trước khi chương trình xử lý dữ liệu. Một cách để thực hiện điều này là chuyển đổi mảng byte thành một đối tượng String sử dụng constructor sau đây:

- public String(byte[] buffer,String encoding)

Tham số đầu tiên, buffer, là mảng các byte chứa dữ liệu từ datagram. Tham số thứ hai cho biết cách thức mã hóa xâu ký tự. Cho trước một DatagramPacket dp được nhận từ mạng, ta có thể chuyển đổi nó thành xâu ký tự như sau:

String s=new String(dp.getData(),"ASCII");

Nếu datagram không chứa văn bản, việc chuyển đổi nó thành dữ liệu Java khó khăn hơn nhiều. Một cách tiếp cận là chuyển đổi mảng byte được trả về bởi phương thức getData() thành luồng ByteArrayInputStream bằng cách sử dụng constructor này:

- public ByteArrayInputStream(byte[] b, int offset, int length)

  b là mảng byte được sử dụng như là một luồng nhập InputStream

- public int getLength()

  Phương thức getLength() trả về số bytes dữ liệu có trong một datagram.

- public getOffset()

Phương thức này trả về vị trí trong mảng được trả về bởi phương thức getData() mà từ đó dữ liệu trong datagram xuất phát.

Các phương thức thiết lập giá trị cho các trường thông tin

Sáu constructor ở trên là đủ để tạo lập ra các datagram. Tuy nhiên, Java cung cấp một số phương thức để thay đổi dữ liệu, địa chỉ của máy ở xa, và cổng trên máy ở xa sau khi datagram đã được tạo ra. Trong một số trường hợp việc sử dụng lại các DatagramPacket đã có sẵn sẽ nhanh hơn việc tạo mới các đối tượng này.

- *public void setData(byte[] b)*: Phương thức này thay đổi dữ liệu của datagram
- *public void setData(byte[] b, int offset, int length)*

Phương thức này đưa ra giải pháp để gửi một khối lượng dữ liệu lớn. Thay vì gửi toàn bộ dữ liệu trong mảng, ta có thể gửi dữ liệu trong từng đoạn của mảng tại mỗi thời điểm.

Ví dụ đoạn mã sau đây sẽ gửi dữ liệu theo từng đoạn 512 byte:

```
int offset=0;
DatagramPacket dp=new DatagramPacket(b,offset,512);
int bytesSent=0;
while(bytesSent<b.length)
{
 ds.send(dp);
 bytesSent+=dp.getLength();
 int bytesToSend=b.length-bytesSent;
 int size=(bytesToSend>512):512:bytesToSend;
 dp.setData(b,byteSent,512);
}
```

- public void setAddress(InetAddress dc)

Phương thức setAddress() thay đổi địa chỉ của máy mà ta sẽ gửi gói tin tới. Điều này sẽ cho phép ta gửi cùng một datagram đến nhiều nơi nhận.

- public void setPort(int port)

Phương thức này thay đổi số hiệu cổng gửi tới của gói tin.

- pubic void setAddress(SocketAddress sa)
- public void setLength(int length)

Phương thức này thay đổi số byte dữ liệu có thể đặt trong vùng đệm.

3. Lớp DatagramSocket

Để gửi hoặc nhận một DatagramPacket, bạn phải mở một DatagramSocket. Trong Java, một datagram socket được tạo ra và được truy xuất thông qua đối tượng DatagramSocket

public class DatagramSocket extends Object

Tất cả các datagram được gắn với một cổng cục bộ, cổng này được sử dụng để lắng nghe các datagram đến hoặc được đặt trên các header của các datagram sẽ gửi đi. Nếu ta viết một client thì không cần phải quan tâm đến số hiệu cổng cục bộ là bao nhiêu

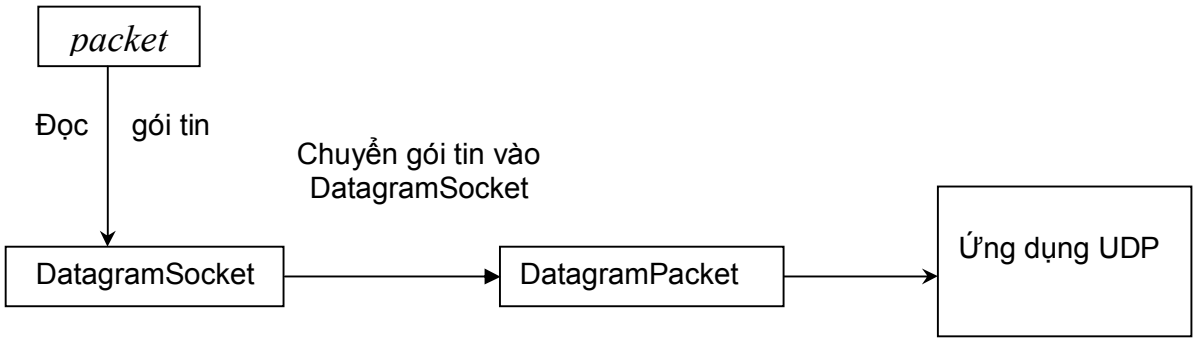
DatagramSocket được sử dụng để gửi và nhận các gói tin UDP. Nó cung cấp các phương thức để gửi và nhận các gói tin, cũng như xác định một giá trị timeout khi sử dụng phương pháp vào ra không phong tỏa (non blocking I/O), kiểm tra và sửa đổi kích thước tối đa của gói tin UDP, đóng socket.

Các phương thức

- void close(): đóng một liên kết và giải phóng nó khỏi cổng cục bộ.
- void connect(InetAddress remote_address, int remote_port)-
- InetAddress getInetAddress():phương thức này trả về địa chỉ remote mà socket kết nối tới, hoặc giá trị null nếu không tồn tại liên kết.
- InetAddress getLocalAddress(): trả về địa chỉ cục bộ
- Int getSoTimeOut() trả về giá trị tùy chọn timeout của socket. Giá trị này xác định thời gian mà thao tác đọc sẽ phong tỏa trước khi nó đưa ra ngoại lệ InterruptedException. Ở chế độ mặc định, giá trị này bằng 0, chỉ ra rằng vào ra không phong tỏa được sử dụng.
- void receive(DatagramPacket dp) throws IOException:phương thức đọc một gói tin UDP và lưu nộ dung trong packet xác định.
- void send(DatagramSocket dp) throws IOException:phương thức gửi một gói tin
- void setSoTimeOut(int timeout): thiết lập giá trị tùy chọn của socket.

4. Nhận các gói tin

Trước khi một ứng dụng có thể đọc các gói tin UDP được gửi bởi các máy ở xa, nó phải gán một socket với một cổng UDP bằng cách sử dụng DatagramSocket, và tạo ra một DatagramPacket sẽ đóng vai trò như là một bộ chứa cho dữ liệu của gói tin UDP. Hình vẽ dưới đây chỉ ra mối quan hệ giữa một gói tin UDP với các lớp Java khác nhau được sử dụng để xử lý nó và các ứng dụng thực tế.

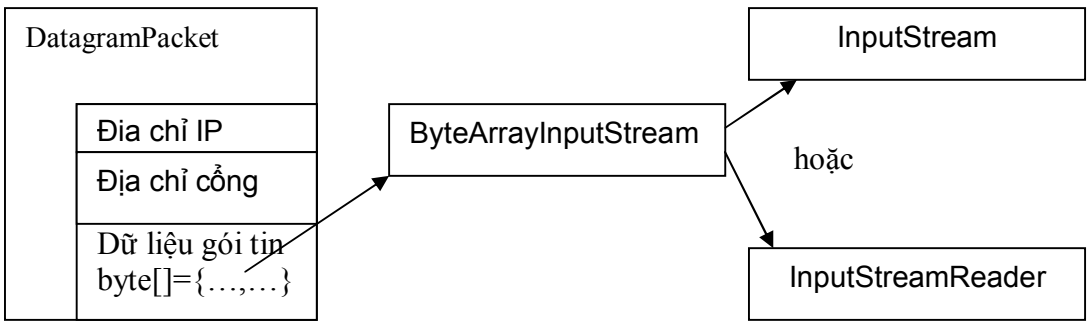


Hình 7.1

Khi một ứng dụng muốn đọc các gói tin UDP, nó gọi phương thức DatagramSocket.receive(), phương thức này sao chép gói tin UDP vào một DatagramPacket xác định. Xử lý nội dung nói tin và tiến trình lặp lại khi cần

```
DatagramPacket dp=new DatagramPacket(new byte[256],256);
DatagramSocket ds=new DatagramSocket(2000);
boolean finished=false;
while(!finished)
{
ds.receive(dp);
//Xử lý gói tin
}
ds.close();
```

Khi xử lý gói tin ứng dụng phải làm việc trực tiếp với một mảng byte. Tuy nhiên nếu ứng dụng là đọc văn bản thì ta có thể sử dụng các lớp từ gói vào ra để chuyển đổi giữa mảng byte và luồng stream và reader. Bằng cách gắn kết luồng nhập ByteArrayInputStream với nội dung của một datagram và sau đó kết nối với một kiểu luồng khác, khi đó bạn có thể truy xuất tới nội dung của gói UDP một cách dễ dàng. Rất nhiều người lập trình thích dùng các luồng vào ra I/O để xử lý dữ liệu, bằng cách sử dụng luồng DataInputStream hoặc BufferedReader để truy xuất tới nội dung của các mảng byte.



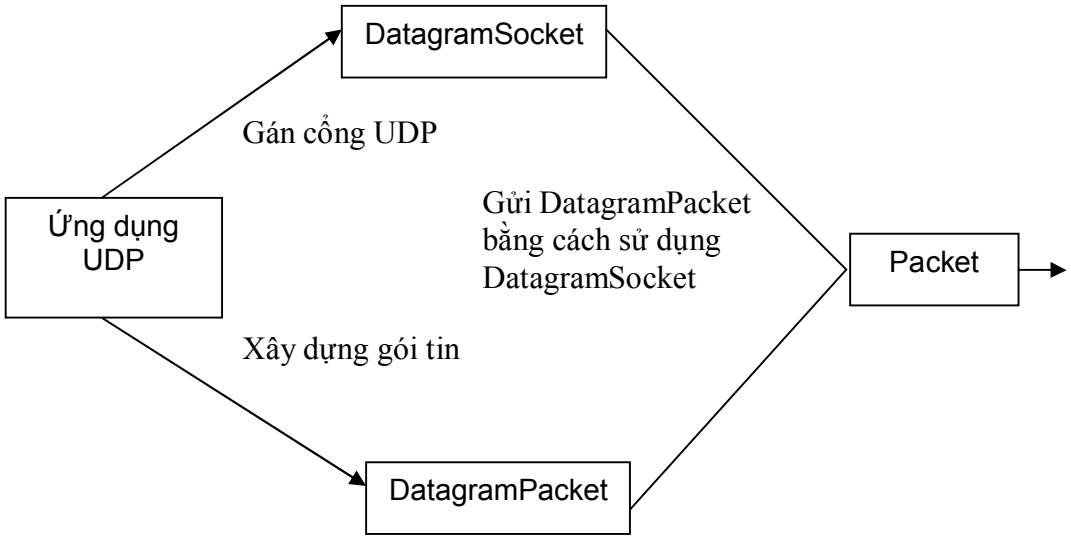
Hình 7.2

Ví dụ, để gắn kết một luồng DataInputStream với nội dung của một DatagramPacket, ta sử dụng đoạn mã sau:

```
ByteArrayInputStream bis=new ByteArrayInputStream(dp.getData());
DataInputStream dis=new DataInputStream(bis);
//đọc nội dung của gói tin UDP
```

5. Gửi các gói tin

Lớp DatagramSocket cũng được sử dụng để gửi các gói tin. Khi gửi gói tin, ứng dụng phải tạo ra một DatagramPacket, thiết lập địa chỉ và thông tin cổng, và ghi dữ liệu cần truyền vào mảng byte. Nếu muốn gửi thông tin phúc đáp thì ta cũng đã biết địa chỉ và số hiệu cổng của gói tin nhận được. Mỗi khi gói tin sẵn sàng để gửi, ta sử dụng phương thức send() của lớp DatagramSocket để gửi gói tin đi.



Hình 7.3

```
//Socket lắng nghe các gói tin đến trên cổng 2000
DatagramSocket socket = new DatagramSocket(2000);
DatagramPacket packet = new DatagramPacket (new byte[256], 256);
packet.setAddress ( InetAddress.getByName ( somehost ) );
packet.setPort ( 2000 );
boolean finished = false;
while !finished )
{
// Ghi dữ liệu vào vùng đệm buffer
```

```
.....
socket.send (packet);
// Thực hiện hành động nào đó, chẳng hạn như đọc gói tin khácі hoặc kiểm tra xemor
// còn gói tin nào cần gửi đi hay không
.....
}
socket.close();
```

6. Ví dụ minh họa giao thức UDP

Để minh họa các gói tin UDP được gửi và nhận như thế nào, chúng ta sẽ viết, biên dịch và chạy ứng dụng sau.

Viết chương trình theo mô hình Client/Server để:

Client thực hiện các thao tác sau đây:

- Client gửi một xâu ký tự do người dùng nhập từ bàn phím cho server
- Client nhận thông tin phản hồi trở lại từ Server và hiển thị thông tin đó trên màn hình

Server thực hiện các thao tác sau:

- Server nhận xâu ký tự do client gửi tới và in lên màn hình
- Server biến đổi xâu ký tự thành chữ hoa và gửi trở lại cho Client

```
import java.net.*;
import java.io.*;
public class UDPClient
{
    public final static int CONG_MAC_DINH=9;

        public static void main(String args[])
        {
                String hostname;
                int port=CONG_MAC_DINH;
                if(args.length>0)
                {
                        hostname=args[0];
                        try{
                        }
                        catch(Exception e){
                                port =Integer.parseInt(args[1]);
                        }
```

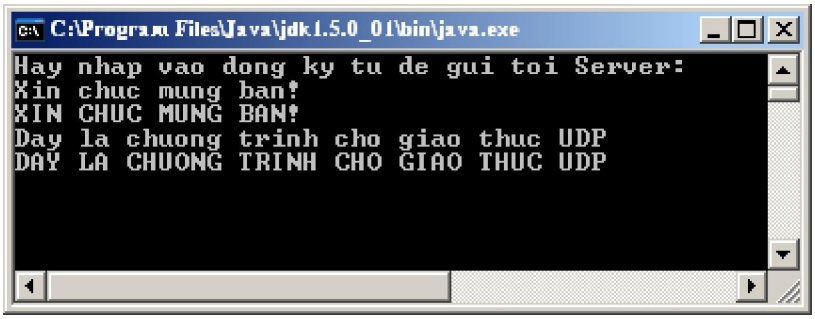
```
                }
                else
                {
                        hostname="127.0.0.1";
                }
                try{
                        InetAddress dc=InetAddress.getByName(hostname);
                        BufferedReader userInput=new BufferedReader(new
InputStreamReader(System.in));
                        DatagramSocket ds =new DatagramSocket(port);
                        while(true){
                                String line=userInput.readLine();
                                if(line.equals("exit"))break;
                                byte[] data=line.getBytes();
                                DatagramPacket dp=new
DatagramPacket(data,data.length,dc,port);
                                ds.send(dp);
                                dp.setLength(65507);
                                ds.receive(dp);
                                ByteArrayInputStream bis =new
ByteArrayInputStream(dp.getData());
                                BufferedReader dis =new BufferedReader(new
InputStreamReader(bis));
                                System.out.println(dis.readLine());

                        }
                }
                catch(UnknownHostException e)
                {
                        System.err.println(e);
                }
                catch(IOException e)
                {
                        System.err.println(e);
                }
        }
```

```
}


import java.net.*;
import java.io.*;

public class UDPServer
{
    public final static int CONG_MAC_DINH=9;

        public static void main(String args[])
        {

            int port=CONG_MAC_DINH;

                try{
                }
                catch(Exception e){
                        port =Integer.parseInt(args[1]);
                }

            try{

                        DatagramSocket ds =new DatagramSocket(port);
                        DatagramPacket dp=new DatagramPacket(new
byte[65507],65507);

                while(true){
                        ds.receive(dp);
                        ByteArrayInputStream bis =new
ByteArrayInputStream(dp.getData());
                        BufferedReader dis =new BufferedReader(new
InputStreamReader(bis));
                        String s=dis.readLine();
                        System.out.println(s);
                        s.toUpperCase();
                        dp.setData(s.getBytes());
```

```
                    dp.setLength(s.length());
                    dp.setAddress(dp.getAddress());
                    dp.setPort(dp.getPort());
                    ds.send(dp);
                }
            }
            catch(UnknownHostException e)
            {
                System.err.println(e);
            }
            catch(IOException e)
            {
                System.err.println(e);
            }
        }

    }
```

C:\>start java UDPServer

C:\>start java UDPClient



Hình 7.4

Chương trình Client/Server sử dụng đa tuyến đoạn

```
import java.net.*;
import java.io.*;
public abstract class UDPServer extends Thread
{
    private int bufferSize;
    protected DatagramSocket ds;
```

```
	public UDPServer(int port, int bufferSize) throws SocketException
	{
		this.bufferSize=bufferSize;
		this.ds=new DatagramSocket(port);
	}
	public UDPServer(int port)throws SocketException
	{
		this(port,8192);
	}
	public void run()
	{
		byte[] buffer=new byte[bufferSize];
		while(true)
		{
			DatagramPacket dp=new DatagramPacket(buffer,buffer.length);
			try{
			ds.receive(dp);
			this.respond(dp);
			}
			catch(IOException e)
			{
				System.err.println(e);
			}
		}
	}
	public abstract void respond(DatagramPacket req);

}
```

Server Echo

```
import java.net.*;
import java.io.*;

public class UDPEchoServer extends UDPServer
{
```

```
        public final static int DEFAULT_PORT=7;

        public UDPEchoServer()throws SocketException
        {
                super(DEFAULT_PORT);
        }
        public void respond(DatagramPacket dp)
        {
                try{
                        DatagramPacket outdp=new
DatagramPacket(dp.getData(),dp.getLength(),dp.getAddress(),dp.getPort());
                        ds.send(outdp);
                }
                catch(IOException e)
                {
                        System.err.println(e);
                }
        }
        public static void main(String[] args)
        {
                try
                {
                        UDPServer server=new UDPEchoServer();
                        server.start();
                        System.out.println("Server dang da san sang lang nghe lien ket...");

                }
                catch(SocketException e)
                {
                        System.err.println(e);
                }
        }
}
```

Client

```
import java.net.*;
import java.io.*;
```

```
public class  ReceiverThread extends Thread
{
        private DatagramSocket ds;
        private boolean stopped=false;

        public ReceiverThread(DatagramSocket ds) throws SocketException
        {
                this.ds=ds;
        }
        public void halt(){
                this.stopped=true;
        }

        public void run()
        {
                byte buffer[]=new byte[65507];

                while(true)
                {
                        if(stopped) return;
                        DatagramPacket dp=new DatagramPacket(buffer,buffer.length);
                        try{
                                ds.receive(dp);
                                String s=new String(dp.getData(),0,dp.getLength());
                                System.out.println(s);
                                Thread.yield();
                        }
                        catch(IOException e)
                        {
                                System.err.println(e);
                        }
                }
        }
}
import java.net.*;
import java.io.*;
```

```
public class  SenderThread extends Thread
{
        private InetAddress server;
        private DatagramSocket ds;
        private boolean stopped=false;
        private int port;
        public SenderThread(InetAddress address, int port) throws SocketException
        {
                this.server=address;
                this.port=port;
                this.ds=new DatagramSocket();
                this.ds.connect(server,port);
        }
        public void halt(){
                this.stopped=true;
        }
        public DatagramSocket getSocket()
        {
                return this.ds;
        }
        public void run()
        {
                try{
                        BufferedReader userInput=new BufferedReader(new
InputStreamReader(System.in));
                        while(true)
                        {
                                if(stopped) return;
                                String line=userInput.readLine();
                                if(line.equals("exit"))break;
                                byte[] data=line.getBytes();
                                DatagramPacket dp=new
DatagramPacket(data,data.length,server,port);
                                ds.send(dp);
                                Thread.yield();
                        }
                }
```

```
                catch(IOException e)
                {
                        System.err.println(e);
                }
        }

}
```

Client Echo

```
import java.net.*;
import java.io.*;

public class  UDPEchoClient
{
        public final static int DEFAULT_PORT=7;
        public static void main(String[] args)
        {
                String hostname="localhost";
                int port= DEFAULT_PORT;
                if(args.length>0)
                {
                        hostname=args[0];
                }
                try{
                        InetAddress ia=InetAddress.getByName(args[0]);
                        SenderThread sender=new SenderThread(ia,DEFAULT_PORT);
                        sender.start();
                        ReceiverThread receiver=new ReceiverThread(sender.getSocket());
                        receiver.start();
                }
                catch(UnknownHostException e)
                {
                        System.err.println(e);
                }
                catch(SocketException e)
                {
```

```
                System.err.println(e);
            }
        }
}
```

7. Kết luận

Trong chương này, chúng ta đã thảo luận những khái niệm căn bản về giao thức UDP và so sánh nó với giao thức TCP. Chúng ta đã đề cập tới việc cài đặt các chương trình UDP trong Java bằng cách sử dụng hai lớp DatagramPacket và DatagramSocket. Một số chương trình mẫu cũng được giới thiệu để bạn đọc tham khảo và giúp hiểu sâu hơn về các vấn đề lý thuyết.

Chương 8

Phân tán đối tượng trong Java bằng RMI

1. Tổng quan

RMI là một cơ chế cho phép một đối tượng đang chạy trên một máy ảo Java này (Java Virtual Machine) gọi các phương thức của một đối tượng đang tồn tại trên một máy ảo Java khác (JVM).

Thực chất RMI là một cơ chế gọi phương thức từ xa đã được thực hiện và tích hợp trong ngôn ngữ Java. Vì Java là một ngôn ngữ lập trình hướng đối tượng, nên phương pháp lập trình trong RMI là phương pháp hướng đối tượng do đó các thao tác hay các lời gọi phương thức đều liên quan đến đối tượng. Ngoài ra, RMI còn cho phép một Client có thể gửi tới một đối tượng đến cho Server xử lý, và đối tượng này cũng có thể được xem là tham số cho lời gọi hàm từ xa, đối tượng này cũng có những dữ liệu bên trong và các hành vi như một đối tượng thực sự.

So sánh giữ gọi phương thức từ xa với các lời gọi thủ tục từ xa

Gọi phương thức từ xa không phải là một khái niệm mới. Thậm chí trước khi ra đời lập trình hướng đối tượng phần mềm đã có thể gọi các hàm và các thủ tục từ xa. Các hệ thống như RPC đã được sử dụng trong nhiều năm và hiện nay vẫn được sử dụng.

Trước hết, Java là một ngôn ngữ độc lập với nền và cho phép các ứng dụng Java truyền tin với các ứng dụng Java đang chạy trên bất kỳ phần cứng và hệ điều hành nào có hỗ trợ JVM. Sự khác biệt chính giữa hai mục tiêu là RPC hỗ trợ đa ngôn ngữ, ngược lại RMI chỉ hỗ trợ các ứng dụng được viết bằng Java.

Ngoài vấn đề về ngôn ngữ và hệ thống, có một số sự khác biệt căn bản giữa RPC và RMI. Gọi phương thức từ xa làm việc với các đối tượng, cho phép các phương thức chấp nhận và trả về các đối tượng Java cũng như các kiểu dữ liệu nguyên tố (premitive type). Ngược lại gọi thủ tục từ xa không hỗ trợ khái niệm đối tượng. Các thông điệp gửi cho một dịch vụ RPC (Remote Procedure Calling) được biểu diễn bởi ngôn ngữ XDR (External Data Representation): dạng thức biểu diễn dữ liệu ngoài. Chỉ có các kiểu dữ liệu có thể được định nghĩa bởi XDR mới có thể truyền đi.

2. Mục đích của RMI

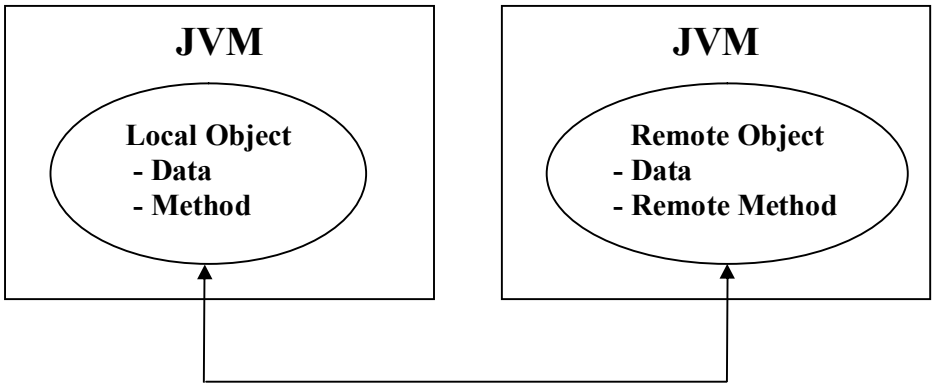
- Hỗ trợ gọi phương thức từ xa trên các đối tượng trong các máy ảo khác nhau
- Hỗ trợ gọi ngược phương thức ngược từ server tới các applet
- Tích hợp mô hình đối tượng phân tán vào ngôn ngữ lập trình Java theo một cách tự nhiên trong khi vẫn duy trì các ngữ cảnh đối tượng của ngôn ngữ lập trình Java
- Làm cho sự khác biệt giữa mô hình đối tượng phân tán và mô hình đối tượng cục bộ không có sự khác biệt.
- Tạo ra các ứng dụng phân tán có độ tin cậy một cách dễ dàng
- Duy trì sự an toàn kiểu được cung cấp bởi môi trường thời gian chạy của nền tảng Java
- Hỗ trợ các ngữ cảnh tham chiếu khác nhau cho các đối tượng từ xa
- Duy trì môi trường an toàn của Java bằng các trình bảo an và các trình nạp lớp.

3. Một số thuật ngữ

Cũng như tất cả các chương trình khác trong Java, chương trình RMI cũng được xây dựng bởi các giao tiếp và lớp. Giao tiếp định nghĩa các phương thức và các lớp thực thi các phương thức đó. Ngoài ra lớp còn thực hiện một vài phương thức khác. Nhưng chỉ có những phương thức khai báo trong giao tiếp thừa kế từ giao tiếp Remote hoặc các

lớp con của nó mới được Client gọi từ JVM khác. Trong mục này ta nêu một số thuật ngữ thường xuyên được sử dụng trong phần này:

- Giao tiếp Remote: Một giao tiếp khai báo các phương thức cho phép gọi từ xa. Trong Java giao tiếp Remote có các đặc điểm sau:
  - Thừa kế giao tiếp có sẵn: java.rmi.Remote.
  - Mỗi phương thức trong giao tiếp Remote phải được khai báo để đưa ra ngoại lệ RemoteException nhờ mệnh đề throws java.rmi.RemoteException và có thể có các ngoại lệ khác.
- Đối tượng Remote: một đối tượng được tạo ra để cho phép những đối tượng khác trên một máy JVM khác gọi tới nó.
- Phương thức Remote: Đối tượng Remote chứa một số các phương thức, những phương thức này có thể được gọi từ xa bởi các đối tượng trong JVM khác .



Hình 8.1

4. Các lớp trung gian Stub và Skeleton

Trong kỹ thuật lập trình phân tán RMI, để các đối tượng trên các máy Java ảo khác nhau có thể truyền tin với nhau thông qua các lớp trung gian: Stub và Skeleton.

Vai trò của lớp trung gian: Lớp trung gian tồn tại cả ở hai phía client (nơi gọi phương thức của các đối tượng ở xa) và server (nơi đối tượng thật sự được cài đặt để thực thi mã lệnh của phương thức). Trong Java trình biên dịch rmic.exe được sử dụng để tạo ra lớp trung gian này. Phía client lớp trung gian này gọi là Stub (lớp móc), phía server lớp trung gian này gọi là Skeleton(lớp nối) chúng giống như các lớp môi giới giúp các lớp ở xa truyền tin với nhau.

5. Cơ chế hoạt động của RMI

Các hệ thống RMI phục vụ cho việc truyền tin thường được chia thành hai loại: client và server. Một server cung cấp dịch vụ RMI, và client gọi các phương thức trên đối tượng của dịch vụ này.

Server RMI phải đăng ký với một dịch vụ tra tìm và đăng ký tên. Dịch vụ này cho phép các client truy tìm chúng, hoặc chúng có thể tham chiếu tới dịch vụ trong một mô hình khác. Một chương trình đóng vai trò như vậy có tên là rmiregistry, chương trình này chạy như một tiến trình độc lập và cho phép các ứng dụng đăng ký dịch vụ RMI hoặc nhận một tham chiếu tới dịch vụ được đặt tên. Mỗi khi server đươc đăng ký, nó sẽ chờ các yêu cầu RMI từ các client. Gắn với mỗi đăng ký dịch vụ là một tên được biểu diễn bằng một xâu ký tự để cho phép các client lựa chọn dịch vụ thích hợp. Nếu một dịch vụ chuyển từ server này sang một server khác, client chỉ cần tra tìm trình đăng ký để tìm ra vị trí mới. Điều này làm cho hệ thống có khả năng dung thứ lỗi-nếu một dịch vụ không khả dụng do một máy bị sập, người quản trị hệ thống có thể tạo ra một thể hiện mới của dịch vụ trên một hệ thống khác và đăng ký nó với trình đăng ký RMI.

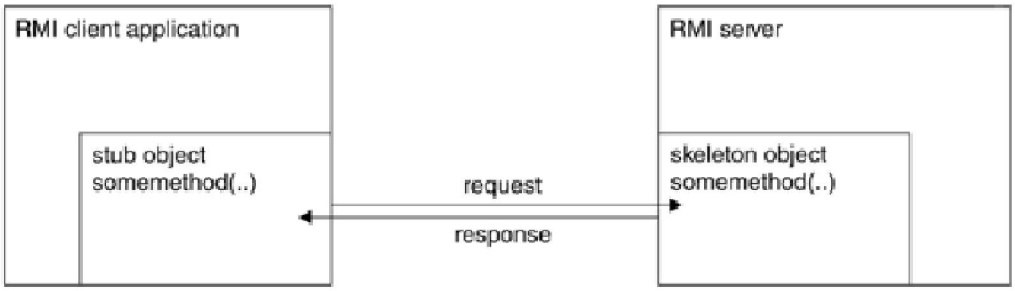
Các client RMI sẽ gửi các thông điệp RMI để gọi một phương thức trên một đối tượng từ xa. Trước khi thực hiện gọi phương thức từ xa, client phải nhận được một tham chiếu từ xa. Tham chiếu này thường có được bằng cách tra tìm một dịch vụ trong trình đăng ký RMI. Ứng dụng client yêu cầu một tên dịch vụ cụ thể, và nhận một URL trỏ tới tài nguyên từ xa. Khuôn dạng dưới đây được sử dụng để biểu diễn một tham chiếu đối tượng từ xa:

rmi://hostname:port/servicename

Trong đó *hostname* là tên của máy chủ hoặc một địa chỉ IP, *port* xác định dịch vụ, và *servicename* là một xâu ký tự mô tả dịch vụ.

Mỗi khi có được một tham chiếu, client có thể tương tác với dịch vụ từ xa. Các chi tiết liên quan đến mạng hoàn toàn được che dấu đối với những người phát triển ứng dụng-làm việc với các đối tượng từ xa đơn giản như làm việc với các đối tượng cục bộ. Điều này có thể có được thông qua sự phân chia hệ thống RMI thành hai thành phần, stub và skeleton.

Đối tượng stub là một đối tượng ủy quyền, truyền tải yêu cầu đối tượng tới server RMI. Cần nhớ rằng mỗi dịch vụ RMI được định nghĩa như là một giao tiếp, chứ không phải là một chương trình cài đặt, các ứng dụng client giống như các chương trình hướng đối tượng khác. Tuy nhiên ngoài việc thực hiện công việc của chính nó, stub còn truyền một thông điệp tới một dịch vụ RMI ở xa, chờ đáp ứng, và trả về đáp ứng cho phương thức gọi. Người phát triển ứng dụng không cần quan tâm đến tài nguyên RMI nằm ở đâu, nó đang chạy trên nền nào, nó đáp ứng đầy đủ yêu cầu như thế nào. Client RMI đơn giản gọi một phương thức trên đối tượng ủy quyền, đối tượng này quản lý tất cả các chi tiết cài đặt.

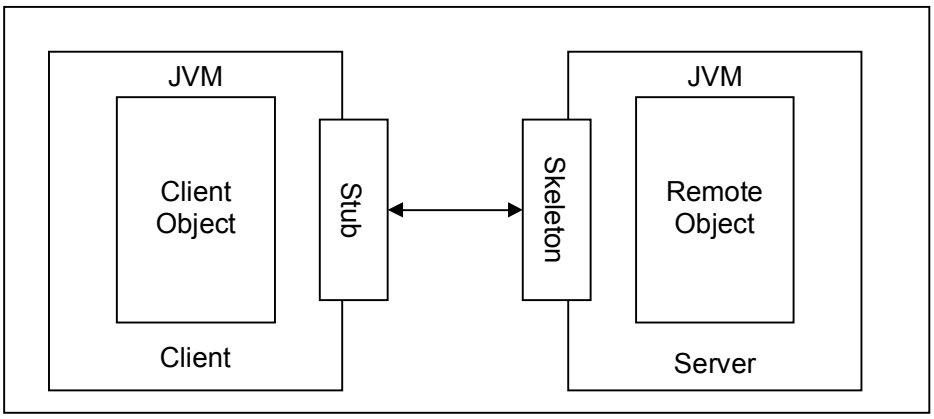


Hình 8.2

Tại phía server, đối tượng skeleton có nhiệm vụ lắng nghe các yêu cầu RMI đến và truyền các yêu cầu này tới dịch vụ RMI. Đối tượng skeleton không cung cấp bản cài đặt của dịch vụ RMI. Nó chỉ đóng vai trò như là chương trình nhận các yêu cầu, và truyền các yêu cầu. Sau khi người phát triển tạo ra một giao tiếp RMI, thì anh ta phải cung cấp một phiên bản cài đặt cụ thể của giao tiếp. Đối tượng cài đặt này được gọi là đối tượng skeleton, đối tượng này gọi phương thức tương ứng và truyền các kết quả cho đối tượng stub trong client RMI. Mô hình này làm cho việc lập trình trở nên đơn giản, vì skeleton được tách biệt với cài đặt thực tế của dịch vụ. Tất cả những gì mà người phát triển dịch vụ cần quan tâm là mã lệnh khởi tạo (để đăng ký dịch vụ và chấp nhận dịch vụ), và cung cấp chương trình cài đặt của giao tiếp dịch vụ RMI.

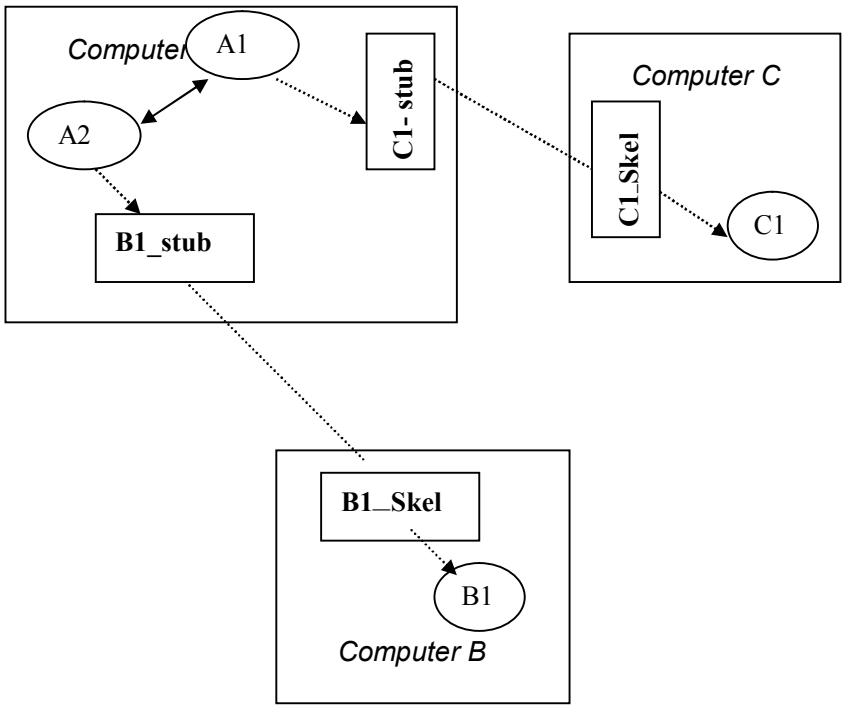
Với câu hỏi các thông điệp được truyền như thế nào, câu trả lời tương đối đơn giản. Việc truyền tin diễn ra giữa các đối tượng stub và skeleton bằng cách sử dụng các socket TCP. Mỗi khi được tạo ra, skeleton lắng nghe các yêu cầu đến được phát ra bởi các đối tượng stub. Các tham số trong hệ thống RMI không chỉ hạn chế đối với các kiểu dữ liệu nguyên tố-bất kỳ đối tượng nào có khả năng tuần tự hóa đều có thể được truyền như một tham số hoặc được trả về từ phương thức từ xa. Khi một stub truyền một yêu cầu tới một đối tượng skeleton, nó phải đóng gói các tham số (hoặc là các kiểu dữ liệu nguyên tố, các đối tượng hoặc cả hai) để truyền đi, quá trình này được gọi là marshalling. Tại phía skeleton các tham số được khôi phục lại để tạo nên các kiểu dữ liệu nguyên tố và các đối tượng, quá trình này còn được gọi là unmarshaling. Để thực hiện nhiệm vụ

này, các lớp con của các *lớp ObjectOutputStream* và *ObjectInputStream* được sử dụng để đọc và ghi nội dung của các đối tượng.



Hình 8.3

Sơ đồ gọi phương thức của các đối tượng ở xa thông qua lớp trung gian được cụ thể hoá như sau:



Hình 8.4

- Ta có đối tượng C1 được cài đặt trên máy C. Trình biên dịch rmic.exe sẽ tạo ra hai lớp trung gian C1_Skel và C1_Stub. Lớp C1_Stub sẽ được đem về máy A. Khi A1 trên máy A gọi C1 nó sẽ chuyển lời gọi đến lớp C1_Stub, C1_Stub chịu trách nhiệm đóng gói tham số, chuyển vào không gian địa chỉ tương thích với đối tượng C1 sau đó gọi phương thức tương ứng.
- Nếu có phương thức của đối tượng C1 trả về sẽ được lớp C1_Skel đóng gói trả ngược về cho C1_Stub chuyển giao kết quả cuối cùng lại cho A1. Nếu khi kết nối mạng gặp sự cố thì lớp trung gian Stub sẽ thông báo lỗi đến đối tượng A1. Theo cơ chế này A1 luôn nghĩ rằng nó đang hoạt động trực tiếp với đối tượng C1 trên máy cục bộ.
- Trên thực tế, C1_Stub trên máy A chỉ làm lớp trung gian chuyển đổi tham số và thực hiện các giao thức mạng, nó không phải là hình ảnh của đối tượng C1. Để

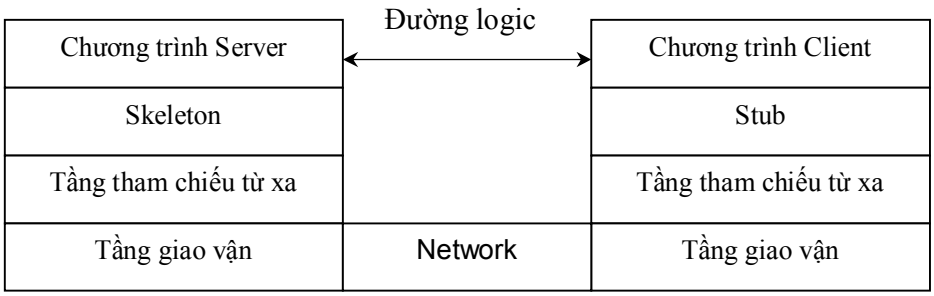
làm được điều này, đối tượng C1 cần cung cấp một giao diện tương ứng với các phương thức cho phép đối tượng A1 gọi nó trên máy A.

6. Kiến trúc RMI

Sự khác biệt căn bản giữa các đối tượng từ xa và các đối tượng cục bộ là các đối tượng từ xa nằm trên một máy ảo khác. Thông thường, các tham số đối tượng được truyền cho các phương thức và các giá trị đối tượng được trả về từ các phương thức thông qua cách truyền theo tham chiếu. Tuy nhiên cách này không làm việc khi các phương thức gọi và các phương thức được gọi không cùng nằm trên một máy ảo.

Vì vậy, có ba cơ chế khác nhau được sử dụng để truyền các tham số cho các phương thức từ xa và nhận các kết quả trả về từ các phương thức ở xa. Các kiểu nguyên tố (int, boolean, double,…) được truyền theo tham trị. Các tham chiếu tới các đối tượng từ xa được truyền dưới dạng các tham chiếu cho phép tất cả phía nhận gọi các phương thức trên các đối tượng từ xa. Các đối tượng không thực thi giao tiếp từ xa (nghĩa là các đối tượng không thực thi giao tiếp Remote) được truyền theo tham trị; nghĩa là các bản sao đầy đủ được truyền đi bằng cách sử dụng cơ chế tuần tự hóa đối tượng. Các đối tượng không có khả năng tuần tự hóa thì không thể được truyền đi tới các phương thức ở xa. Các đối tượng ở xa chạy trên server nhưng có thể được gọi bởi các đối tượng đang chạy trên client. Các đối tượng không phải ở xa, các đối tượng khả tuần tự chạy trên các hệ thống client.

Để quá trình truyền tin là trong suốt với người lập trình, truyền tin giữa client và server được cài đặt theo mô hình phân tầng như hình vẽ dưới đây



Hình 8.5

Đối với người lập trình, client dường như truyền tin trực tiếp với server. Thực tế, chương trình client chỉ truyền tin với đối tượng stub là đối tượng ủy quyền của đối tượng thực sự nằm trên hệ thống từ xa. Stub chuyển cuộc đàm thoại cho tầng tham chiếu, tầng này truyền tin trực tiếp với tầng giao vận. Tầng giao vận trên client truyền dữ liệu đi trên mạng máy tính tới tầng giao vận bên phía server. Tầng giao vận bên phía server truyền tin với tầng tham chiếu, tầng này truyền tin một phần của phần mềm server được gọi là skeleton. Skeleton truyền tin với chính server. Theo hướng khác từ server đến client thì luồng truyền tin được đi theo chiều ngược lại.

Cách tiếp cận có vẻ phức tạp nhưng ta không cần quan tâm đến vấn đề này. Tất cả đều được che dấu đi, người lập trình chỉ quan tâm đến việc lập các chương trình có khả năng gọi phương thức từ xa giống như đối với chương trình cục bộ.

Trước khi có thể gọi một phương thức trên một đối tượng ở xa, ta cần một tham chiếu tới đối tượng đó. Để nhận được tham chiếu này, ta yêu cầu một trình đăng ký tên rmiregistry cung cấp tên của tham chiếu. Trình đăng ký đóng vai trò như là một DNS nhỏ cho các đối tượng từ xa. Một client kết nối với trình đăng ký và cung cấp cho nó một URL của đối tượng từ xa. Trình đăng ký cung cấp một tham chiếu tới đối tượng đó và client sử dụng tham chiếu này để gọi các phương thức trên server.

Trong thực tế, client chỉ gọi các phương thức cục bộ trên trong stub. Stub là một đối tượng cục bộ thực thi các giao tiếp từ xa của các đối tượng từ xa.

Tầng tham chiếu từ xa thực thi giao thức tầng tham chiếu từ xa cụ thể. Tầng này độc lập với các đối tượng stub và skeleton cụ thể. Tầng tham chiếu từ xa có nhiệm vụ hiểu tầng tham chiếu từ xa có ý nghĩa như thế nào. Đôi khi tầng tham chiếu từ xa có thể tham chiếu tới nhiều máy ảo trên nhiều host.

Tầng giao vận gửi các lời gọi trên Internet. Phía server, tầng giao vận lắng nghe các liên kết đến. Trên cơ sở nhận lời gọi phương thức, tầng giao vận chuyển lời gọi cho tầng tham chiếu trên server. Tầng tham chiếu chuyển đổi các tham chiếu được gửi bởi client thành các tham chiếu cho các máy ảo cục bộ. Sau đó nó chuyển yêu cầu cho skeleton. Skeleton đọc tham số và truyền dữ liệu cho chương trình server, chương trình server sẽ thực hiện lời gọi phương thức thực sự. Nếu lời gọi phương thức trả về giá trị, giá trị được gửi xuống cho skeleton, tầng tham chiếu ở xa, và tầng giao vận trên phía server, thông qua Internet và sau đó chuyển lên cho tầng giao vận, tầng tham chiếu ở xa, stub trên phía client.

7. Cài đặt chương trình

Để lập một hệ thống client/server bằng RMI ta sử dụng ba gói cơ bản sau: java.rmi, java.rmi.server, java.rmi.registry. Gói java.rmi bao gồm các giao tiếp, các lớp và các ngoại lệ được sử dụng để lập trình cho phía client. Gói java.rmi.server cung cấp các giao tiếp, các lớp và các ngoại lệ được sử dụng để lập trình cho phía server. Gói java.rmi.registry có các giao tiếp, các lớp và các ngoại lệ được sử dụng để định vị và đặt tên các đối tượng.

7.1. Cài đặt chương trình phía Server

Để minh họa cho kỹ thuật lập trình RMI ở đây tác giả xin giới thiệu cách lập một chương trình FileServer đơn giản cho phép client tải về một tệp tin.

- **Bước 1:** Đặc tả giao tiếp Remote

```
import java.rmi.*;
public interface FileInterface extends Remote
{
 public byte[] downloadFile(String fileName)throws RemoteException;
}
```

- **Bước 2**: Viết lớp thực thi giao tiếp

```
import java.rmi.*;
import java.rmi.server.*;
import java.io.*;

public class  FileImpl extends UnicastRemoteObject implements FileInterface
{
 private String name;
 public FileImpl(String s)throws RemoteException
 {
        super();
        name=s;
 }
 public byte[] downloadFile(String fileName)throws RemoteException
 {
```

```
try{
        File file=new File(fileName);
        //Tạo một mảng b để lưu nội dung của tệp
        byte b[]=new byte[(int)file.length()];
        BufferedInputStream bis=new BufferedInputStream(new
FileInputStream(fileName));
        bis.read(b,0,b.length);
        bis.close();
        return b;
    }
    catch(Exception e)
    {
        System.err.println(e);
        return null;
    }
  }

}
```

- **Bước 3:** Viết chương trình phía server

```
import java.io.*;
import java.rmi.*;
import java.net.*;
public class FileServer
{
 public static void main(String[] args) throws Exception
 {
        FileInterface fi=new FileImpl("FileServer");
        InetAddress dc=InetAddress.getLocalHost();
        Naming.rebind("rmi://"+dc.getHostAddress()+"/FileServer",fi);
        System.out.println("Server ready for client requests...");
 }
}
```

- **Bước 4:** Cài đặt client

```
import java.rmi.*;
import java.io.*;
public class FileClient
{
```

```
    public static void main(String[] args) throws Exception
    {
        if(args.length!=2)
        {
            System.out.println("Su dung:java FileClient fileName machineName
");
            System.exit(0);
        }
        String name="rmi://"+args[1]+"/FileServer";
        FileInterface fi=(FileInterface)Naming.lookup(name);
        byte[] filedata=fi.downloadFile(args[0]);
        File file =new File(args[0]);
        BufferedOutputStream bos=new BufferedOutputStream(new
FileOutputStream(file.getName()));
        bos.write(filedata,0,filedata.length);
        bos.flush();
        bos.close();
    }
}
```

8. Triển khai ứng dụng

Để triển khai ứng dụng RMI ta cần thực hiện các bước sau:

- **Bước 1:** Biên dịch các tệp chương trình

C:\MyJava>javac FileInterface.java

C:\MyJava>javac FileImpl.java

C:\MyJava>javac FileServer.java

C:\MyJava>javac FileClient.java

Ta sẽ thu được các lớp sau:

FileInterface.class, FileImpl.class, FileServer.class, FileClient.class

Để tạo ra các lớp trung gian ta dùng lệnh sau:

C:\MyJava>rmic FileImpl

Sau khi biên dịch ta sẽ thu được hai lớp trung gian là FileImpl_Stub.class và FileImpl_Skel.class.

- **Bước 2:** Tổ chức chương trình

Ta tổ chức chương trình trên hai máy client và server như sau:

Phía Server	Phía Client
FileInterface.class	FileInterface.class
FileImpl.class	FileImpl_Stub.class
FileImpl_Skel.class	FileClient.class
FileServer.class	

Bảng 8.1

- **Bước 3:** Khởi động chương trình

Ở đây ta giả lập chương trình trên cùng một máy. Việc triển khai trên mạng không có gì khó khăn ngoài việc cung cấp hostname hoặc địa chỉ IP của server cung cấp dịch vụ

Khởi động trình đăng ký:

```
C:\MyJava>start rmiregistry
Khởi động server
C:\MyJava>start java FileServer
Khởi động client
C:\MyJava>java FileClient D:\RapidGet.exe localhost
```

9. Các lớp và các giao tiếp trong gói java.rmi

Khi viết một applet hay một ứng dụng sử dụng các đối tượng ở xa, người lập trình cần nhận thức rằng các giao tiếp và các lớp cần dùng cho phía client nằm ở trong gói java.rmi

9.1. *Giao tiếp Remote*

Giao tiếp này không khai báo bất kỳ phương thức nào. Các phương thức được khai báo trong phương thức này là các giao tiếp có thể được gọi từ xa.

9.2. Lớp Naming

Lớp java.rmi.Naming truyền tin trực tiếp với một trình đăng ký đang chạy trên server để ánh xạ các URL rmi://hostname/myObject thành các đối tượng từ xa cụ thể trên host xác định. Ta có thể xem trình đăng ký như là một DNS cho các đối tượng ở xa. Mỗi điểm vào trong trình đăng ký bao gồm một tên và một tham chiếu đối tượng. Các client cung cấp tên và nhận về một tham chiếu tới URL.

URL rmi giống như URL http ngoại trừ phần giao thức được thay thế bằng rmi. Phần đường dẫn của URL là tên gắn với đối tượng từ xa trên server chứ không phải là tên một tệp tin.

Lớp Naming cung cấp các phương thức sau:

- Public static String[] list(String url) throws RemotException

Phương thức này trả về một mảng các xâu ký tự, mỗi xâu là một URL đã được gắn với một tham chiếu. Tham số url là URL của trình đăng ký Naming.

- Public static Remote lookup(String url)throws RemotException, NotBoundException, AccessException, MalformedURLException

Client sử dụng phương thức này để tìm kiếm một đối tượng từ xa gắn liền với tên đối tượng.

Phương thức này đưa ra ngoại lệ NotBoundException nếu server ở xa không nhận ra tên của nó. Nó đưa ra ngoại lệ RemoteException nếu trình không thể liên lạc được với trình đăng ký . Nó đưa ra ngoại lệ AccessException nếu server từ chối tra tìm tên cho host cụ thể. Cuối cùng nếu URL không đúng cú pháp nó sẽ đưa ra ngoại lệ MalformedURLException.

- Public static void bind(String url, Remote object)throws RemotException, AlreadyBoundException, MalformedURLException, AccessException

Server sử dụng phương thức bind() để liên kết một tên với một đối tượng ở xa. Nếu việc gán là thành công thì client có thể tìm kiếm đối tượng stub từ trình đăng ký.

Có rất nhiều tình huống có thể xảy ra trong quá trình gán tên. Phương thức này đưa ra ngoại lệ MalformedURLException nếu url không đúng cú pháp. Nó đưa ra ngoại lệ RemoteException nếu không thể liên lạc được với trình đăng ký. Nó đưa ra ngoại lệ AccessException nếu client không được phép gán các đối tượng trong trình đăng ký. Nếu đối tượng URL đã gắn với một đối tượng cục bộ nó sẽ đưa ra ngoại lệ AlreadyBoundException.

- Public static void rebind(String url, Remote obj)throws RemoteException, AccessException, MalformedURLException

Phương thức này giống như phương thức bind() ngoại trừ việc là nó gán URL cho đối tượng ngay cả khi URL đã được gán.

10. Các lớp và các giao tiếp trong gói java.rmi.registry

Làm thế nào để nhận được một tham chiếu tới đối tượng? Client tìm ra các đối tượng ở xa hiện có bằng cách thực hiện truy vấn với trình đăng ký của server. Trình đăng ký cho ta biết những đối tượng nào đang khả dụng bằng cách truy vấn trình đăng ký của server. Ta đã biết lớp java.rmi.Naming cho phép chương trình giao tiếp với trình đăng ký.

Giao tiếp Registry và lớp LocateRegistry cho phép các client tìm kiếm các đối tượng ở xa trên một server theo tên. RegistryImpl là lớp con của lớp RemoteObject, lớp này liên kết các tên với các đối tượng RemoteObject. Client sử dụng lớp LocateRegistry để tìm kiếm RegistryImpl cho một host và cổng cụ thể.

10.1. Giao tiếp Registry

Giao tiếp này cung cấp năm phương thức:

- Bind() để gán một tên với một đối tượng từ xa cụ thể
- List() liệt kê tất cả các tên đã được đăng ký với trình đăng ký
- Lookup() tìm một đối tượng từ xa cụ thể với một URL cho trước gắn với nó
- Rebind() gán một tên với một đối tượng ở xa khác
- Unbind() loại bỏ một tên đã được gán cho một đối tượng ở xa trong trình đăng ký

Registry.REGISTRY_PORT là cổng mặc định để lắng nghe các các yêu cầu. Giá trị mặc định là 1099.

10.2. Lớp LocateRegistry

Lớp java.rmi.registry.LocateRegistry cho phép client tìm trong trình đăng ký trước tiên.

- Public static Registry getRegistry() throws RemoteException
- Public static Registry getRegistry(int port) throws RemoteException
- Public static Registry getRegistry(String host) throws RemoteException
- Public static Registry getRegistry(String host, int port) throws RemoteException
- Public static Registry getRegistry(String host, int port, RMIClientSocketFactory factory) throws RemoteException

Mỗi phương thức trên trả về một đối tượng Registry được sử dụng để nhận các đối tượng từ xa thông qua tên.

Ví dụ để client có tìm thấy đối tượng ta có đăng ký đối tượng đó với trình đăng ký thông qua lớp Registry:

Registry r=LocateRegistry.getRegistry();

r.bind("MyName",this);

Client muốn gọi phương thức trên đối tượng từ xa có thể dùng các lệnh sau:

Registry r=LocateRegistry.getRegistry(www.somehose.com);

RemoteObjectInterface obj=(RemoteObjectInterface)r.lookup("MyName");

Obj.invokeRemoteMethod();

Ví dụ dưới đây minh họa cách tạo ra một trình đăng ký ngay trong server

import java.io.;*

import java.rmi.;*

```
import java.net.*;
import java.rmi.registry.*;
public class FileServer
{
 public static void main(String[] args) throws Exception
 {
        FileInterface fi=new FileImpl("FileServer");
        InetAddress dc=InetAddress.getLocalHost();
        LocateRegistry.createRegistry(1099);
        Naming.bind("rmi://"+dc.getHostAddress()+"/FileServer",fi);
        System.out.println("Server ready for client requests...");
 }
}
```

Như vậy khi thực thi chương trình ta không cần phải khởi động trình đăng ký vì việc tạo ra trình đăng ký và khởi động nó đã được tiến hành ở ngay trong chương trình phía server.

11. Các lớp và các giao tiếp trong gói java.rmi.server

11.1. Lớp RemoteObject

Về mặt kỹ thuật đối tượng từ xa không phải là một thể hiện của lớp RemoteObject. Thực tế, phần lớn các đối tượng từ xa là thể hiện của các lớp con của lớp RemoteObject.

11.2. Lớp RemoteServer

Lớp này là lớp con của lớp RemoteObject; nó là lớp cha của lớp UnicastRemoteObject.

Lớp này có các constructor này:

- Protected RemoteServer()
- Protected RemoteServer(RemoteRef r)

Nhận các thông tin về client

Lớp RemoteServer có một phương thức để xác định thông tin về client mà ta đang truyền tin với nó:

- public static String getClientHost() throws ServerNotActiveException

Phương thức này trả về hostname của client mà gọi phương thức từ xa.

11.3. Lớp UnicastRemoteObject

Lớp UnicastRemoteObject là một lớp con cụ thể của lớp RemoteServer. Để tạo ra một đối tượng ở xa, ta phải thừa kế lớp UnicastRemoteServer và khai báo lớp này thực thi giao tiếp Remote.

12. Kết luận

RMI là một công nghệ phân tán cho phép các phương thức trên các máy ảo Java được gọi từ xa. Đây là cách đơn giản để truyền tin giữa một ứng dụng này với ứng dụng khác so với truyền tin trực tiếp với TCP socket, cách truyền tin này đòi hỏi cả hai phía đều sử dụng cùng một giao thức. Thay vì viết các chương trình cài đặt giao thức, những người phát triển có thể tương tác với các phương thức đối tượng được định nghĩa bởi một giao tiếp dịch vụ RMI. Mỗi khi có được một tham chiếu tới đối tượng từ xa, tham

chiếu này có thể được xem như là một đối tượng cục bộ, đây là cách trực quan để phát triển các ứng dụng mạng.

Chương 9

Xử lý cơ sở dữ liệu trong Java

Các ứng dụng Internet ngày nay thường được dựa trên các cơ sở dữ liệu lớn được cài đặt bằng cách sử dụng công nghệ cơ sở dữ liệu quan hệ. Kể từ khi xuất hiện từ năm 1995, Java được yêu cầu cần cung cấp khả năng kết nối với các cơ sở dữ liệu quan hệ hiện có như Ingres, Oracle, Access, và SQL Server,…Các tiện ích cho phép truy xuất cơ sở dữ liệu nằm trong gói java.sql.

Ngày nay các thông tin với dung lượng lớn đều được lưu trữ trong các kho dữ liệu lớn. Khả năng truy xuất tới các cơ sở dữ liệu là điều không thể thiếu đối với các ứng dụng. Điều này lại càng đúng với các ứng dụng chạy trên mạng máy tính nói chung và Internet nói riêng. Trong chương này chúng ta sẽ đi vào tìm hiểu giao diện lập trình ứng dụng JDBC của Java và cách thức để kết nối với một cơ sở dữ liệu từ một ứng dụng Java thông qua JDBC.

1. JDBC Java Database Connectivity API

SUN đã phát triển một giao diện lập trình ứng dụng API để truy xuất cơ sở dữ liệu-JDBC. Mục tiêu đặt ra của SUN là:

- JDBC là một giao diện lập trình ứng dụng mức SQL.
- JDBC cần có được những kinh nghiệm làm việc với các API cơ sở dữ liệu hiện có.
- JDBC cần đơn giản

Giao diện lập trình ứng dụng mức SQL nghĩa là JDBC cho phép ta xây dựng các lệnh SQL và nhúng các lệnh SQL bên trong các lời gọi Java API. Nói tóm lại, về cơ bản ta vẫn sử dụng SQL nhưng JDBC cho phép ta dịch một cách trôi chảy giữa thế giới cơ sở dữ liệu và thế giới ứng dụng Java. Kết quả của bạn từ cơ sở dữ liệu, được trả về dưới dạng các đối tượng Java và nếu có vấn đề khi truy xuất nó sẽ đưa ra các ngoại lệ.

JDBC API đã chuẩn hóa:

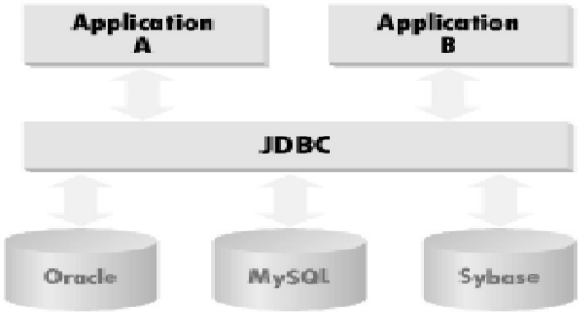
- Cách thiết lập tới cơ sở dữ liệu
- Cách tiếp cận để khởi tạo các truy vấn
- Cách thức để tạo ra các truy vấn có tham số
- Chuẩn hóa cấu trúc dữ liệu của kết quả truy vấn
    - Xác định số cột
    - Tra tìm các metadata.

JDBC API chưa chuẩn hóa cú pháp SQL. JDBC không phải là SQL nhúng. Lớp JDBC nằm trong gói java.sql. Nó bao gồm hai phần:

- JDBC API là một giao diện lập trình ứng dụng viết bằng ngôn ngữ Java thuần túy.
- Trình quản lý Driver JDBC truyền tin với các trình điều khiển cụ thể của nhà sản xuất, các trình điều khiển cơ sở dữ liệu của nhà sản xuất truyền tin với cơ sở dữ liệu.

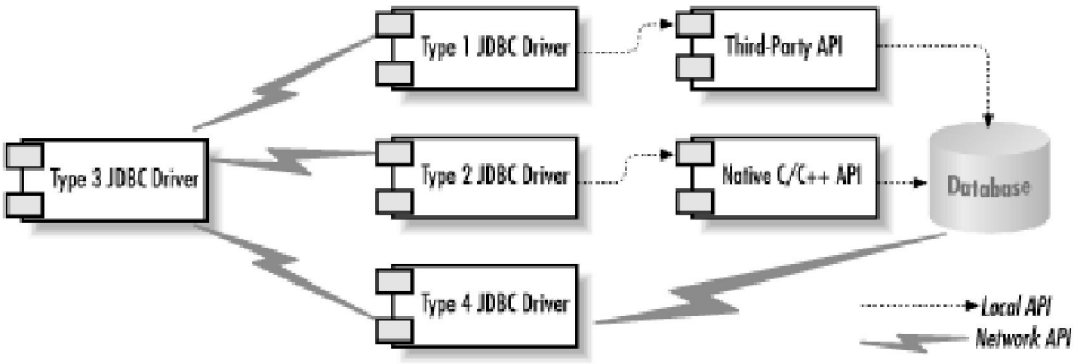
2. Cấu trúc của JDBC

JDBC thực hiện các mục tiêu của nó thông qua một tập hợp các giao tiếp JDBC, mỗi giao tiếp thực được thực hiện bởi từng nhà sản xuất. Tập hợp các lớp thực thi các giao tiếp JDBC cho một mô tơ cơ sở dữ liệu cụ thể được gọi là một trình điều khiển JDBC. Khi xây dựng một ứng dụng cơ sở dữ liệu, ta không phải xem xét đến tất cả các lớp cơ sở. JDBC che dấu các chi tiết của từng cơ sở dữ liệu và như vậy ta chỉ cần quan tâm đến ứng dụng của mình.



Hình 9.1

Các cơ sở dữ liệu và các trình điều khiển



Hình 9.2

2.1. Kiểu 1

Các trình điều khiển này sử dụng một công nghệ cầu nối để truy xuất tới một cơ sở dữ liệu. Cầu nối JDBC-ODBC được bắt đầu đưa vào từ JDK 1.2 là một ví dụ điển hình cho kiểu driver này. Nó cung cấp một gateway tới API ODBC. Cài đặt của API này thực hiện truy xuất tới cơ sở dữ liệu thực tế. Giải pháp cầu nối thường yêu cầu phần mềm phải được cài đặt trên hệ thống client, nghĩa là chúng không phải là các giải pháp tốt cho các ứng dụng mà không cho phép cài đặt phần mềm trên client.

Cầu nối JDBC-ODBC cung cấp cách truy xuất thông qua một hay nhiều trình điều khiển ODBC.

- Ưu điểm:
  - Đây là một cách tiếp cận tốt để học JDBC.
  - Hữu ích cho các công ty đã cài đặt trình điều khiển ODBC trên từng máy client.
  - Đây là cách duy nhất để truy xuất được tới các cơ sở dữ liệu trên máy tính để bàn mức thấp.
- Nhược điểm:
  - Không phù hợp với các ứng dụng quy mô lớn. Hiệu năng thấp vì có cần nhiều công đoạn cần thực hiện để chuyển từ JDBC sang ODBC.
  - Không hỗ trợ tất cả các đặc trưng của Java.
  - Người sử dụng bị hạn chế bởi chức năng do trình điều khiển ODBC cung cấp.

2.2. Kiểu 2

Các trình điều khiển kiểu 2 là các trình điều khiển API-trình điều khiển gốc. Điều này nghĩa là mã Java gọi các phương thức C hoặc C++ được cung cấp bởi từng nhà sản xuất hệ quản trị cơ sở dữ liệu để thực hiện truy xuất tới cơ sở dữ liệu. Giải pháp này vẫn

yêu cầu phải có phần mềm trên hệ thống client. JDBC chuyển các lời gọi tới JDBC API thành các lời gọi kết nối với giao diện lập trình ứng dụng của máy khác cho một cơ sở dữ liệu cụ thể như IBM, Informix, Oracle, hoặc Sybase.

- Ưu điểm:

Hiệu năng tốt hơn kiểu 1, vì trình điều khiển kiểu 2 chứa các mã lệnh đã được biên dịch được tối ưu hóa cho hệ điều hành của server có sở dữ liệu hoạt động ở chế độ hậu trường,

- Nhược điểm
    - Người sử dụng cần đảm bảo rằng trình điều khiển JDBC của nhà sản xuất cơ sở dữ liệu có trên từng máy khách.
    - Phải có chương trình đã được biên dịch cho mỗi hệ điều hành mà ứng dụng sẽ chạy.
    - Chỉ sử dụng có hiệu quả trong các môi trường có kiểm soát như một mạng intranet

2.3. Kiểu 3

Các trình điều khiển kiểu 3 cung cấp cho client một API mạng chung, API này sau đó chuyển thành thao tác truy xuất cơ sở dữ liệu mức server. Mặt khác, trình điều khiển JDBC trên client sử dụng các socket để gọi một ứng dụng trung gian (middleware) trên server để chuyển các yêu cầu của client thành một API cụ thể đối với từng server. Kết quả là trình điều khiển này đặc biệt linh hoạt, vì nó không cần phải có phần mềm cài đặt trên client và một trình điều khiển có thể cung cấp khả năng truy xuất tới nhiều cơ sở dữ liệu.

Java Middleware thuần tuý

Trình điều khiển Java thuần túy cho các chương trình trung gian cơ sở dữ liệu để dịch các lời gọi JDBC cho giao thức của nhà sản xuất phần mềm trung gian, trình điều khiển này sau đó được chuyển cho một giao thức gắn với cơ sở dữ liệu cụ thể bởi phần mềm server trung gian.

- Ưu điểm:
    - Được sử dụng khi một công ty có nhiều cơ sở dữ liệu và muốn sử dụng một trình điều khiển JDVC để kết nối với tất cả các cơ sở dữ liêu.
    - Trình điều khiển nằm trên server, vì thế không cần trình điều khiển JDBC trên từng máy client
    - Thành phần server được tối ưu hóa cho hệ điều hành đang chạy ở chế độ hậu trường
- Nhược điểm:
    - Cần mã lệnh cho cơ sở dữ liệu cụ thể trên server trung gian

2.4. Kiểu 4

Sử dụng các giao thức mạng được tích hợp sẵn vào engine cơ sở dữ liệu, các driver kiểu 4 truyền tin trực tiếp với cơ sở dữ liệu bằng cách sử dụng socket Java. Đây là trình điều khiển Java thuần túy nhất. Kiểu trình điều khiển này thường do nhà sản xuất cơ sở dữ liệu cung cấp.

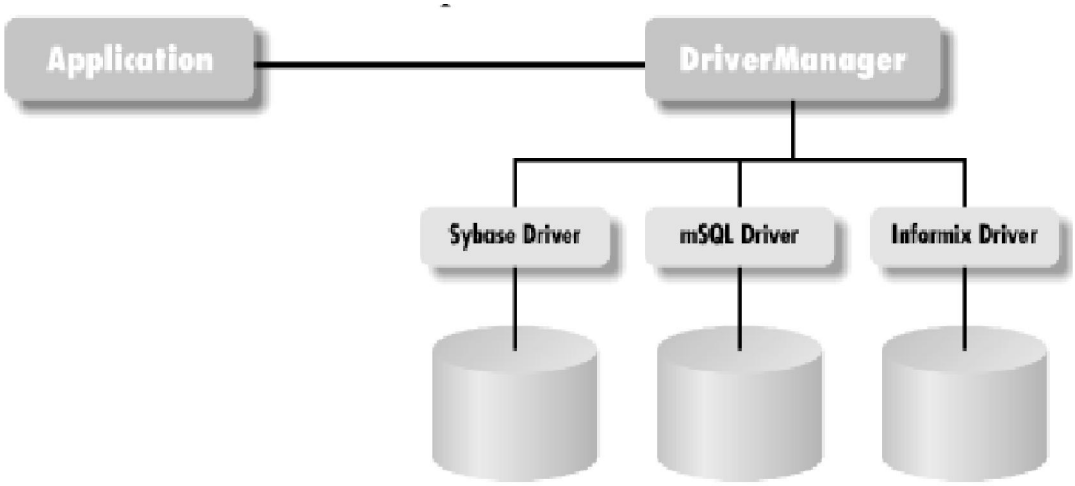
Trình điều khiển Java thuần túy tới kết nối trực tiếp với cơ sở dữ liệu chuyển các lời gọi JDBC thành các gói tin được truyền đi trên mạng theo một khuôn dạng được sử dụng bởi cơ sở dữ liệu cụ thể. Cho phép một lời gọi trực tiếp từ máy client tới cơ sở dữ liệu.

- Ưu điểm:

- Không cần cài phần mềm đặc biệt nào trên client hoặc server. Có thể được tải về một cách linh hoạt

- Nhược điểm
  - Không tối ưu cho hệ điều hành server vì vậy trình điều khiển không thể tận dụng các đặc trưng ưu việt của hệ điều hành

3. Kết nối cơ sở dữ liệu



Hình 10.3

Hình vẽ trên cho thấy cách thức mà một ứng dụng JDBC truyền tin với một hoặc nhiều cơ sở dữ liệu mà không cần biết đến các chi tiết có liên quan đến cài đặt driver cho cơ sở dữ liệu đó. Một ứng dụng sử dụng JDBC như là một giao tiếp, thông qua đó nó truyền tất cả các yêu cầu liên quan đến cơ sở dữ liệu của nó.

Khi ta viết các applet hay ứng dụng cơ sở dữ liệu, ta có thể cung cấp các thông tin cụ thể về trình điều khiển JDBC là URL cơ sở dữ liệu. Thậm chí ta có thể nhập vào URL cơ sở dữ liệu cho ứng dụng và applet vào thời gian chạy dưới dạng các tham số.

JDBC là gói kết nối cơ sở dữ liệu bao gồm giao diện lập trình ứng dụng căn bản Java API. Java cung cấp một interface độc lập với cơ sở dữ liệu để mở một kết nối tới cơ sở dữ liệu, bằng cách phát ra các lời gọi SQL tới cơ sở dữ liệu và nhận về kết quả là một tập hợp các dữ liệu. Ở góc độ kỹ thuật, JDBC đóng vai trò như là một chương trình cài đặt giao tiếp mức lời gọi SQL được định nghĩa bởi X/Open và được hỗ trợ bởi hầu hết các nhà cung cấp cơ sở dữ liệu quan hệ. Để thực hiện giao tác với một kiểu cơ sở dữ liệu cụ thể, ta cần phải có một trình điều khiển JDBC đóng vai trò như là một cầu nối giữa các lời gọi phương thức JDBC và interface cơ sở sữ liệu.

3.1. DriverManager

DriverManager cung cấp phương tiện để nạp các trình điều khiển cơ sở dữ liệu vào một ứng dụng Java hoặc một applet; nó chính là cách để JDBC thiết lập một liên kết với cơ sở dữ liệu. Ứng dụng Java, trước tiên tạo một đối tượng DriverManager, kết nối với cơ sở dữ liệu bằng cách gọi phương thức tĩnh getConnection() của lớp DriverManager, với tham chiếu truyền vào giống như một URL được gọi là URL cơ sở dữ liệu. DriverManager tìm kiếm một driver hỗ trợ việc kết nối trong tập hợp các driver hiện có. Nếu tìm thấy driver nó truyền địa chỉ cơ sở dữ liệu cho driver và yêu cầu driver tạo ra một kết nối. Kết nối tới cơ sở dữ liệu được trả về dưới dạng một đối tượng Connection.

Tất cả các driver JDBC cung cấp một cài đặt giao tiếp java.sql.Driver. Khi một DriverManager được tạo ra, nó tải một tập hợp các driver được xác định bởi thuộc tính của java.sql.Driver. Driver được nạp vào thời gian chạy Java, nó có nhiệm vụ tạo ra một đối tượng và đăng ký đối tượng với DriverManager. Các driver cần cho ứng dụng có thể được nạp bởi phương thức Class.forName()

Driver myDriver=(Driver)Class.forName("specialdb.Driver");

3.2. Connection

Mỗi khi các driver cần thiết được nạp bởi DriverManager, sẽ có một liên kết với một cơ sở dữ liệu được tạo ra nhờ phương thức getConnection() của lớp DriverManager. Cơ sở dữ liệu cần làm việc được xác định thông qua một tham số String đóng vai trò như là địa chỉ tham chiếu tới cơ sở dữ liệu. Không có một khuôn dạng chuẩn nào cho địa chỉ xâu cơ sở dữ liệu; DriverManager truyền xâu địa chỉ cho từng driver JDBC đã được nạp và xem nó có hiểu và hỗ trợ kiểu cơ sở dữ liệu đã được xác định.

Jdbc:odbc:financedata

Trong đó financedata là nguồn cơ sở dữ liệu cục bộ. Để truy xuất tới một cơ sở dữ liệu từ xa từ một máy client ta có thể dùng cú pháp sau:

Jdbc:odbc:drvr://dataserver.foobar.com:500/financedata.

Đặc tả JDBC API khuyến cáo một URL cơ sở dữ liệu nên có dạng như sau:

Jdbc:<sub-protocol>:<sub-name>

Trong đó <sub-protocol> xác định dịch vụ kết nối cơ sở dữ liệu và <sub-name> cung cấp tất cả các thông tin cần thiết để dịch vụ tìm cơ sở dữ liệu và kết nối tới nó.

Phương thức getConnection() trên DriverManager hoặc là trả về một đối tượng Connection biểu diễn liên kết tới cơ sở dữ liệu đã được chỉ ra, hoặc là đưa ra ngoại lệ nếu liên kết không được thiết lập.

3.3. Statement

Giao tiếp Connection cho phép người sử dụng tạo ra một câu lệnh truy vấn tới cơ sở dữ liệu. Các lệnh truy vấn được biểu diễn dưới dạng các đối tượng Statement hoặc các lớp con của nó. Giao tiếp Connection cung cấp ba phương thức để tạo ra các lệnh truy vấn cơ sở dữ liệu là: createStatement(), prepareStatement(), và precpareCall(). createStatement() được sử dụng cho các lệnh SQL đơn giản không liên quan đến các tham số. Phương thức này trả về một đối tượng Statement được sử dụng để phát tra các truy vấn SQL tới cơ sở dữ liệu, bằng cách sử dụng phương thức executeQuery(). Phương thức này chấp nhận một lệnh SQL như là một xâu và các kết quả trả về là ở dưới dạng một đối tượng ResultSet. Các phương thức khác có trong giao tiếp Statement để phát ra các lệnh SQL tới các cơ sở dữ liệu là phương thức execute(), phương thức này được sử dụng cho các truy vấn SQL và trả về nhiều resultset và phương thức executeUpdate() được sử dụng để phát ra các lệnh INSERT, UPDATE, hoặc DELETE.

Ngoài giao tiếp Statement cơ bản, một đối tượng Connection có thể được sử dụng để tạo ra một đối tượng PreparedStatement và các CallableStatement biểu diễn các thủ tục stored procedure trong cơ sở dữ liệu. Một lệnh SQL có thể liên quan đến nhiều tham số đầu vào, hoặc một lệnh mà ta muốn xử lý nhiều lần, có thể được tạo ra bằng cách sử dụng lệnh prepareStatement() trên đối tượng Connection, phương thức này trả về đối tượng PreparedStatement. Lệnh SQL được truyền cho phương thức prepareStatement() là một lệnh được biên dịch trước vì vậy việc xử lý nhiều lần một lệnh sẽ hiệu quả hơn. Lớp con của lớp Statement hỗ trợ việc thiết lập các giá trị của các tham số đầu vào được biên dịch trước thông qua các phương thức setXXX(). Đối tượng PreparedStatement có phương thức executeQuery() không cần tham số, thay vào đó nó xử lý các lệnh SQL được biên dịch trước trên cơ sở dữ liệu. Chú ý rằng không phải tất cả các nhà sản xuất cơ sở dữ iệu hoặc các driver JDBC đều hỗ trợ các lệnh được biên dịch trước.

3.4. ResultSet

Các dòng dữ liệu được trả về từ việc xử lý một lệnh được biểu diễn bằng một ResultSet trong JDBC. Ví dụ, phương thức executeQuery() của Statement trả về một đối tượng ResultSet. Đối tượng ResultSet cung cấp các cách để duyệt qua các dòng dữ liệu được trả về từ việc xử lý câu lệnh truy vấn SQL thông qua phương thức next() của nó; các trường dữ liệu trong mỗi hàng có thể được tìm kiếm thông qua các tên hoặc chỉ mục cột bằng cách sử dụng phương thức getXXX(). Người dùng cần phải biết kiểu dữ liệu

trong mỗi cột dữ liệu được trả về, vì mỗi mục dữ liệu được tìm kiếm thông qua các phương thức getXXX() có kiểu cụ thể.

Tùy thuộc vào kiểu trình điều khiển JDBC được cài đặt, việc duyệt qua các hàng dữ liệu trong đối tượng ResultSet có thể tạo ra hiệu ứng lấy dữ liệu từ cơ sở dữ liệu, hoặc đơng giản là trả về từng hàng dữ liệu từ cache. Nếu hiệu năng của các giao dịch là vấn đề đối với ứng dụng, ta cần xác định dữ liệu trả về được quản lý như thế nào bởi các trình điều khiển của nhà sản xuất.

Lưu ý: Giá trị trả lại của hàm *getXXX(args)* là dữ liệu của trường có tên là *args* của các dòng dữ liệu đã được chọn ra. Ngoài ra cũng cần phân biệt các kiểu của Java với các kiểu dữ liệu của SQL. Bảng dưới đây mô tả các kiểu dữ liệu tương ứng của Java, SQL và các hàm *getXXX().*

Kiểu của SQL	Kiểu của Java	Hàm getXXX()
CHAR	String	getString()
VARCHAR	String	getString()
LONGVARCHAR	String	getString()
NUMBERIC	java.math.BigDecimal	getBigDecimal()
DECIMAL	java.math.BigDecimal	getBigDecimal()
BIT	Boolean (boolean)	getBoolean()
TINYINT	Integer (byte)	getByte()
SMALLINT	Integer (short)	getShort()
INTEGER	Integer (int)	getInt()
BIGINT	Long (long)	getLong()
REAL	Float (float)	getFloat()
FLOAT	Double (double)	getDouble()
DOUBLE	Double (double)	getDouble()
BINARY	byte[]	getBytes()
VARBINARY	byte[]	getBytes()
LONGVARBINARY	byte[]	getBytes()
DATE	java.sql.Date	getDate()
TIME	java.sql.Time	getTime()
TIMESTAMP	java.sql.Timestamp	getTimestamp()

Bảng 10.1

4. Lớp DatabaseMetaData

Muốn xử lý tốt các dữ liệu của một CSDL thì chúng ta phải biết được những thông tin chung về cấu trúc của CSDL đó như: hệ QTCSDL, tên của các bảng dữ liệu, tên gọi của các trường dữ liệu, v.v .

Để biết được những thông tin chung về cấu trúc của một hệ CSDL, chúng ta có thể sử dụng giao diện *java.sql.DatabaseMetaData* thông qua hàm *getMetaData().*

DatabaseMetaData dbmeta = con.getMetaData();

trong đó, *con* là đối tượng kết nối đã được tạo ra bởi lớp *Connection.*

Lớp DatabaseMetaData cung cấp một số hàm được nạp chồng để xác định được những thông tin về cấu hình của một CSDL. Một số hàm cho lại đối tượng của String (*getURL()*), một số trả lại giá trị logic (*nullsAreSortedHigh()*) hay trả lại giá trị nguyên như

hàm *getMaxConnection()*). Những hàm khác cho lại kết quả là các đối tượng của *ResultSet* như: *getColumns(), getTableType(), getPrivileges(), v.v.*

5. Lớp ResultSetMetaData

Giao diện *ResultSetMetaData* cung cấp các thông tin về cấu trúc cụ thể của *ResultSet*, bao gồm cả số cột, tên và giá trị của chúng. Ví dụ sau là một chương trình hiển thị các kiểu và giá trị của từng trường của một bảng dữ liệu.

Ví dụ 9.3 Chương trình hiển thị một bảng dữ liệu.

```
import java.sql.*;
import java.util.StringTokenizer;
public class TableViewer {
 final static String jdbcURL = "jdbc:odbc:StudentDB";
 final static String jdbcDriver =
                                    "sun.jdbc:odbc:JdbcOdbcDriver";
 final static String table = "STUDENT";
 public static void main(java.lang.String[]args) {
  System.out.println("---Table Viewer ---");
  try {
        Class.forName(jdbcDriver);
        Connection con =
              DriverManager.getConnection(jdbcURL, "", "");
        Statement stmt = con.createStatement();
        // Đọc ra cả bảng Student và đưa vào đối tượng rs
        ResultSet rs = stmt.executeQuery("SELECT * FROM " + table);
        // Đọc ra các thông tin về rs
        ResultSetMetaData rsmd = rs.getMetaData();
        // Xác định số cột của rsmd
        int colCount = rsmd.getColumnCount();
        for(int col = 1; col <= colCount; col++)
        {
              // In ra tên và kiểu của từng trường dữ liệu trong rsmd
              System.out.print(rsmd.getColumnLabel(col));
              System.out.print(" (" + rsmd.getColumnTypeName(col) + ")");
              if(col < colCount)
                 System.out.print(", ");
              }
        System.out.println();

        while(rs.next()){
              // In ra dòng dữ liệu trong rsmd
              for(int col = 1; col <= colCount; col++)
```

```
                    {
                    System.out.print(rs.getString(col));
                    if(col < colCount)
                      System.out.print("  ");
                }
                System.out.println();
            }
            rs.close();
            stmt.close();
            con.close();
            }
            catch (ClassNotFoundException e) {
                System.out.println("Unable to load database driver class");
              }
              catch (SQLException se) {
                  System.out.println("SQL Exception: " + se.getMessage());
              }
     }
    }
```

6. Các bước cơ bản để kết nối với cơ sở dữ liệu từ một ứng dụng Java

- **Bước 1:** Nạp trình điều khiển

```
try{
Class.forName(“oracle.jdbc.driver.OracleDriver”);
}
catch(ClassNotFoundException e)
{
System.out.println(“Loi nap trinh dieu khien:”+e);
}
```

- **Bước 2:** Xác định URL cơ sở dữ liệu

```
String host=”dbhost.yourcompany.com”;
String dbName=”someName”;
int port=1234;
String oracaleURL=”jdbc:oracle:thin:@”+host+”:”+port+dbName;
```

- **Bước 3:** Thiết lập liên kết

```
String username=”hoan_td2001”;
String password=”topsecret”;
Connection con=DriverManager.getConnecton(oracleURL,username,password);
```

- **Bước 4:** Tạo ra một đối tượng Statement

  Statement s=con.createStatement();
- **Bước 5:** Xử lý truy vấn

  String q="Select col1, col2, col3 from sometable";

  ResultSet rs=s.executeQuery(q);
- **Bước 6:** Xử lý kết quả

  while(rs.next())

  {

  System.out.println(rs.getString(1)+" "+

  rs.getString(2)+" "+

  rs.getString(3));

  }

  Cột đầu tiên có chỉ mục là 1 chứ không phải là 0.
- **Bước 7: Đóng liên kết**

  con.close();

  Các ví dụ về kết nối cơ sở dữ liệu từ ứng dụng Java.

Ví dụ về kết nối kiểu 1:

```
import java.sql.*;
class DBOracle1
{
    public static void main(String args[])throws ClassNotFoundException, SQLException
    {

    try{

            //Co the dung lenh nay de tai driver
            Class.forName("oracle.jdbc.OracleDriver");

            DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

            //Lien ket toi co so du lieu
            Connection conn = DriverManager.getConnection("jdbc:oracle:oci8:@HOAN", "scott", "tiger");

            Statement stmt = conn.createStatement( );

            ResultSet rset = stmt.executeQuery("select empno, ename from emp");
           ResultSetMetaData rst=rset.getMetaData();
            int numcol=rst.getColumnCount();
           System.out.println("So cot cua bang la:"+numcol);
           System.out.println("Schema Name:" + rst.getTableName(1));

           for(int i=1;i<numcol+1;i++)
```

```
                System.out.println(rst.getColumnName(i)+" "+rst.getColumnTypeName(i));

                while(rset.next( ))
                {
                        System.out.println(rset.getString("empno"));
                        System.out.println(rset.getString("ename"));

                }

                rset.close( );
                stmt.close( );
                conn.close( );
        }
        catch(Exception e)
        {
                System.err.println("Ex : "+e);
        }
    }
}
```

Ví dụ về kết nối kiểu 2:

```
import java.io.*;
import java.sql.*;
import java.text.*;
public class DBOracle2 {
 Connection conn;
 public DBOracle2( )
 {
        try
        {
        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

    Connection conn = DriverManager.getConnection("jdbc:oracle:oci8:@HOAN",
"scott", "tiger");

        }
        catch (SQLException e)
        {
                System.err.println(e.getMessage( ));
                e.printStackTrace( );
        }
 }

 public static void main(String[] args)throws Exception, IOException
```

```
{
        new DBOracle2().process( );
}

public void process( ) throws IOException, SQLException
{
        int rows = 0;
        ResultSet rslt = null;
        PreparedStatement pstmt = null;
        String insert ="insert into EMP " +"( EMPNO, ENAME,JOB) " +"values " +"(
?, ?, ?)";

        try {
                System.out.println(insert);
                pstmt = conn.prepareStatement(insert);
                pstmt.setString( 1, "EMPNO" );
                pstmt.setString( 2, "ENAME" );
                pstmt.setString( 3,"JOB" );
                rows = pstmt.executeUpdate( );
                pstmt.close( );
                pstmt = null;
                System.out.println(rows + " rows inserted");
                System.out.println("");
        }
        catch (SQLException e) {
                System.err.println(e.getMessage( ));
        }
        finally {
                        if (pstmt != null)
                try {
                    pstmt.close( );
                }
                catch(SQLException ignore)
                {
                }
        }


}
```

```
            protected void finalize( )throws Throwable {
                if (conn != null)
                try { conn.close( ); } catch (SQLException ignore) { }
            super.finalize( );
            }
        }
```

Ví dụ về kết nối kiểu 4:

```
        //Type 4 Driver
        import java.sql.*;
        import java.util.*;

        class DBOracle4
        {
         public static void main(String args[])throws ClassNotFoundException,
SQLException
         {

         try{

                //Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

                DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

                  Enumeration drivers = DriverManager.getDrivers();
                  while(drivers.hasMoreElements())
                  {
                    Driver driver = (Driver)drivers.nextElement();
                    System.out.println("Registered Driver:"+driver.getClass().getName());
                  }


                //Lien ket toi co so du lieu
                Connection conn =
DriverManager.getConnection("jdbc:Oracle:thin:@neworacle02:1521:HOAN", "scott",
"tiger");
                  DatabaseMetaData dbmd=conn.getMetaData();
                  System.out.println(dbmd.getDatabaseProductName());
                System.out.println(dbmd.getDatabaseProductVersion());
                Statement stmt = conn.createStatement( );
```

```
                ResultSet rset = stmt.executeQuery("select empno, ename from emp");
                  ResultSetMetaData rst=rset.getMetaData();
                int numcol=rst.getColumnCount();
                  System.out.println("So cot cua bang la:"+numcol);
                  System.out.println(rst.getTableName(1));

                  for(int i=1;i<numcol+1;i++)
                    System.out.println(rst.getColumnName(i)+"
"+rst.getColumnTypeName(i));

                while(rset.next( ))
                {
                        System.out.println(rset.getString("empno")+"
"+rset.getString("ename"));

                }

                rset.close( );
                stmt.close( );
                conn.close( );
                }
         catch(Exception e)
         {
                System.err.println("Ex : "+e);
         }
          }
        }
```

7. Sử dụng PreparedStatement

Đôi khi việc sử dụng một đối tượng PreparedStatent hiệu quả và tiện lợi hơn nhiều so với việc sử dụng đối tượng Statement. Kiểu lệnh đặc biệt này là lớp con của lớp Statement.

Khi nào cần sử dụng đối tượng PreparedStatement

Nếu ta muốn xử lý một đối tượng Statement nhiều lần, ta có thể sử dụng đối tượng PreparedStatement để giảm thời gian xử lý.

Đặc trưng chính của một đối tượng PreparedStatement là nó được cung cấp trước một lệnh SQL trước khi tạo ra đối tượng. Đối tượng PreparedStatement là một lệnh SQL đã được biên dịch trước. Điều này nghĩa là khi đối tượng PreparedStatement được xử lý, hệ quản trị cơ sở dữ liệu chỉ cần xử lý lệnh SQL của PreparedStatement mà không phải biên dịch nó.

Mặc dù PreparedStatement có thể được sử dụng với các lệnh SQL không có tham số nhưng ta thường hay sử dụng các lệnh SQL có tham số. Ưu điểm của việc sử dụng lệnh SQL có tham số là ta có thể sử dụng cùng một lệnh và cung cấp cho nó các giá trị khác nhau mỗi khi xử lý. Ta sẽ thấy điều này trong ví dụ ở phần sau.

Tạo một đối tượng PreparedStatement

Giống như các đối tượng Statement, bạn đọc có thể tạo ra các đối tượng PrepraredStatement với một phương thức Connection. Sử dụng một kết nối mở trong ví dụ trước là *con*, có thể tạo ra đối tượng PreparedStatement nhận hai tham số đầu vào như sau:

```
PreparedStatement updateSales = con.prepareStatement(
"UPDATE COFFEES SET SALES = ? WHERE COF_NAME LIKE ?");
```

Cung cấp các giá trị cho các tham số của đối tượng PreparedStatement

Ta cần cung cấp các giá trị được sử dụng thay cho vị trí của các dấu hỏi nếu có trước khi xử lý một đối tượng PreparedStatement. Ta có thể thực hiện điều này bằng cách gọi một trong các phương thức setXXX đã được định nghĩa trong lớp PreparedStatement. Nếu giá trị ta muốn thay thế cho dấu hỏi (?) là kiểu int trong Java, ta có thể gọi phương thức setInt. Nếu giá trị ta muốn thay thế cho dấu (?) là kiểu String trong Java, ta có thể gọi phương thức setString,…Một cách tổng quát, ứng với mỗi kiểu trong ngôn ngữ lập trình Java sẽ có một phương thức setXXX tương ứng.

Ví dụ:

```
import java.sql.*;
public class PreparedStmt{
public static void main(String args[]){

int empid;
String LastName;
String FirstName;
String query = "SELECT * FROM Employees where EmployeeID=?;";
try {
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
Connection con =DriverManager.getConnection ("jdbc:odbc:MyData");
PreparedStatement pstmt = con.prepareStatement(query);
pstmt.setInt(1,2);
ResultSet rs = pstmt.executeQuery();
while (rs.next()) {
empid = rs.getInt("EmployeeID");
LastName = rs.getString("LastName");
FirstName = rs.getString("FirstName");
System.out.println(empid+", "+LastName+"\t"+FirstName+"\t");
}
}
catch(ClassNotFoundException e){
e.printStackTrace();
```

```
}
catch(SQLException e){
e.printStackTrace();
}
}
}
```

- Sử dụng một vòng lặp để thiết lập các giá trị

Ta có thể sử dụng vòng lặp để thiết lập các giá trị cho các tham số đầu vào.

```
PreparedStatement updateSales;
String updateString = "update COFFEES " +
"set SALES = ? where COF_NAME like ?";
updateSales = con.prepareStatement(updateString);
int [] salesForWeek = {175, 150, 60, 155, 90};
String [] coffees = {"Colombian", "French_Roast", "Espresso",
"Colombian_Decaf", "French_Roast_Decaf"};
int len = coffees.length;
for(int i = 0; i < len; i++) {
updateSales.setInt(1, salesForWeek[i]);
updateSales.setString(2, coffees[i]);
updateSales.executeUpdate();
}
```

Các giá trị trả về của phương thức executeUpdate

Phương thức executeQuery trả về một đối tượng ResultSet chứa các kết quả của truy vấn được gửi tới hệ quản trị cơ sở dữ liệu, giá trị trả về khi xử lý phương thức executeUpdate là một số nguyên int chỉ ra số hàng trong bảng đã được cập nhật.

```
updateSales.setInt(1, 50);
updateSales.setString(2, "Espresso");
int n = updateSales.executeUpdate();
// n = 1 because one row had a change in it
```

8. Sử dụng các giao tác

Quản lý giao tác

Một giao tác là một tập hợp một hoặc nhiều lệnh được xử lý cùng với nhau như một chỉnh thể thống nhất (đơn vị). Khi xử lý một giao tác hoặc tất cả các lệnh được xử lý hoặc không lệnh nào được xử lý. Nhiều trường hợp ta không muốn một lệnh có hiệu lực ngay nếu lệnh khác không thành công.

Điều này có thể được thực hiện nhờ phương thức setAutoCommit() của đối tượng Connection. Phương thức này nhận một giá trị boolean làm tham số..

Ngăn chế độ Auto-commit

Khi một liên kết được tạo ra, thì liên kết đó ở chế độ auto-commit.

Mỗi lệnh SQL được xem như là một giao tác và sẽ được tự động hoàn thành ngay khi nó được xử lý.

Cách để cho phép hai hoặc nhiều lệnh được nhóm cùng với nhau thành một giao tác là cấm chế độ auto-commit.

Ví dụ:

```
con.setAutoCommit(false);
```

Xác nhận hoàn thành một giao tác

Mỗi khi chế độ auto-commit bị cấm, không có lệnh SQL nào sẽ được xác nhận hoàn thành cho tới khi ta gọi phương thức commit().

Ta có thể thực hiện điều này bằng cách gọi phương thức commit() của các đối tượng liên kết.
Nếu ta cố gắng xử lý một hay nhiều lệnh trong một giao tác và nhận được một ngoại lệ SQLException, ta cần gọi phương thức rollback() để hủy bỏ giao tác và khởi động lại toàn bộ giao tác.

```
con.setAutoCommit(false);
PreparedStatement updateName =null;
String query = null;
Query="UPDATE license SET name = ? WHERE id = 126"
updateName= con.prepareStatement(query);
updateName.setString(1, name);
updateName.executeUpdate();
PreparedStatement updateSex = null;
query = "UPDATE test SET test_value =?"
updateSex = con.prepareStatement(query);
updateSex.setString(1, "Male");
updateSex.executeUpdate();
con.commit();
con.setAutoCommit(true);
```

Ví dụ:

```
import java.sql.*;
public class PreparedUpdate{
public static void main(String args[]) throws Exception{

int empid;
int rows=0;
String LastName;
String FirstName;
String query = "insert into EMP " +"(EmployeeID,LASTNAME,FIRSTNAME) "
+"values " +"(?,?, ?)";
try {
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
Connection con =DriverManager.getConnection ("jdbc:odbc:MyData");
con.setAutoCommit(false);
PreparedStatement pstmt = con.prepareStatement(query);
pstmt.setInt(1,Integer.parseInt(args[0]));
pstmt.setString(2,args[1]);
pstmt.setString(3,args[2]);
rows = pstmt.executeUpdate( );
pstmt.close( );
```

```
                pstmt = null;
                System.out.println(rows + " rows inserted");
                System.out.println("");
                con.commit();

        }
        catch(ClassNotFoundException e){
        e.printStackTrace();

        }
        catch(SQLException e){
        e.printStackTrace();
        }
        }
        }
```

8. Sử dụng các giao tác

Sau đó thực hiện các lệnh: Các chương trình Java chỉ thực hiện được các lệnh trên CSDL thông qua đối tượng **Statement**.

Các câu lệnh SQL có thể được thực hiện tức thì thông qua đối tượng **Statement**, có thể là một câu lệnh biên dịch trước (đối tượng **PreparedStatement**) hay có thể là một lệnh gọi các thủ tục cài sẵn (Stored Procedure) trong CSDL (đối tượng **CallableStatement**).

Các câu lệnh SQL có thể được thực hiện thông qua phương thức executeQuery() – kết quả là một đối tượng ResultSet, hay phương thức executeUpdate() – kết quả là một số nguyên cho biết tổng số các record chịu ảnh hưởng của câu lệnh vừa thực hiện (thường là các câu lệnh sửa đổi dữ liệu Update - Delete).

Trong trường hợp có sử dụng trình quản lý transaction, các phương thức rollback() được dùng để phục hồi trạng thái trước đó và commit() để xác nhận việc thực hiện lệnh.

Để chấm dứt cần xóa kết nối, xóa các đối tượng để giải phóng tài nguyên của hệ thống.

.

Chương 10

TUẦN TỰ HÓA ĐỐI TƯỢNG VÀ ỨNG DỤNG TRONG LẬP TRÌNH MẠNG

1. Tuần tự hóa đối tượng

1.1. Khái niệm

Tuần tự hóa là quá trình chuyển tập hợp các thể hiện đối tượng chứa các tham chiếu tới các đối tượng khác thành một luồng byte tuyến tính, luồng này có thể được gửi đi qua một *Socket*, được lưu vào tệp tin hoặc được xử lý dưới dạng một luồng dữ liệu. Tuần tự hóa là cơ chế được sử dụng bởi RMI để truyền các đối tượng giữa các máy ảo JVM hoặc dưới dạng các tham số trong lời gọi phương thức từ client tới server hoặc là các giá trị trả về từ một lời gọi phương thức.

Tuần tự hóa là một cơ chế đã được xây dựng và được đưa vào các lớp thư viện Java căn bản để chuyển một đồ thị các đối tượng thành các luồng dữ liệu. Luồng dữ liệu này sau đó có thể được xử lý bằng cách lập trình và ta có thể tạo lại các bản sao của đối tượng ban đầu nhờ quá trình ngược lại được gọi là giải tuần tự hóa.

Tuần tự hóa có ba mục đích chính sau

- Cơ chế ổn định: Nếu luồng được sử dụng là *FileOuputStream*, thì dữ liệu sẽ được tự động ghi vào tệp.
- Cơ chế sao chép: Nếu luồng được sử dụng là *ByteArrayObjectOuput*, thì dữ liệu sẽ được ghi vào một mảng byte trong bộ nhớ. Mảng byte này sau đó có thể được sử dụng để tạo ra các bản sao của các đối tượng ban đầu.
- Nếu luồng đang được sử dụng xuất phát từ một *Socket* thì dữ liệu sẽ được tự động gửi đi tới *Socket* nhận, khi đó một chương trình khác sẽ quyết định phải làm gì đối với dữ liệu nhận được.

Một điều quan trọng khác cần chú ý là việc sử dụng tuần tự hóa độc lập với thuật toán tuần tự hóa.

1.2. Khả tuần tự (*Serializable*)

Chỉ có đối tượng thực thi giao diện *Serializable* mới có thể được ghi lại và được phục hồi bởi các tiện ích tuần tự hóa. Giao diện *Serializable* không định nghĩa các thành phần. Nếu một lớp thực thi giao diện *Serializable* thì lớp đó có khả năng tuần tự hóa. Một lớp là khả tuần tự thì tất cả các lớp con của nó cũng là khả tuần tự.

Giao diện *ObjectOutput* thừa kế từ giao diện DataOutput và hỗ trợ tuần tự hóa đối tượng. Lớp *ObjectOuputStream* là lớp con của lớp *ObjectOuput* và thực thi giao diện *ObjectOutput*. Nó có nhiệm vụ ghi các đối tượng vào một luồng bằng cách sử dụng phương thức *writeObject*(Object obj).

ObjectInput thừa kế giao diện *DataInput* và định nghĩa các phương thức. Nó hỗ trợ cho việc tuần tự hóa đối tượng. Phương thức *readObject*() được gọi để giải tuần tự hóa một đối tượng.

ObjectInputStream được định nghĩa trong gói *java.io* là một luồng cài đặt cơ chế đọc trạng thái của luồng nhập đối tượng.

Một vấn đề đặt ra là: liệu mọi lớp trong Java đều có khả năng tuần tự hóa? Câu trả lời là không, bởi vì không cần thiết hoặc sẽ không có ý nghĩa khi tuần tự hóa một số lớp nhất định. Để xác định xem một lớp có khả tuần tự hay không ta sử dụng công cụ serialver có trong bộ JDK.

Hình 1

Hình 2

Với kết quả trên cho ta thấy lớp này là khả tuần tự. Nhưng không phải mọi lớp trong Java đều khả tuần tự chẳng hạn ta thử kiểm tra với lớp *java.net.Socket*

Hình 3

Khi đó kết quả hiển thị là *Class java.net.Socket is not Serializable* (Lớp *java.net.Socket* không khả tuần tự).

1.3. Xây dựng lớp một lớp khả tuần tự

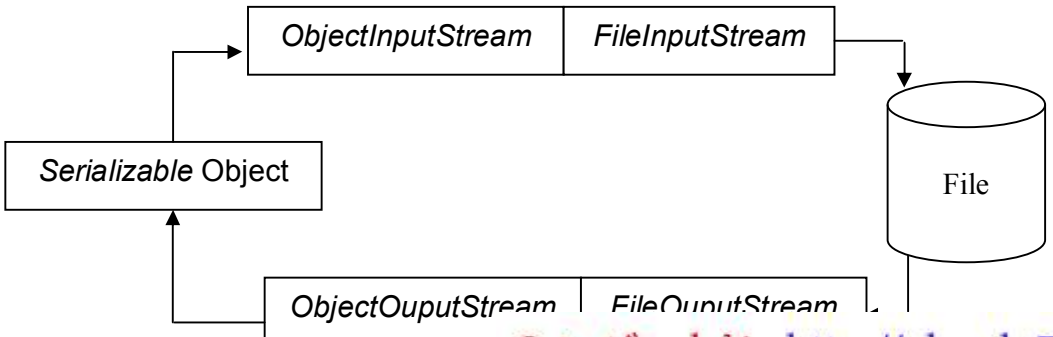
Đối với các lớp do người lập trình định nghĩa ta phải khai báo để báo hiệu cho hệ thống biết nó có khả tuần tự hay không. Một lớp do người dùng định nghĩa có khả năng tuần tự hóa khi lớp đó thực thi giao diện *Serializable*. Trong ví dụ dưới đây ta định nghĩa lớp *Point* để lớp này có khả năng tuần tự hóa.

```
public class Point implements Serializable
{
    private double x,y;
```

```
    public Point(double x,double y){
        this.x=x;
        this.y=y;
    }
    public double getX(){
        return x;
    }
    public double getY(){
        return y;
    }
    public void move(double dx,double dy){
        x+=dx;
        y+=dy;
    }
    public void print(){
        System.out.println("Toa do cua diem la:");
        System.out.println("Toa do x="+x);
        System.out.println("Toa do y="+y);
    }
}
```

1.4. Cơ chế đọc và ghi đối tượng trên thiết bị lưu trữ ngoài

Chúng ta đều biết rằng tất cả các thao tác nhập và xuất dữ liệu trong Java thực chất là việc đọc và ghi trên các luồng dữ liệu vào và luồng dữ liệu ra. Việc đọc và ghi đối tượng trên thiết bị lưu trữ ngoài cũng không phải là một ngoại lệ. Chúng ta có thể thấy được cơ chế này qua hình 4.



Hình 4

Giả sử đối tượng obj là một đối tượng khả tuần tự. Bản thân đối tượng này có thể đã là khả tuần tự hoặc do người lập trình định nghĩa nên thuộc tính khả tuần tự cho nó.

Cơ chế ghi đối tượng được tiến hành rất đơn giản: Trước tiên ta tạo ra một tệp để ghi thông tin, thực chất là tạo ra đối tượng *FileOuputStream*, sau đó ta tạo ra một luồng ghi đối tượng *ObjectOuputStream* gắn với luồng ghi tệp và gắn kết hai luồng với nhau. Việc ghi đối tượng được thực hiện bởi *phương thức writeObject().*

```
FileOuputStream fos=new FileOuputStream("date.out");
ObjectOuputStream oos=new ObjectOuputStream(fos);
Date d=new Date();
oos.writeObject(d);
```

Quá trình trên được gọi là quá trình tuần tự hóa.

Chúng ta nhận thấy rằng để phục hồi lại trạng thái của một đối tượng ta phải mở một tệp để đọc dữ liệu. Nhưng ta không thể đọc được trực tiếp mà phải thông qua luồng nhập đối tượng *ObjectInputStream* gắn với luồng nhập tệp tin *FileInputStream*. Việc đọc lại trạng thái đối tượng được tiến hành nhờ phương thức *readObject*()

```
FileInputStream fis=new FileInputStream("date.out");
ObjectInputStream ois=new ObjectInputStream(fis);
Date d=(Date)ois.readObject();
```

Quá trình trên còn được gọi là giải tuần tự hóa

Công việc đọc và ghi trạng thái của đối tượng khả tuần tự do người lập trình định nghĩa được tiến hành hoàn toàn tương tự như trên.

2. Truyền các đối tượng thông qua *Socket*

Chúng ta đã biết cách ghi và đọc các đối tượng từ các luồng vào ra trong một tiến trình đơn, bây giờ chúng ta sẽ xem xét cách truyền đối tượng thông qua *Socket*.

Mô hình lập trình *Socket* cho giao thức TCP là mô hình rất phổ biến trong lập trình mạng. Để lập chương trình client/server trong Java ta cần hai lớp *Socket* và *ServerSocket*.

2.1. Lớp Socket

Lớp *Socket* của Java được sử dụng bởi cả client và server, nó có các phương thức tương ứng với bốn thao tác đầu tiên. Ba thao tác cuối chỉ cần cho server để chờ các client liên kết với chúng. Các thao tác này được cài đặt bởi lớp *ServerSocket*. Các *Socket* cho client thường được sử dụng theo mô hình sau:

1. Một *Socket* mới được tạo ra bằng cách sử dụng hàm dựng *Socket*().
2. *Socket* cố gắng liên kết với một host ở xa.
3. Mỗi khi liên kết được thiết lập, các host ở xa nhận các luồng vào và luồng ra từ *Socket*, và sử dụng các luồng này để gửi dữ liệu cho nhau. Kiểu liên kết này được gọi là song công (full-duplex), các host có thể nhận và gửi dữ liệu đồng thời. Ý nghĩa của dữ liệu phụ thuộc vào từng giao thức.
4. Khi việc truyền dữ liệu hoàn thành, một hoặc cả hai phía ngắt liên kết. Một số giao thức, như HTTP, đòi hỏi mỗi liên kết phải bị đóng sau mỗi khi yêu cầu được phục vụ. Các giao thức khác, chẳng hạn như FTP, cho phép nhiều yêu cầu được xử lý trong một liên kết đơn.

2.2. Lớp ServerSocket

Lớp *ServerSocket* có đủ mọi thứ ta cần để viết các server bằng Java. Nó có các constructor để tạo các đối tượng *ServerSocket* mới, các phương thức để lắng nghe các liên kết trên một cổng xác định và các phương thức trả về một *Socket* khi liên kết được thiết lập, vì vậy ta có thể gửi và nhận dữ liệu.

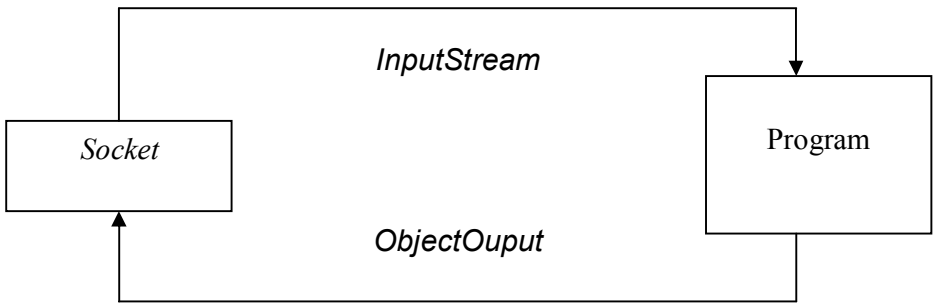
Vòng đời của một server

1. Một *ServerSocket* mới được tạo ra trên một cổng xác định bằng cách sử dụng một constructor *ServerSocket*.
2. *ServerSocket* lắng nghe liên kết đến trên cổng đó bằng cách sử dụng phương thức accept(). Phương thức accept() phong tỏa cho tới khi một client thực hiện một liên kết, phương thức accept() trả về một đối tượng *Socket* biểu diễn liên kết giữa client và server.
3. Tùy thuộc vào kiểu server, hoặc phương thức *getInputStream*(), *getOuputStream*() hoặc cả hai được gọi để nhận các luồng vào ra phục vụ cho việc truyền tin với client.
4. Server và client tương tác theo một giao thức thỏa thuận sẵn cho tới khi ngắt liên kết.
5. Server, client hoặc cả hai ngắt liên kết

Server trở về bước hai và đợi liên kết tiếp theo.

2.3. Truyền và nhận dữ liệu trong mô hình lập trình Socket

Việc truyền và nhận dữ liệu thực chất là các thao tác đọc và ghi dữ trên *Socket*. Ta có thể thấy điều này qua sơ đồ dưới đây:



Hình 5

Giả sử s là một đối tượng *Socket*. Nếu chương trình nhận dữ liệu thì ta sẽ lấy dữ liệu từ luồng nhập đến từ *Socket*:

InputStream is=s.getInputStream()

Để phục hồi trạng thái đối tượng ta gắn kết luồng nhập thô lấy được từ *Socket* với luồng đọc đối tượng *ObjectInputStream*:

ObjectInputStream ois=new ObjectInputStream(is);

Khi đó đối tượng được phục hồi lại trạng thái bởi câu lệnh:

Object obj=(Object)ois.readObject();

Nếu chương trình gửi dữ liệu thì ta sẽ lấy dữ liệu từ luồng xuất đến từ *Socket*:

ObjectOuput os=s.getObjectOuput();

Để tiến hành ghi đối tượng ta gắn kết luồng xuất thô lấy được từ *Socket* với luồng xuất đối tượng *ObjectOuputStream*:

ObjectOuputStream oos=new *ObjectOutputStream*(os);

Việc truyền đối tượng lúc này trở thành một công việc rất đơn giản:

oos.writeObject(obj);

oos.flush();

2.4. Ví dụ minh họa

Để minh họa kỹ thuật chúng ta viết một server thực hiện phép nhân hai mảng số nguyên với nhau. Client gửi hai đối tượng, mỗi đối tượng biểu diễn một mảng nguyên; server nhận các đối tượng này, thực hiện lời gọi phương nhân hai mảng số nguyên với nhau và gửi kết quả trả về cho client.

Trước tiên chúng ta định nghĩa đối tượng để có thể sử dụng trong việc truyền các đối tượng.

```
public class ArrayObject implements java.io.Serializable{
    private int[] a=null;
    public ArrayObject(){
    }
    public void setArray(int a[]){
        this.a=a;
    }
    public int[] getArray(){
        return a;
    }
}
```

Lớp *ArrayObject* là lớp được xây dựng để đóng gói các mảng số nguyên và có khả năng truyền đi qua lại trên mạng. Cấu trúc lớp như sau: trường thông tin là một mảng số nguyên a[]; phương thức setArray() thiết lập giá trị cho mảng. Phương thức getArray() trả về một mảng số nguyên từ đối tượng *ArrayObject*.

Mô hình client/server tối thiểu phải có hai mođun client và server. Trong ví dụ này cũng vậy ta sẽ xây dựng một số mođun chương trình như sau:

Đầu tiên chúng ta phát triển client. Client tạo ra hai thể hiện của các đối tượng *ArrayObject* và ghi chúng ra luồng xuất (thực chất là gửi tới server).

```
public class ArrayClient{
    public static void main(String[] args)throws Exception{
        ObjectOuputStream oos=null;
        ObjectInputStream ois=null;
        int dat1[]={3,3,3,3,3,3,3};
        int dat2[]={5,5,5,5,5,5,5};
            Socket s=new Socket("localhost",1234);
            oos=new ObjectOuputStream(s.getObjectOuput());
            ois=new ObjectInputStream(s.getInputStream());
            ArrayObject a1=new ArrayObject();
            a1.setArray(dat1);

            ArrayObject a2=new ArrayObject();
            a2.setArray(dat2);
            ArrayObject res=null;

            int r[]=new int[7];
            oos.writeObject(a1);
            oos.writeObject(a2);
            oos.flush();
            res=(ArrayObject)ois.readObject();
            r=res.getArray();
            System.out.println("The result received from server...");
            System.out.println();
            for(int i=0;i<r.length;i++)System.out.print(r[i]+" ");
        }
}
```

Bước tiếp theo chúng ta phát triển server. Server là một chương trình cung cấp dịch vụ phục vụ các yêu cầu của client. Server nhận hai đối tượng *ArrayObject* và nhận về hai mảng từ hai đối tượng này và sau đó đem nhân chúng với nhau và gửi kết quả trở lại cho client.

```
public class ArrayServer extends Thread {
	private ServerSocket ss;
	public static void main(String args[])throws Exception
	{
		new ArrayServer();
	}
	public ArrayServer()throws Exception{
		ss=new ServerSocket(1234);
		System.out.println("Server running on port "+1234);
		this.start();
	}
	public void run(){
		while(true){
			try{
				System.out.println("Waiting for client...");
				Socket s=ss.accept();
				System.out.println("Accepting a connection from:"+s.getInetAddress());
				Connect c=new Connect(s);
			}
			catch(Exception e){
				System.out.println(e);
			}
		}
	}
}
```

Trong mô hình client/server tại một thời điểm server có thể phục vụ các yêu cầu đến từ nhiều client, điều này có thể dẫn đến các vấn đề tương tranh. Chính vì lý do này mà lớp *ArrayServer* thừa kế lớp *Thread* để giải quyết vấn đề trên. Ngoài ra để nâng cao hiệu suất của chương trình thì sau khi đã chấp nhận liên kết từ một client nào đó, việc xử lý dữ liệu sẽ được dành riêng cho một tuyến đoạn để server có thể tiếp tục chấp nhận các yêu cầu khác. Hay nói cách khác, mỗi một yêu cầu của client được xử lý trong một tuyến đoạn riêng biệt.

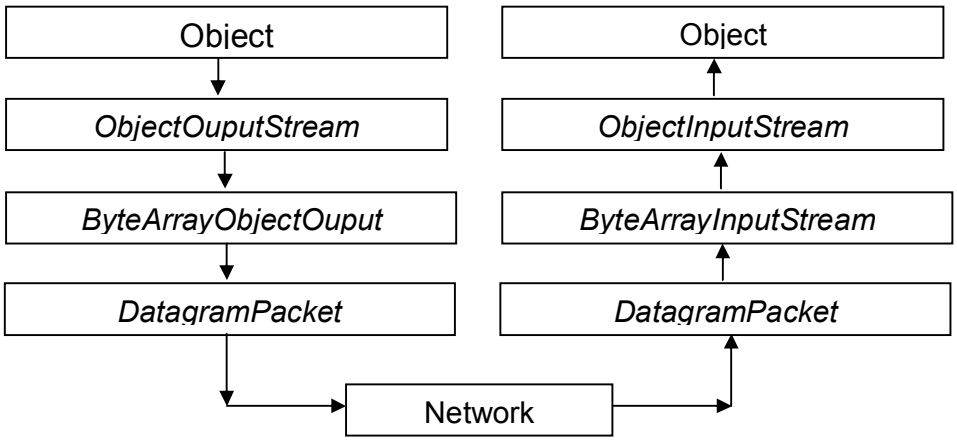
```
class Connect extends Thread{
	private Socket client=null;
	private ObjectInputStream ois;
```

```
private ObjectOuputStream oos;
public Connect(){
}
public Connect(Socket client){
        this.client=client;
        try{
                ois=new ObjectInputStream(client.getInputStream());
                oos=new ObjectOuputStream(client.getObjectOuput());
        }
        catch(Exception e){
                System.err.println(e);
        }
        this.start();
}
public void run(){
        ArrayObject x=null;
        ArrayObject y=null;
        int a1[]=new int[7];
        int a2[]=new int[7];
        int r[]=new int[7];
        try{
                x=(ArrayObject)ois.readObject();
                y=(ArrayObject)ois.readObject();
                a1=x.getArray();
                a2=y.getArray();
                for(int i=0;i<a1.length;i++)r[i]=a1[i]*a2[i];
                ArrayObject res=new ArrayObject();
                res.setArray(r);
                oos.writeObject(res);
                oos.flush();
                ois.close();
                client.close();
                }
        catch(Exception e){
                }
```

```
        }
}
```

3. Truyền các đối tượng thông qua giao thức UDP

Một giao thức gần với giao thức TCP là giao thức UDP. Java hỗ trợ cho kiểu ứng dụng truyền tin phi liên kết trên giao thức UDP thông qua lớp *DatagramSocket* và *DatagramPacket*. Liệu chúng ta có thể viết được các chương trình nhập và xuất đối tượng bằng truyền tin datagram? Thực hiện điều này không thể tiến hành trực tiếp như với luồng *Socket*. Vấn đề là *DatagramSocket* không được gắn với bất kỳ luồng nào; mà nó sử dụng một tham số mảng byte để gửi và nhận dữ liệu.



Hình 6

Có thể thấy rằng để xây dựng một gói tin datagram, đối tượng phải được chuyển thành một mảng byte. Việc chuyển đổi này rất khó để thực hiện nếu bản thân đối tượng có liên quan đến một số đối tượng phức tạp trong đồ thị đối tượng.

Hình 6 minh họa dòng luân chuyển dữ liệu khi truyền một đối tượng thông qua một datagram. Dưới đây là bảy bước ta cần thực hiện để cài đặt mô hình truyền dữ liệu cho giao thức UDP

- Bước 1. Chuẩn bị: Tạo đối tượng cần truyền đi, giả sử đối tượng này là obj, làm cho nó khả tuần tự bằng cách thực thi giao tiếp *Serializable*.
- Bước 2. Tạo một luồng *ByteArrayObjectOuput* và đặt tên cho nó là baos.
- Bước 3. Xây dựng đối tượng *ObjectOuputStream* và đặt tên cho nó là oos. Tham số cho cấu tử *ObjectOuputStream* là baos
- Bước 4. Ghi đối tượng obj vào luồng baos bằng cách sử dụng phương thức *writeObject*() của oos.
- Bước 5. Tìm kiếm vùng đệm dữ liệu mảng byte từ bằng cách sử dụng phương thức *toByteAray*().

- Bước 6. Xây dựng đối tượng *DatagramPacket* và đặt tên là dp với dữ liệu đầu vào là vùng đệm dữ liệu đã tìm được ở bước 5.
- Bước 7. Gửi dp thông qua *DatagramSocket* bằng cách gọi phương thức *send*() của nó.

Ví dụ minh họa chi tiết quá trình gửi một đối tượng

```
InetAddress ia=InetAddress.getByName("localhost");
Student st=new Student("Peter",7,8,9);
DatagramSocket ds=new DatagramSocket();
ByteArrayObjectOuput baos=new ByteArrayObjectOuput(5000);
ObjectOuputStream oos=new ObjectOuputStream(new
BufferedObjectOuput(baos));
oos.flush();
oos.writeObject(st);
oos.flush();
byte[] b=baos.toByteAray();
DatagramPacket dp=new DatagramPacket(b,b.length,ia,1234);
ds.send(dp);
oos.close();
```

Để nhận một đối tượng ta cũng tiến hành các bước như trên nhưng theo thứ tự ngược lại, thay thế luồng *ObjectOuputStream* bằng *ObjectInputStream* và *ByteArrayObjectOuput* bằng *ByteArrayInputStream*.

Ví dụ dưới đây minh họa chi tiết quá trình nhận một đối tượng

```
DatagramSocket ds=new DatagramSocket(1234);
    while(true){
            byte b[]=new byte[5000];
            DatagramPacket dp=new DatagramPacket(b,b.length);
            ds.receive(dp);
            ByteArrayInputStream bais=new
             ByteArrayInputStream(new BufferedInputStream(b));
            ObjectInputStream ois =new ObjectInputStream(bais);
            Student st=(Student)ois.readObject();
            st.computeAverage();
            st.print();
            ois.close();
            bais.close();
  }
```

4. Kết luận

Qua bài báo này tôi đã giới thiệu tổng quan về tuần tự hóa đối tượng. Thông qua các ví dụ chúng ta thấy không quá khó để làm việc với tuần tự hóa đối tượng và điều quan trọng hơn là chúng ta đã biết cách để truyền đi các đối tượng có cấu trúc phức tạp thông qua các *Socket*.

Ngoài ra, bài báo cũng đã đề cập tới cách truyền đối tượng bằng cách sử dụng các gói tin datagram. Nhờ những ưu điểm của tiện ích tuần tự hóa đối tượng, tôi đã minh họa một cách truyền các đối tượng bằng cách sử dụng các gói tin datagram. Như chúng ta đã thấy, mặc dù trong giao thức này không hỗ trợ xử lý theo luồng dữ liệu nhưng tôi đã "luồng hóa" các đối tượng để đưa các đối tượng vào các mảng byte.

Sự lựa chọn giữa việc sử dụng RMI hay giải pháp *Socket* kết hợp với tuần tự hóa phụ thuộc vào từng dự án và các yêu cầu của nó. Sự lựa chọn giải pháp nào chính là sự thỏa hiệp giữa các đặc trưng của mỗi giải pháp: nếu đối với RMI thì đó là tính đơn giản khi triển khai, ngược lại với *Socket* kết hợp với tuần tự hóa đối tượng thì đó lại là ưu thế về mặt hiệu năng. Nếu vấn đề hiệu năng có tầm quan trọng thì giải pháp lập trình *Socket* kết hợp tuần tự hóa đối tượng là giải pháp tốt hơn so với RMI.

TÀI LIỆU THAM KHẢO

[1] Elliotte Rusty Harold, ***Java Network Programming***

[2] Nguyễn Phương Lan- Hoàng Đức Hải, ***Java lập trình mạng***, Nhà xuất bản Giáo dục

[3] Darrel Ince & Adam Freemat, ***Programming the Internet with Java***, Addison-Wesley

[4] Mary Campione&Kathy Walrath&Alison Huml, ***Java™ Tutorial, Third Edition: A Short Course on the Basics***, Addison Wesley

[5] The Complete Java 2Reference

[6] Nguyễn Thúc Hải, ***Mạng máy tính và các hệ thống mở***, Nhà xuất bản Giáo dục

[7] Đoàn Văn Ban***, Lập trình hướng đối tượng với Java***, Nhà xuất bản Khoa học và Kỹ thuật

Tài liệu tham khảo

[1] **Douglas E.Comer, David L.Stevens**, *Client-Server Programming And Applications*. In book: Internetworking with TCP/IPVolume III, Pearson Education, Singapore, 2004.

[2] **Herbert Schildt**, *Java™ 2: The Complete Reference Fifth Edition*, Tata McGraw-Hill Publishing Company Limited, India, 2002.

[3] **Elliote Rusty Harold**, *Java™ Network Programming, Third Edition, Oreilly*, 2005.

[4] **Qusay H. Mahmoud**, *Advanced Socket Programming*, http://java.sun.com, December 2001

[5] **Shengxi Zhou**, *Transport Java objects over the network with datagram packets*, http://www.javaworld.com, 2006