HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG (PTIT)

GIÁO TRÌNH

LẬP TRÌNH MẠNG

(PHẦN LẬP TRÌNH MẠNG CƠ SỞ)

HÀ MẠNH ĐÀO

HÀ NỘI, 07/2010

MỞ ĐẦU

Ngày nay do nhu cầu thực tế và do sự phát triển mạnh mẽ của nhiều công nghệ tích hợp, dẫn đến các chương trình ứng dụng hiện này hầu hết đều có khả năng thực hiện trên môi trường mạng máy tính nói riêng và mạng tích hợp nói chung. Chính vì vậy giáo trình này nhằm cung cấp cho sinh viên những kiến thức và kỹ thuật cơ bản nhất để phát triển các chương trình ứng dụng mạng. Giáo trình này bao gồm 4 phần lớn và 7 chương: Phần thứ nhất trình bày các kiến thức cơ sở cho lập trình mạng, chủ yếu là kiến thức mạng máy tính, ngôn ngữ lập trình và mô hình lập trình mạng. Phần 2 và 3 cung cấp cho sinh viên 2 kỹ thuật lập trình cơ bản nhất và phổ biến nhất hiện này là lập trình mạng với socket và lập trình phân tán thông qua ngôn ngữ Java. Đồng thời cũng rèn sinh viên cách lập trình với giao thức truyền thông có sẵn và khả năng tích hợp trong các ứng dụng khác nhau, nhất là các giao thức truyền thông thời gian thực(RTP). Phần 4 sẽ đề cập đến lập trình truyền thông qua mạng điện thoại công cộng, để sinh viên bước đầu làm quen với kỹ thuật lập trình cơ bản truyền thông qua hệ thống mạng này. Từ đó sinh viên dễ dàng tiếp cận phát triển các ứng dụng trên cơ sở mạng này như hội thoại video, các dịch vụ truy cập từ xa, VPN, IPTV.. và nói chung là công nghệ IP. Phần 5 cung cấp cho sinh viên làm quen kiến thức lập trình mạng an toàn bảo mật mà chủ yếu là giao thức SSL. Cách bố trí của chúng tôi thành từng phần rõ ràng, mỗi phần có thể có một hoặc nhiều chương với mục đích hướng mở cho từng phần trong tương lai.

Để nắm được kiến thức lập trình mạng, sinh viên phải học qua kiến thức các môn: Mạng máy tính, lập trình OOP, phân tích và thiết kế hệ thống, ngôn ngữ lập trình java cơ bản.

Giáo trình biên soạn phiên bản đầu, chắc không tránh khỏi lỗi, rất mong nhận được ý kiến đóng góp của đồng nghiệp và những độc giả quan tâm.

Xin chân thành cảm ơn!

Hà Nội, tháng 07 năm 2010

Tác giả

MỤC LỤC

PHẦN I. KIẾN THỨC CƠ SỞ CHO LẬP TRÌNH

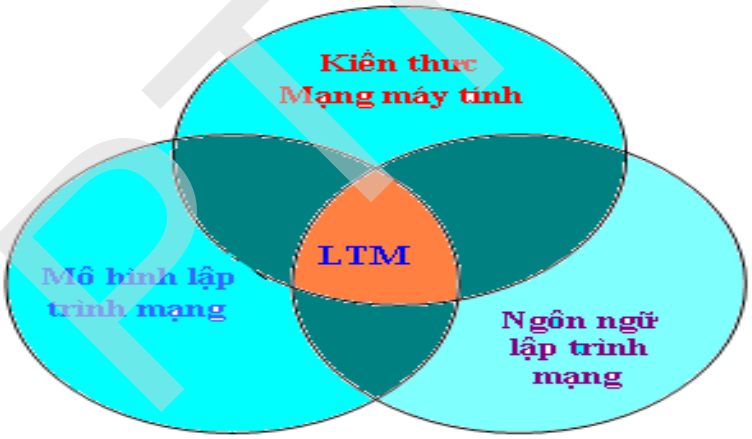
CHƯƠNG I

MỘT SỐ KIẾN THỨC CƠ SỞ CHO LẬP TRÌNH

I. GIỚI THIỆU VỀ LẬP TRÌNH MẠNG(LTM)

Ngày này khi nói đến phát triển các ứng dụng phần mềm, đa số là người ta muốn nói đến chương trình có khả năng làm việc trong môi trường mạng tích hợp nói chung và mạng máy tính nói riêng. Từ các chương trình kế toán doanh nghiệp, quản lý, trò chơi, điều khiển... đều là các chương trình ứng dụng mạng.

Vấn đề lập trình mạng liên quan đế nhiều lĩnh vực kiến thức khác nhau. Từ kiến thức sử dụng ngôn ngữ lập trình, phân tích thiết kế hệ thống, kiến thức hệ thống mạng, mô hình xây dựng chương trình ứng dụng mạng, kiến thức về cơ sở dữ liệu... cho đến kiến thức truyền thông, các kiến thức các lĩnh vực liên quan khác như mạng điện thoại di động, PSTN, hệ thống GPS, các mạng như BlueTooth, WUSB, mạng sensor.... Nhưng có thể nói vấn đề lập trình mạng có 3 vấn đề chính cốt lõi tích hợp trong lập trình ứng dụng mạng và được thể hiện như hình 1.



Hình 1.1. Các kiến thức cơ sở cho lập trình mạng

Hay nói cách khác, vấn đề lập trình mạng có thể được định nghĩa với công thức sau:

LTM=KTM+MH+NN

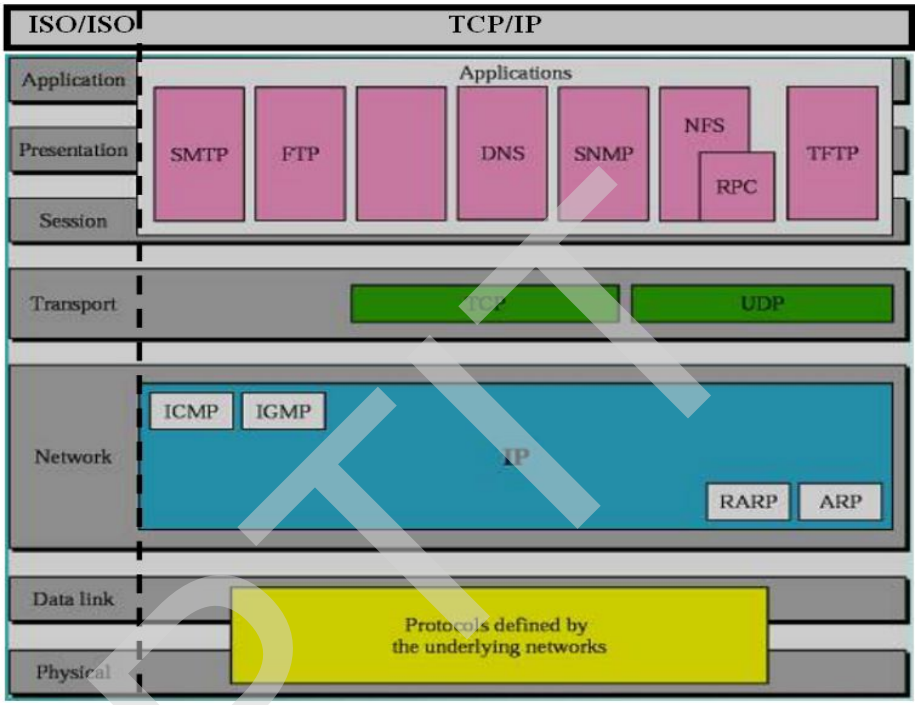
Trong đó:

- LTM: Lập trình mạng
- KTM: Kiến thức mạng truyền thông(mạng máy tính, PSTN....)
- MH: Mô hình lập trình mạng
- NN: Ngôn ngữ lập trình mạng

Trong giao trình này, chúng tôi tập trung chủ yếu vào các kỹ thuật phát triển chương trình ứng dụng mạng. Còn các vấn đề khác can thiệp sâu xuống phía thấp hơn trong hệ thống mạng như các trình tiện ích mạng, thu thập bắt và phân tích gói tin...các bạn có thể tham khảo các tài liệu khác, nhất là các tài liệu liên quan đến lập trình với Raw socket.

II. MỘT SỐ KIẾN THỨC MẠNG CƠ SỞ LẬP TRÌNH MẠNG

1. Mô hình OSI/ISO và họ giao thức TCP/IP



Hình 1.2. Mô hình OSI/ISO và họ giao thức TCP/IP

1.2. Giao thức truyền thông và phân loại(protocol)

Giao thức truyền thông là tập các qui tắc, qui ước mà mọi thực thể tham ra truyền thông phải tuân theo để mạng có thể hoạt động tốt. Hai máy tính nối mạng muốn truyền thông với nhau phải cài đặt và sử dụng cùng một giao thức thì mới "hiểu" nhau được.

Dựa vào phương thức hoạt động, người ta có thể chia giao thức truyền thông thành 2 loại: Giao thức hướng kết nối và giao thức hướng không kết nối.

1.2.1. Giao thức hoạt động theo hướng có kết nối

Loại giao thức truyền thông này sử dụng kết nối(ảo) để truyền thông. Đặc điểm của loại giao thức này là:

- Truyền thông theo kiểu điểm-điểm
- Dữ liệu truyền qua mạng là một dòng các byte liên tục truyền từ nơi gửi tới nơi nhận, mỗi byte có một chỉ số xác định.

- Quá trình truyền thông được thực hiện thông qua 3 giai đoạn:
    - ❖ Thiết lập kết nối
    - ❖ Truyền dữ liệu kèm theo cơ chế kiểm soát chặt chẽ
    - ❖ Huỷ bỏ kết nối
- Giao thức tiêu biểu là giao thức TCP

1.2.2. Giao thức hoạt động hướng không kết nối

Kiểu giao thức này khi thực hiện truyền thông không cần kết nối (ảo) để truyền dữ liệu. Giao thức kiểu này có đặc điểm sau:

- Truyền thông theo kiểu điểm-đa điểm
- Quá trình truyền thông chỉ có một giai đoạn duy nhất là truyền dữ liệu, không có giai đoạn thiết lập kết nối cũng như huỷ bỏ kết nối.
- Dữ liệu truyền được tổ chức thành các tin gói tin độc lập, trong mỗi gói dữ liệu có chứa địa chỉ nơi nhận.
- Giao thức tiêu biểu loại này là giao thức UDP

1.2.3. Một số giao thức truyền thông Internet phổ biến

- Giao thức tầng Internet: IP, ARP, RARP, ICMP, IGMP
- Giao thức tầng giao vận: TCP, UDP
- Giao thức dịch vụ: Telnet, FTP, TFTP, SMTP, POP3, IMAP4, DNS, HTTP...

1.3. Địa chỉ IP, mặt nạ(mask)

1.3.1. Địa chỉ IP

Hai phiên bản địa chỉ IP thông dụng: IPv4 và IPv6. Hiện thế giới cũng như Việt Nam đang chuyển dần sang sử dụng IPv6.

1.3.2. Mặt nạ(mask)

Mặt nạ là một giá trị hằng(một số nhị phân 32 bít) cho phép phân tách địa chỉ mạng từ địa chỉ IP(địa chỉ đầu khối địa chỉ IP). Cụ thể khi cho bất kỳ một địa chỉ IP nào trong khối địa chỉ, bằng cách thực hiện phép toán AND mức bít, mặt nạ sẽ giữ nguyên phần netid và xoá toàn bộ các bít phần hostid về giá trị 0, tức là trả về địa chỉ đầu khối địa chỉ đó. Mặt nạ của một mạng con có thể là mặt nạ có chiều dài cố định hoặc biến đổi. Các mặt nạ mặc định của các lớp địa chỉ A, B, C tương ứng là: 255.0.0.0, 255.255.0.0, 255.255.255.0. Trong kỹ thuật chia một mạng thành nhiều mạng con(subnet), hoặc để tạo thành siêu mạng(supernet) đối với lớp C, người ta phải tìm được mặt nạ mạng và định danh cho các mạng đó bằng cách mượn một số bít phần hostid(subnet) hoặc phần netid(supernet). Mặt nạ có vai trò quan trọng trong việc định tuyến cho một gói tin đi đến đúng mạng đích

1.3.3. Một số địa chỉ IP đặc biệt

- Địa chỉ mạng: nettid là định danh của mạng, các bít hostid đều băng 0.

- Địa chỉ Broadcast trực tiếp: Là địa chỉ đích, có phần netid của mạng, các bít phần hostid đều có giá trị 1.
- Địa chỉ Broadcast hạn chế: Là địa chỉ đích và có tất cả các bít phần netid và hostid đều có giá trị 1. Gói tin có địa chỉ này sẽ bị chặn bởi các router.

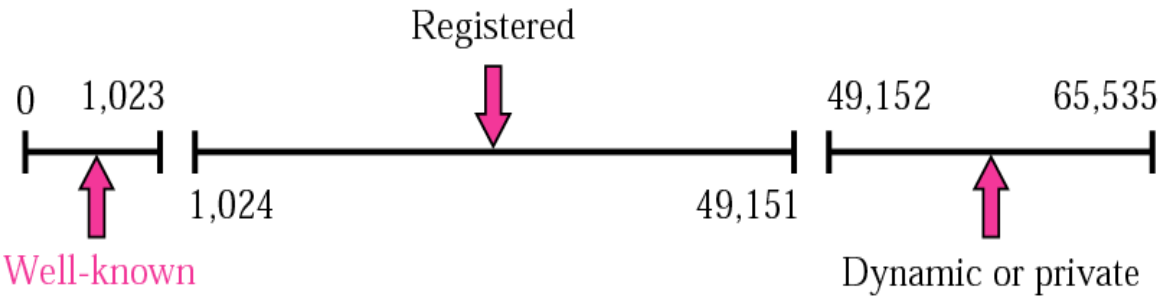
Địa chỉ *this host on this network*: có tất cả các bít netid và hostid đều bằng 0. Địa chỉ này là địa chỉ nguồn được máy trạm sử dụng tại thời điểm Bootstrap để truyền thông khi nó biết địa chỉ IP của nó.

- Địa chỉ máy trạm cụ thể trong một mạng: có tất cả các bít netid bằng 0 và phần hostid là địa chỉ host cụ thể trong mạng.
- Địa chỉ Loopback: Địa chỉ này có byte đầu tiên là 127, còn các byte còn lại có thể có giá trị bất kỳ: 127.X.Y.Z. Địa chỉ này được dùng để chạy thử các chương trình ứng dụng mạng trên cùng một máy, nhất là khi không có mạng. Địa chỉ loopback là địa chỉ đích, khi địa chỉ này được sử dụng, gói tin sẽ không bao giờ truyền ra khỏi máy. Địa chỉ loopback tiêu biểu là 127.0.0.1 hoặc có thể dùng chuỗi "localhost" thay thế.
- Địa chỉ riêng: Một số khối địa chỉ trong các lớp được qui định chỉ sử dụng cho mạng riêng(mạng cục bộ) mà không được phép sử dụng trên mạng Internet. Khi các gói tin truyền thông trên mạng Internet, các router và switch trên mạng xương sống Internetđược cấu hình loại bỏ gói tin sử dụng các địa chỉ trong các khối địa chỉ riêng này. Các dải địa chỉ riêng:
    - Lớp A: 10.0.0.0 -> 10.255.255.255
    - Lớp B: 172.16.0.0 -> 172.31.255.255
    - Lớp C: 192.168.0.0 -> 192.168.255.255

Ngoài ra người ta còn sử dụng các địa chỉ không theo lớp mà cho các khối địa chỉ có chiều dài biến đổi, các địa chỉ này có dạng CIDR: a.b.c.d/n.

1.4. Địa chỉ cổng(port)

Đa số các hệ điều hành mạng hiện nay đều đa nhiệm nên cho phép nhiều tiến trình truyền thông chạy đồng thời trên cùng một máy tính và đều chung một địa chỉ IP. Chính vì như vậy, 2 tiến trình trên 2 máy tính muốn truyền thông với nhau mà chỉ sử dụng địa chỉ IP là chưa thể thực hiện được. Để phân biệt các tiến trình chạy trên cùng một máy tính đồng thời, người ta gán cho mỗi tiến trình một nhãn duy nhất để phân biệt các tiến trình với nhau. Trong kỹ thuật mạng máy tính, người ta sử dụng một số nguyên 16 bít để làm nhãn và nó được gọi là số hiệu cổng hoặc địa chỉ cổng(port). Địa chỉ cổng này được sử dụng và được quản lý bởi tầng giao vận và nó có giá trị từ 0 đến 65535, được chia làm 3 giải:



Hình 1.3. Các dải địa chỉ cổng

- Giải địa chỉ từ 0 đến 1023: Gải này dùng cho hệ thống, người sử dụng không nên dùng. Các địa chỉ cổng trong dải này thường được gán mặc định cho các giao thức truyền thông phổ biến như bảng sau:

port	Giao thức	Mô tả
7	Echo	Phản hồi Datagram nhận được trở lại nơi gửi
9	Discard	Loại bỏ mọi Datagram nhận được
13	Daytime	Trả về ngày và giờ
19	Chargen	Trả về một chuỗi ký tự
20	FTP,Data	Phía server FTP(Kết nối dữ liêu)
21	FTP,Control	Phía server FTP(Kết nối điều khiển)
23	Telnet	Mạng đầu cuối
25	SMTP	Giao thức gửi thư Internet
53	DNS	Giao thức DNS
67	BOOTP	Giao thức Bootrap
79	Finger	Finger
80	HTTP	Giao thức truyền siêu văn bản
111	RPC	Giao thức gọi thủ tục từ xa
110	POP3	Giao thức truy cập Email
143	IMAP4	Giao thức truy cập Email

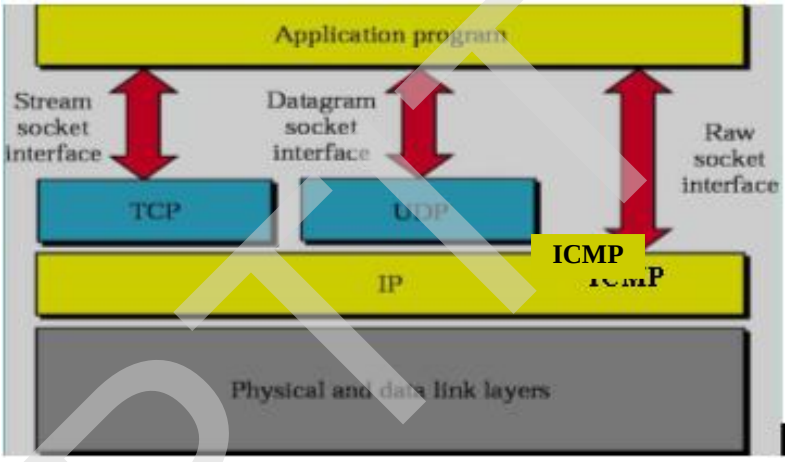
- Giải địa chỉ từ 1024 đến 49151: Giải địa chỉ cổng này người sử dụng được phép dùng, nhưng phải đăng ký để tránh trùng lặp.

- ❖ Giải địa chỉ từ 49152 đến 65535: Đây là giải địa chỉ động hoặc dùng riêng. Người sử dụng dùng địa chỉ trong giải này không phải đăng ký và cũng không phải chịu trách nhiệm khi xảy ra xung đột địa chỉ.

1.5. Giao diện socket, địa chỉ socket

Socket là gì? Chúng ta có thể hiểu socket là giao diện và là một cấu trúc truyền thông đóng vai trò như là một điểm cuối(end point) để truyền thông. Mỗi tiến trình khi muốn truyền thông bằng socket, đầu tiên nó phải tạo ra một socket và socket đó phải được gán một định danh duy nhất được gọi là địa chỉ socket. Một địa chỉ socket là một tổ hợp gồm 2 địa chỉ: địa chỉ IP và địa chỉ cổng(port). Như vậy địa chỉ socket xác định một đầu mút cuối truyền thông. Nó chỉ ra tiến trình truyền thông nào(port) và chạy trên trên máy nào(IP) sẽ thực hiện truyền thông.

Để hỗ trợ người phát triển ứng dụng mạng sử dụng socket, các nhà sản xuất phần mềm đã xây dựng sẵn một tập các hàm thư viện API và gọi là tập hàm thư viện giao diện socket. Giao diện socket được phân làm 3 loại socket(hình 2).



Hình 1.4. Các kiểu giao diện socket

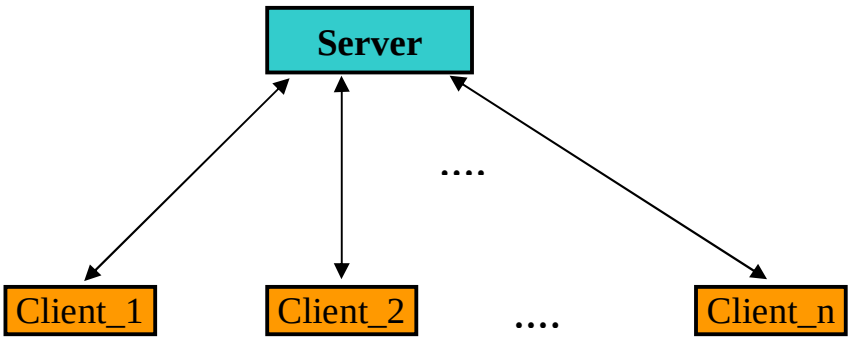
- ❖ Stream socket: cho phép truyền thông với các giao thức truyền thông hướng kết nối mà tiêu biểu là giao thức TCP(TCPSocket). TCP sử dụng một cặp stream socket để kết nối một chương trình ứng dụng với một chương trình ứng dụng khác qua mạng Internet.
- ❖ Datagram socket: Cho phép truyền thông với các giao thức hướng không kết nối, tiêu biểu là giao thức UDP (UDP socket). UDP sử dụng một cặp datagram socket để gửi thông điệp từ một chương trình ứng dụng tới một chương trình ứng dụng khác qua mạng Internet.
- ❖ Raw socket: Đây là kiểu giao socket cho phép truyền thống đến các giao thức ở tầng mạng thập hơn cả tầng giao vậnmà tiêu biểu nhất là giao thức ICMP của tầng Internet hoặc OSPF. Ví dụ chương trình ping sử dụng kiểu socket này.

II. CÁC MÔ HÌNH LẬP TRÌNH MẠNG

1. Mô hình client/server

Chương trình ứng dụng mạng tổ chức theo mô hình client/server được sử dụng phổ biến trong thực tế. Chương trình ứng dụng mạng theo mô hình này gồm có 2 phần mềm: Phần mềm

server(phục vụ) và phần mềm client(máy khách) và nó thể hiện như hình 2. Một chương trình server có thể phục vụ nhiều chương trình client đồng thời hoặc tuần tự(kiểu lặp).



Hình 1.5. Mô hình client/server

1.1. Chương trình client: client là một chương trình chạy trên máy cục bộ mà đưa ra yêu cầu dịch vụ đối với server. Chương trình client có thời gian chạy hữu hạn. Nó được khởi đầu bởi người sử dụng(hoặc một chương trình ứng dụng khác) và kết thúc khi dịch vụ đã thực hiện hoàn thành. Sau khi khởi tạo, client thực hiện mở một kênh truyền thông sử dụng địa chỉ IP của máy trạm từ xa và địa chỉ cổng(nhãn) đã biết rõ của chương trình server cụ thể chạy trên máy tính từ xa đó. Cách mở đó của client được gọi là mở tích cực(active open). Sau khi kênh truyền thông được mở client sẽ gửi yêu cầu tới server và nhận đáp ứng trả về từ server.

1.2. Chương trình server: Chương trình này có đặc điểm là có thời gian chạy vô tận và chỉ dừng chạy bởi người sử dụng hoặc tắt máy tính. Chương trình này sau khi khởi tạo, nó sẽ thực hiện mở thụ động(passive Open) và được đặt ở trạng thái "nghe" chờ tín hiệu gửi tới từ client, nếu có, nó sẽ nhận yêu cầu gửi tới từ client, thực hiện xử lý và đáp ứng yêu cầu đó.

2. Mô hình peer-to-peer

Chương trình ứng dụng mạng làm việc theo mô hình peer-to-peer(ngang cấp, bình đẳng) có thể nói là các chương trình mà có thể thực hiện vai trò của cả server và của client. Chương trình này khi chạy có thể yêu cầu chương trình khác phục vụ nó và nó cũng có thể phục vụ yêu cầu gửi tới từ chương trình khác.

3. Mô hình đa tầng

Mô hình đa tầng gồm nhiều tầng mà tiêu biểu nhất là mô hình 3 tầng. Trong mô hình này, tầng thấp nhất là tầng thông tin, tầng trung gian và tầng đỉnh. Một ví dụ tiểu biểu của mô hình 3 tầng đó là dịch vụ Web với tầng đỉnh là trình duyệt, tầng trung gian là webserver và tầng thông tin là cơ sở dữ liệu. Mô hình nhiều tầng sẽ được khảo sát kỹ trong phần lập trình ứng dụng mạng nâng cao với các kỹ thuật Servlet, EJB, Portlet..

III . NGÔN NGỮ LẬP TRÌNH MẠNG

1. Giới thiệu chung

Nói chung tất cả các ngôn ngữ lập trình đều có thể sử dụng để lập trình mạng. Nhưng mỗi ngôn ngữ có những ưu, nhược điểm khác nhau và được hỗ trợ thư viện API ở các mức độ khác nhau. Tuỳ từng ứng dụng mạng cụ thể, hệ điều hành mạng cụ thể và thói quen lập trình mà người lập trình có thể chọn ngôn ngữ phù hợp để phát triển các ứng dụng mạng. Các ngôn ngữ lập trình phổ biến hiện nay gồm những ngôn ngữ sau:

- Hợp ngữ(Assembly Language)
- C/C^{++}
- VC^{++}, VB, Delphi
- Java
- .NET
- ASP

Đối với phát triển ứng dụng mạng hiện nay có 2 ngôn ngữ lập trình được sử dụng phổ biến nhất, đó là .NET và JAVA. Người lập trình có thể sử dụng thành thạo một trong 2 dòng ngôn ngữ đó để phát triển ứng dụng mạng(ở với Việt Nam nói chung nên nắm tốt cả 2 công nghệ này). Trong giáo trình này chúng tôi sẽ sử dụng ngôn ngữ lập trình JAVA và các công nghệ liên quan đến nó để phát triển ứng dụng mạng. Sau khi nắm chắc kỹ thuật, tư tưởng lập trình mạng thông qua ngôn ngữ Java, sinh viên có thể sử dụng bất kể ngôn ngữ lập trình nào phù hợp như VB.NET, C#, ...

2. Lập trình mạng bằng ngôn ngữ Java

Để lập trình mạng bằng ngôn ngữ Java, sinh viên phải nắm chắc một số kiến thức lập trình java sau:

- Tổng quan công nghệ Java, các gói thư viện(J2SE, J2ME, J2EE)
- Lập trình Java cơ sở
- Lập trình Java OOP
- Lập trình giao diện đồ hoạ người sử dụng(GUI) và applet
- I/O theo luồng và thao tác tệp
- Lập trình kết nối với cơ sở dữ liệu
- Kỹ thuật lập trình đa luồng
- Ngoại lệ và xử lý ngoại lệ
- Lập trình an toàn bảo mật trong Java

Ngoài ra sinh viên còn phải hiểu về máy ảo java dành cho các ứng dụng java khác nhau(JVM, KVM, máy ảo cho dòng SPOT...).

IV. KỸ THUẬT LẬP TRÌNH MẠNG

Có nhiều kỹ thuật lập trình mạng khác nhau, nhưng trong giáo trình này chủ yếu chỉ tập trung vào 3 kỹ thuật lập trình mạng chính:

- Kỹ thuật lập trình mạng với socket: Trong kỹ thuật này, chương trình ứng dụng mạng sẽ được xây dựng với các kiểu socket khác nhau. Kỹ thuật này cho phép mối

quan hệ qua mạng giữa các chương trình chạy lỏng lẻo vì bản thân socket là giao diện mạng , không phải cơ chế truyền thông.

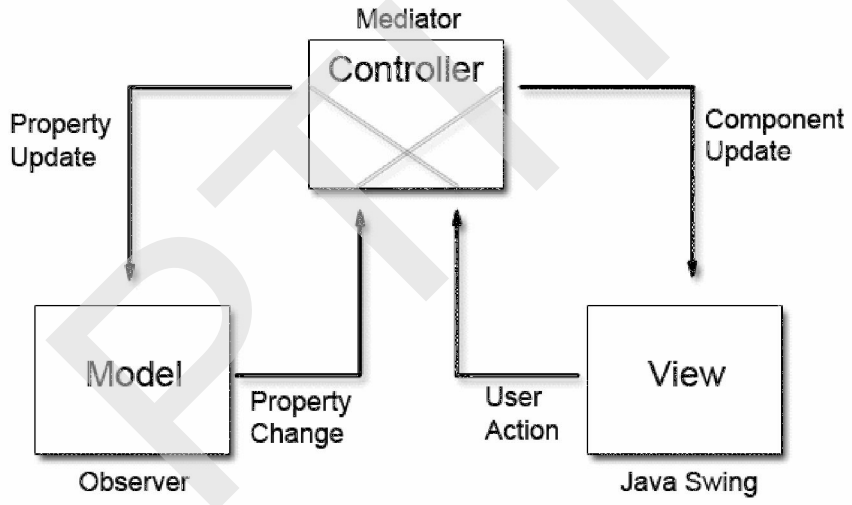
- Kỹ thuật lập trình phân tán: Trái với kỹ thuật lập trình socket, trong kỹ thuật này mối quan hệ giữa chương trình client và server là gắn kết chặt chẽ. Kỹ thuật lập trình này thực chất là kỹ thuật lập trình phân tán mã lệnh(đối tượng), cho phép phân tải tính toán lên các máy tính kết nối với nhau với quan hệ hữu cơ thay vì tập trung trên cùng một máy. Điều này cho phép tận dụng tài nguyên mạng để giải quyết các bài toán với khối lượng tính toán lớn, thời gian thực.
- Kỹ thuật lập trình truyền thông qua mạng điện thoại công cộng PSTN.

Các kỹ thuật này sẽ được khảo sát chi tiết trong các chương tiếp theo.

V. THIẾT KẾ VÀ CÀI ĐẶT THEO MÔ HÌNH MVC

1. Giới thiệu mô hình MVC

Mô hình MVC (Model – View - Control) được sử dụng khá rộng rãi để thiết kế các phần mềm hiện nay. Theo đó, hệ thống được nhóm thành 3 thành phần chính (Hình 1.6):



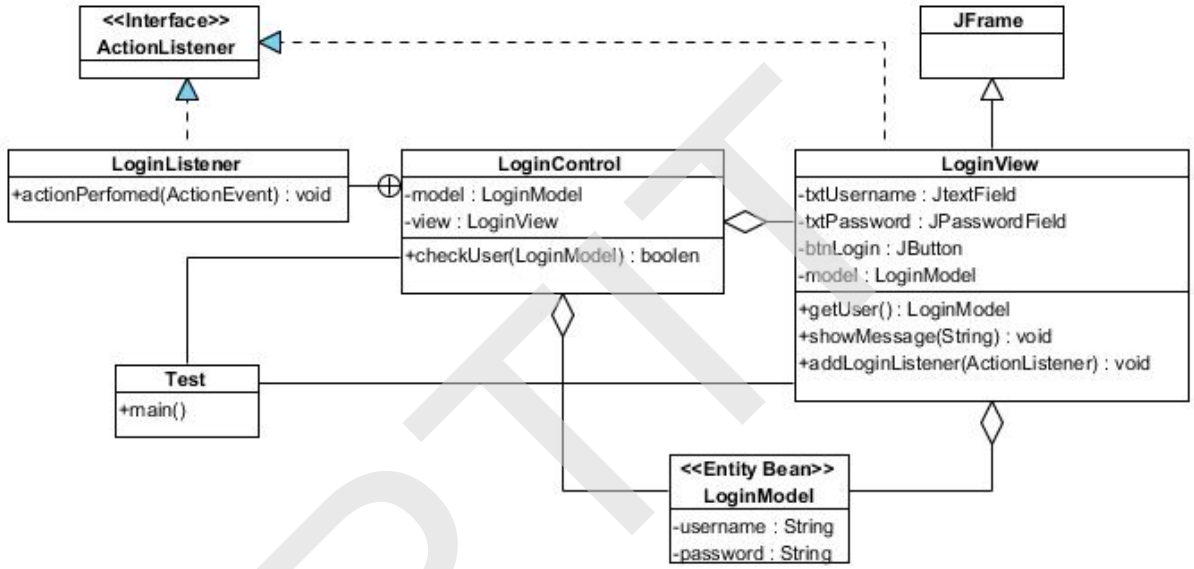
Hình 1.6: Mô hình MVC tổng quan

- Thành phần Model (M): mô hình, hay còn được gọi với nhiều tên khác như thực thể (entity, bean). Là các lớp chứa thông tin để xử lí của hệ thống. Các thông tin không nên để riêng lẻ mà nên hợp lại thành các lớp thực thể để trao đổi, truyền/nhận, và xử lí giữa các lớp thuộc các phần còn lại như Control và View cho tiện lợi.
- Thành phần View (V): trình diễn, hay còn được gọi với các tên khác như giao diện (interface), biên (boundary). C nhiệm vụ hiển thị các form để nhập dữ liệu và hiển thị kết quả xử lí từ hệ thống cho người dùng.
- Thành phần Control (C): điều khiển, hay còn được gọi là nghiệp vụ (business). Chứa toàn bộ các hoạt động xử lí, tính toán, điều khiển luồng, điều khiển form, và có thể cả các thao tác truy cập cơ sở dữ liệu.

2. Case study: thiết kế ứng dụng login theo mô hình MVC

Bài toán đặt ra như sau: Xây dựng một ứng dụng cho phép người dung đăng nhập theo tài khoản của mình

- Trên giao diện đang nhập có 2 ô văn bản cho phép người dùng nhập username/password, và một nút nhấn Login để người dùng click vào đăng nhập.
- Khi người dùng click vào nút Login, hệ thống phải kiểm tra trong cơ sở dữ liệu xem có username/password đấy không. Nếu có thì thông báo thành công, nếu sai thì thông báo username/password không hợp lệ.
- Hệ thống phải được thiết kế và cài đặt theo mô hình MVC



Hình 1. 7: Sơ đồ lớp của hệ thống

Sơ đồ lớp của hệ thống được thiết kế theo mô hình MVC trong Hình 1.7, bao gồm 3 lớp chính tương ứng với sơ đồ M-V-C như sau:

- Lớp LoginModel: là lớp tương ứng với thành phần model (M), bao gồm hai thuộc tính username và password, các hàm khởi tạo và các cặp getter/setter tương ứng với các thuộc tính.
- Lớp LoginView: là lớp tương ứng với thành phần view (V), là lớp form nên phải kế thừa từ lớp JFrame của Java, nó chứa các thuộc tính là các thành phần đồ họa bao gồm ô text nhập username, ô text nhập password, nút nhất Login.
- Lớp LoginControl: là lớp tướng ứng với thành phần control (C), nó chứa một lớp nội tại là LoginListener. Khi nút Login trên tầng view bị click thì nó sẽ chuyển tiếp sự kiện xuống lớp nội tại này để xử lí. Tất cả các xử lí đều gọi từ trong phương thức actionPerformed của lớp nội tại này. Điều này đảm bảo nguyên tắc control điều khiển các phần còn lại trong hệ thống, đúng theo nguyên tắc của mô hình MVC.

Tuần tự các bước xử lí như sau:

1. Người dùng nhập username/password và click vào giao diện của lớp LoginView
2. Lớp Loginview sẽ đóng gói thông tin username/password trên form vào một đối tượng model LoginModel bằng phương thức getUser() và chuyển xuống cho lớp LoginControl xử lí
3. Lớp LoginControl chuyển sang cho lớp nội tại LoginListener xử lí trong phương thức actionPerformed
4. Lớp LoginListener sẽ gọi phương thức checkLogin() của lớp LoginControl để kểm tra thông tin đăng nhập trong cơ sở dữ liệu.
5. Kết quả kiểm tra sẽ được chuyển cho lớp LoginView hiển thị bằng phương thức showMessage()
6. Lớp LoginView hiển thị kết quả đăng nhập lên cho người dùng

3. Cài đặt ứng dụng login theo mô hình MVC

Lớp LoginModel.java

```
package login_GUI_MVC;

public class LoginModel {
	private String userName;
	private String password;

	public LoginModel(){
	}

	public LoginModel(String username, String password){
		this.userName = username;
		this.password = password;
	}

	public String getPassword() {
		return password;
	}

	public void setPassword(String password) {
		this.password = password;
	}

	public String getUserName() {
		return userName;
	}

	public void setUserName(String userName) {
		this.userName = userName;
	}
}
```

Lớp LoginView.java

```
package login_GUI_MVC;
import java.awt.FlowLayout;
```

```java
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JPasswordField;
import javax.swing.JTextField;

public class LoginView extends JFrame implements ActionListener{
	private JTextField txtUsername;
	private JPasswordField txtPassword;
	private JButton btnLogin;
	private LoginModel model;

	public LoginView(){
		super("Login MVC");

		txtUsername = new JTextField(15);
		txtPassword = new JPasswordField(15);
		txtPassword.setEchoChar('*');
		btnLogin = new JButton("Login");

		JPanel content = new JPanel();
		content.setLayout(new FlowLayout());
		content.add(new JLabel("Username:"));
		content.add(txtUsername);
		content.add(new JLabel("Password:"));
		content.add(txtPassword);
		content.add(btnLogin);

		btnLogin.addActionListener(this);

		this.setContentPane(content);
		this.pack();

		this.addWindowListener(new WindowAdapter(){
			public void windowClosing(WindowEvent e){
				System.exit(0);
			}
		});
	}

	public void actionPerformed(ActionEvent e) {
	}

	public LoginModel getUser(){
		model = new LoginModel(txtUsername.getText(), txtPassword.getText());
		return model;
	}

	public void showMessage(String msg){
		JOptionPane.showMessageDialog(this, msg);
	}
```

```
	public void addLoginListener(ActionListener log) {
		btnLogin.addActionListener(log);
	    }
}
```

Lớp LoginControl.java

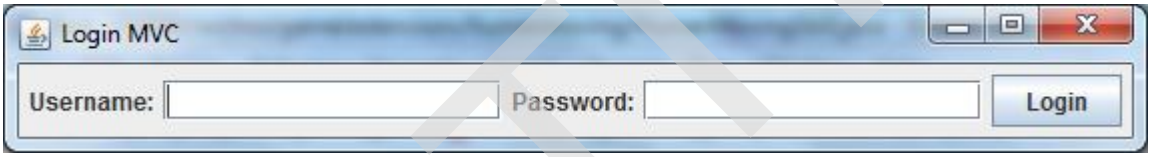
```
package login_GUI_MVC;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;


public class LoginControl {
	private LoginModel model;
	private LoginView view;

	public LoginControl(LoginView view){
		this.view = view;

		view.addLoginListener(new LoginListener());
	}

	class LoginListener implements ActionListener {
	  public void actionPerformed(ActionEvent e) {
	      try {
	          model = view.getUser();
	          if(checkUser(model)){
	              view.showMessage("Login succesfully!");
	          }else{
	              view.showMessage("Invalid username and/or password!");
	          }
	      } catch (Exception ex) {
	          view.showMessage(ex.getStackTrace().toString());
	      }
	  }
    }

	public boolean checkUser(LoginModel user) throws Exception {

	    String dbUrl = "jdbc:mysql://localhost:3306/usermanagement";
	    String dbClass = "com.mysql.jdbc.Driver";
	    String query = "Select * FROM users WHERE username ='"
	            + user.getUserName()
	            + "' AND password ='" + user.getPassword() + "'";

	    try {
	      Class.forName(dbClass);
	      Connection con = DriverManager.getConnection(dbUrl,
	                        "root", "12345678");
	      Statement stmt = con.createStatement();
	      ResultSet rs = stmt.executeQuery(query);
```

```
            if (rs.next()) {
                  return true;
            }
            con.close();
          }catch(Exception e) {
            throw e;
          }
          return false;
        }
}
```

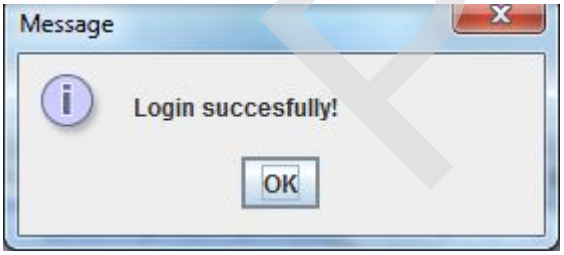
Lớp Test.java

```
package login_GUI_MVC;
public class Test {
       public static void main(String[] args) {
        LoginView view       = new LoginView();
         LoginControl controller = new LoginControl(view);
         view.setVisible(true);
       }
}
```

Kết quả



Login thành công:



Login lỗi:

VI. KẾT LUẬN

Trong chương này chúng ta đã điểm qua một số kiến thức cơ sở cho lập trình mạng bao gồm kiến thức mạng truyền thông, mô hình lập trình mạng và ngôn ngữ lập trình mạng. Và thông qua chương này sinh viên cũng nắm được mục đích của môn lập trình mạng. Các chương tiếp theo sẽ làm rõ các kỹ thuật lập trình mạng cơ bản và chỉ ra lập trình mạng an toàn bảo mật. Còn những kỹ thuật lập trình mạng phức tạp khác như CORBA, EJB, PORTAL, JAVAMAIL hoặc công nghệ đám mây(cloud) cũng như mô hình đa tầng, kỹ thuật lập trình hướng dịch vụ SOA sẽ được xét trong giáo trình lập trình mạng nâng cao. Còn kỹ thuật lập trình các dịch vụ mạng di động như SMS, MMS, các dịch vụ mạng di động khác và mạng Bluetooth, mạng Sensor, ZeeBig, WUSB, GPS...sinh viên sẽ được cung cấp qua môn lập trình thiết bị di động, qua các bài tập thực hành và hệ thống bài tập lớn của môn lập trình mạng.

PHẦN II. KỸ THUẬT LẬP TRÌNH MẠNG VỚI SOCKET

CHƯƠNG II

LẬP TRÌNH ỨNG DỤNG MẠNG VỚI SOCKET

I. GIỚI THIỆU CHUNG

Lập trình ứng dụng mạng với socket là kỹ thuật hiện nay được sử dụng cực kỳ phổ biến trong thực tế. Các ngôn ngữ lập trình mạng hầu hết đều có thư viện hỗ trợ lập trình với socket như: Ngôn ngữ c/c^{++} có thư viện socket, VC^{++} có , VB có thư viện WinSock, C# có thư viện system.socket...Trong Java các lớp thư viện hỗ trợ lập trình với socket hầu hết nằm trong gói java.net. Khi phát triển các ứng dụng mạng thì java và .NET hỗ trợ rất mạnh đối với socket sử dụng giao thức TCP(TCPsocket) và UDP(UDPsocket), nhưng lập trình Raw socket với java thì cực kỳ phức tạp. Chính vì vậy, khi lập trình các ứng dụng tiện ích mạng như chương trình ping, tracer,.. hoặc các ứng dụng can thiệt sâu hệ thống mạng mà sử dụng raw socket thì tốt nhất sử dụng ngôn ngữ C/C^{++}(Linux), VC^{++} hoặc .NET(Windows).

Trong chương này chúng tôi sẽ tập trung lập trình ứng dụng mạng sử dụng TCPSocket, UDPSocket và sử dụng ngôn ngữ lập trình Java. Đối với các ứng dụng này, Java hỗ trợ rất mạnh trong các gói java.net, java.nio. Các lớp quan trong nhất trong gói java.net gồm 6 lớp: InetAddress, ServerSocket, Socket, DatagramPacket, DatagramSocket, URL. Với 6 lớp này Java cho phép phát triển tất cả các ứng dụng mạng từ chương trình ứng dụng đơn giản cho đến phức tạp, từ các ứng dụng cỡ nhỏ đến các ứng dụng lớn. Ngoài ra còn một số lớp khác cũng được sử dụng phổ biến như NetworkInterface... Sau đây chúng ta sẽ khảo sát những kỹ thuật lập trình mạng cơ bản nhất sử dụng socket trong Java.

II. LẬP TRÌNH THAO TÁC VỚI ĐỊA CHỈ MÁY TRẠM

1. Lập trình thao tác với địa chỉ IP

1.1. Lớp InetAddress

Java có các lớp quan trọng để thao tác với địa chỉ IP trong gói java.net. Lớp quan trọng nhất là lớp InetAddress. Lớp này cho phép lấy địa chỉ của một máy trạm bất kỳ trên mạng và cho phép dễ dàng hoán chuyển giữa địa chỉ IP và tên của một máy trạm(host). Mỗi đối tượng InetAddress chứa 2 thành phần chính của một máy trạm là hostname và địa chỉ IP của máy trạm đó. Ngoài ra còn có 2 lớp khác kết thừa trực tiếp từ lớp InetAddress dành cho các phiên bản IPv4 và IPv6 là lớp Inet4Address, Inet6Address và 2 lớp khác là lớp SocketAddress , InetSocketAddress liên quan tới địa chỉ socket .

Hình 2.1. Lớp kế thừa từ lớp InetAddress và SocketAddress

Lớp InetAddress được sử dụng phổ biến trong các lớp `Socket`, `ServerSocket`, `URL`, `DatagramSocket`, `DatagramPacket` và nó được kế thừa từ lớp `Object`:

public class InetAddress extends Object implements Serializable

Đặc điểm của lớp InetAddress là lớp không có cấu tử nên không thể tạo ra đối tượng InetAddress bằng toán tử new. Nhưng bù lại, lớp InetAddress có một số phương thức có thuộc tính static cho phép lấy địa chỉ của máy trạm bất kỳ trên mạng, cụ thể là có các phương thức sau:

Tóm tắt các phương thức của lớp InetAddress	
boolean	***equals**(Object obj)* So sánh đối tượng với đối tượng obj
byte[]	***getAddress()*** Trả về địa chỉ IP chứa trong đối tượng InetAddress dạng mảng byte
static InetAddress[]	***getAllByName**(String host)* Trả về mảng địa chỉ của tất cả các máy trạm có cùng tên trên mạng
static InetAddress	***getByAddress**(byte[] addr)* Trả về đối tượng InetAddress tương ứng với địa chỉ IP truyền cho phương thức dưới dạng mảng byte
static InetAddress	***getByAddress**(String host,byte[] addr)* Tạo đối tượng InetAddress dựa trên tên và địa chỉ IP
static InetAddress	***getByName**(String host)* Xác định địa chỉ IP của máy trạm từ tên của máy trạm(host)
String	***getCanonicalHostName()*** Lấy tên miền của địa chỉ IP
String	***getHostAddress()*** Trả về địa chỉ IP chứa trong đối tượng INetAddress là chuỗi dạng a.b.c.d
String	***getHostName()*** Trả về tên máy trạm chưa trong đối tượng
static InetAddress	***getLocalHost()*** Lấy đối tượng InetAddress của máy cục bộ
int	***hashCode()***

	Trả về hashcode của địa chỉ IP cục thể
boolean	***isAnyLocalAddress()*** Kiểm tra địa chỉ InetAddress có phải địa chỉ wildcard không?
boolean	***isLinkLocalAddress()*** Kiểm tra địa chỉ có phải là một địa chỉ link-local hay không.
boolean	***isLoopbackAddress()*** Kiểm tra địa chỉ có phải là địa chỉ Loopback không.
boolean	***isMCGlobal()*** Kiểm tra địa chỉ multicast có phạm vi toàn cục hay không?
boolean	***isMCLinkLocal()*** Kiểm tra địa chỉ multicast có phải là địa chỉ có phạm vi liên kết hay không?
boolean	***isMCNodeLocal()*** Kiểm tra địa chỉ multicast có phải là địa chỉ phạm vi nút mạng hay không?
boolean	***isMulticastAddress()*** Kiểm tra địa chỉ InetAddress có phải là địa chỉ IP multicast hay không.
String	***toString()*** Chuyển địa chỉ IP thành chuỗi.

- Phương thức *getByName():*

Phương thức này có cú pháp sau:

public static InetAddress getByName(String hostName)

throws UnknownHostException

Phương thức này cho phép trả về địa chỉ của một máy trạm bất kỳ trên mạng được chỉ ra bởi tham số hostName. Tham số này có thể PCname, là tên miền DNS hoặc địa chỉ IP. Trong trường hợp không tồn tại máy trạm có tên chỉ ra trên mạng, phương thức ném trả về ngoại lệ *UnknownHostException.* Ví dụ đoạn chương trình sau để lấy địa chỉ của máy trạm có tên miền là www.yahoo.com và hiển thị địa chỉ ra màn hình:

```
try {
  InetAddress address = InetAddress.getByName("www.yahoo.com");
```

```
  System.out.println(address);

}
catch (UnknownHostException ex) {
  System.out.println("Could not find www.yahoo.com");
}
```

Lệnh *InetAddress.getByName()* sử dụng được do phương thức *getByName()* có thuộc tính static. Nếu máy trạm với tên miền chỉ ra không tồn tại thì ngoại lệ *UnknownHostException được* ném trả về và được xử lý.

- Phương thức *getAllByName():*

Phương thức này cho phép trả về địa chỉ của tất cả các máy trạm có cùng tên trên mạng dưới dạng là một mảng đối tượng InetAddress. Phương thức có cú pháp sau:

InetAddress[] addresses = InetAddress.getAllByName(String name)

throws UnknownHostException

Ví dụ: Hãy in ra địa chỉ của tất cả các máy trạm trên mạng mà có cùng tên miền www.microsoft.com:

```
//AllAddr.java
import java.net.*;
public class AllAddr{
  public static void main (String[] args) {
    try {
      InetAddress[] addresses =
       InetAddress.getAllByName("www.microsoft.com");
      for (int i = 0; i < addresses.length; i++) {
        System.out.println(addresses[i]);
      }
    }
  catch (UnknownHostException ex) {
   System.out.println("Could not find www.microsoft.com");
  }
 }
}
```

Dịch chạy chương trình trên máy tính có kết nối mạng Internet, kết quả trả về như sau:

```
www.microsoft.com/63.211.66.123
www.microsoft.com/63.211.66.124
www.microsoft.com/63.211.66.131
www.microsoft.com/63.211.66.117
```

```
www.microsoft.com/63.211.66.116
www.microsoft.com/63.211.66.107
www.microsoft.com/63.211.66.118
www.microsoft.com/63.211.66.115
www.microsoft.com/63.211.66.110
```

- Phương thức *getLocalHost():*

Phương thức này cho phép trả về địa chỉ của máy cục bộ, nếu không tìm thấy nó cũng ném trả về ngoại lệ tượng tự như phương thức *getByName().* Nó cũng có cú pháp:

public static InetAddress getLocalHost( ) throws UnknownHostException

Ngoài các phương thức static trên, một số phương thức khác cho phép trả về địa chỉ IP hoặc tên của một máy trạm từ đối tượng InetAddress của máy trạm sau khi đã lấy được địa chỉ của máy trạm. Các phương thức tiêu biểu là:

- Phương thức *getHosName():*Trả về tên máy trạm từ đối tượng InetAddress của máy trạm đó. Cú pháp:

  public String getHostName( )

  Ví dụ: Cho địa chỉ, in ra tên máy trạm:

```
import java.net.*;
public class ReverseTest {
  public static void main (String[] args) {
    try {
      InetAddress ia = InetAddress.getByName("208.201.239.37");
      System.out.println(ia.getHostName( ));
    }
    catch (Exception ex) {
      System.err.println(ex);
    }
  }
}
```

- Phương thức *getHostAddress():* Trả về địa chỉ IP của máy trạm từ đối tượng InetAddress tương ứng là chuỗi địa chỉ dạng a.b.c.d. Phương thức có cú pháp:

  public String getHostAddress( )

  Ví dụ: In ra địa chỉ IP của máy cục bộ

```
import java.net.*;
public class MyAddress {
  public static void main(String[] args) {
    try {
      InetAddress me = InetAddress.getLocalHost( );
      String dottedQuad = me.getHostAddress( );
      System.out.println("My address is " + dottedQuad);
```

```
    }
    catch (UnknownHostException ex) {
      System.out.println("I'm sorry. I don't know my own address.");
    }
  }
}
```

- Phương thức *getAddress():* Trả về địa chỉ IP của máy trạm từ đối tượng InetAddress của máy trạm tương ứng dưới dạng mảng byte. Phương thức có cú pháp:

public byte[] getAddress( )

Ví dụ: Phương thức getVersion() lấy phiên bản địa chỉ IP của máy cục bộ:

```
import java.net.*;
public class AddressTests {
  public static int getVersion(InetAddress ia) {
    byte[] address = ia.getAddress( );
    if (address.length == 4) return 4;
    else if (address.length == 16) return 6;
    else return -1;
  }
```

Lưu ý: Khi in ra các byte địa chỉ IP, nếu giá trị của byte địa chỉ mà vượt qua 127 thi phải cộng với 256 để ra giá trị đúng(vì kiểu byte chỉ có giá trị trong khoảng từ 0128 đến +127), nếu không nó sẽ có giá trị âm. Ví dụ với mảng address trong ví dụ trên:

```
for(int i=0;i<address.length;i++)
    System.out.println((address[i]>0)?address[i]: (address[i]+256));
```

<u>Các phương thức khác của lớp InetAddress</u>:

public boolean isAnyLocalAddress( ): Phương thức này trả về giá trị true với địa chỉ wildcard, false nếu không phải. Địa chỉ wildcard tương hợp với bất cứ địa chỉ nào của máy cục bộ. Phương thức này quan trọng nếu hệ thống cục bộ có nhiều card giao tiếp mạng, nhất là đối với server và gateway. Trong IPv4, địa chỉ wildcard là 0.0.0.0, trong IPv6 là 0:0:0:0:0:0:0:0.

public boolean isLoopbackAddress( ): Phương thức này kiểm tra một địa chỉ có phải là địa chỉ loopback hay không, nếu không phải trả về false. Địa chỉ loopback kết nối trực tiếp trong máy trạm trong lớp IP mà không sử dụng bất kỳ phần cứng vật lý nào. Với IPv4, địa chỉ loopback là 127.0.0.1, với IPv6 là 0:0:0:0:0:0:0:1.

public boolean isLinkLocalAddress( ): Phương thức này trả về giá trị true nếu một địa chỉ là địa chỉ link-local IPv6, nếu không phải thì trả về giá trị false. Địa chỉ link-local là địa chỉ chỉ được hỗ trợ trong mạng IPv6 để tự cấu hình, tương tự như DHCP trên mạng IPv4 nhưng không cần server. Bộ định tuyến sẽ không cho phép truyền qua các gói tin có địa chỉ này ra khỏi mạng con cục bộ. Tất cả địa chỉ link-local đều bắt đầu với 8 byte:

FE80:0000:0000:0000

8 byte tiếp theo sẽ là địa chỉ cục bộ thường là địa chỉ lấy từ địa chỉ MAC trong thẻ Ethernet(NIC).

public boolean isMulticastAddress( ): Trae về true nếu địa chỉ là địa chỉ multicast, nếu không trả về giá trị false. Trong IPv4, địa chỉ multicast nằm trong dải địa chỉ IP: 224.0.0.0->239.255.255.255(lớp D), trong IPv6 thì chúng được bắt đầu với byte có giá trị FF.

1. 2. Ví dụ sử dụng các phương thức lớp InetAddress

Chương trình sau cho phép sử dụng các phương thức của lớp InetAddresss để hiển thị các đặc trưng của một địa chỉ IP được nhập vào từ trên dòng lệnh. Mã chương trình ví dụ được thể hiện như sau:

```
//IPCharacteristics.java
import java.net.*;
public class IPCharacteristics {
  public static void main(String[] args) {
     try {
      InetAddress address = InetAddress.getByName(args[0]);
      if (address.isAnyLocalAddress( )) {
        System.out.println(address + " is a wildcard address.");
      }
      if (address.isLoopbackAddress( )) {
        System.out.println(address + " is loopback address.");
      }
      if (address.isLinkLocalAddress( )) {
        System.out.println(address + " is a link-local address.");
      }
      else if (address.isSiteLocalAddress( )) {
        System.out.println(address + " is a site-local address.");
      }
      else {
        System.out.println(address + " is a global address.");
      }
      if (address.isMulticastAddress( )) {
        if (address.isMCGlobal( )) {
          System.out.println(address + " is a global multicast address.");
        }
        else if (address.isMCOrgLocal( )) {
          System.out.println(address
           + " is an organization wide multicast address.");
        }
        else if (address.isMCSiteLocal( )) {
          System.out.println(address + " is a site wide multicast
```

```
                        address.");
        }
        else if (address.isMCLinkLocal( )) {
          System.out.println(address + " is a subnet wide multicast
                        address.");
        }
        else if (address.isMCNodeLocal( )) {
          System.out.println(address
           + " is an interface-local multicast address.");
        }
        else {
          System.out.println(address + " is an unknown multicast
                        address type.");
        }
      }
      else {
        System.out.println(address + " is a unicast address.");
      }
    }
    catch (UnknownHostException ex) {
      System.err.println("Could not resolve " + args[0]);
    }
  }
}
```

Sau khi biên dịch chương trình, chạy chương trình với lệnh:

java *IPCharacteristics* <*addresss*> [Enter]

III. LẬP TRÌNH ỨNG DỤNG MẠNG VỚI TCPSOCKET

1. Giao thức TCP và cơ chế truyền thông của TCP

<Tham khảo giáo trình mạng máy tính>

2. Một số lớp Java hỗ trợ lập trình với TCPSocket

2.1. Lớp Socket

Lớp Socket dùng để tạo đối tượng socket cho phép truyền thông với giao thức TCP hoặc UDP. (Với giao thức UDP người ta thường sử dụng lớp DatagramSocket thay vì lớp Socket).

2.1.1. Các cấu tử:

- *public Socket(String host, int port)*

  throws UnknownHostException, IOException

Cấu tử này cho phép tạo ra đối tượng Socket truyền thông với giao thức TCP và thực hiện kết nối với máy trạm từ xa có địa chỉ và số cổng được chỉ ra bởi tham số host và port tương ứng. Tham số host có thể là tên máy trạm, tên miền hoặc địa chỉ IP. Nếu không tìm thấy máy trạm từ

xa hoặc đối tượng Socket không được mở thì nó ném trả về ngoại lệ *UnknownHostException* hoặc *IOException.* Ví dụ đoạn chương trình sau cho phép mở socket và kết nối tới máy trạm từ xa có tên miền www.yahoo.com và số cổng là 80.

```
try {
  Socket toYahoo = new Socket("www.yahoo.com", 80);
  // Hoạt động gửi /nhận dữ liệu
}
catch (UnknownHostException ex) {
  System.err.println(ex);
}
catch (IOException ex) {

  System.err.println(ex);
}
```

- *public Socket(InetAddress host, int port) throws IOException*

Cấu tử này tương tự như cấu tử trên, nhưng tham số thứ nhất là đối tượng InetAddress của máy trạm từ xa. Đối tượng InetAddress của máy trạm từ xa có thể lấy được bằng phương thức getByName() của lớp InetAddress.

- *public Socket(String host, int port, InetAddress interface, int localPort) throws IOException, UnknownHostException*

Cấu tử này cho phép tạo ra đối tượng Socket và kết nối với máy trạm từ xa. Hai tham số đầu là tên và số cổng của máy trạm từ xa, 2 tham số sau là giao tiếp mạng vật lý(NIC) hoặc ảo và số cổng được sử dụng trên máy cục bộ. Nếu số cổng cục bộ localPort mà bằng 0 thì Java sẽ chọn sử dụng một số cổng cho phép ngẫu nhiên trong khoảng 1024 đến 65535.

- *public Socket(InetAddress host, int port, InetAddress interface, int localPort) throws IOException*

Tương tự như cấu tử trên, nhưng tham số thứ nhất là đối tượng InetAddress của máy trạm từ xa.

- *protected Socket( )*

Cấu tử này tạo đối tượng socket mà không kết nối với máy trạm từ xa. Cấu tử này được sử dụng khi chương trình có các socket lớp con.

2.1.2. Một số phương thức quan trọng của lớp Socket

- *public InetAddress getInetAddress( ):* Phương thức cho phép trả về địa chỉ của máy trạm từ xa hiện đang kết nối với socket.
- *public int getPort( ):* Trả về số cổng trên máy trạm từ xa mà hiện đang kết nối với socket.
- *public int getLocalPort( )*: Trả về số cổng trên máy cục bộ
- *public InputStream getInputStream( ) throws IOException:* Trả về luồng nhập của socket là đối tượng InputStream.

- *public OutputStream getOutputStream( ) throws IOException*: Trả về luồng xuất của socket là đối tượng OutputStream.
- *public void close( ) throws IOException*: Đóng socket

2.1.3. Thiết đặt các tuỳ chọn Socket

Tuỳ chọn socket chỉ ra làm thế nào lớp Java Socket có thể gửi /nhận dữ liệu trên native socket. Socket két có các tuỳ chọn sau:

- TCP_NODELAY
- SO_BINDADDR
- SO_TIMEOUT
- SO_LINGER
- SO_SNDBUF (Java 1.2 and later)
- SO_RCVBUF (Java 1.2 and later)
- SO_KEEPALIVE (Java 1.3 and later)
- OOBINLINE (Java 1.4 and later)

Để thiết lập các tuỳ chọn và trả về trạng thái các tuỳ chọn, lớp socket có các phương thức tương ứng. Ví dụ để thiết đặt và trả về trạng thái tuỳ chọn TCP_NODELAY, lớp Socket có các phương thức sau:

public void setTcpNoDelay(boolean on) throws SocketException

public boolean getTcpNoDelay( ) throws SocketException

2.2. Lớp ServerSocket

Lớp ServerSocket cho phép tạo đối tượng socket phía server và truyền thông với giao thức TCP. Sau khi được tạo ra, nó được đặt ở trạng thái lắng nghe(trạng thái thụ động) chờ tín hiệu kết nới gửi tới từ client.

2.2.1 Các cấu tử

- *public ServerSocket(int port) throws BindException, IOException*

Cấu tử này cho phép tạo ra đối tượng ServerSocket với số cổng xác định được chỉ ra bởi tham số port. Nếu số cổng port=0 thì nó cho phép sử dụng một số cổng cho phép nào đó(anonymous port). Cấu tử sẽ ném trả về ngoại lệ khi socket không thể tạo ra được. Socket được tạo bởi cấu tử này cho phép đáp ứng cực đại tới 50 kết nối đồng thời.

- *public ServerSocket(int port, int queueLength)*

throws IOException, BindException

Tương tự như cấu tử trên nhưng cho phép chỉ ra số kết nối cực đại mà socket có thể đáp ứng đồng thời bởi tham số queueLenth.

- *public ServerSocket( ) throws IOException*

Cấu tử này cho phép tạo đối tượng ServerSocket nhưng không gắn kết thực sự socket với một số cổng cụ thể nào cả. Và như vậy nó sẽ không thể chấp nhận bất cứ kết nối nào gửi tới. Nó sẽ được gắn kết địa chỉ sau sử dụng phương thức bind(). Ví dụ:

```
ServerSocket ss = new ServerSocket( );
// set socket options...
SocketAddress  http = new InetSocketAddress(80);
ss.bind(http);
```

2.2.2. Phương thức

- Phương thức accept()

Phương thức này có cú pháp sau:

public Socket accept( ) throws IOException

Phương thức này khi thực hiện nó đặt đối tượng ServerSocket ở trạng thái "nghe" tại số cổng xác định chờ tín hiệu kết nối gửi đến từ client. Khi có tín hiệu kết nối gửi tới phương thức sẽ trả về đối tượng Socket mới để phục vụ kết nối đó. Khi xảy ra lỗi nhập/xuất, phương thức sẽ ném trả về ngoại lệ IOException. Ví dụ:

```
ServerSocket server = new ServerSocket(5776);

while (true) {

  Socket connection = server.accept( );

  OutputStreamWriter out

   = new OutputStreamWriter(connection.getOutputStream( ));

  out.write("You've connected to this server. Bye-bye now.\r\n");

connection.close( );

}
```

- Phương thức close()

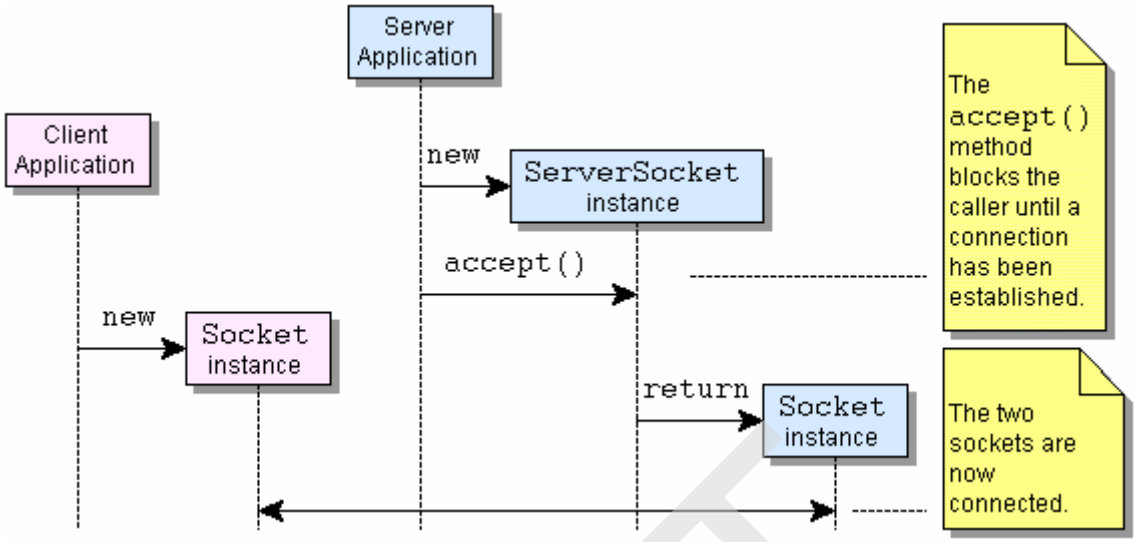
Phương thức close() có cú pháp sau:

public void close( ) throws IOException

Phương thức này cho phép đóng soccket và giải phóng tài nguyên cấp cho socket.

3. Kỹ thuật lập trình truyền thông với giao thức TCP

Trong chương trình ứng dụng mạng xây dựng theo mô hình client/server, để chương trình client và chương trình server có thể truyền thông được với nhau thì mỗi phía phải thực hiện tối thiểu các thao tác cơ bản sau đây(Hình 2.2):



Hình 2.2. Quá trình khởi tạo truyền thông với TCPSocket

3.1. Chương trình phía server:

- Tạo đối tượng ServerSocket với một số hiệu cổng xác định
- Đặt đối tượng ServerSocket ở trạng thái nghe tín hiệu đến kết nối bằng phương thức accept(). Nếu có tín hiệu đến kết nối phương thức accept() tạo ra đối tượng Socket mới để phục vụ kết nối đó.
- Khai báo luồng nhập/xuất cho đối tượng Socket mới(tạo ra ở bước trên). Luồng nhập/xuất có thể là luồng kiểu byte hoặc kiểu char.
- Thực hiện truyền dữ liệu với client thông qua luồng nhập/xuất
- Server hoặc client hoặc cả 2 đóng kết nối
- Server trở về bước 2 và đợi kết nối tiếp theo.

3.2. Chương trình client

- Tạo đối tượng Socket và thiết lập kết nối tới server bằng cách chỉ ra các tham số của server.
- Khai báo luồng nhập/xuất cho Socket. Luồng nhập/xuất có thể là luồng kiểu byte hoặc kiểu char.
- Thực hiện truyền dữ liệu qua mạng thông qua luồng nhập/xuất
- Đóng Socket, giải phóng các tài nguyên khác, kết thúc chương trình nếu cần.

Lưu ý:

- Bình thường chương trình server luôn chạy trước chương trình client
- Một chương trình server có thể phục vụ nhiều client đồng thời hoặc lặp.

Ví dụ:

```
import java.io.*;
import java.net.*;

public class EchoClient {
    public static void main(String[] args) throws IOException {

        Socket echoSocket = null;
        PrintWriter out = null;
        BufferedReader in = null;

        try {
            echoSocket = new Socket("taranis", 7);
            out = new PrintWriter(echoSocket.getOutputStream(), true);
            in = new BufferedReader(new InputStreamReader(
                                        echoSocket.getInputStream()));
        } catch (UnknownHostException e) {
            System.err.println("Don't know about host: taranis.");
            System.exit(1);
        } catch (IOException e) {
            System.err.println("Couldn't get I/O for "
                               + "the connection to: taranis.");
            System.exit(1);
        }

  BufferedReader stdIn = new BufferedReader(
                                    new InputStreamReader(System.in));
  String userInput;

  while ((userInput = stdIn.readLine()) != null) {
      out.println(userInput);
      System.out.println("echo: " + in.readLine());
  }

  out.close();
  in.close();
  stdIn.close();
  echoSocket.close();
    }}
```

3.3. Luồng nhập/xuất mạng và đọc/ghi dữ liệu qua luồng nhập/xuất

Luồng nhập/xuất mạng cho phép chương trình client và server trao đổi dữ liệu với nhau qua mạng. Luồng nhập/xuất của socket có thể là luồng kiểu byte hoặc kiểu ký tự. Ở đây chúng tôi nêu lên một cách thông dụng nhất tạo luồng kiểu byte và kiểu ký tự để chuơng trình thực hiện đọc ghi dữ liệu với mạng.

- **Luồng kiểu byte**

Giả sử đối tượng Socket được tạo ra với biến tham chiếu là cl.

- Với luồng nhập:

+ Tạo luồng nhập cho socket:

InputStream inp=cl.getInputStream();

+ Đọc dữ liệu: Có ba cách

-/ Đọc mỗi lần một byte: *inp.read()*

-/Đọc một khối dữ liệu và cất vào mảng b:

byte b=new byte[1024];

inp.read(b) hoặc inp.read(b,offset, len)

- Với luồng xuất:

+Tạo luồng xuất:

OutputStream outp=cl.getOutputStream();

+ Viết dữ liệu:

-/Viết mỗi lần một byte b: *outp.write(b);*

-/ Viết cả khối dữ liệu chứa trong mảng b kiểu byte:

//byte[] b;

outp.write(b) hoặc *outp.write(b,offset,len);*

- **Luồng kiểu char:**

- Với luồng nhập:

+Tạo luồng nhập:

BufferedReader inp=new BuferedReader(

new InputStreamReader(cl.getInputStream()));

+ Đọc dữ liệu:

-/Đọc từng ký tự: *int ch=inp.read()*

-/ Đọc chuỗi*:* *String s=inp.readLine();*

- Với luồng xuất:

+ Tạo luồng xuất:

PrintWriter outp=new PrintWriter(cl.getOutputStream(),true);

+ Viết dữ liệu:

outp.println(<data>);

4. Một số ví dụ

4.1. Chương trình quét cổng sử dụng Socket

```
//PortScanner.java
import java.net.*;
import java.io.*;
public class PortScanner {
  public static void main(String[] args) {
     String host = "localhost";
    if (args.length > 0) {
      host = args[0];
    }
    try {
      InetAddress theAddress = InetAddress.getByName(host);
      for (int i = 1; i < 65536; i++) {
        Socket connection = null;
        try {
          connection = new Socket(host, i);
          System.out.println("There is a server on port "
           + i + " of " + host);
        }
        catch (IOException ex) {
          // must not be a server on this port
        }
        finally {
          try {
            if (connection != null) connection.close( );
          }
          catch (IOException ex) {}
        }
      } // end for

    } // end try
    catch (UnknownHostException ex) {
      System.err.println(ex);
    }
  }  // end main
}  // end PortScanner
```

4.2. Chương trình quét cổng cục bộ dùng lớp ServerSocket

```
import java.net.*;
```

```
import java.io.*;
public class LocalPortScanner {
  public static void main(String[] args) {

    for (int port = 1; port <= 65535; port++) {
      try {
        // the next line will fail and drop into the catch block if
        // there is already a server running on the port
        ServerSocket server = new ServerSocket(port);
      }
      catch (IOException ex) {
        System.out.println("There is a server on port " + port + ".");
      } // end catch
    } // end for
  }
}
```

4.3. Chương trình finger client

Finger là một giao thức truyền thẳng theo RFC 1288, client tạo kết nối TCP tới server với số cổng 79 và gửi một truy vấn on-line tới server. Server đáp ứng truy vấn và đóng kết nối.

```
import java.net.*;
import java.io.*;
public class FingerClient {
  public final static int DEFAULT_PORT = 79;
  public static void main(String[] args) {
    String hostname = "localhost";
    try {
      hostname = args[0];
    }
    catch (ArrayIndexOutOfBoundsException ex) {
      hostname = "localhost";
    }
    Socket connection = null;
    try {
      connection = new Socket(hostname, DEFAULT_PORT);
      Writer out = new OutputStreamWriter(
       connection.getOutputStream( ), "8859_1");
      for (int i = 1; i < args.length; i++) out.write(args[i] + " ");
      out.write("\r\n");
      out.flush( );
      InputStream raw = connection.getInputStream( );
```

```
      BufferedInputStream buffer = new BufferedInputStream(raw);
      InputStreamReader in = new InputStreamReader(buffer, "8859_1");
      int c;
      while ((c = in.read( )) != -1) {
 // filter non-printable and non-ASCII as recommended by RFC 1288
      if ((c >= 32 && c < 127) || c == '\t' || c == '\r' || c == '\n')
        {
          System.out.write(c);
        }
      }
    }
    catch (IOException ex) {
      System.err.println(ex);
    }
    finally {
      try {
        if (connection != null) connection.close( );
      }
      catch (IOException ex) {}
    }
  }
}
```

4.4. Chương trình cho phép lấy thời gian server về client.

//TimeClient.java

```
import java.net.*;
import java.io.*;
import java.util.*;
public class TimeClient {
  public final static int    DEFAULT_PORT = 37;
  public final static String DEFAULT_HOST = "time.nist.gov";
  public static void main(String[] args) {
    String hostname = DEFAULT_HOST ;
    int port = DEFAULT_PORT;
   if (args.length > 0) {
      hostname = args[0];
    }
    if (args.length > 1) {
     try {
        port = Integer.parseInt(args[1]);
      }
```

```
    catch (NumberFormatException ex) {
      // Stay with the default port
    }
  }
   // The time protocol sets the epoch at 1900,
  // the Java Date class at 1970. This number
  // converts between them.
    long differenceBetweenEpochs = 2208988800L;
    // If you'd rather not use the magic number, uncomment
  // the following section which calculates it directly.

 /*
  TimeZone gmt = TimeZone.getTimeZone("GMT");
  Calendar epoch1900 = Calendar.getInstance(gmt);

  epoch1900.set(1900, 01, 01, 00, 00, 00);
  long epoch1900ms = epoch1900.getTime( ).getTime( );
  Calendar epoch1970 = Calendar.getInstance(gmt);
  epoch1970.set(1970, 01, 01, 00, 00, 00);
  long epoch1970ms = epoch1970.getTime( ).getTime( );
    long differenceInMS = epoch1970ms - epoch1900ms;
  long differenceBetweenEpochs = differenceInMS/1000;
  */
    InputStream raw = null;
  try {
    Socket theSocket = new Socket(hostname, port);
    raw = theSocket.getInputStream( );
        long secondsSince1900 = 0;
    for (int i = 0; i < 4; i++) {
      secondsSince1900 = (secondsSince1900 << 8) | raw.read( );
    }
    long secondsSince1970
     = secondsSince1900 - differenceBetweenEpochs;
    long msSince1970 = secondsSince1970 * 1000;
    Date time = new Date(msSince1970);
         System.out.println("It is " + time + " at " + hostname);
   }  // end try
  catch (UnknownHostException ex) {
    System.err.println(ex);
  }
  catch (IOException ex) {
```

```
      System.err.println(ex);
    }
    finally {
      try {
        if (raw != null) raw.close( );
      }
      catch (IOException ex) {}
    }
  }  // end main
} // end TimeClient
//TimeServe.java
import java.net.*;
import java.io.*;
import java.util.Date;
public class TimeServer {
 public final static int DEFAULT_PORT = 37;
  public static void main(String[] args) {
   int port = DEFAULT_PORT;
   if (args.length > 0) {
     try {
        port = Integer.parseInt(args[0]);
        if (port < 0 || port >= 65536) {
          System.out.println("Port must between 0 and 65535");
          return;
        }
     }
     catch (NumberFormatException ex) {}
   }
   // The time protocol sets the epoch at 1900,
   // the Date class at 1970. This number
   // converts between them.
   long differenceBetweenEpochs = 2208988800L;
   try {
     ServerSocket server = new ServerSocket(port);
       while (true) {
         Socket connection = null;
         try {
           connection = server.accept( );
           OutputStream out = connection.getOutputStream( );
           Date now = new Date( );
           long msSince1970 = now.getTime( );
```

```
            long secondsSince1970 = msSince1970/1000;
            long secondsSince1900 = secondsSince1970
             + differenceBetweenEpochs;
           byte[] time = new byte[4];
      time[0]= (byte) ((secondsSince1900 & 0x00000000FF000000L) >> 24);
      time[1 = (byte) ((secondsSince1900 & 0x0000000000FF0000L) >> 16);
      time[2] = (byte) ((secondsSince1900 & 0x000000000000FF00L) >> 8);
            time[3] = (byte) (secondsSince1900 & 0x00000000000000FFL);
            out.write(time);
           out.flush( );
          } // end try
          catch (IOException ex) {
          } // end catch
          finally {
           if (connection != null) connection.close( );
         }
        }  // end while
    }  // end try
    catch (IOException ex) {
      System.err.println(ex);
    } // end catch
  } // end main
} // end TimeServer
```

5. Case study: Login từ xa dùng giao thức TCP/IP

5.1 Bài toán

Bài toán login từ xa dùng giao thức TCP/IP đặt ra như sau:

- Cở sở dữ liệu đọc lưu trữ và quản lí trên server TCP, trong đó có bảng users chứa ít nhất hai cột: cột username và cột password.
- Chương trình phía client TCP phải hiện giao diện đồ họa, trong đó có một ô text để nhập username, một ô text để nhập password, và một nút nhấn Login.
- Khi nút Login được click, chương trình client sẽ gửi thông tin đăng nhập (username/password) trên form giao diện, và gửi sang server theo giao thức TCP
- Tại phía server, mỗi khi nhận được thông tin đăng nhập gửi từ client, nó sẽ tiến hành kiểm tra trong cơ sở dữ liệu xem có tài khoản nào trùng với thông tin đăng nhập nhận được hay không.
- Sau khi có kết quả kiểm tra (đăng nhập đúng, hoặc sai), server TCP sẽ gửi kết quả này về cho client tương ứng, theo đúng giao thức TCP.
- Ở phía client, sau khi nhận được kết quả đăng nhập (đăng nhập đúng, hoặc sai) từ server, nó sẽ hiển thị thông báo tương ứng với kết quả nhận được: nếu đăng nhập

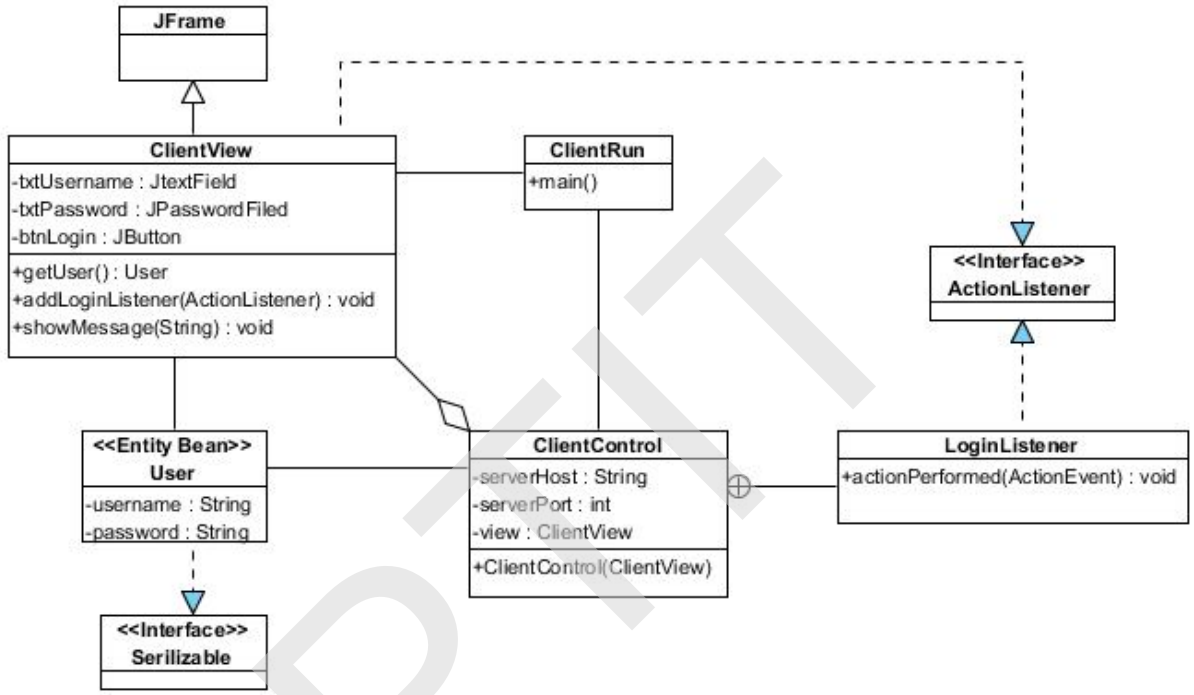
đúng thì thông báo login thành công. Nếu đăng nhập sai thì thông báo là username/password không đúng.

- Yêu cầu kiến trúc hệ thống ở cả hai phía client và server đều được thiết kế theo mô hình MVC

5.2 Kiến trúc hệ thống theo mô hình MVC

Vì hệ thống được thiết kế theo mô hình client/server dùng giao thức TCP/IP nên mỗi phía client, server sẽ có một sơ đồ lớp riêng, các sơ đồ này được thiết kế theo mô hình MVC.

5.2.1 Sơ đồ lớp phía client



Hình 2.3: Sơ đồ lớp phía client TCP/IP

Sơ đồ lớp của phía client được thiết kế theo mô hình MVC trong Hình 2.3, bao gồm 3 lớp chính tương ứng với sơ đồ M-V-C như sau:

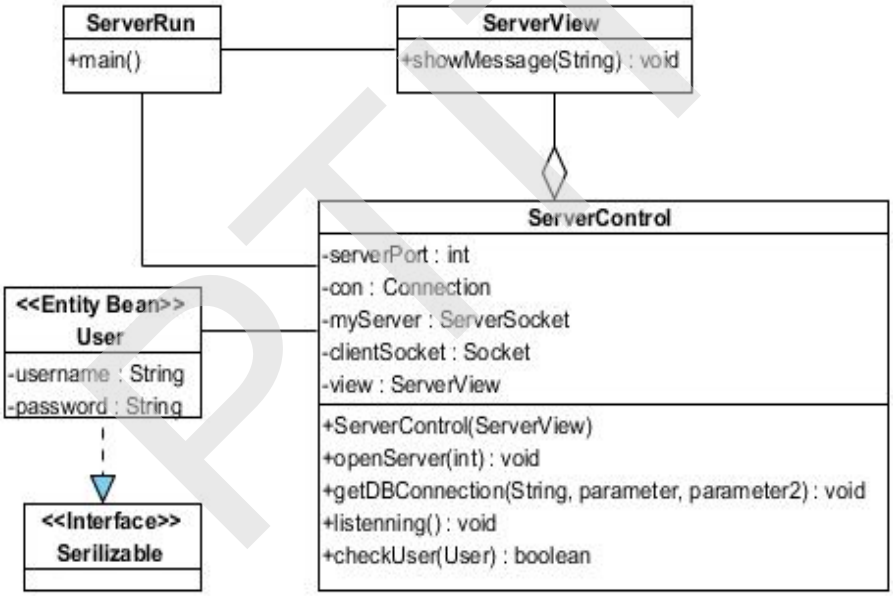
- Lớp User: là lớp tương ứng với thành phần model (M), bao gồm hai thuộc tính username và password, các hàm khởi tạo và các cặp getter/setter tương ứng với các thuộc tính.
- Lớp ClientView: là lớp tương ứng với thành phần view (V), là lớp form nên phải kế thừa từ lớp JFrame của Java, nó chứa các thuộc tính là các thành phần đồ họa bao gồm ô text nhập username, ô text nhập password, nút nhất Login.
- Lớp ClientControl: là lớp tương ứng với thành phần control (C), nó chứa một lớp nội tại là LoginListener. Khi nút Login trên tầng view bị click thì nó sẽ chuyển tiếp sự kiện xuống lớp nội tại này để xử lí. Tất cả các xử lí đều gọi từ trong phương thức actionPerformed của lớp nội tại này, bao gồm: lấy thông tin trên form giao diện và gửi sang server theo giao thức TCP/IP, nhận kết quả đăng nhập từ server về và yêu cầu

form giao diện hiển thị. Điều này đảm bảo nguyên tắc control điều khiển các phần còn lại trong hệ thống, đúng theo nguyên tắc của mô hình MVC.

5.2.2 Sơ đồ lớp phía server

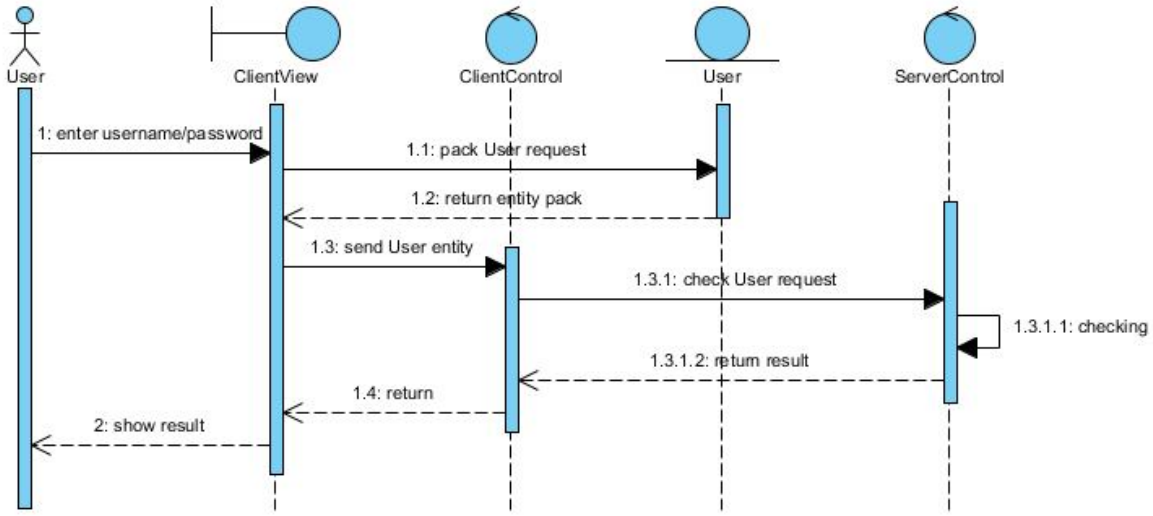
Sơ đồ lớp của phía server được thiết kế theo mô hình MVC trong Hình 2.4, bao gồm 3 lớp chính tương ứng với sơ đồ M-V-C như sau:

- Lớp User: là lớp thực thể, dùng chung thống nhất với lớp phía bên client.
- Lớp ServerView: là lớp tương ứng với thành phần view (V), là lớp dùng hiển thị các thông báo và trạng thái hoạt động bên server TCP.
- Lớp ServerControl: là lớp tương ứng với thành phần control (C), nó đảm nhiệm vai trò xử lí của server TCP, bao gồm: nhận thông tin đăng nhập từ phía các client, kiểm tra trong cơ sở dữ liệu xem các thng tin này đúng hay sai, sau đó gửi kết quả đăng nhập về cho client tương ứng.



Hình 2.4: Sơ đồ lớp phía server TCP/IP

5.2.3 Tuần tự các bước thực hiện



Hình 2.5: Tuần tự các bước thực hiện theo giao thức TCP/IP

Tuần tự các bước xử lí như sau (Hình 2.5):

1. Ở phía client, người dùng nhập username/password và click vào giao diện của lớp ClientView
2. Lớp ClientView sẽ đóng gói thông tin username/password trên form vào một đối tượng model User bằng phương thức getUser() và chuyển xuống cho lớp ClientControl xử lí
3. Lớp ClientControl gửi thông tin chứa trong đối tượng User này sang phía server để kiểm tra đăng nhập
4. Bên phía server, khi nhận được thông tin đăng nhập trong đối tượng User, nó sẽ gọi phương thức checkLogin() để kểm tra thông tin đăng nhập trong cơ sở dữ liệu.
5. Kết quả kiểm tra sẽ được trả về cho lớp ClientControl
6. Ở phía client, khi nhận được kết quả kiểm tra đăng nhập, lớp ClientControl sẽ chuyển cho lớp LoginView hiển thị bằng phương thức showMessage()
7. Lớp LoginView hiển thị kết quả đăng nhập lên cho người dùng

5.3 Cài đặt

5.3.1 Các lớp phía client

User.java

```
package tcp.client;
import java.io.Serializable;

public class User implements Serializable{
	private String userName;
	private String password;

	public User(){
	}
```

```
	public User(String username, String password){
		this.userName = username;
		this.password = password;
	}

	public String getPassword() {
		return password;
	}

	public void setPassword(String password) {
		this.password = password;
	}

	public String getUserName() {
		return userName;
	}

	public void setUserName(String userName) {
		this.userName = userName;
	}
}
```

ClientView.java

```
package tcp.client;
import java.awt.FlowLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JPasswordField;
import javax.swing.JTextField;

public class ClientView extends JFrame implements ActionListener{
	private JTextField txtUsername;
	private JPasswordField txtPassword;
	private JButton btnLogin;

	public ClientView(){
		super("TCP Login MVC");

		txtUsername = new JTextField(15);
		txtPassword = new JPasswordField(15);
		txtPassword.setEchoChar('*');
		btnLogin = new JButton("Login");

		JPanel content = new JPanel();
		content.setLayout(new FlowLayout());
		content.add(new JLabel("Username:"));
		content.add(txtUsername);
		content.add(new JLabel("Password:"));
		content.add(txtPassword);
```

```
		content.add(btnLogin);

		this.setContentPane(content);
		this.pack();

		this.addWindowListener(new WindowAdapter(){
			public void windowClosing(WindowEvent e){
				System.exit(0);
			}
		});
	}

	public void actionPerformed(ActionEvent e) {
	}

	public User getUser(){
		User model = new User(txtUsername.getText(), txtPassword.getText());
		return model;
	}

	public void showMessage(String msg){
		JOptionPane.showMessageDialog(this, msg);
	}

	public void addLoginListener(ActionListener log) {
		  btnLogin.addActionListener(log);
		}
}
```

ClientControl.java

```
package tcp.client;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.Socket;

public class ClientControl {
	private ClientView view;
	private String serverHost = "localhost";
	private int serverPort = 8888;

	public ClientControl(ClientView view){
		this.view = view;
		this.view.addLoginListener(new LoginListener());
	}

	class LoginListener implements ActionListener {
	  public void actionPerformed(ActionEvent e) {
	      try {
		  User user = view.getUser();
		  Socket mySocket = new Socket(serverHost, serverPort);
		  ObjectOutputStream oos =
		      new ObjectOutputStream(mySocket.getOutputStream());
		  oos.writeObject(user);
```

```
                ObjectInputStream ois =
                    new ObjectInputStream(mySocket.getInputStream());
                Object o = ois.readObject();
                if(o instanceof String){
                    String result = (String)o;
                    if(result.equals("ok"))
                           view.showMessage("Login succesfully!");
                    else view.showMessage("Invalid username and/or password!");
                }
                mySocket.close();
            } catch (Exception ex) {
                view.showMessage(ex.getStackTrace().toString());
            }
        }
    }
}
```

ClientRun.java

```
package tcp.client;

public class ClientRun {
	public static void main(String[] args) {
		ClientView view = new ClientView();
		ClientControl control = new ClientControl(view);
		view.setVisible(true);
	}
}
```

5.3.2 Các lớp phía server

ServerView.java

```
package tcp.server;

public class ServerView {
	public ServerView(){
	}

	public void showMessage(String msg){
		System.out.println(msg);
	}
}
```

ServerControl.java

```
package tcp.server;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.ServerSocket;
import java.net.Socket;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
```

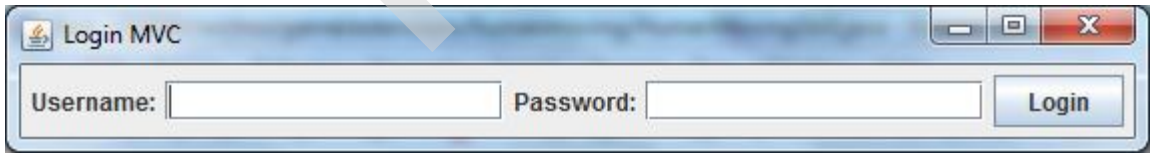
```java
import tcp.client.User;

public class ServerControl {
	private ServerView view;
	private Connection con;
	private ServerSocket myServer;
	private Socket clientSocket;
	private int serverPort = 8888;

	public ServerControl(ServerView view){
		this.view = view;
		getDBConnection("usermanagement", "root", "12345678");
		openServer(serverPort);
		view.showMessage("TCP server is running...");

		while(true){
			listenning();
		}
	}

	private void getDBConnection(String dbName,
					String username, String password){
	    String dbUrl = "jdbc:mysql://localhost:3306/" + dbName;
	    String dbClass = "com.mysql.jdbc.Driver";

	    try {
	       Class.forName(dbClass);
	       con = DriverManager.getConnection (dbUrl, username, password);
	    }catch(Exception e) {
	       view.showMessage(e.getStackTrace().toString());
	    }
	}

	private void openServer(int portNumber){
		try {
			myServer = new ServerSocket(portNumber);
		}catch(IOException e) {
			view.showMessage(e.toString());
		}
	}

	private void listenning(){
		try {
			clientSocket = myServer.accept();
			ObjectInputStream ois =
				new ObjectInputStream(clientSocket.getInputStream());
			ObjectOutputStream oos =
				new ObjectOutputStream(clientSocket.getOutputStream());

			Object o = ois.readObject();
			if(o instanceof User){
				User user = (User)o;
				if(checkUser(user)){
					oos.writeObject("ok");
				}
				else
					oos.writeObject("false");
```

```
                }
            }catch (Exception e) {
                view.showMessage(e.toString());
            }
        }


        private boolean checkUser(User user) throws Exception {
            String query = "Select * FROM users WHERE username ='"
                    + user.getUserName()
                    + "' AND password ='" + user.getPassword() + "'";
            try {
               Statement stmt = con.createStatement();
               ResultSet rs = stmt.executeQuery(query);
               if (rs.next()) {
                     return true;
               }
            }catch(Exception e) {
               throw e;
            }
            return false;
          }
}
```

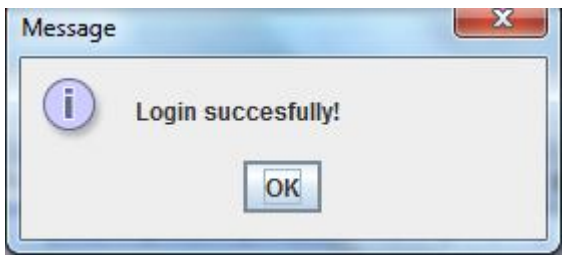
ServerRun.java

```
package tcp.server;

public class ServerRun {
        public static void main(String[] args) {
                ServerView view       = new ServerView();
                ServerControl control = new ServerControl(view);
        }
}
```

5.5 Kết quả



Login thành công:



Login lỗi:

IV. LẬP TRÌNH ỨNG DỤNG MẠNG VỚI UDPSOCKET

1. Giao thức UDP và cơ chế truyền thông của UDP

<Tham khảo giáo trình mạng máy tính>

2. Một số lớp Java hỗ trợ lập trình với UDPSocket

2.1. Lớp DatagramPacket

Lớp này cho phép tạo gói tin truyền thông với giao thức UDP. Lớp này kết thừa trực tiếp từ lớp Object.

public final class DatagramPacket extends Object

Gói tin là đối tượng của lớp này chứa 4 thành phần quan trọng: Địa chỉ, dữ liệu truyền thật sự, kích thước của gói tin và số hiệu cổng chứa trong gói tin.

2.1.1. Cấu tử

Lớp này có các cấu tử tạo gói tin gửi và gói tin nhận khác nhau:

* Cấu tử tạo gói tin nhận từ mạng:

public DatagramPacket(byte[] inBuffer, int length)

Tham số:

- inBuffer: Bộ đệm nhập, chứa dữ liệu của gói tin nhận
- length: kích cỡ của dữ liệu của gói tin nhận, nó thường được xác định bằng lệnh: *length= buffer.length.*

Ví dụ tạo gói tin nhận:

```
byte[] inBuff=new byte[512];//bộ đệm nhập
DatagramPacket inData=new DatagramPacket(inBuf, inBuff.length);
```

* Cấu tử tạo gói tinh gửi:

public DatagramPacket(byte[] outBuffer , int length,
InetAddress destination, int port)

Tham số:

- outBuffer: Bộ đệm xuất chưa dữ liệu của gói tin gửi
- length: kích cỡ dữ liệu của gói tin gửi tính theo số byte và thường bằng outBuffer.length.

- destination: Địa chỉ nơi nhận gói tin
- port: Số hiệu cổng đích, nơi nhận gói tin.

Ví dụ:

```
      String s=" Hello World!";
//Bộ đệm xuất và gán dữ liệu cho bộ đệm xuất
      byte[] outBuff=s.getBytes();
//Địa chỉ đích
      InetAddress addrDest=InetAddress.getByName("localhost");
//Số cổng đích
      int   portDest=3456;
//Tạo gói tin gửi
      DatagramPacket outData=new DatagramPacket(outBuff,
                              outBuff.length, addrDest, portDest);
```

2.1.2. Phương thức

- *public InetAddress getAddress( ):* Phương thức này trả về đối tượng InetAddress của máy trạm từ xa chứa trong gói tin nhận.
- *public int getPort( ):* Trả về số hiệu cổng của máy trạm từ xa chứa trong gói tin.
- *public byte[] getData( )*: Trả về dữ liệu chứa trong gói tin dưới dạng mảng byte.
- *public int getLength( ):* Trả về kích cỡ của dữ liệu chưa trong gói tin tính theo số byte.

Tương ứng với 4 phương thức getXXXX..(), lớp DatagramPacket có 4 phương thức setXXXX..() để thiết lập 4 tham số cho gói tin gửi.

2.2. Lớp DatagramSocket

Lớp DatagramSocket cho phép tạo ra đối tượng socket truyền thông với giao thức UDP. Socket này cho phép gửi/nhận gói tin DatagramPacket. Lớp này được khai báo kế thừa từ lớp Object.

public class DatagramSocket extends Object

2.2.1. Các cấu tử (phương thức khởi tạo)

- *public DatagramSocket( ) throws SocketException:*

Cấu tử này cho phép tạo ra socket với số cổng nào đó(anonymous) và thường được sử dụng phía chương trình client. Nếu tạo socket không thành công, nó ném trả về ngoại lệ *SocketException.*
Ví dụ:

```
      try {

        DatagramSocket client = new DatagramSocket( );

        // send packets...

      }
```

```
catch (SocketException ex) {

  System.err.println(ex);

}
```

- *public DatagramSocket(int port) throws SocketException:*

Cấu tử này cho phép tạo socket với số cổng xác định và chờ nhận gói tín truyền tới. Cấu tử này được sử dụng phía server trong mô hình client/server. Ví dụ chương trình sau sẽ cho phép hiển thị các cổng cục bộ đã được sử dụng:

```
//UDPPortScanner.java
import java.net.*;
public class UDPPortScanner {
  public static void main(String[] args) {
    for (int port = 1024; port <= 65535; port++) {
      try {
        // the next line will fail and drop into the catch block if
        // there is already a server running on port i
       DatagramSocket server = new DatagramSocket(port);
        server.close( );
      }
      catch (SocketException ex) {
        System.out.println("There is a server on port " + port + ".");
      } // end try
    } // end for
  }
  }
```

2.2.2. Các phương thức

- *public void send(DatagramPacket dp) throws IOException:*

Phương thức này cho phép gửi gói tin UDP qua mạng. Ví dụ chương trình sau nhận một chuỗi từ bàn phím, tạo gói tin gửi và gửi tới server.

```
//UDPDiscardClient.java
import java.net.*;
import java.io.*;
public class UDPDiscardClient {
  public final static int DEFAULT_PORT = 9;
  public static void main(String[] args) {
    String hostname;
    int port = DEFAULT_PORT;
    if (args.length > 0) {
      hostname = args[0];
```

```
        try {
                port = Integer.parseInt(args[1]);
        }
        catch (Exception ex) {
          // use default port
        }
      }
      else {
        hostname = "localhost";
      }
      try {
        InetAddress server = InetAddress.getByName(hostname);
        BufferedReader userInput
         = new BufferedReader(new InputStreamReader(System.in));
        DatagramSocket theSocket = new DatagramSocket( );
        while (true) {
          String theLine = userInput.readLine( );
          if (theLine.equals(".")) break;
          byte[] data = theLine.getBytes( );
          DatagramPacket theOutput
           = new DatagramPacket(data, data.length, server, port);
          theSocket.send(theOutput);
        }  // end while
      }  // end try
      catch (UnknownHostException uhex) {
        System.err.println(uhex);
      }
      catch (SocketException sex) {
        System.err.println(sex);
      }
      catch (IOException ioex) {
        System.err.println(ioex);
      }
    }  // end main
}
```

- *public void receive(DatagramPacket dp) throws IOException:*

Phương thức nhận gói tin UDP qua mạng. Ví dụ chương trình sau sẽ tạo đối tượng DatagramSocket với số cổng xác định, nghe nhận gói dữ liệu gửi đến, hiển thị nội dung gói tin và địa chỉ, số cổng của máy trạm gửi gói tin.

```
//UDPDiscardServer.java
import java.net.*;
import java.io.*;
public class UDPDiscardServer {
  public final static int DEFAULT_PORT = 9;
  public final static int MAX_PACKET_SIZE = 65507;
  public static void main(String[] args) {
    int port = DEFAULT_PORT;
  byte[] buffer = new byte[MAX_PACKET_SIZE];
    try {
     port = Integer.parseInt(args[0]);
    }
    catch (Exception ex) {
      // use default port
    }
    try {
      DatagramSocket server = new DatagramSocket(port);
      DatagramPacket packet = new DatagramPacket(buffer, buffer.length);
      while (true) {
        try {
          server.receive(packet);
          String s = new String(packet.getData( ), 0, packet.getLength( ));
          System.out.println(packet.getAddress( ) + " at port "
           + packet.getPort( ) + " says " + s);
          // reset the length for the next packet
          packet.setLength(buffer.length);
        }
        catch (IOException ex) {
          System.err.println(ex);
        }
       } // end while
    }  // end try
    catch (SocketException  ex) {
      System.err.println(ex);

    }  // end catch
  }  // end main
}
```

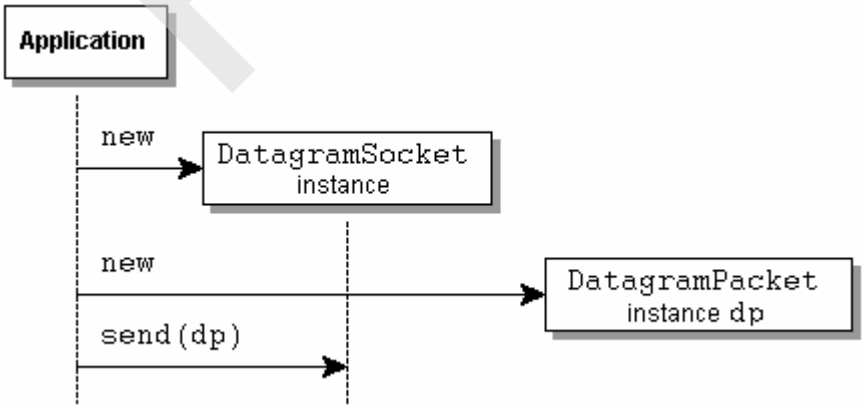
- *public void close( ):* Phương thức đóng socket.

Các phương thức khác thể hiện trong bảng sau:

Một số phương thức của lớp DatagramSocket	
void	***bind***(*SocketAddress* `addr`) Gắn kết DatagramSocket với địa chỉ và số cổng cụ thể
void	***connect***(*InetAddress* `address,int port`) Kết nối socket với địa chỉ máy trạm từ xa
void	***connect***(*SocketAddress* `addr`) Kết nối socket với địa chỉ socket từ xa.
void	***disconnect***() Huỷ bỏ kết nối
boolean	***isBound***() Trả về trạng thái kết nối của socket.
boolean	***isClosed***() Kiểm tra socket đã đóng hay chưa
boolean	***isConnected***() Kiểm tra trạng thái kết nối

3. Kỹ thuật lập trình truyền thông với giao thức UDP

Trong mô hình client/server, để chương trình client và server có thể truyền thông được với nhau, mỗi phía phải thực hiện một số thao tác cơ bản sau đây(Hình 2.3)



Hình 2.6. Quá trình khởi tạo truyền thông UDPSocket

3.1. Phía server:

- Tạo đối tượng DatagramSocket với số cổng xác định được chỉ ra
- Khai báo bộ đệm nhập /xuất inBuffer/outBuffer dạng mảng kiểu byte

- Khai báo gói tin nhận gửi inData/outData là đối tượng DatagramPacket.
- Thực hiện nhận/gửi gói tin với phương thức receive()/send()
- Đóng socket, giải phóng các tài nguyên khác, kết thúc chương trình nếu cần, không quay về bước 3.

3.2. Phía client

- Tạo đối tượng DatagramSocket với số cổng nào đó
- Khai báo bộ đệm xuất/nhập outBuffer/inBuffer dạng mảng kiểu byte
- Khai báo gói tin gửi/nhận outData/inData là đối tượng DatagramPacket.
- Thực hiện gửi /nhận gói tin với phương thức send()/receive()
- Đóng socket, giải phóng các tài nguyên khác, kết thúc chương trình nếu cần, không quay về bước 3.

3.3. Một số lưu ý:

- Chương trình server phải chạy trước chương trình client và chương trình client phải gửi gói tin đến server trước. Để từ gói tín nhận được phía server, server mới tách được địa chỉ và số hiệu cổng phía client , từ đó mới tạo gói tin gửi cho client.
- Chương trình server có thể phục vụ nhiều máy khách kiểu lặp.

4. Một số chương trình ví dụ

4.1. Chương trình minh hoạ

```
//UDPEchoClient.java
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(hostname);
      Thread sender = new SenderThread(ia, DEFAULT_PORT);
      sender.start( );
      Thread receiver = new ReceiverThread(sender.getSocket( ));
      receiver.start( );
    }
    catch (UnknownHostException ex) {
      System.err.println(ex);
```

```
    }
    catch (SocketException ex) {
      System.err.println(ex);
    }
  }  // end main
}
```

```
//UDPEchoServer.java
import java.net.*;
import java.io.*;
public class  extends UDPServer {
  public final static int DEFAULT_PORT = 7;
  public UDPEchoServer( ) throws SocketException {
   super(DEFAULT_PORT);
  }
  public void respond(DatagramPacket packet) {
    try {
      DatagramPacket outgoing = new DatagramPacket(packet.getData( ),
       packet.getLength( ), packet.getAddress( ), packet.getPort( ));
      socket.send(outgoing);
    }
    catch (IOException ex) {
      System.err.println(ex);
    }
  }
  public static void main(String[] args) {
   try {
     UDPServer server = new UDPEchoServer( );
     server.start( );
  }
   catch (SocketException ex) {
     System.err.println(ex);
   }
  }
}
```

V. LẬP TRÌNH VỚI THẺ GIAO TIẾP MẠNG(NIC)

1. Giới thiệu về thẻ giao tiếp mạng(*network interface card-NIC)*

Thẻ giao tiếp mạng là điểm liên kết giữa máy tính với mạng riêng hoặc mạng công cộng. Giao tiếp mạng nói chung là một thẻ giao tiếp mạng(NIC) nhưng nó cũng có thể không phải giao tiếp vật lý. Mà thay vào đó giao tiếp mạng có thể được thực hiện trong dạng phần mềm. Ví dụ giao

tiếp loopback(127.0.0.1 đối với IPv4 và ::1 đối với IPv6) không phải là dạng thiết bị vật lý mà là một phần mềm phỏng theo giao tiếp mạng vật lý. Giao tiếp loopback noi chung được sử dụng trong môi trường thử nghiệm.

2. Lớp NetworkInterface

Lớp này dùng cho cả thẻ giao tiếp vật lý như Ethernet Card hoặc thẻ giao tiếp ảo mà được tạo ra tương tự giống như thẻ giao tiếp vật lý. Lớp NetworkInterface cung cấp các phương thức để liệt kê tất cả các địa chỉ cục bộ và tạo ra đối tượng InetAddress từ chúng. Các đối tượng InetAddress này có thể được sử dụng để tạo các socket, server socket...

Đối tượng NetworkInterface thể hiện phần cứng vật lý hoặc địa chỉ ảo và chúng không thể được xây dựng tuỳ ý. Cũng tương tự như lớp InetAdddress, nó cũng có một số phương thức có thuộc tính static cho phép trả về đối tượng NetworkInterface gắn kết với bộ giao tiếp mạng cụ thể. Sau đây chúng ta sẽ khảo sát một số phương thức quan trọng của lớp NetworkInterface.

2.1. Các phương thức static

- Phương thức *getByName():*

Cú pháp:

public static NetworkInterface getByName(String name)

throws SocketException

Phương thức này trả về đối tượng NetworkInterface biểu diễn một bộ giao tiếp mạng với tên cụ thể. Nếu không có tên đó thì nó trả về giá trị null. Nếu các tầng mạng nền tảng xẩy ra vấn đề, phương thức trả về ngoại lệ *SocketException.* Dạng tên giao tiếp mạng phụ thuộc vào nền cụ thể. Với hệ điều hành Unix, tên của giao tiếp Ethernet có dạng eth0, eth1,...Địa chỉ loopback cục bộ có thể đặt tên chẳng hạn như "lo". Đối với hệ điều hành Windows, tên là các chuỗi "CE31", "ELX100" mà được lấy từ các nhà cung cấp và mô hình phần cứng trên phần cứng giao tiếp mạng đó. Ví dụ đoạn chương trình sau thực hiện tìm giao tiếp mạng Etthernet cơ sở trên hệ điều hành Unix:

```
try {

  NetworkInterface ni = NetworkInterface.getByName("eth0");

  if (ni == null) {

    System.err.println("No such interface:  eth0" );

  }

}
catch (SocketException ex) {

 System.err.println("Could not list sockets." );
```

```
}
```

- Phương thức *getByInetAddress():*

Cú pháp:

public static NetworkInterface getByInetAddress(InetAddress address)
throws SocketException

Phương thức này trả về đối tượng NeworkInterface biểu diễn giao tiếp mạng được gắn với với một địa chỉ IP cụ thể, Nếu không có giao tiếp mạng gắn với địa chỉ IP đó trên máy trạm cục bộ thì nó trả về null. Khi xẩy ra lỗi nó ném trả về ngoại lệ SocketException. ví dụ đoạn chương trình sau minh hoạ cách sử dụng phương thức để tìm giao tiếp mạng đối với địa chỉ loopback cục bộ:

```
try {
  InetAddress local = InetAddress.getByName("127.0.0.1");
  NetworkInterface ni = NetworkInterface.getByName(local);
  if (ni == null) {
    System.err.println("That's weird. No local loopback address.");
  }
}
catch (SocketException ex) {
  System.err.println("Could not list sockets." );
}
catch (UnknownHostException ex) {
  System.err.println("That's weird. No local loopback address.");
}
```

- Phương thức *getNetworkInterfaces( ):*

Cú pháp:

public static Enumeration getNetworkInterfaces( ) throws SocketException

Phương thức này trả về đối tượng java.util.Enumeration là một danh sách liệt kê tất cả các giao tiếp mạng có trên máy cục bộ. Chương trình ví dụ sau minh hoạ cách sử dụng phương thức để đưa ra một danh sách tất cả các giao tiếp mạng trên máy cục bộ:

```
//InterfaceLister.java
import java.net.*;
import java.util.*;
public class InterfaceLister {
    public static void main(String[] args) throws Exception {
      Enumeration interfaces = NetworkInterface.getNetworkInterfaces( );
  while (interfaces.hasMoreElements( )) {
   NetworkInterface ni = (NetworkInterface)      interfaces.nextElement(
);
        System.out.println(ni);
```

```
        }
      }
  }
```

2.2. Các phương thức khác:

- *public Enumeration getInetAddresses( )*: Phương thức này trả về đối tượng `java.util.Enumeration` chứa đối tượng InetAddress đối với mỗi địa chỉ IP mà giao tiếp mạng được với nó. Mà mỗi giao tiếp mạng đơn có thể gắn với các địa chỉ IP khác nhau. Ví dụ sau hiển thị tất cả các địa chỉ IP gắn với giao diện mạng eth0:

```
NetworkInterface eth0 = NetworkInterrface.getByName("eth0");
Enumeration addresses = eth0.getInetAddresses( );
while (addresses.hasMoreElements( )) {
  System.out.println(addresses.nextElement( ));
}
```

- *public String getName( )*: Phương thức này trả về tên của đối tượng NetworkInterface cụ thể, chẳng hạn như eth0 hoặc lo.
- *public String getDisplayName( )*:

Phương thức trả về tên "thân thiện" hơn của một giao tiếp mạng cụ thể. Trong mạng Unix, nó trả về chuỗi giống như phương thức *getName(),* Trong mạng Windows, no trả về chuỗi tên "thân thiện" như "Local Area Connection" hoặc "Local Area Connection 2".

Ngoài ra trong lớp NetworkInterface còn định nghĩa các phương thức equals(), hashCode(), toString().

3. Lập trình với giao tiếp mạng(NIC)

Lớp NetworkInterface thể hiện cả 2 kiểu giao diện vật lý và giao tiếp mềm. Lớp này đầy hữu ích đối với các hệ thống multihome có nhiều NIC. Với lớp này, chương trình có thể chỉ ra NIC cho một hoạt động mạng cụ thể.

Để gửi dữ liệu, hệ thống xác định giao tiếp nào sẽ được sử dụng. Nhưng cũng có thể truy vấn hệ thống đối với các giao tiếp phù hợp và tìm một địa chỉ trên giao tiếp muốn sử dụng. Khi chương trình tạo ra một socket và gắn nó với địa chỉ đó, hệ thống sẽ sử dụng giao tiếp được gắn kết đó.

Ví dụ:

```
NetworkInterface nif = NetworkInterface.getByName("bge0");
  Enumeration nifAddresses = nif.getInetAddresses();
  Socket soc = new java.net.Socket();
  soc.bind(nifAddresses.nextElement());
  soc.connect(new InetSocketAddress(address, port));
```

Người sử dụng cũng có thể sử dụng NetworkInterface để nhận biết giao tiếp cục bộ mà một nhóm multicast được ghép nối, ví dụ:

```
NetworkInterface nif = NetworkInterface.getByName("bge0");
MulticastSocket() ms = new MulticastSocket();
ms.joinGroup(new InetSocketAddress(hostname, port) , nif);
```

3.1. Lấy các giao tiếp mạng

Lớp NetworkInterface không có cấu tử public. Do đó không thể tạo được đối tượng với toán tử new. Thay vào đó nó có các phương thức static(giống InetAddress) cho phép lấy được các chi tiết giao tiếp từ hệ thống: *getByInetAddress()*, *getByName()* và *getNetworkInterfaces()*. Hai phương thức đầu tiên được sử dụng khi có sẵn địa chỉ IP hoặc tên của giao tiếp mạng cục thể. Phương thức thứ 3, *getNetworkInterfaces( )*, trả về một danh sách đầy đủ các giao tiếp mạng trên máy tính.

Giao tiếp mạng cũng có thể tổ chức theo kiểu phân cấp. Lớp NetworkInterface sử dụng 2 phương thức *getParent()* và *getSubInterface()* đối với cấu trúc giao tiếp mạng phân cấp. Nếu giao tiếp mạng là giao tiếp con, *getParent()* trả về giá trị none-null. Phương thức *getSubInterfaces()* sẽ trả về tất cả các giao tiếp con của giao tiếp mạng. Ví dụ sau đây sẽ hiển thị tên của tất cả các giao tiếp mạng và giao tiếp con(n nếu nó tồn tại) trên một máy:

```
  //ListNIFs.java
import java.io.*;
import java.net.*;
import java.util.*;
import static java.lang.System.out;

public class ListNIFs
{
    public static void main(String args[]) throws SocketException {
        Enumeration<NetworkInterface> nets =
NetworkInterface.getNetworkInterfaces();

        for (NetworkInterface netIf : Collections.list(nets)) {
            out.printf("Display name: %s\n", netIf.getDisplayName());
            out.printf("Name: %s\n", netIf.getName());
            displaySubInterfaces(netIf);
            out.printf("\n");
        }
    }

    static void displaySubInterfaces(NetworkInterface netIf) throws
SocketException {
        Enumeration<NetworkInterface> subIfs = netIf.getSubInterfaces();

        for (NetworkInterface subIf : Collections.list(subIfs)) {
            out.printf("\tSub Interface Display name: %s\n",
subIf.getDisplayName());
            out.printf("\tSub Interface Name: %s\n", subIf.getName());
        }
     }
}
```

Kết quả chạy trên máy tính của chúng tôi hiện ra như sau:

```
Display name: bge0
Name: bge0
    Sub Interface Display name: bge0:3
    Sub Interface Name: bge0:3
    Sub Interface Display name: bge0:2
    Sub Interface Name: bge0:2
    Sub Interface Display name: bge0:1
    Sub Interface Name: bge0:1

Display name: lo0
```

```
Name: lo0
```

3.2. Lấy danh sách địa chỉ giao tiếp mạng

Một phần thông tin cực kỳ hữu ích mà người sử dụng cần lấy được từ giao tiếp mạng là danh sách địa chỉ IP mà được gắn cho các giao tiếp mạng. Người sử dụng có thể thu được thông tin từ một thể hiện NetworkInterface bằng cách sử dụng một trong 2 phương thức sau: Phương thức *getInetAddresses()* trả về một *Enumeration* của các đối tượng *InetAddress*, còn phương thức *getInterfaceAddresses()* trả về một danh sách của các thể hiện *java.net.InterfaceAddress*. Phương thứcc này được sử dụng khi người sử dụng cần thông tin nhiều hơn về địa chỉ giao tiếp ngoài địa chỉ IP của nó. Ví dụ, khi bạn cần thông tin bổ sung về mặt nạ mạng con và địa chỉ broardcast khi địa chỉ là một địa chỉ IPv4 và chiều dài prefix mạng trong địa chỉ IPv6. Ví dụ sau đây hiển thị danh sách tất cả các giao tiếp mạng và địa chỉ của chúng trên một máy:

```
import java.io.*;
import java.net.*;
import java.util.*;
import static java.lang.System.out;

public class ListNets
{
    public static void main(String args[]) throws SocketException {
        Enumeration<NetworkInterface> nets =
NetworkInterface.getNetworkInterfaces();
        for (NetworkInterface netint : Collections.list(nets))
            displayInterfaceInformation(netint);
    }

    static void displayInterfaceInformation(NetworkInterface netint)
throws SocketException {
        out.printf("Display name: %s\n", netint.getDisplayName());
        out.printf("Name: %s\n", netint.getName());
        Enumeration<InetAddress> inetAddresses =
netint.getInetAddresses();
        for (InetAddress inetAddress : Collections.list(inetAddresses))
{
            out.printf("InetAddress: %s\n", inetAddress);
        }
        out.printf("\n");
     }
}
```

Kết quả chạy chương trình trên máy tính của chúng tôi như sau:

```
Display name: bge0
Name: bge0
InetAddress: /fe80:0:0:0:203:baff:fef2:e99d%2
InetAddress: /121.153.225.59
Display name: lo0
Name: lo0
InetAddress: /0:0:0:0:0:0:0:1%1
InetAddress: /127.0.0.1
```

3.3. Truy cập các tham số giao tiếp mạng

Người sử dụng có thể truy cập các tham số về giao tiếp mạng ngoài tên và địa chỉ IP gán cho nó. Và chương trình có thể phát hiện giao tiếp mạng đang chạy với phương thức *isUp().* các phương thức sau chỉ thị kiểu giao tiếp mạng:

- *isLoopback():* chỉ thị giao tiếp mạng là một giao tiếp loopback.
- *isPointToPoint()* chỉ thị nếu giao tiếp là giao tiếp point-to-point.
- *isVirrtual():* chỉ thị nếu giao tiếp là giao tiếp ảo(giao tiếp mềm).

Phương thức *supportsMulticast()* chỉ thị một khi giao tiếp mạng hỗ trợ multicast. Phương thức *getHardwareAddress()* trả về địa chỉ phần cứng vật lý của giao tiếp mạng, địa chỉ MAC, khi nó có khả năng. Phương thức *getMTU()* trả về đơn vị truyền cực đại(MTU) là kích cỡ gói tin lớn nhất. Ví dụ sau mở rộng của ví dụ trên bằng cách thêm các tham số mạng bổ sung:

```
//ListNetsEx.java
import java.io.*;
import java.net.*;
import java.util.*;
import static java.lang.System.out;

public class ListNetsEx
{
    public static void main(String args[]) throws SocketException {
        Enumeration<NetworkInterface> nets =
NetworkInterface.getNetworkInterfaces();
        for (NetworkInterface netint : Collections.list(nets))
            displayInterfaceInformation(netint);
    }

    static void displayInterfaceInformation(NetworkInterface netint) throws
SocketException {
        out.printf("Display name: %s\n", netint.getDisplayName());
        out.printf("Name: %s\n", netint.getName());
        Enumeration<InetAddress> inetAddresses = netint.getInetAddresses();

        for (InetAddress inetAddress : Collections.list(inetAddresses)) {
            out.printf("InetAddress: %s\n", inetAddress);
        }

        out.printf("Up? %s\n", netint.isUp());
        out.printf("Loopback? %s\n", netint.isLoopback());
        out.printf("PointToPoint? %s\n", netint.isPointToPoint());
        out.printf("Supports multicast? %s\n", netint.supportsMulticast());
        out.printf("Virtual? %s\n", netint.isVirtual());
        out.printf("Hardware address: %s\n",
                   Arrays.toString(netint.getHardwareAddress()));
        out.printf("MTU: %s\n", netint.getMTU());

        out.printf("\n");

     }
}
```

Kết quả chạy chương trình trên máy tính của chúng tôi như sau:

```
Display name: bge0
Name: bge0
InetAddress: /fe80:0:0:0:203:baff:fef2:e99d%2
```

```
InetAddress: /129.156.225.59
Up? true
Loopback? false
PointToPoint? false
Supports multicast? false
Virtual? false
Hardware address: [0, 3, 4, 5, 6, 7]
MTU: 1500

Display name: lo0
Name: lo0
InetAddress: /0:0:0:0:0:0:0:1%1
InetAddress: /127.0.0.1
Up? true
Loopback? true
PointToPoint? false
Supports multicast? false
Virtual? false
Hardware address: null
MTU: 8232
```

4. Một số chương trình ví dụ minh hoạ sử dụng lớp NetworkInterface và InetAddress

```
//InetExample.java
 import java.util.Enumeration;
 import java.net.*;
 public class InetExample {
public static void main(String[] args) {
 // Get the network interfaces and associated addresses for this host
try {
Enumeration<NetworkInterface>              interfaceList              =
NetworkInterface.getNetworkInterfaces();
if (interfaceList == null) {
System.out.println("--No interfaces found--");
 } else {
while (interfaceList.hasMoreElements()) {
 NetworkInterface iface = interfaceList.nextElement();
 System.out.println("Interface " + iface.getName() + ":");
Enumeration<InetAddress> addrList = iface.getInetAddresses();
if (!addrList.hasMoreElements()) {
System.out.println("\t(No addresses for this interface)");
 }
while (addrList.hasMoreElements()) {
 InetAddress address = addrList.nextElement();
 System.out.print("\tAddress  "  +  ((address  instanceof  Inet4Address  ?
"(v4)"
```

```
 : (address instanceof Inet6Address ? "(v6)" : "(?)")));
 System.out.println(": " + address.getHostAddress());
}
 }
}
} catch (SocketException se) {
 System.out.println("Error getting network interfaces:" +
se.getMessage());
 }
 // Get name(s)/address(es) of hosts given on command line
 for (String host : args) {
 try {
System.out.println(host + ":");
 InetAddress[] addressList = InetAddress.getAllByName(host);
 for (InetAddress address : addressList) {
System.out.println("\t" + address.getHostName() + "/" +
address.getHostAddress());
 }
 } catch (UnknownHostException e) {
System.out.println("\tUnable to find address for " + host);
}
}
}
 }
```


5. Case study: Login từ xa dùng giao thức UDP

5.1 Bài toán

Bài toán login từ xa dùng giao thức UDP đặt ra như sau:

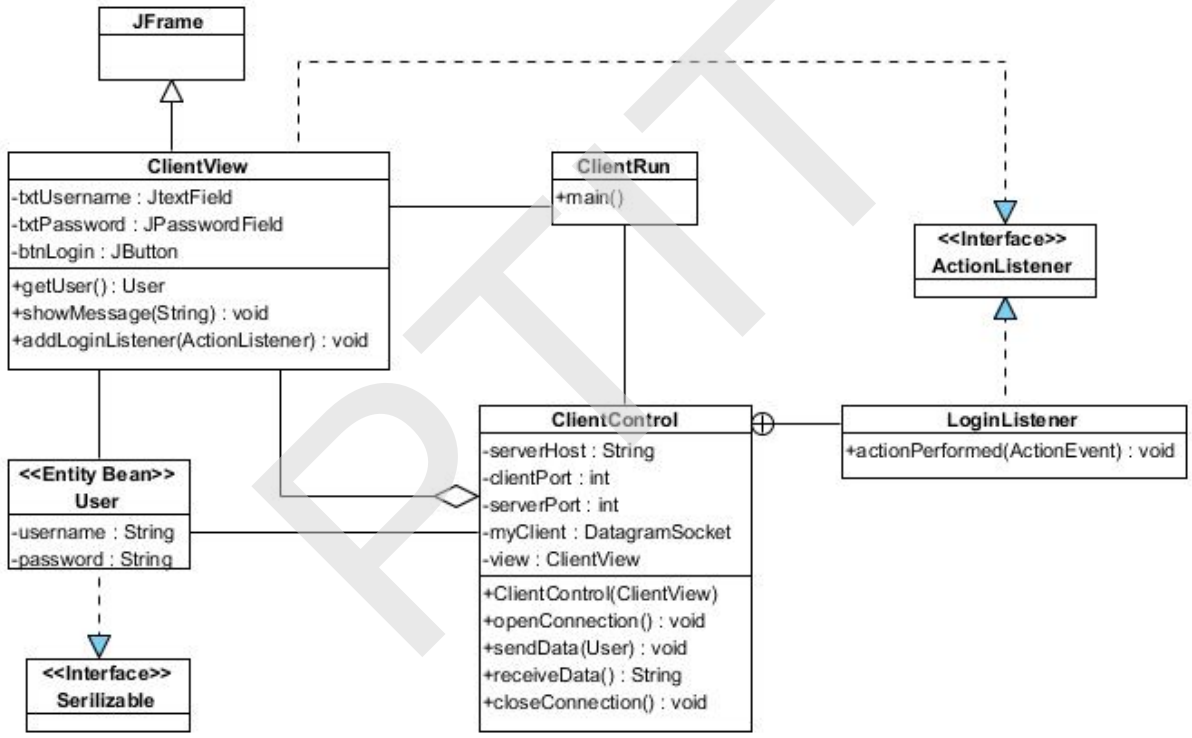
- Cở sở dữ liệu đợc lưu trữ và quản lí trên server UDP, trong đó có bảng users chứa ít nhất hai cột: cột username và cột password.
- Chương trình phía client UDP phải hiện giao diện đồ họa, trong đó có một ô text để nhập username, một ô text để nhập password, và một nút nhấn Login.
- Khi nút Login được click, chương trình client sẽ gửi thông tin đăng nhập (username/password) trên form giao diện, và gửi sang server theo giao thức UDP
- Tại phía server, mỗi khi nhận được thông tin đăng nhập gửi từ client, nó sẽ tiến hành kiểm tra trong cơ sở dữ liệu xem có tài khoản nào trùng với thông tin đăng nhập nhận được hay không.

- Sau khi có kết quả kiểm tra (đăng nhập đúng, hoặc sai), server UDP sẽ gửi kết quả này về cho client tương ứng, theo đúng giao thức UDP.
- Ở phía client, sau khi nhận được kết quả đăng nhập (đăng nhập đúng, hoặc sai) từ server, nó sẽ hiển thị thông báo tương ứng với kết quả nhận được: nếu đăng nhập đúng thì thông báo login thành công. Nếu đăng nhập sai thì thông báo là username/password không đúng.
- Yêu cầu kiến trúc hệ thống ở cả hai phía client và server đều được thiết kế theo mô hình MVC

5.2 Kiến trúc hệ thống theo mô hình MVC

Vì hệ thống được thiết kế theo mô hình client/server dùng giao thức UDP nên mỗi phía client, server sẽ có một sơ đồ lớp riêng, các sơ đồ này được thiết kế theo mô hình MVC.

5.2.1 Sơ đồ lớp phía client



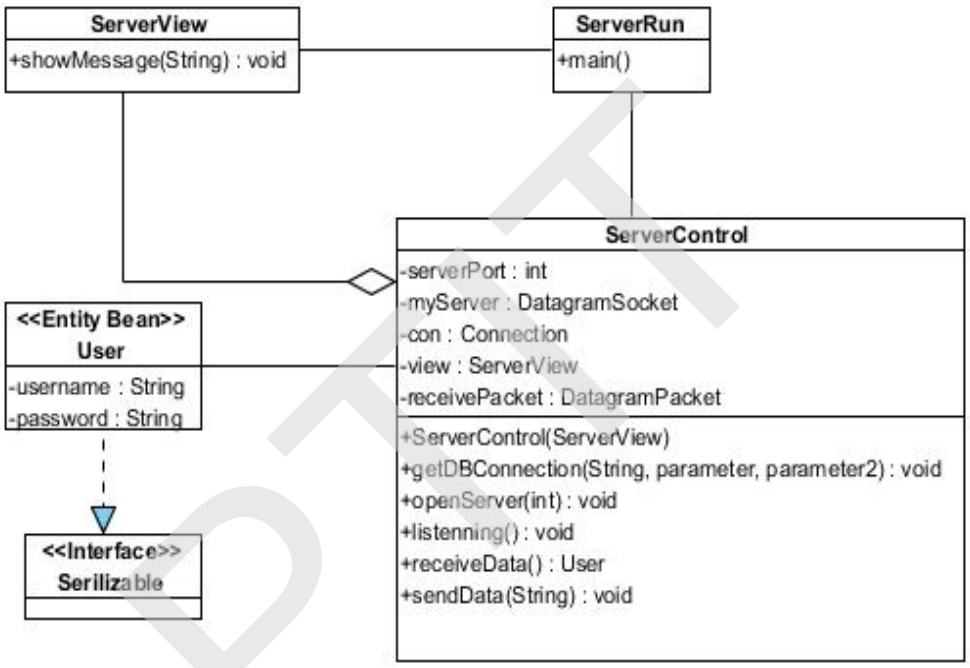
Hình 2.7: Sơ đồ lớp phía client UDP

Sơ đồ lớp của phía client được thiết kế theo mô hình MVC trong Hình 2.7, bao gồm 3 lớp chính tương ứng với sơ đồ M-V-C như sau:

- Lớp User: là lớp tương ứng với thành phần model (M), bao gồm hai thuộc tính username và password, các hàm khởi tạo và các cặp getter/setter tương ứng với các thuộc tính.
- Lớp ClientView: là lớp tương ứng với thành phần view (V), là lớp form nên phải kế thừa từ lớp JFrame của Java, nó chứa các thuộc tính là các thành phần đồ họa bao gồm ô text nhập username, ô text nhập password, nút nhất Login.

- Lớp ClientControl: là lớp tương ứng với thành phần control (C), nó chứa một lớp nội tại là LoginListener. Khi nút Login trên tầng view bị click thì nó sẽ chuyển tiếp sự kiện xuống lớp nội tại này để xử lí. Tất cả các xử lí đều gọi từ trong phương thức actionPerformed của lớp nội tại này, bao gồm: lấy thông tin trên form giao diện và gửi sang server theo giao thức UDP, nhận kết quả đăng nhập từ server về và yêu cầu form giao diện hiển thị. Điều này đảm bảo nguyên tắc control điều khiển các phần còn lại trong hệ thống, đúng theo nguyên tắc của mô hình MVC.

5.2.2 Sơ đồ lớp phía server

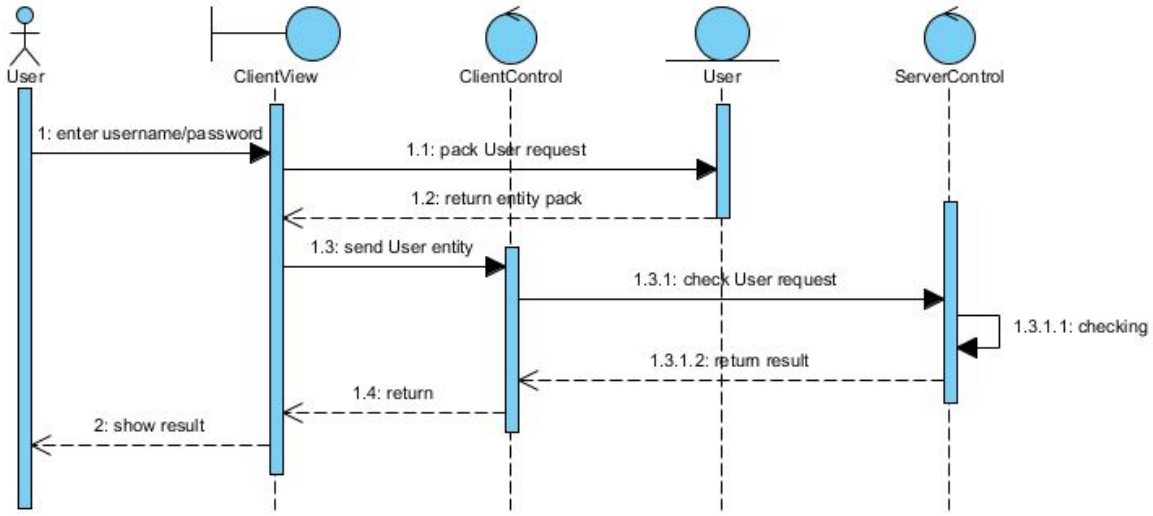


Hình 2.8: Sơ đồ lớp phía server UDP

Sơ đồ lớp của phía server được thiết kế theo mô hình MVC trong Hình 2.8, bao gồm 3 lớp chính tương ứng với sơ đồ M-V-C như sau:

- Lớp User: là lớp thực thể, dùng chung thống nhất với lớp phía bên client.
- Lớp ServerView: là lớp tương ứng với thành phần view (V), là lớp dùng hiển thị các thông báo và trạng thái hoạt động bên server UDP.
- Lớp ServerControl: là lớp tương ứng với thành phần control (C), nó đảm nhiệm vai trò xử lí của server UDP, bao gồm: nhận thông tin đăng nhập từ phía các client, kiểm tra trong cơ sở dữ liệu xem các thng tin này đúng hay sai, sau đó gửi kết quả đăng nhập về cho client tương ứng.

5.2.3 Tuần tự các bước thực hiện



Hình 2.9: Tuần tự các bước thực hiện theo giao thức UDP

Tuần tự các bước xử lí như sau (Hình 2.9):

1. Ở phía client, người dùng nhập username/password và click vào giao diện của lớp ClientView
2. Lớp ClientView sẽ đóng gói thông tin username/password trên form vào một đối tượng model User bằng phương thức getUser() và chuyển xuống cho lớp ClientControl xử lí
3. Lớp ClientControl gửi thông tin chứa trong đối tượng User này sang phía server để kiểm tra đăng nhập
4. Bên phía server, khi nhận được thông tin đăng nhập trong đối tượng User, nó sẽ gọi phương thức checkLogin() để kểm tra thông tin đăng nhập trong cơ sở dữ liệu.
5. Kết quả kiểm tra sẽ được trả về cho lớp ClientControl
6. Ở phía client, khi nhận được kết quả kiểm tra đăng nhập, lớp ClientControl sẽ chuyển cho lớp LoginView hiển thị bằng phương thức showMessage()
7. Lớp LoginView hiển thị kết quả đăng nhập lên cho người dùng

5.3 Cài đặt

5.3.1 Các lớp phía client

User.java

```
package udp.client;
import java.io.Serializable;

public class User implements Serializable{
	private String userName;
	private String password;

	public User(){
	}
```

```
	public User(String username, String password){
		this.userName = username;
		this.password = password;
	}

	public String getPassword() {
		return password;
	}

	public void setPassword(String password) {
		this.password = password;
	}

	public String getUserName() {
		return userName;
	}

	public void setUserName(String userName) {
		this.userName = userName;
	}
}
```

ClientView.java

```
package udp.client;
import java.awt.FlowLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JPasswordField;
import javax.swing.JTextField;

public class ClientView extends JFrame implements ActionListener{
	private JTextField txtUsername;
	private JPasswordField txtPassword;
	private JButton btnLogin;

	public ClientView(){
		super("UDP Login MVC");

		txtUsername = new JTextField(15);
		txtPassword = new JPasswordField(15);
		txtPassword.setEchoChar('*');
		btnLogin = new JButton("Login");

		JPanel content = new JPanel();
		content.setLayout(new FlowLayout());
		content.add(new JLabel("Username:"));
		content.add(txtUsername);
		content.add(new JLabel("Password:"));
		content.add(txtPassword);
```

```
		content.add(btnLogin);

		this.setContentPane(content);
		this.pack();

		this.addWindowListener(new WindowAdapter(){
			public void windowClosing(WindowEvent e){
				System.exit(0);
			}
		});
	}

	public void actionPerformed(ActionEvent e) {
	}

	public User getUser(){
		User model = new User(txtUsername.getText(), txtPassword.getText());
		return model;
	}

	public void showMessage(String msg){
		JOptionPane.showMessageDialog(this, msg);
	}

	public void addLoginListener(ActionListener log) {
		  btnLogin.addActionListener(log);
		}
}
```

ClientControl.java

```
package udp.client;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;

public class ClientControl {
	private ClientView view;
	private int serverPort = 5555;
	private int clientPort = 6666;
	private String serverHost = "localhost";
	private DatagramSocket myClient;

	public ClientControl(ClientView view){
		this.view = view;
		this.view.addLoginListener(new LoginListener());
	}

	class LoginListener implements ActionListener {
	  public void actionPerformed(ActionEvent e) {
		openConnection();
```

```
        User user = view.getUser();
        sendData(user);

        String result = receiveData();
        if(result.equals("ok"))
                view.showMessage("Login succesfully!");
         else
                view.showMessage("Invalid username and/or password!");

         closeConnection();
    }
}

   private void openConnection(){
         try {
        myClient = new DatagramSocket(clientPort);
    } catch (Exception ex) {
        view.showMessage(ex.getStackTrace().toString());
    }
   }

   private void closeConnection(){
         try {
        myClient.close();
    } catch (Exception ex) {
        view.showMessage(ex.getStackTrace().toString());
    }
   }

   private void sendData(User user){
     try {
        ByteArrayOutputStream baos = new ByteArrayOutputStream();
        ObjectOutputStream oos = new ObjectOutputStream(baos);
        oos.writeObject(user);
        oos.flush();

        InetAddress IPAddress = InetAddress.getByName(serverHost);
        byte[] sendData = baos.toByteArray();
        DatagramPacket sendPacket = new DatagramPacket(sendData,
                        sendData.length, IPAddress, serverPort);
        myClient.send(sendPacket);

    } catch (Exception ex) {
        view.showMessage(ex.getStackTrace().toString());
    }
   }

   private String receiveData(){
         String result = "";
         try {
                byte[] receiveData = new byte[1024];
                DatagramPacket receivePacket =
                        new DatagramPacket(receiveData, receiveData.length);
                myClient.receive(receivePacket);

                ByteArrayInputStream bais =
```

```
                        new ByteArrayInputStream(receiveData);
                    ObjectInputStream ois = new ObjectInputStream(bais);
                    result = (String)ois.readObject();
          } catch (Exception ex) {
              view.showMessage(ex.getStackTrace().toString());
          }
               return result;
        }
}
```

ClientRun.java

```
package udp.client;

public class ClientRun {
	public static void main(String[] args) {
		ClientView view = new ClientView();
		ClientControl control = new ClientControl(view);
		view.setVisible(true);
	}
}
```

5.3.2 Các lớp phía server

ServerView.java

```
package udp.server;

public class ServerView {
	public ServerView(){
	}

	public void showMessage(String msg){
		System.out.println(msg);
	}
}
```

ServerControl.java

```
package udp.server;
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
import udp.client.User;


public class ServerControl {
```

```
private ServerView view;
private Connection con;
private DatagramSocket myServer;
private int serverPort = 5555;
private DatagramPacket receivePacket = null;

public ServerControl(ServerView view){
	this.view = view;
	getDBConnection("usermanagement", "root", "12345678");
	openServer(serverPort);
	view.showMessage("UDP server is running...");

	while(true){
		listenning();
	}
}

private void getDBConnection(String dbName,
                                String username, String password){
	String dbUrl = "jdbc:mysql://localhost:3306/" + dbName;
    String dbClass = "com.mysql.jdbc.Driver";

    try {
	Class.forName(dbClass);
	con = DriverManager.getConnection (dbUrl, username, password);
    }catch(Exception e) {
	view.showMessage(e.getStackTrace().toString());
    }
}

private void openServer(int portNumber){
	try {
		myServer = new DatagramSocket(portNumber);
	}catch(IOException e) {
		view.showMessage(e.toString());
	}
}

private void listenning(){
	User user = receiveData();

	String result = "false";
	if(checkUser(user)){
		result = "ok";
	}

	sendData(result);
}

private void sendData(String result){
  try {
      ByteArrayOutputStream baos = new ByteArrayOutputStream();
      ObjectOutputStream oos = new ObjectOutputStream(baos);
      oos.writeObject(result);
      oos.flush();

      InetAddress IPAddress = receivePacket.getAddress();
```

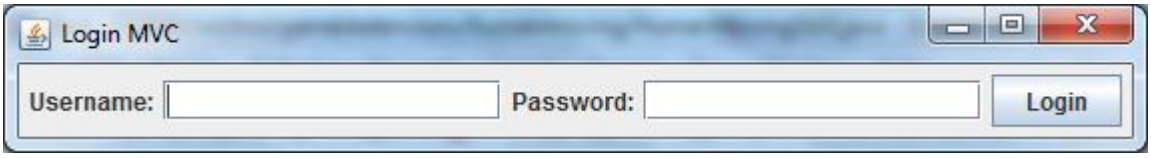
```
            int clientPort = receivePacket.getPort();
            byte[] sendData = baos.toByteArray();
            DatagramPacket sendPacket = new DatagramPacket(sendData,
                    sendData.length, IPAddress, clientPort);
            myServer.send(sendPacket);

          } catch (Exception ex) {
              view.showMessage(ex.getStackTrace().toString());
          }
        }

        private User receiveData(){
               User user = null;
               try {
                      byte[] receiveData = new byte[1024];
                      receivePacket =
                             new DatagramPacket(receiveData, receiveData.length);
                      myServer.receive(receivePacket);

                      ByteArrayInputStream bais =
                             new ByteArrayInputStream(receiveData);
                      ObjectInputStream ois = new ObjectInputStream(bais);
                      user = (User)ois.readObject();

          } catch (Exception ex) {
              view.showMessage(ex.getStackTrace().toString());
          }
               return user;
        }


        private boolean checkUser(User user) {
            String query = "Select * FROM users WHERE username ='"
                    + user.getUserName()
                    + "' AND password ='" + user.getPassword() + "'";

            try {
              Statement stmt = con.createStatement();
              ResultSet rs = stmt.executeQuery(query);
              if (rs.next()) {
                    return true;
              }
            }catch(Exception e) {
              view.showMessage(e.getStackTrace().toString());
            }
            return false;
          }
}
```

ServerRun.java

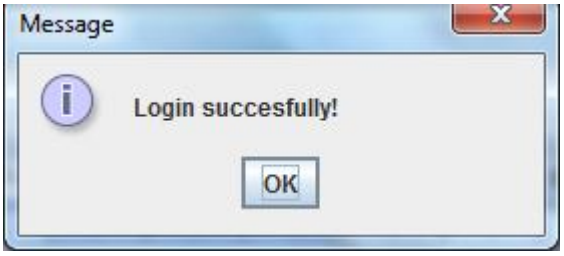
```
package udp.server;

public class ServerRun {
        public static void main(String[] args) {
               ServerView view       = new ServerView();
               ServerControl control = new ServerControl(view);
```

```
        }
}
```

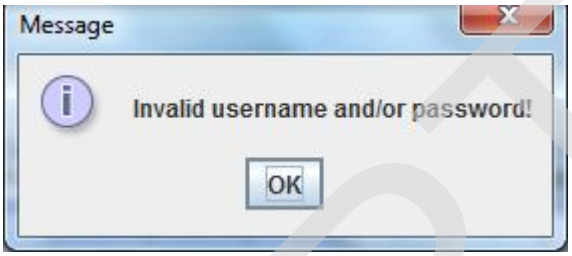
5.5 Kết quả



Login thành công:



Login lỗi:



VI. LẬP TRÌNH TRUYỀN THÔNG MULTICAST

1. Giới thiệu truyền thông multicast và lớp MulticastSocket

Trong truyền thông multicast cho phép truyền gói tin tới một nhóm client nhờ sử dụng địa chỉ multicast của lớp D từ địa chỉ 224.0.0.0 đến 239.255.255.255. Truyền thông multicast có nhiều ứng dụng trong thực tế như:

- Videoconferencing
- Usenet news
- Computer configuration

Các địa chỉ multicast:

- 224.0.0.1: Tất cả các hệ thống ở trên mạng con cục bộ
- 224.0.0.2 : Tất cả các router tren mạng con cục bộ.
- 224.0.0.11: Các tác tử di động(agent) trên mạng con cục bộ
- 224.0.1.1 : Giao thức định thời mạng
- 224.0.1.20: Thử nghiệm mà không cho vượt ra khỏi mạng con cục bộ

- 224.2.X.X (Multicast Backbone on the Internet (MBONE)): Được sử dụng cho audio và video quảng bá trên mạng Internet .

Java hỗ trợ lớp MulticastSocket cho phép tạo ra socket thực hiện truyền thông kiểu này. Lớp MulticastSocket được kế thừ từ lớp DatagramSocket

*public class **MulticastSocket** extends DatagramSocket*

MuticastSocket là một DatagramSocket mà thêm khả năng ghép nối gộp nhóm các máy trạm multicast trên mạng Internet. Một nhóm multicast được chỉ ra bởi địa chỉ lớp D và một địa chỉ cổng UDP chuẩn. Lớp MulticastSocket được sử dụng phía bên nhận. Các cấu tử và phương thức của lớp MulticastSocket được trình bày tóm tắt trong bảng sau:

Cấu tử lớp MulticastSocket
MulticastSocket() Tạo socket muticast
***MulticastSocket**(int port)* Tạo socket muticast và gắn với socket đó một địa chỉ cổng cụ thể.
***MulticastSocket**(SocketAddress bindaddr)* Tạo socket muticast và gắn với socket đó một địa chỉ socket cụ thể.

Các phương thức của lớp MulticastSocket	
InetAddress	***getInterface()*** Lấy địa chỉ giao tiếp mạng được sử dụng cho các gói tin multicast
boolean	***getLoopbackMode()*** Lấy chuỗi thiết đặt đối local loopback của gói tin multicast
NetworkInterface	***getNetworkInterface()*** Lấy tập giao tiếp mạng multicast
int	***getTimeToLive()*** Lấy tham số time to live mặc định của các gói tin multicast gửi ra socket
byte	***getTTL()*** Lấy tham số time- to -live
void	***joinGroup**(InetAddress mcastaddr)* Ghép nhóm multicast
void	***joinGroup**(SocketAddress mcastaddr, NetworkInterface netIf)* Ghép nhóm multicast cụ thể tại giao tiếp mạng cụ thể
void	***leaveGroup**(InetAddress mcastaddr)* Loại bỏ một nhóm multicast
void	***leaveGroup**(SocketAddress mcastaddr, NetworkInterface netIf)* Loại bỏ một nhóm multicast trên giao tiếp mạng cục bộ được chỉ ra.

void	***send**(DatagramPacket p, byte ttl)* Gửi gói tin
void	***setInterface**(InetAddress inf)* Đặt giao tiếp mạng multicast được sử dụng bởi phương thức mà hành vi của nó bị ảnh hưởng bởi giá trị của gao tiếp mạng.
void	***setLoopbackMode**(boolean disiao tiếp mạng**able**)* Cho phép hoặc làm mất hiệu lực vòng phản hồi cục bộ của lược đồ dữ liệu multicast
void	***setNetworkInterface**(NetworkInterface netIf)* Chỉ ra giao tiếp mạng để gửi các lược đồ dữ liệu multicast qua
void	***setTimeToLive**(int ttl)* Thiết đặt tham số TTL mặc định cho các gói tin multicast gửi trên MulticastSocket nhằm mục đích điều khiển phạm vi multicast.
void	***setTTL**(byte ttl)* Thiết đặt tham số TTL

Để tạo ra kết nối một nhóm multicast, đầu tiên phải tạo ra đối tượng MulticastSocket với một địa chỉ cổng xác định bằng cách gọi phương thức jointGroup() của lớp MulticastSocket. Ví dụ:

```
// Kết nối một nhóm multicast và gửi lời chào tới nhóm ...
 String msg = "Hello";
 InetAddress group = InetAddress.getByName("228.5.6.7");
 MulticastSocket s = new MulticastSocket(6789);
 s.joinGroup(group);
 DatagramPacket hi = new DatagramPacket(msg.getBytes(), msg.length(),
                         group, 6789);
 s.send(hi);
 // Nhận đáp ứng của chúng
 byte[] buf = new byte[1000];
 DatagramPacket recv = new DatagramPacket(buf, buf.length);
 s.receive(recv);
 ...
 // OK, I'm done talking - leave the group...
 s.leaveGroup(group);
```

Khi gửi thông điệp tới group, tất cả các máy trạm phía nhận là các thành viên của nhóm sẽ nhận được gói tin, để loại bỏ nhóm, phương thức leaveGroup() sẽ được gọi.

2. Một số ví dụ gửi/nhận dữ liệu multicast

2.1. Ví dụ gửi dữ liệu multicast

```
import java.net.*;
// Which port should we send to
int port = 5000;
// Which address
String group = "225.4.5.6";
// Which ttl
```

```
int ttl = 1;
// Create the socket but we don't bind it as we are only going to send data
MulticastSocket s = new MulticastSocket();
// Note that we don't have to join the multicast group if we are only
// sending data and not receiving
// Fill the buffer with some data
byte buf[] = byte[10];
for (int i=0; i<buf.length; i++) buf[i] = (byte)i;
// Create a DatagramPacket
DatagramPacket pack = new DatagramPacket(buf, buf.length,
                                    InetAddress.getByName(group), port);
// Do a send. Note that send takes a byte for the ttl and not an int.
s.send(pack,(byte)ttl);
// And when we have finished sending data close the socket
s.close();
```

2.2. Ví dụ nhận dữ liệu multicast

```
import java.net.*;
// Which port should we listen to
int port = 5000;
// Which address
String group = "225.4.5.6";
// Create the socket and bind it to port 'port'.
MulticastSocket s = new MulticastSocket(port);
// join the multicast group
s.joinGroup(InetAddress.getByName(group));
// Now the socket is set up and we are ready to receive packets
// Create a DatagramPacket and do a receive
byte buf[] = byte[1024];
DatagramPacket pack = new DatagramPacket(buf, buf.length);
s.receive(pack);
// Finally, let us do something useful with the data we just received,
// like print it on stdout :-)
System.out.println("Received data from: " + pack.getAddress().toString() +
                ":" + pack.getPort() + " with length: " +
                pack.getLength());
System.out.write(pack.getData(),0,pack.getLength());
System.out.println();
// And when we have finished receiving data leave the multicast group and
// close the socket
s.leaveGroup(InetAddress.getByName(group));
s.close();
```

2.3. Một số ví dụ khác

```
//MulticastJoin.java
 import java.net.*;
 import java.io.*;
 public class MulticastJoin {
 public static void main(String [ ] args){
 try {
 MulticastSocket mSocket = new MulticastSocket(4001);
```

```
InetAddress mAddr = InetAddress.getByName("224.0.0.1");
mSocket.joinGroup(mAddr);
byte [ ] buffer = new byte[512];
while (true) {
DatagramPacket dp = new DatagramPacket(buffer,
buffer.length);
mSocket.receive(dp);
String str = new String(dp.getData(), "8859_1");
System.out.println(str);
}//end of while
}//end of try
catch (SocketException se){
System.out.println("Socket Exception : " + se); }
catch (IOException e) { System.out.println("Exception : " + e); }
}//end of main
}// end of class definition
```

//MulticastListener.java

```
import java.net.*;
import java.io.*;
public class MulticastListener {
public static void main( String [ ] args) {
InetAddress mAddr=null;
MulticastSocket mSocket=null;
final int PORT_NUM= 4001;
try {
mAddr = InetAddress.getByName("audionews.mcast.net");
mSocket = new MulticastSocket(PORT_NUM);
String hostname = InetAddress.getLocalHost().getHostName();
byte [ ] buffer = new byte[8192];
mSocket.joinGroup(mAddr);
System.out.println("Listening from " + hostname + " at " +
mAddr.getHostName());
while (true){
DatagramPacket dp = new DatagramPacket(buffer, buffer.length);
mSocket.receive(dp);
String str = new String(dp.getData(), "8859_1");
System.out.println(str);
}//end of while
}
catch (SocketException se) {
System.out.println("Socket Exception : " + se);
}
catch (IOException e) {
System.out.println("Exception : " + e);
}
finally {
if (mSocket != null){
try {
mSocket.leaveGroup(mAddr);
```

```
 mSocket.close();
 }
 catch (IOException e){ }
 }//end of if
 }//end of finally
 }//end of main
}
```

VII. KẾT LUẬN

Trong chương này chúng ta đã nghiên cứu các kỹ thuật lập trình mạng cơ bản sử dụng socket: TCP Socket, UDP Socket. Sau đó chúng ta đã nghiên cứu cách lập trình với địa chỉ mạng, với giao tiếp mạng và kỹ thuật lập trình truyền thông multicacst. Trong chương tiếp theo chúng ta sẽ mở rộng kiến thức trong chương này để phát triển các chương trình server phục vụ đồng thời nhiều chương trình máy khác cũng như tuần tự.

CHƯƠNG III

KỸ THUẬT XÂY DỰNG ỨNG DỤNG MẠNG PHÍA SERVER

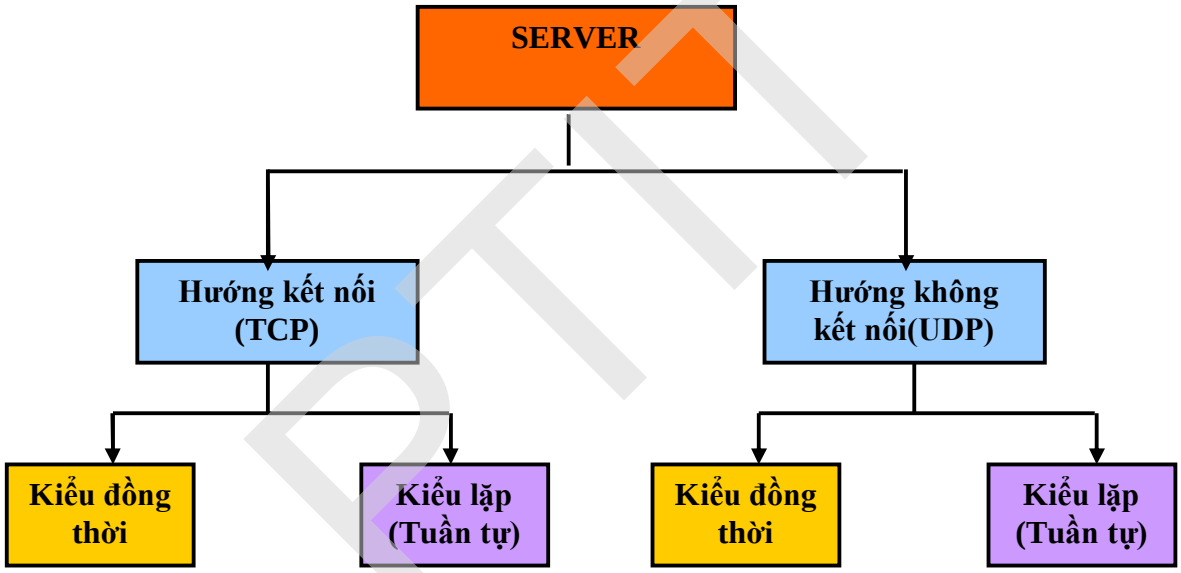
I. GIỚI THIỆU VỀ CÁC KIỂU SERVER

Trong mô hình client/server, chương trình server đóng vai trò phục vụ yêu cầu gửi tới từ chương trình client. Chương trình server có thể phục vụ một hoặc nhiều client đồng thời hoặc phục vụ kiểu lặp.

Server có thể phân thành các loại sau:

- Server chạy chế độ đồng thời hướng không kết nối(TCP)
- Server chạy chế độ lặp hướng không kết nối(TCP)
- Server chạy chế độ đồng thời hướng không kết nối(UDP)
- Server chạy chế độ lặp hướng không kết nối(UDP)

và sự phân loại này được thể hiện như hình 3.



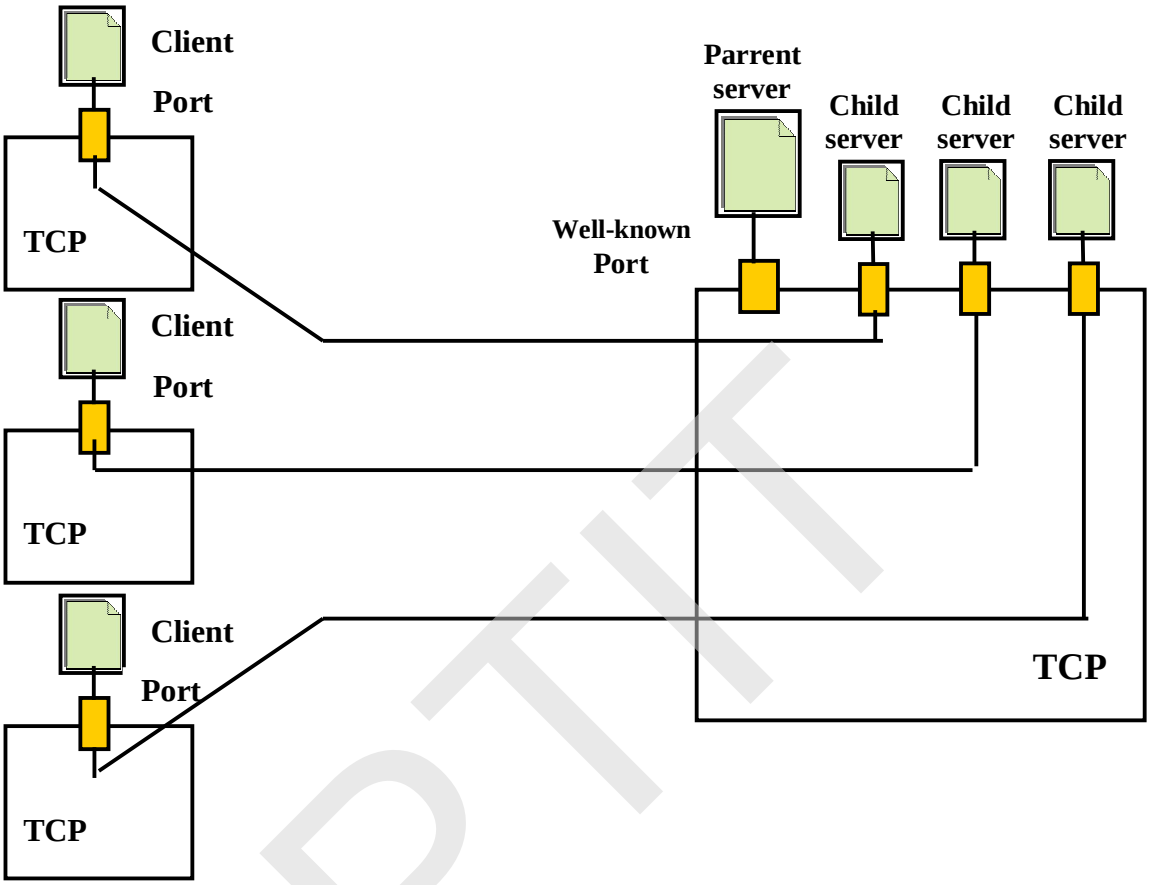
Hình 3.1. Các kiểu server

Trong các kiểu server này, kiểu server đồng thời hướng kết nối và server kiểu lặp hướng không kết nối được sử dụng phổ biến. Chính vì vậy chúng ta chỉ tập trung vào xét 2 kiểu server này.

1. Server chạy chế độ đồng thời hướng kết nối

Đây là loại server chuẩn, sử dụng giao thức truyền thông TCP. Server này có thể phục vụ nhiều client đồng thời. Kết nối được thiết lập giữa server với mỗi client và kết nối được duy trì hoạt động cho đến khi toàn bộ luồng được xử lý, cuối cùng kết nối được kết thúc. Server hướng kết nối đồng thời không thể chỉ sử dụng một cổng đã biết bởi mỗi kết nối cần một địa chỉ cổng và có nhiều kết nối sẽ được thiết lập tại cùng thời điểm. Chính vì vậy server phải sử dụng nhiều cổng và nó chỉ sử dụng một cổng biết rõ trước. Khi khởi tạo, server sẽ thực hiện mở thụ động tại cổng biết rõ đó và đặt ở trạng thái nghe tín hiệu đến kết nối từ client. Mỗi khi có một client thiết lập kết nối với server qua cổng đó, server sẽ sinh ra các server con với một số cổng khác để phục vụ

client đó. Còn server chính sẽ tiếp tục đặt ở trạng thái nghe tín hiệu kết nối khác.Server cũng có thể sử dụng bộ đệm cho mỗi kết nối. Các segment truyền từ client tới sẽ được cất vào bộ đệm phù hợp và sẽ được phục vụ đồng thời bởi server. Mô hình hoạt động của server này được thể hiện như hình 3.2.



Hình 3.2. Mô hình server phục vụ đồng thời hướng kết nối

2. Server chạy chế độ lặp hướng không kết nối

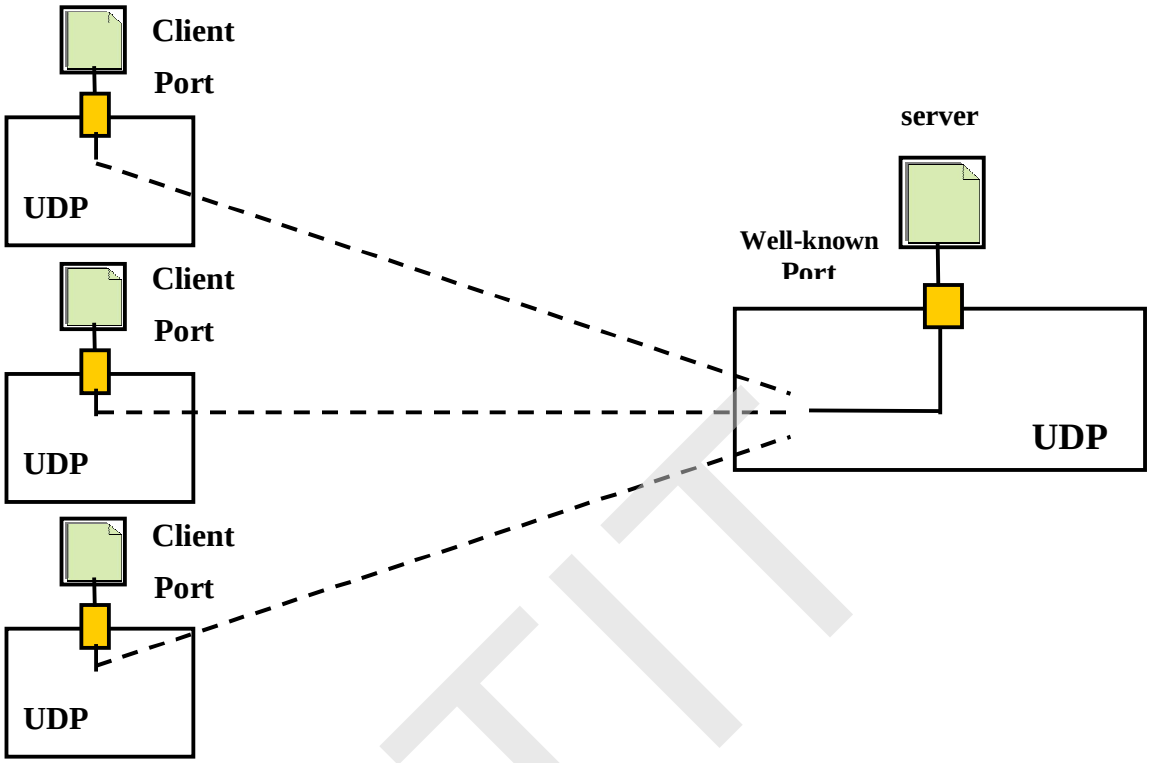
Server kiểu lặp hướng không kết nối thường sử dụng giao thức UDP. Trong kiểu server này, tại mỗi thời điểm nó chỉ xử lý một yêu cầu. Server lấy yêu cầu từ UDP, xử lý yêu cầu và trả đáp ứng về cho UDP để gửi về client. Khi các client gửi gói tin đến sẽ được chứa trong hàng đợi để chờ phục vụ. Các gói tin này có thể đi tới từ một client hoặc nhiều client và server sẽ thực hiện sử lý tuần tự từng yêu cầu theo trật tự trong hàng đợi. Mô hình hoạt động của server này được thể hiện như hình 3.3.

II. XÂY DỰNG CHƯƠNG TRÌNH SERVER PHỤC VỤ NHIỀU CLIENT HƯỚNG KẾT NỐI

1. Giới thiệu

Để cài đặt chương trình server TCP phục vụ nhiều client đồng thời, trong lập trình mạng có 2 kỹ thuật phổ biến:

- Xây dựng chương trình đa tiến trình: Trong chương trình này tiến trình cha sẽ sinh ra tiến trình con mỗi khi có một client gửi yêu cầu tới server. Nhược điểm của kỹ thuật lập trình này là không tận dụng hiệu quả CPU. Vì khi chương trình chạy, nó phải sử dụng cơ chế ngắt để chuyển từ tiến trình này sang tiến trình khác, nên CPU sẽ rảnh rỗi trong quá trình này.



Hình 3.3. Mô hình server kiểu lặp hướng không kết nối

trong quá trình chuyển tiến trình đó. Kỹ thuật này được sử dụng phổ biến trong các ngôn ngữ lập trình C/C^{++}(Linux, Unix), VC^{++}...

- Xây dựng chương trình đa luồng(đa tiểu trình): Chương trình server kiểu này sẽ sinh ra một luồng mới mỗi khi có một client gửi yêu cầu tới đòi phục vụ. Kiểu chương trình này khi chạy tận dụng hiệu quả CPU vì chương trình không có việc chuyển từ tiến trình này sang tiến trình khác. Kỹ thuật lập trình này được các ngôn ngữ lập trình phổ biến hiện nay hỗ trợ mạnh mẽ như VC^{++}, Java, .NET...Sau đây chúng ta sẽ lướt qua kỹ thuật lập trình đa luồng trong java và sử dụng để xây dựng chương trình server đáp ứng nhiều kết nối đồng thời với giao thức truyền thông TCP.

2. Kỹ thuật lập trình đa luồng trong Java(MultiThread)

Một luồng là một thuộc tính duy nhất của Java. Nó là đơn vị nhỏ nhất của đoạn mã có thể thi hành được mà thực hiện một công việc riêng biệt. Ngôn ngữ Java và máy ảo Java cả hai là các hệ thống đươc phân luồng. Java hỗ trợ đa luồng, mà có khả năng làm việc với nhiều luồng. Một ứng dụng có thể bao hàm nhiều luồng. Mỗi luồng được đăng ký một công việc riêng biệt, mà chúng được thực thi đồng thời với các luồng khác.

Đa luồng giữ thời gian nhàn rỗi của hệ thống thành nhỏ nhất. Điều này cho phép bạn viết các chương trình có hiệu quả cao với sự tận dụng CPU là tối đa. Mỗi phần của chương trình được

gọi một luồng, mỗi luồng định nghĩa một đường dẫn khác nhau của sự thực hiện. Đây là một thiết kế chuyên dùng của sự đa nhiệm.

Trong sự đa nhiệm, nhiều chương trình chạy đồng thời, mỗi chương trình có ít nhất một luồng trong nó(luồng chính). Một vi xử lý thực thi tất cả các chương trình. Cho dù nó có thể xuất hiện mà các chương trình đã được thực thi đồng thời, trên thực tế bộ vi xử lý nhảy qua lại giữa các luồng.

Cấu trúc của một chương trình đa luồng gồm một luồng chính(main) và các luồng con. Luồng chính được khởi tạo ngay khi chương trình chạy và nó có đặc điểm:

- Là luồng sinh ra các luồng con
- Là luồng kết thúc sau cùng.

Mỗi luồng trong chương trình Java được đăng ký cho một quyền ưu tiên. Máy ảo Java không bao giờ thay đổi quyền ưu tiên của luồng. Quyền ưu tiên vẫn còn là hằng số cho đến khi luồng bị ngắt.

Mỗi luồng có một giá trị ưu tiên nằm trong khoảng của Thread.MIN_PRIORITY(Giá trị 1), và Thread.MAX_PRIORITY(giá trị 10). Mỗi luồng phụ thuộc vào một nhóm luồng, và mỗi nhóm luồng có quyền ưu tiên của chính nó. Mỗi luồng mặc định có mức độ ưu tiên bằng 5. Mỗi luồng mới thừa kế quyền ưu tiên của luồng mà tạo ra nó.

Để hỗ trợ lập trình đa luồng, java có giao diện Runnable và lớp Thread, ThreadGroup thuộc gói *java.lang(*gói mặc định)). Các phương thức của lớp Thread như bảng sau:

Phương thức	Mô tả
Enumerate(Thread t)	Sao chép tất cả các luồng hiện hành vào mảng được chỉ định từ nhóm của các luồng, và các nhóm con của nó.
getName()	Trả về tên của luồng
isAlive()	Kiểm tra một luồng có còn tồn tại (sống)
getPriority()	Trả về quyền ưu tiên của luồng
setName(String name)	Đặt tên của luồng là tên mà luồng được truyền như là một tham số
join()	Đợi cho đến khi luồng kết thúc
resume()	Chạy lại một luồng
sleep()	Tạm dừng một luồng sau khoảng thời gian nào đó
start()	Khởi tạo một luồng, thực chất là gọi thi hành phương thức run
run()	Điểm vào của một luồng(tương tự phương thức main())

Mỗi luồng con trong chương trình java có điểm vào là phương thức run() là phương thức của giao diện Runnable hoặc lớp Thread.

```
public void run()
{
//Khối lệnh của luồng
}
```

Chu kỳ sống của luồng được thể hiện như hình 3.4.

Để tạo một luồng mới:

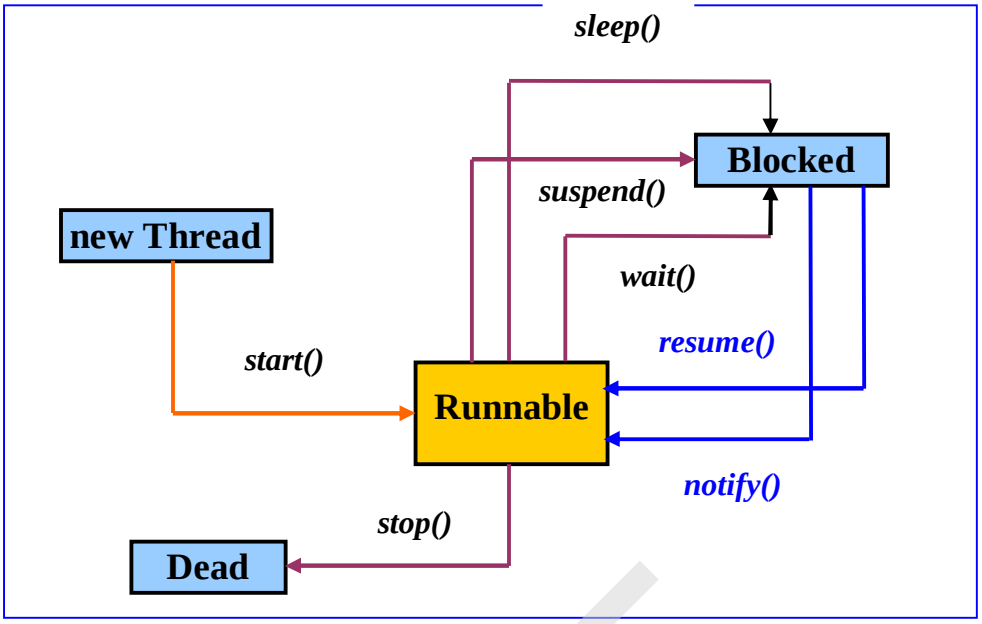
Để tạo luồng mới có 2 cách khai báo:

- Cách 1: Khai báo một lớp kế thừa lớp Thread, từ đó cài đặt mã lệnh thực thi của luồng vào phương thức run() bằng cách khai báo nạp chồng phương thức run.

Ví dụ:

Viết chương trình sinh ra 10 luồng, mỗi luồng in ra số thứ tự của luồng.

```
//TestThread.java
class NewThread   extends  Thread
{
private  int  count;
//Khai bao cau tu
NewThread(int count)
{
      super();
      this.count=count;
      start();
}
public void run()
{
System.out.println("Luong thu:"+count);
}}
class TestThread{
public static void main(String[] args)
{
int  i=0;
while(i<10)
{
new  NewThread(i);
i++;
}
}}
```



Hình 3.4. Chu kỳ sống của luồng(thread)

- Cách 2: Khai báo lớp thực thi giao diện Runnable. Lớp này cho phép tạo ra đối tượng Thread và cài đặt phần thân cho phương thức run() của giao diện. Ví dụ viết lại chương trình trên, chương trình chỉ khác phần khai báo lớp NewThread.

```
//TestThread.java
class NewThread   implêmnts  Runnable
{
private  int  count;
//Khai bao cau tu
NewThread(int count)
{
      Thread  t=new  Thread();
      this.count=count;
public void start()
{  run(); }
public void run()
{
System.out.println("Luong thu:"+count);
}}
```

Một vấn đề quan trọng khác là vấn đề đồng bộ. Để giải quyết vấn đề này Java sử dụng một cơ chế đặc biệt gọi là Monitor.

3. Xây dựng chương trình server phục vụ nhiều client đồng thời hướng kết nối

Để minh hoạ kỹ thuật này, chúng ta tiến hành xây dựng một chương trình ví dụ:

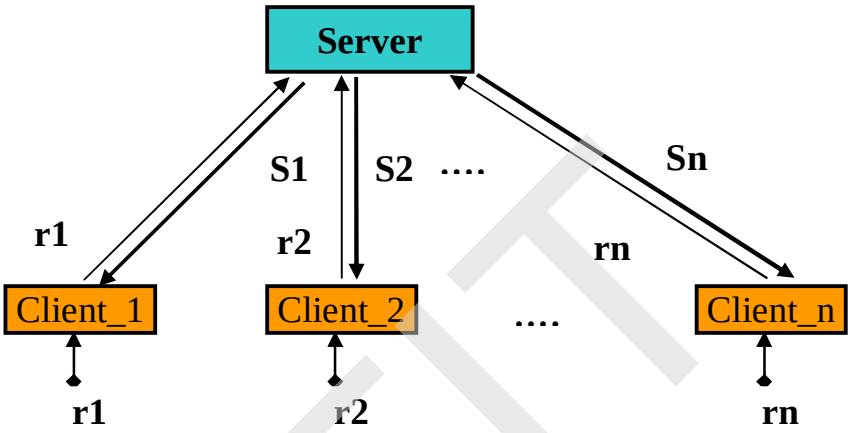
Hãy viết chương trình server phục vụ nhiều client đồng thời sử dụng giao thức truyền thông TCP. Chương trình cho phép nhận bán kính đường tròn gửi đến từ các client, tính diện tích hình tròn,

hiển thị tên, địa chỉ IP, số cổng, bán kính r, diện tích của client tương ứng. Sau đó trả kết quả về cho client.

a) Chạy chương trình sử dụng trình telnet

b) Viết chương trình client.

Để viết chương trình này chúng ta sẽ sử dụng kỹ thuật đa luồng trong Java. Mỗi khi một chương trình client gửi yêu cầu kết nối đến, server sẽ sinh ra một luồng mới để phục vụ kết nối đó. Sau khi phục vụ xong kết nối nào thì luồng đó được giải phóng. Mô hình xây dựng chương trình thể hiện như hình 3.5.



Hình 3.5. Mô hình client/server của bài toán

3.1. Chương trình client

Chương trình client thực hiện các công việc sau:

- Gửi kết nối tới server
- Nhập bán kính r từ bàn phím
- Gửi bán kính tới server
- Nhận kết quả trả về và hiển thị
- Kết thúc chương trình

```
//areaClient.java
import  java.io.*;
import  java.net.*;
class areaClient{
public static void main(String[] args)
{
//Khai bao bien
Socket  cl=null;
BufferedReader  inp=null;//luong nhap
PrintWriter     outp=null.//luong xuat
BufferedReader  key=null;//luong nhap tu ban phim
String ipserver= "127.0.0.1";//Chuoi dia chi server
int     portserver=3456; //dia chi cong server
```

```
String   r; //ban kinh r la chuoi so
//Tao socket va ket noi toi server
try{
cl=new Socket(ipserver,portserver);
//tao luong nha/xuatp kieu ky tu cho socket
inp=new BufferedReader(new InputStreamReader(cl.getInputStream()));
outp=new PrintWriter(cl,getOutputStream(),true);
//tao luong nhap tu ban phim
key=new BufferedReader(new InputStreamReader(System.in));
//Nhap ban kinh r tu ban phim
System.out.print("r=");
r=key.readLine().trim();
//gui r toi server
outp.println(r);
//Nhan dien tich tra ve tu server va hien thi
System.out.println("Area:"+inp.readLine());
//ket thuc chuong trinh
if(inp!=null)
inp.close();
if(key!=null)
key.close();
if(outp!=null)
outp.close();
if(cl!=null)
cl.close();
}
cacth(IOException  e)
{
System.out.println(e);
}
}
}
```

3.2. Chương trình server

Chương trình server phục vụ nhiều client thực hiện các công việc sau:

- Khởi tạo đối tượng ServerSocket và nghe tại số cổng 3456.
- Thực hiện lặp lại các công việc sau:
  - ❖ Nhận kết nối mới, tạo socket mới
  - ❖ Phát sinh một luồng mới và nhận socket
  - ❖ Nhận bán kính gửi tới từ client
  - ❖ Tính diện tích
  - ❖ Hiển thị số thứ tự luồng, tên, địa chỉ IP, số cổng, bán kính r, diện tích của client
  - ❖ Gửi diện tích về cho client
  - ❖ Kết thúc luồng

```
//AreaThreadServer.java
import  java.io.*;
import  java.net.*;
//Khai báo lớp NewThread cho phép tạo ra luồng mới
```

```
class NewThread   extends     Thread
{
   private   int count;
   private  Socket  cl=null;
   private BufferedReader  inp=null;//luong nhap
   private  PrintWriter     outp=null;//luong xuat
   NewThread(Socket   cl, int  count)
   {
   super();//Truy xuất cấu tử lớp Thread
   this.cl=cl;
   this.count=count;
   start();
   }
 //cai dat phuong thuc run-Luong moi
 public  void   run()
 {
 try{
 //tao luong nhap /xuat cho socket cl
 inp=new  BufferedReader(new  InputStreamReader(cl.getInputStream()));
 outp=new PrintWriter(cl.getOutputStream(),true);
 //Doc ban kinh gui toi tu client
 double  r=Double.parseDouble(inp.readLine().trim());
 // lay dia chi client
 InetAddress  addrclient=cl.getInetAddress();
 //lay so cong phia client
 int   portclient=cl.getPort();
 //Tinh dien tich
 double  area=3.14*r*r;
 //Hien thi
 System.out.println("Luong                                        thu:"+count+",
 client:"+addrclient.getHostName()+
               ", ip:"+addrclient.getHostAddress()+",port:"+portclient+
               ", r="+r+",area:"+area);
 //Gui dien tich ve cho client tuong ung
 outp.println(area);
 //ket thuc luong
 inp.close();
 outp.close();
 cl.close();
 }
 catch(IOException e)
 {
 System.out.println(e);
 }
 }
 }
 //Chuong trinh server
 class   AreaThreadServer{
 public static void main(String[] args)
 {
```

```
//Khai bao bien
int   count;
ServerSocket  svr=null;
Socket   cl=null;
int portserver=3456;
try{
svr=new ServerSocket(portserver);
count=0;
while(true){
cl=svr.accept();
new  NewThread(cl, count);
count++;
}
}
catch(IOException   e)
{
System.out.println(e);
}
}
}
```

3.3. Dịch và chạy chương trình

Dịch chương trình:

Mở cửa sổ lệnh và đến thư mục chứa chương trình client và server, thực hiện biên dịch chương trình:

javac areaClient.java [Enter]

javac AreaThreadServer.java [Enter]

Chạy chương trình:

❖ ***Chạy chương trình với trình telnet:***

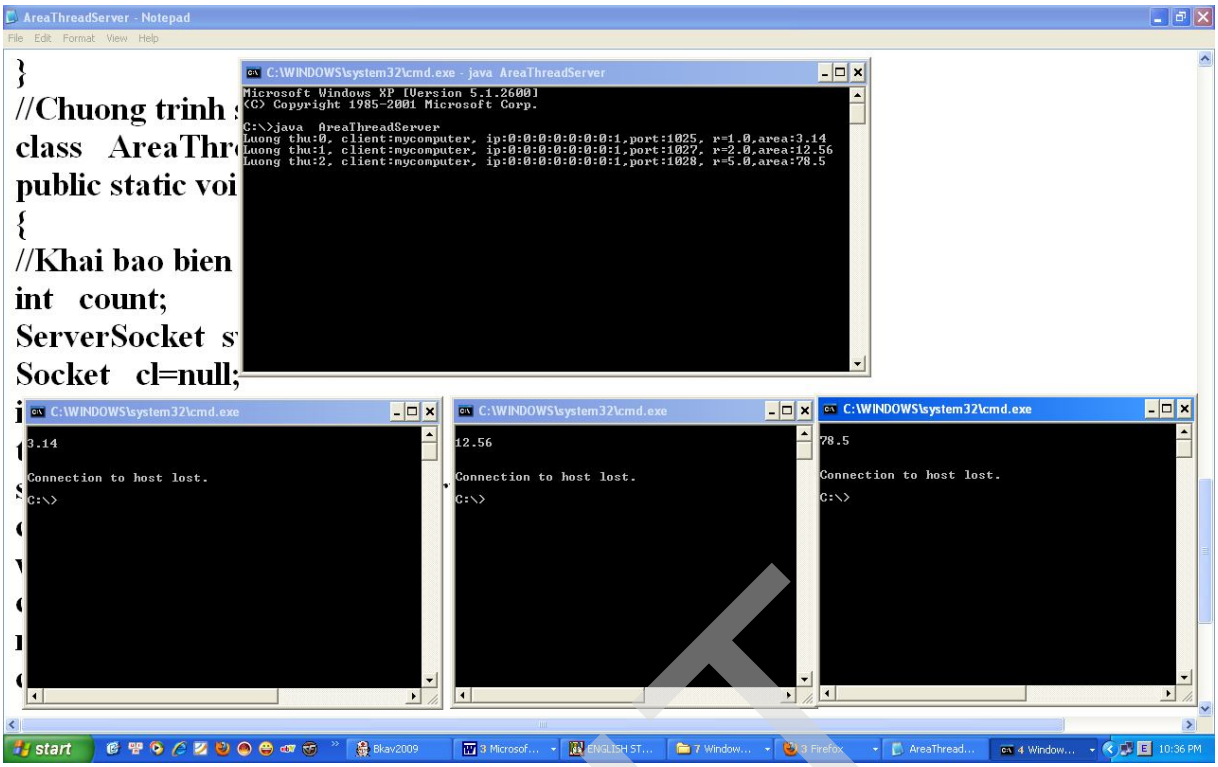
- Mở 1 cửa sổ lệnh, chạy chương trình server:

java AreaThreadServer [Enter]

- Giả sử mở 3 cửa sổ, mỗi cửa sổ là chạy một chương trình client sử dụng trình telnet được chạy với cú pháp sau:

telnet localhost 3456 [Enter]

Kết quả chạy chương trình thể hiện như cửa sổ hình 3.6.



Hình 3.6. Kết quả chạy chương trình với trình telnet

❖ Chạy chương trình với chương trình client:

Thay vì chạy trình telnet, sử dụng chương trình client areaClient. Chương trình chạy trong các cửa sổ với cú phát sau:

java areaClient [Enter]

❖ Chạy chương trình trên mạng cục bộ:

Bước 1: Sửa lại chương trình client trong cấu lệnh new Socket(....,....) với địa chỉ ipserver là địa chỉ của máy trạm trên đó chạy chương trình server. Sau đó dịch lại chương trình.

Bước 2: Copy chương trình server tới máy có địa chỉ dùng để sửa ở bước 1 và chạy chương trình.

Bước 3: Copy chương trình client đã dịch ở bước 1 tới các máy tính khác trên mạng và thực hiện chạy chương trình client đó.

Bước 4: Nhập giá trị bán kính r từ của sổ client, quan sát kết quả chạy chương trình trên client và server.

III. KẾT LUẬN

Trong chương 3 này chúng ta đã khảo sát các kiểu chương trình server, khảo sát kỹ thuật lập trình đa luồng và ứng dụng nó vào xây dựng chương trình server phục vụ nhiều client đồng thời. Cuối cùng chúng ta đã xây dựng một chương trình ví dụ đơn giản để minh hoạ kỹ thuật xây dựng server. Từ chương trình ví dụ, sinh viên có thể sửa chương trình để ứng dụng nhiều bài toán thực tế như bài toán tra cứu tuyển sinh, bài toán nhập dữ liệu từ xa, bài toán tra cứu thời tiết ... mà có

kết nối với các cơ sở dữ liệu như Access, SQL hoặc Oracle. Các kỹ thuật lập trình mạng này sẽ được củng cố hơn ở các chương tiếp theo.

CHƯƠNG IV
LẬP TRÌNH VỚI GIAO THỨC DỊCH VỤ MẠNG PHÍA CLIENT

I. GIỚI THIỆU

Chương này sẽ hướng sinh viên sử dụng kỹ thuật lập trình socket đã được trang bị trong các chương trước để lập trình với một số giao thức dịch vụ mạng phổ biến trên internet như: DSN, Telnet, FTP, TFTP, SMTP, POP3, IMAP4, HTTP, RTP.

Để lập trình được với các giao thức truyền thông có sẵn, người lập trình phải:

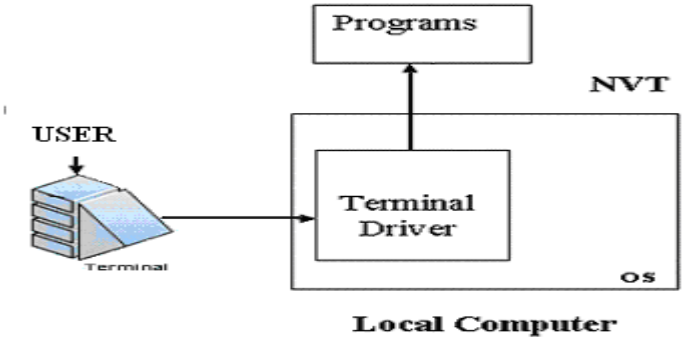
- Khảo sát kỹ đặc điểm, mô hình và cơ chế truyền thông của giao thức;
- Tập lệnh(command), tập đáp ứng(response) và tập tham số của các giao thức;
- Các chế độ hoạt động của giao thức
- Kỹ thuật cài đặt giao thức bằng các ngôn ngữ lập trình

Thông qua đó sinh viên nắm được kỹ thuật lập trình với các giao thức truyền thông có sẵn khác để phát triển các ứng dụng hoặc phát triển các modul tích hợp giải quyết các bài toán thực tế.

II. LẬP TRÌNH GIAO THỨC DỊCH VỤ TELNET

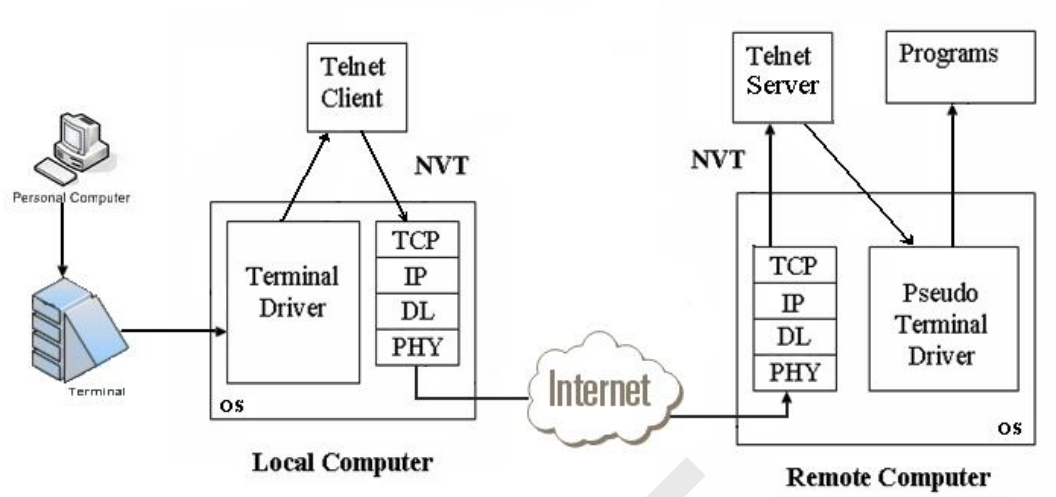
1. Một số khái niệm và đặc điểm của dịch vụ Telnet

- *Đầu cuối*: Trong dịch vụ Telnet, đầu cuối có thể coi là tổ hợp của bàn phím và màn hình. Thiết bị đầu cuối này cho phép người sử dụng nhập dữ liệu gửi tới trung tâm xử lý và nhận kết quả trả về.
- *Môi trường chia sẻ thời gian*: đây thực chất là một mạng các đầu cuối, các đầu cuối được kết nối với nhau thông qua trung tâm xử lý thường là một máy tính mạnh. Trong môi trường chia sẻ thời gian, các ký tự được người sử dụng nhập vào bàn phím đều được chuyển tới trung tâm xử lý. Sau khi xử lý xong kết quả được trả về màn hình người sử dụng.
- *Đầu cuối ảo*: khi một máy tính kết nối qua mạng Internet với máy tính từ xa với vai trò như một đầu cuối cục bộ trên máy tính từ xa đó gọi là đầu cuối ảo. Mạng gồm nhiều đầu cuối ảo được gọi là mạng đầu cuối ảo (Network Virtual Terminal).
- *Đăng nhập*: đây là quá trình người sử dụng mã tài khoản để truy nhập vào hệ thống từ xa. Có hai loại đăng nhập:
  - ❖ Đăng nhập cục bộ: là quá trình đăng nhập vào môi trường chia sẻ thời gian cục bộ.



Hình 4.1. Đăng nhập cục bộ

- Đăng nhập từ xa: máy tính cục bộ phải cài phần mềm Telnet client, máy tính từ xa phải cài phần mềm Telnet server.



Hình 4.2. Đăng nhập từ xa

Quá trình đăng nhập: Khi người sử dụng nhập các ký tự thông qua đầu cuối, ký tự đó sẽ được gửi tới Hệ điều hành của máy tính cục bộ (hệ điều hành không dịch ký tự đó mà nó gửi đến cho chương trình Telnet Client). Chương trình Telnet Client dịch ký tự đó ra dạng tập ký tự chung NVT-ASCII 7 bít và gửi đến các tầng TCP/IP để chuyển qua mạng Internet, tới các tầng TCP/IP của máy tính từ xa. Hệ điều hành gửi các ký tự đó đến chương trình Telnet Server, chương trình này sẽ dịch các ký tự đó ra dạng mà máy tính từ xa có thể hiểu được. Nhưng do hệ điều hành được thiết kế không cho phép gửi ký tự ngược lại hệ điều hành. Để giải quyết vấn đề này, trên máy tính từ xa bổ sung thêm modul phần mềm giả lập đầu cuối (Pseudo Terminal Driver). Từ đó Telnet Server gửi ký tự đó đến cho phần mềm này và chuyển tiếp đến hệ điều hành. Hệ điều hành sẽ gửi các ký tự đó đến chương trình phù hợp.

- Đặc điểm của dịch vụ Telnet:
  - TELNET= TErminaL NETwork
  - Telnet sử dụng kết nối TCP với số cổng mặc định là 23
  - Telnet gồm 2 phần mềm: Telnet client cài trên máy cục bộ, Telnet Server cài trên máy từ xa.
  - Telnet là dịch vụ đăng nhập từ xa. Sau khi đăng nhập thành công, máy cục bộ trở thành đầu cuối ảo của máy từ xa(màn hình , bàn phím... trở thành của máy từ xa). Dịch vụ cho phép truy cập và thao tác với tài nguyên trên máy từ xa.
  - Dịch vụ Telnet hiện đã được tích hợp vào hệ điều hành mạng và được coi như là giao thức chuẩn của TCP/IP.
- Đối với lập trình ứng dụng mạng, bài toán quan trọng nhất là xây dựng chương trình phần mềm phía client. Điều này cho phép người sử dụng có thể tạo ra được phần mềm với giao diện phù hợp và dễ dàng tích hợp với các dịch vụ khác. Để lập trình đượoc dịch vụ Telnet phía người sử dụng,người lập trình phải nắm chắc tập ký tự NVT, các tuỳ chọn và các chính sách

thoả thuận tuỳ chọn của Telnet, các lệnh điều khiển server và cấu trúc lệnh Telnet. Cuối cùng người sử dụng phải nắm được các chế độ hoạt động của Telnet trước khi cài đặt chương trình Telnet.

2. Một số kiến thức giao thức Telnet cơ bản

2.1. Tập ký tự chung NVT

Để tạo ra sự độc lập giữa máy tính cục bộ và máy tính từ xa trong các mạng không đồng nhất, telnet định nghĩa một giao diên chung gọi là tập kí tự mạng đầu cuối ảo NVT (Network Virtual Terminal). NVT gồm 2 tập kí tự:

- Tập ký tự dữ liệu: có bit cao nhất bằng 0 và có mã thuộc [0,127] .
- Tập ký tự điều khiển: có bit cao nhất bằng 1 và có mã thuộc [128,255] .

Name	**Code**	**Decimal Value**	**Function**
NULL	NUL	0	No operation
Line Feed	LF	10	Di chuyển máy in tới hàng in tiếp theo, định vị vị trí nằm ngang.
Carriage			Di chuyển máy in sang bên trái
Return	CR	13	Lề của hàng hiện thời
BELL	BEL	7	Sinh ra một tín hiệu nghe được hoặc rõ ràng (mà không di chuyển đầu in).
Back Space	BS	8	Di chuyển đầu in một ký tự định vị về phía lề trái (trên thiết bị in, mà thiết bị này thông thường được sử dụng tới mẫu văn bản ký tự hoàn chỉnh bằng cách in hai ký tự cơ bản trên phần đầu lẫn nhau).
Horizontal Tab	HT	9	Di chuyển máy in tới Horizontal Tab tiếp theo (Nó giữ nguyên không được chỉ rõ phải làm như thế nào để mỗi nhóm xác định hoặc thiết lập nơi được định vị).
Vertical Tab	VT	11	Tương tự như HT
Form Feed	FF	12	Di chuyển máy in tới phần đầu của trang tiếp theo và giữ vị trí nằm ngang (trên hiển thị trực quan, việc xóa màn hình và di chuyển con trỏ tới góc trái)

Một số kí tự dữ liệu quan trọng

Name	**Decimal Code**	**Meaning**

SE	240	End of subnegotiation parameters: Kết thúc của tham số thỏa thuận
NOP	241	No operation: không thao tác
DM	242	Data mark: Chỉ ra vị trí của sự kiện đồng bộ bên trong luồng dữ liệu. (Cái này luôn phải được kèm theo cảnh báo TCP).
BRK	243	Break: chỉ ra sự thoát
IP	244	Interupt Process: dùng để ngắt tiến trình đang chạy trên máy từ xa.
AO	245	Abort output: cho phép tiến trình hiện thời chạy hoàn thành nhưng không gửi đầu ra của nó cho người sử dụng
AYT	246	Are you there: gửi đến cho server và hỏi xem server còn hoạt động không.
EC	247	Erase character: người nhận nên xóa ký tự trước lần cuối từ luồng dữ liệu.
EL	248	Erase line: xóa ký tự từ luồng dữ liệu nhưng không bao gồm CRLF
GA	249	Go ahead: người dùng, dưới những hoàn cảnh nhất định có thể diễn tả kết thúc khác mà nó có thể truyền.
SB	250	SubOption Begin: chỉ thị bắt đầu một tùy chọn thành phần.
WILL	251	Chỉ ra sự mong muốn bắt đầu được thực hiện hoặc sự xác nhận mà bạn đang thực hiện.
WONT	252	Chỉ ra sự từ chối thực hiện hoặc tiếp tục thực hiện.
DO	253	Chỉ ra yêu cầu mà một nhóm thực hiện khác hoặc xác nhận điều bạn đang mong đợi của nhóm khác thực hiện.
DON'T	254	Chỉ ra sự yêu cầu mà nhóm khác ngừng thực hiện xác nhận điều mà bạn không mong chờ nhóm khác thực hiện.
IAC	255	Interpret as command: Đây là ký tự không dịch lệnh

Một số kí tự điều khiển quan trọng

2.2. Các tùy chọn

Các tùy chọn: được sử dụng để bổ sung thêm thông tin cho các lệnh:

Echo: hiển thị trả lời.

Terminal Type: tuỳ chọn kiểu đầu cuối.

Terminal Speed: thỏa thuận về tốc độ đầu cuối.

Binary : cho phép người nhận dịch mọi kí tự 8 bit như là dữ liệu nhị phân, trừ kí tụ IAC

Echo: cho phép Server phản hồi dữ liệu nhận được trở lại client để hiện lên màn hình

Suppress go head : loại bỏ kí tự CA

Timing: cho phép một thành viên phát sinh dấu hiệu định thời, để chỉ thị rằng tất cả dữ liệu nhận được trước đó đã được xử lý. Mã của các tuỳ chọn được thể hiện trong bảng sau:

Decimal Code	Name	RFC
1	Echo	857
3	Suppress go ahead	858
5	Status	859
6	Timing mark	860
24	Terminal type	1091
31	Window size	1073
32	Terminal speed	1079
33	Remote flow control	1372
34	Linemode	1184
36	Environment variables	1408

2.3. Sự thỏa thuận các tùy chọn

Trong Telnel trước khi sử dụng một tùy chọn nào đó thì giữa Client và Server phải có thỏa thuận về tùy chọn đó. Có hai phương thức thỏa thuận là: đề nghị và yêu cầu.

Với hai hình thức này thì có hai kiểu thỏa thuận:

- ✓ Cho phép một tùy chọn
- ✓ Làm mất hiệu lực một tùy chọn

Các lệnh dùng trong thỏa thuận tùy chọn:WILL, DO, WONT, DONT

2.4. Sự nhúng trong telnet

Trong telnet để gửi các lệnh và dữ liệu thì sử dụng một kết nối duy nhất, các lệnh được nhúng ở trong dòng dữ liệu để bên nhận phân biệt được lệnh với dữ liệu trước mỗi kí tự điều khiển đều có kí tự IAC. Trong trường hợp có 2 kí tự IAC đi liền nhau thì kí tự IAC thứ nhất sẽ bị bỏ qua và kí tự IAC thứ hai sẽ là dữ liệu.

2.5. Các chế độ làm việc của Telnet

- *Chế độ mặc định*: được sử dụng khi không có sự thỏa thuận dùng một chế độ khác. Trong chế độ này, khi các ký tự được nhập vào từ bàn phím, nó sẽ phản hồi ngay lên màn

hình cục bộ và chỉ khi nhập hoàn chỉnh cả dòng ký tự thì dòng đó mới được gửi sang server và nó phải chờ tín hiệu GA (go Ahead) từ server trả về mới chấp nhận dòng mới (truyền theo kiểu half-duplex).

- *Chế độ Character*: trong chế độ này, mỗi khi có ký tự nhập vào từ bàn phím, trình Telnet Client gửi ký tự đó đến cho Server, Server sẽ gửi phản hồi ký tự đó lại trình Client để hiển thị lên màn hình cục bộ.
- *Chế độ Line Mode*: chế độ này bổ sung sự khiếm khuyết của hai chế độ trên. Mỗi khi Client nhận một dòng, nó gửi tới Server và nó sẽ nhận dòng mới mà không cần chờ tín hiệu GA gửi về từ Server (truyền thông theo kiểu full-duplex).

3. Cài đặt dịch vụ Telnet Client với Java

Chương trình Telnet phía người sử dụng phải thực hiện các công việc sau:

- Tạo một đối tượng Socket và thiết lập kết nối tới TelnetServer với địa chỉ máy mà trên đó trình Telnet Server đang chạy, và số cổng mà Telnet Server đang nghe.

Ví dụ: Giả sử telnet server chạy trên may tính có địa chỉ IP là 192.168.1.10, địa chỉ cổng là 23:

Socket telnetclient=new Socket("192.168.1.10",23);

- Tạo luồng nhập/xuất cho socket.
- Thực hiện gửi/ nhận các lệnh của Telnet thông qua luồng nhập/xuất

ví dụ khi thoả thuận, client cần phải gửi lệnh WONT có mã là 252, IAC là 255 với lệnh:

```
if(c2==255)
{
        out.write(new byte[] {(byte)255, (byte)254, (byte)c2});
}
```

- Xây dựng giao diện GUI cho chương trình nếu muốn.

Sau đây là một chương trình ví dụ cài đặt dịch vụ Telnet đơn giản với giao thức Telnet:

```
// TelnetClient.java
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.net.*;
// Terminal hiển thị chữ trên cửa sổ
class Terminal extends Canvas
{
      // Kích cỡ font chữ
      private int charWidth, charHeight;
      // text[0] là dòng thao tác hiện tại
      private String[] text;
```

```
// Khoảng cách với viền cửa sổ chính chương trình
private final int margin=4;
// Số dòng lệnh tối đa được lưu lại
private final int lines=50;

// Constructor, khởi tạo các giá trị ban đầu
Terminal()
{
	charHeight=12;
	setFont(new Font("Monospaced", Font.PLAIN, charHeight));
	charWidth=getFontMetrics(getFont()).stringWidth(" ");
	text=new String[lines];
	for (int i=0; i<lines; ++i)
	text[i]="";
	setSize(80*charWidth+margin*2, 25*charHeight+margin*2);
	requestFocus();
	// Lắng nghe sự kiện con trỏ chuột
	addMouseListener(new MouseAdapter() {
		public void mousePressed(MouseEvent e)
		{
			requestFocus();
		}
	});
}
// In và lưu lại các kí tự người dùng nhập từ bàn phím
public void put(char c)
{
	Graphics g=getGraphics();
	if (c=='\r')
	{  // Return
		for (int i=lines-1; i>0; --i)
			text[i]=text[i-1];
		text[0]="";
		update(g);  // Clear screen and paint
	}
	// Các kí tự điều khiển: backspace, delete, telnet EC
	else if (c==8 || c==127 || c==247)
	{
		int len=text[0].length();
		if (len>0)
		{
```

```
                        --len;
        text[0]=text[0].substring(0, len);
        g.setColor(getBackground());
        g.fillRect(len*charWidth+margin, getSize().height-margin-charHeight,
                    (len+1)*charWidth+margin, getSize().height-margin);
                   }
              }
        else if (c=='\t')
              {     // Tab với khoảng cách 8 space
              text[0]+="        ";
              text[0].substring(0, text[0].length()&-8);
              }
        else if (c>=32 && c<127)
              {  // Kí tự có thể in
         g.drawString(""+c, margin+text[0].length()*charWidth,
                getSize().height-margin);
        text[0]+=c;
              }
        g.dispose();
        }
        // Hiển thị những gì đã gõ từ bàn phím
        public void paint(Graphics g)
        {
              int height=getSize().height;
              for (int i=0; i<lines; ++i)
              g.drawString(text[i], margin, height-margin-i*charHeight);
        }
}
// luồng nhận sẽ chờ các kí tự đến từ một luồng vào (Input
// stream) và gửi đến Termial. Đàm phán các lựa chọn đầu cuối
class Receiver extends Thread
{
        private InputStream in;
        private OutputStream out;
        private Terminal terminal;
        public Receiver(InputStream in, OutputStream out, Terminal terminal)
        {
            this.in=in;
            this.out=out;
            this.terminal=terminal;
            start();
```

```
	}
	// Đọc các kí tự và gửi đến đầu cuối
	public void run()
	{
	while (true)
	{
	try {
	int c=in.read();
	if (c<0)
	{	// EOF
	System.out.println("Connection closed by remote host");
		return;
	}
	else if (c==255)
	{	// Đàm phán các lựa chọn đầu cuối
	int c1=in.read();  // 253=do, 251=will
	int c2=in.read();  // option
	if (c1==253)  // do option, send "won't do option"
	out.write(new byte[] {(byte)255, (byte)252, (byte)c2});
	else if (c1==251) // will do option, send "don't do option"
	out.write(new byte[] {(byte)255, (byte)254, (byte)c2});
	}
	else
	terminal.put((char)c);
	}
	catch (IOException x) {
		System.out.println("Receiver: "+x);
	}}}}
// TelnetWindow. Gửi dữ liệu bàn phím từ terminal đến một socket từ
// xa và bắt đầu nhận các kí tự từ socket và hiển thị các kí tự đó  trên
terminal
class TelnetWindow extends Frame
{
	Terminal terminal;
	InputStream in;
	OutputStream out;
	// Constructor
	TelnetWindow(String hostname, int port)
	{
		super("telnet "+hostname+" "+port);  // Set title\
		// Thiết lập cửa sổ
```

```
            add(terminal=new Terminal());
            // Xử lý việc đóng cửa sổ
            addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e)
            {
            dispose();
            try {
            out.close();
               }
            catch (IOException x) {
            System.out.println("Closing connection: "+x);
            }
                  }
                  public void windowClosed(WindowEvent e) {
                        System.exit(0);
                  }
            });
            // Xử lý các thao tác với bàn phím
            terminal.addKeyListener(new KeyAdapter() {
            public void keyTyped(KeyEvent e) {
                  char k=e.getKeyChar();
                  try {
                        terminal.put(k);
                        out.write((int)k);
                        if (k=='\r')
                        {
                              out.write('\n');  // Convert CR to CR-LF
                              out.flush();
                        }
                        }
                        catch (IOException x) {
                        System.out.println("Send: "+x);
                        }}});
          try {
            // Mở một connection
            System.out.println("Opening  connection  to  "+hostname+"  on  port
"+port);
            Socket socket=new Socket(hostname, port);
            InetAddress addr=socket.getInetAddress();
            System.out.println("Connected to "+addr.getHostAddress());
            in=socket.getInputStream();
```

```
            out=socket.getOutputStream();
            // Hiển thị cửa sổ
            pack();
            setVisible(true);
            // Bắt đầu nhận dữ liệu từ server
            new Receiver(in, out, terminal);
            System.out.println("Ready");
          }
          catch (UnknownHostException x) {
            System.out.println("Unknown host: "+hostname+" "+x);
            System.exit(1);
            }
          catch (IOException x) {
                  System.out.println(x);
                  System.exit(1);
            }}}
// Chương trình chính
public class TelnetClient
{
      public static void main(String[] argv)
      {
          // Phân tách các đối số: telnet hostname port
          String hostname="";
          int port=23;
          try {
                  hostname=argv[0];
                  if (argv.length>1)
                  port=Integer.parseInt(argv[1]);
          }   catch (ArrayIndexOutOfBoundsException x) {
            System.out.println("Usage: java telnet hostname [port]");
            System.exit(1);
          }
          catch (NumberFormatException x) {}
          TelnetWindow t1=new TelnetWindow(hostname, port);
         }
      }
```

4. Chạy thử chương trình

Bước 1: Dịch chương trình TelnetClient.java

Bước 2: Kiểm tra xem trên máy từ xa, trình Telnet server đã được khởi tạo chạy chưa, nếu chưa thì chạy nó và dùng trình quản trị Telnet Server, thiết lập các tham số phù hợp.

Bước 3: Chạy chương trình Telnet Client từ máy cục bộ.

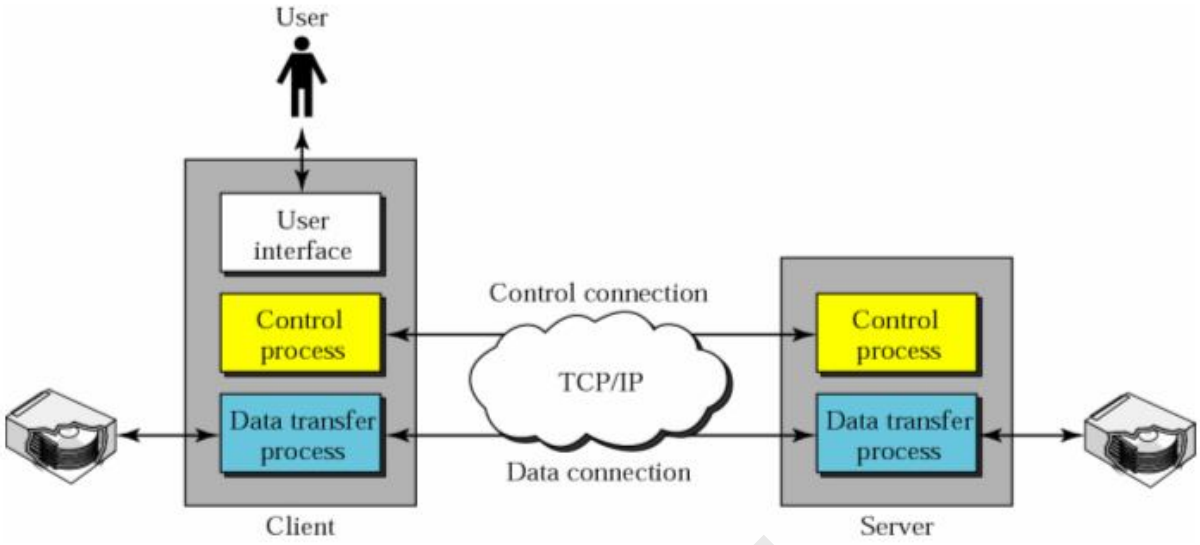
III. LẬP TRÌNH DỊCH VỤ TRUYỀN TỆP VỚI GIAO THỨC FTP

1. Dịch vụ truyền tệp FTP

1.1. Giao thức FTP

1.1.1. Đặc điểm

- FTP là giao thức chuẩn của TCP/IP
- FTP sử dụng kết nối TCP, là kết nối truyền thông tin cậy
- FTP gồm 2 phần mềm: Phần mềm FTPClient cài trên máy cục bộ và FTPServer cài trên máy từ xa(File Server).
- FTP sử dựng 2 kết nối truyền thông đồng thời để tăng hiệu quả của việc truyền tệp qua mạng:
  - ❖ Kết nối điều khiển: Sử dụng phương thức truyền thông đơn giản và dữ liệu truyền dưới dạng text(NVT-ASCII 7bít). Kết nối này cho phép truyền lệnh từ client tới server và truyền đáp ứng từ server về client. Kết nối này sử dụng số cổng mặc định là 21 phía server.
  - ❖ Kết nối dữ liệu: Kết nối này sử dụng các phương thức truyền thông phức tạp vì phải truyền nhiều kiểu dữ liệu khác nhau. Kết nối này được thiết lập mỗi khi truyền một tệp và huỷ sau khi truyền xong tệp đó. Kết nối này bao giờ cũng được khởi tạo sau kết nối điều khiển và kết thúc trước khi huỷ bỏ kết nối điều khiển(kết nối điều khiển duy trì trong suốt phiên làm việc). Kết nối dữ liệu sử dụng số cổng mặc định phía server là 20. Có 2 cách thiết lập kết nối dữ liệu: dùng lệnh PORT và lệnh PASV.
- FTP có 3 chế độ truyền tệp:
  - ❖ Cất tệp trên máy cục bộ lên máy tính từ xa dưới sự giám sát của lệnh STOR.
  - ❖ Lấy một tệp trên máy tính từ xa về máy tính cục bộ dưới sự giám sát của lệnh RETR.
  - ❖ Lấy danh sách các mục trong một thư mục trên máy từ xa về máy cục bộ dưới sự giám sát của lệnh LIST.
- Mô hình hoạt động của FTP thể hiện như hình vẽ



Hình 4.3. Mô hình FTP

1.1.2. *Tập lệnh và đáp ứng của FTP*

1.1.2.1. Tập lệnh:

Tập lệnh FTP chỉ được thi hành phía FTP Server, không dùng cho người sử dụng. Khi client gửi một lệnh FTP đến FTPServer, lệnh đó sẽ được FTPServer thi hành và trả đáp ứng về cho client. Cú pháp lệnh FTP có dạng:

<COMMAND> <SPACE> [PARAMS]

FTP có hơn ba mươi lệnh được chia làm sáu nhóm và được liệt kê trong bảng sau:

Nhóm lệnh truy cập:

Command	*Argument(s)*	*Description*
USER	User id	User information
PASS	User password	Password
ACCT	Account to be charged	Account information
REIN		Reinitialize
QUIT		Log out of the system
ABOR		Abort the previous command

Nhóm lệnh quản lý tệp:

Command	*Argument(s)*	*Description*
CWD	Directory name	Change to another directory
CDUP		Change to the parent directory
DELE	File name	Delete a file
LIST	Directory name	List subdirectories or files
NLIST	Directory name	List the names of subdirectories or files without other attributes
MKD	Directory name	Create a new directory
PWD		Display name of current directory
RMD	Directory name	Delete a directory
RNFR	File name (old file name)	Identify a file to be renamed
RNTO	File name (new file name)	Rename the file
SMNT	File system name	Mount a file system

Nhóm lệnh định dạng dữ liệu:

Command	*Argument(s)*	*Description*
TYPE	A (ASCII), E (EBCDIC), I (Image), N (Nonprint), or T (TELNET)	Define the file type and if necessary the print format
STRU	F (File), R (Record), or P (Page)	Define the organization of the data
MODE	S (Stream), B (Block), or C (Compressed)	Define the transmission mode

Nhóm lệnh định nghĩa cổng:

Command	*Argument(s)*	*Description*
PORT	6-digit identifier	Client chooses a port
PASV		Server chooses a port

Nhóm lệnh truyền tệp:

Command	*Argument(s)*	*Description*
ALLO	File name(s)	Allocate storage space for the files at the server
REST	File name(s)	Position the file marker at a specified data point
STAT	File name(s)	Return the status of files

Nhóm lệnh còn lại:

Command	*Argument(s)*	*Description*
HELP		Ask information about the server
NOOP		Check if server is alive
SITE	Commands	Specify the site-specific commands
SYST		Ask about operating system used by the server

1.2.1.2. Tập đáp ứng(response)

Đáp ứng FTP được gửi từ FTP server về client sau mỗi khi FTP server thực thi một lệnh FTP gửi từ client đến server. Cú pháp của một đáp ứng của FTP có dạng sau:

<XYZ> <SPACE> <TEXT>

Với XYZ là phần mã gồm 3 số nguyên, mỗi chữ số và giá trị số được ấn định với một ý nghĩa xác định:

Code	*Description*
Positive Preliminary Reply	
120	Service will be ready shortly
125	Data connection open; data transfer will start shortly
150	File status is OK; data connection will be open shortly

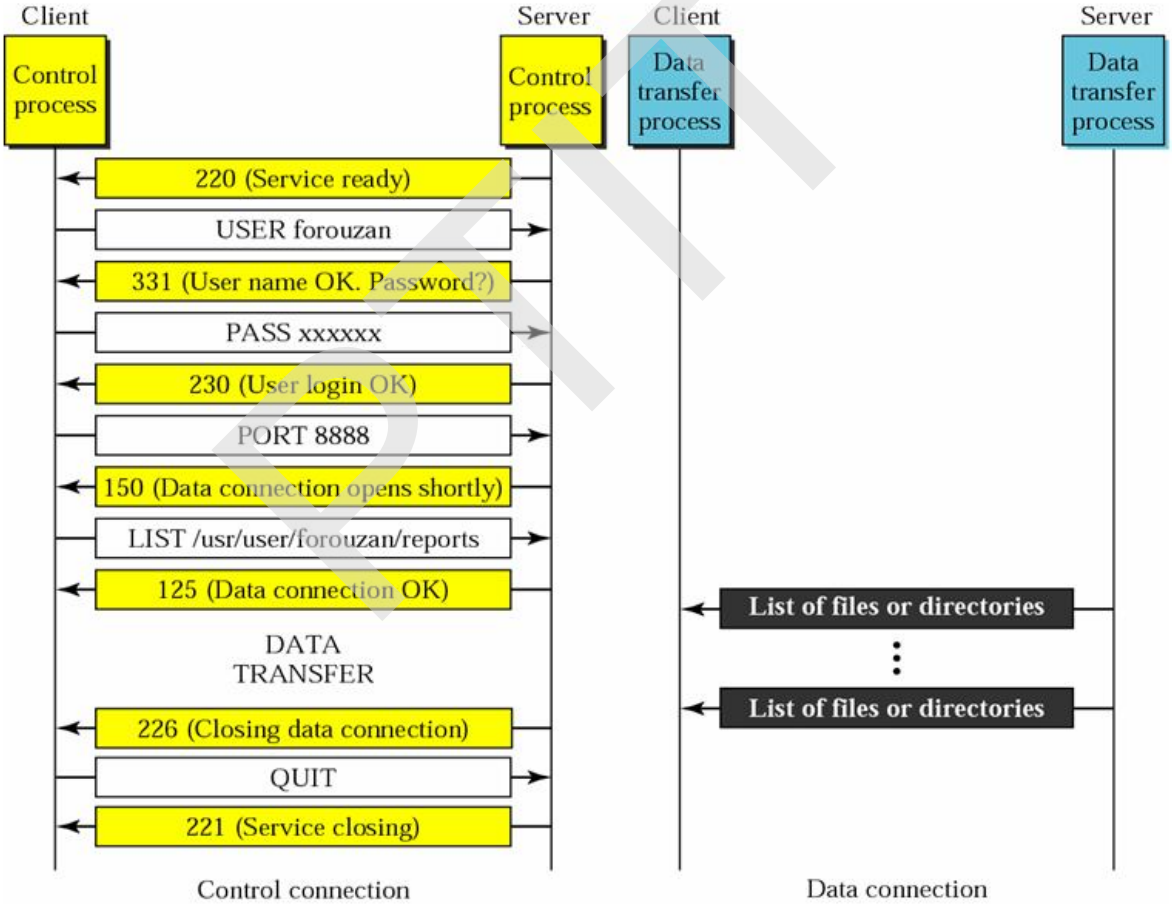
Positive Completion Reply	
200	Command OK
211	System status or help reply
212	Directory status
213	File status
214	Help message
215	Naming the system type (operating system)
220	Service ready
221	Service closing
225	Data connection open
226	Closing data connection
227	Entering passive mode; server sends its IP address and port number
230	User login OK
250	Request file action OK

Positive Intermediate Reply	
331	User name OK; password is needed
332	Need account for logging
350	The file action is pending; more information needed

Transient Negative Completion Reply	
425	Cannot open data connection
426	Connection closed; transfer aborted
450	File action not taken; file not available
451	Action aborted; local error
452	Action aborted; insufficient storage
Permanent Negative Completion Reply	
500	Syntax error; unrecognized command

501	Syntax error in parameters or arguments
502	Command not implemented
503	Bad sequence of commands
504	Command parameter not implemented
530	User not logged in
532	Need account for storing file
550	Action is not done; file unavailable
552	Requested action aborted; exceeded storage allocation
553	Requested action not taken; file name not allowed

1.1.3. Ví dụ quá trình truyền tệp giữa FTPclient và FTPserver



Hình 4.4. Ví dụ quá trình truyền tệp FTP

2. Kỹ thuật cài đặt giao thức FTP với java

2.1. Các bước cài đặt:

Để có thể truyền tệp với máy chủ truyền tệp với giao thức FTP, chương trình phải:

- Thiết lập và huỷ bỏ kết nối điều khiển.

- Thiết lập và huỷ bỏ kết nối dữ liệu sử dụng lệnh PORT hoặc PASV
- Gửi các lệnh từ client tới server và nhận đáp ứng từ server trả về. Tốt nhất là viết các phương thức bao lấy các lệnh của FTP và phương thức xử lý đáp ứng trả về.
- Nắm chắc trình tự để có thể thực hiện download hoặc upload tệp sử dụng giao thức FTP.

2.2.Chương trình truyền tệp FTP

Trong chương trình này, chúng tôi thực hiện các công việc sau:

- Khai báo tạo đối tượng Socket và thiết lập kết nối tới FTPServer để tạo kết nối điều khiển và tạo luồng nhập xuất cho socket:

Ví du: Giả sử FTPServer nằm trên máy cục bộ và sử dụng số cổng mặc định 21

Socket clientFTP=new Socket("localhost",21);

Hoặc viết phương thức kết nối như ví dụ sau:

```
public boolean connect(String host, int port)
        throws UnknownHostException, IOException
    {
        connectionSocket = new Socket(host, port);
        outputStream = new
PrintStream(connectionSocket.getOutputStream());
        inputStream = new BufferedReader(new

InputStreamReader(connectionSocket.getInputStream()));

        if (!isPositiveCompleteResponse(getServerReply())){
            disconnect();
            return false;
        }

        return true;
    }
```

Hoặc hàm giải phóng kết nối:

```
 public void disconnect()
     {
         if (outputStream != null) {
             try {
                   if (loggedIn) { logout(); };
                 outputStream.close();
                 inputStream.close();
                 connectionSocket.close();
             } catch (IOException e) {}

             outputStream = null;
             inputStream = null;
             connectionSocket = null;
         }
     }
```

- Khai báo các phương thức để thực hiện gửi các lệnh của FTP tới FTPServer:

Ví dụ:

❖ Phương thức thực hiện đăng nhập với lệnh USER và PASS

```
public boolean login(String username, String password)
        throws IOException
    {
        int response = executeCommand("user " + username);
        if (!isPositiveIntermediateResponse(response)) return false;
        response = executeCommand("pass " + password);
        loggedIn = isPositiveCompleteResponse(response);
        return loggedIn;
    }
```

Trong đó phương thức *executeCommand()* để thực thi một lệnh FTP bất kỳ:

```
public int executeCommand(String command)
        throws IOException
    {
        outputStream.println(command);
        return getServerReply();
    }
```

❖ **Phương thức đọc/ghi dữ liệu:**

```
public boolean readDataToFile(String command, String fileName)
        throws IOException
    {
        // Open the local file
        RandomAccessFile outfile = new RandomAccessFile(fileName, "rw");
        // Do restart if desired
        if (restartPoint != 0) {
            debugPrint("Seeking to " + restartPoint);
            outfile.seek(restartPoint);
        }
        // Convert the RandomAccessFile to an OutputStream
        FileOutputStream fileStream = new FileOutputStream(outfile.getFD());
        boolean success = executeDataCommand(command, fileStream);
        outfile.close();
        return success;
    }
}
public boolean writeDataFromFile(String command, String fileName)
        throws IOException
    {
        // Open the local file
        RandomAccessFile infile = new RandomAccessFile(fileName, "r");
        // Do restart if desired
        if (restartPoint != 0) {
            debugPrint("Seeking to " + restartPoint);
            infile.seek(restartPoint);
        }
        // Convert the RandomAccessFile to an InputStream
        FileInputStream fileStream = new FileInputStream(infile.getFD());
        boolean success = executeDataCommand(command, fileStream);
        infile.close();
        return success;
    }
```

❖ **Phương thức download và Upload tệp:**

```
public boolean downloadFile(String fileName)
        throws IOException
```

```
        {
            return readDataToFile("retr " + fileName, fileName);
        }

public boolean downloadFile(String serverPath, String localPath)
        throws IOException
    {
        return readDataToFile("retr " + serverPath, localPath);
    }
```

❖ **Một số phương thức thực hiện các lệnh FTP được liệt kê trong bảng sau:**

STT	Phương thức cài đặt	Lênh FTP
1	*public boolean changeDirectory(String directory) throws IOException*	**CD**
2	*public boolean renameFile(String oldName, String newName) throws IOException*	**RNFR, RNTO**
3	*public boolean removeDirectory(String directory) throws IOException*	**RMD**
4	*public boolean deleteFile(String fileName) throws IOException*	**DELE**
5	*public String getCurrentDirectory() throws IOException*	**PWD**

2.3. Chương trình ví dụ

Đoạn chương trình sau là ví dụ minh hoạ các phương thức đã cài đặt trên:

```
try {
    if (connection.connect(host)) {
         if (connection.login(username, password)) {
             connection.downloadFile(serverFileName);
              connection.uploadFile(localFileName);
          }
          connection.disconnect();
      }
  } catch (UnknownHostException e) {
      // handle unknown host
  } catch (IOException e) {
      // handle I/O exception
  }
```

IV. LẬP TRÌNH GỬI/NHẬN THƯ VỚI GIAO THỨC SMTP và POP3

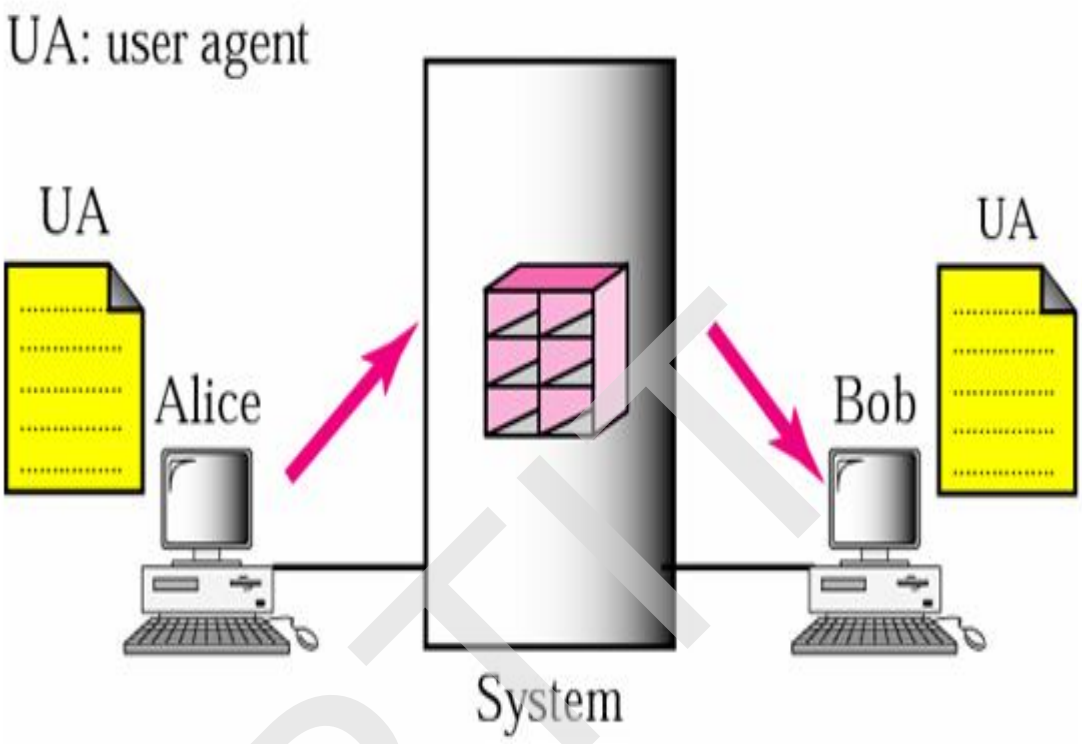
1. Giao thức SMTP

1.1 Giới thiệu

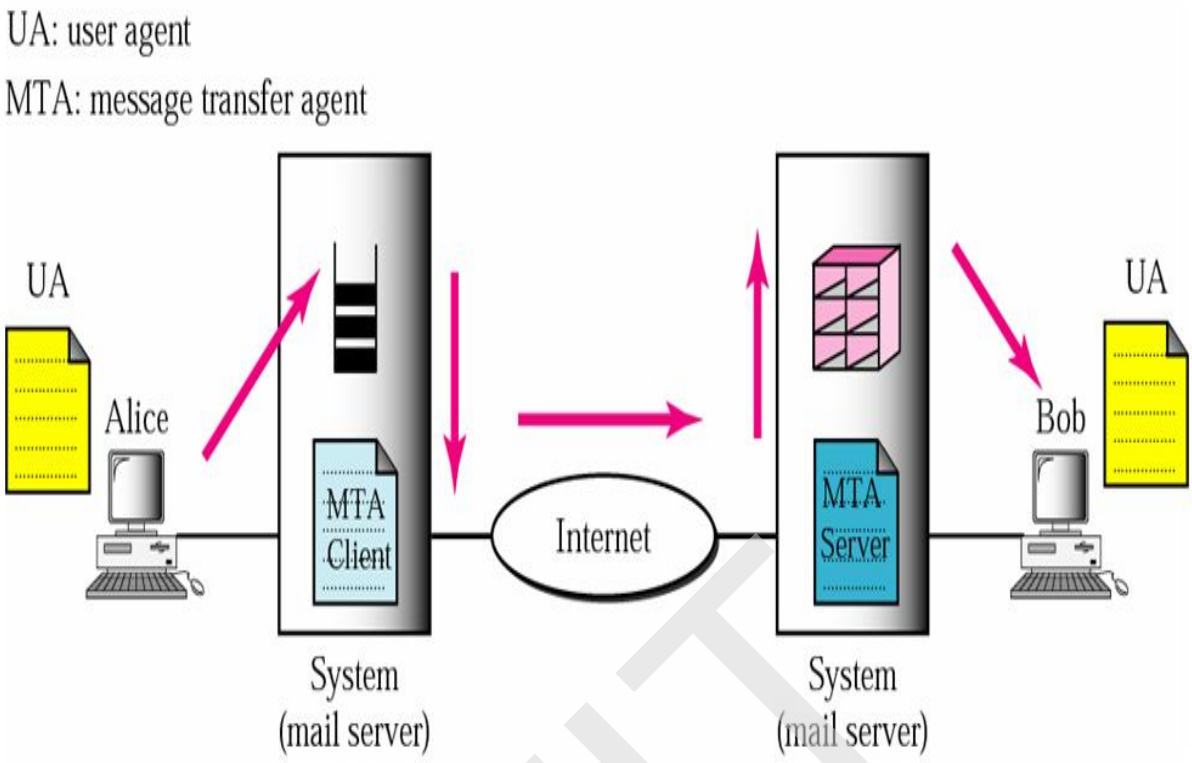
Mục đích của giao thức SMTP là truyền mail một cách tin cậy và hiệu quả. Giao thức SMTP không phụ thuộc vào bất kỳ hệ thống đặc biệt nào và nó chỉ yêu cầu trật tự của dữ liệu truyền trên kênh đảm bảo tin cậy.

1.2 Mô hình của giao thức SMTP

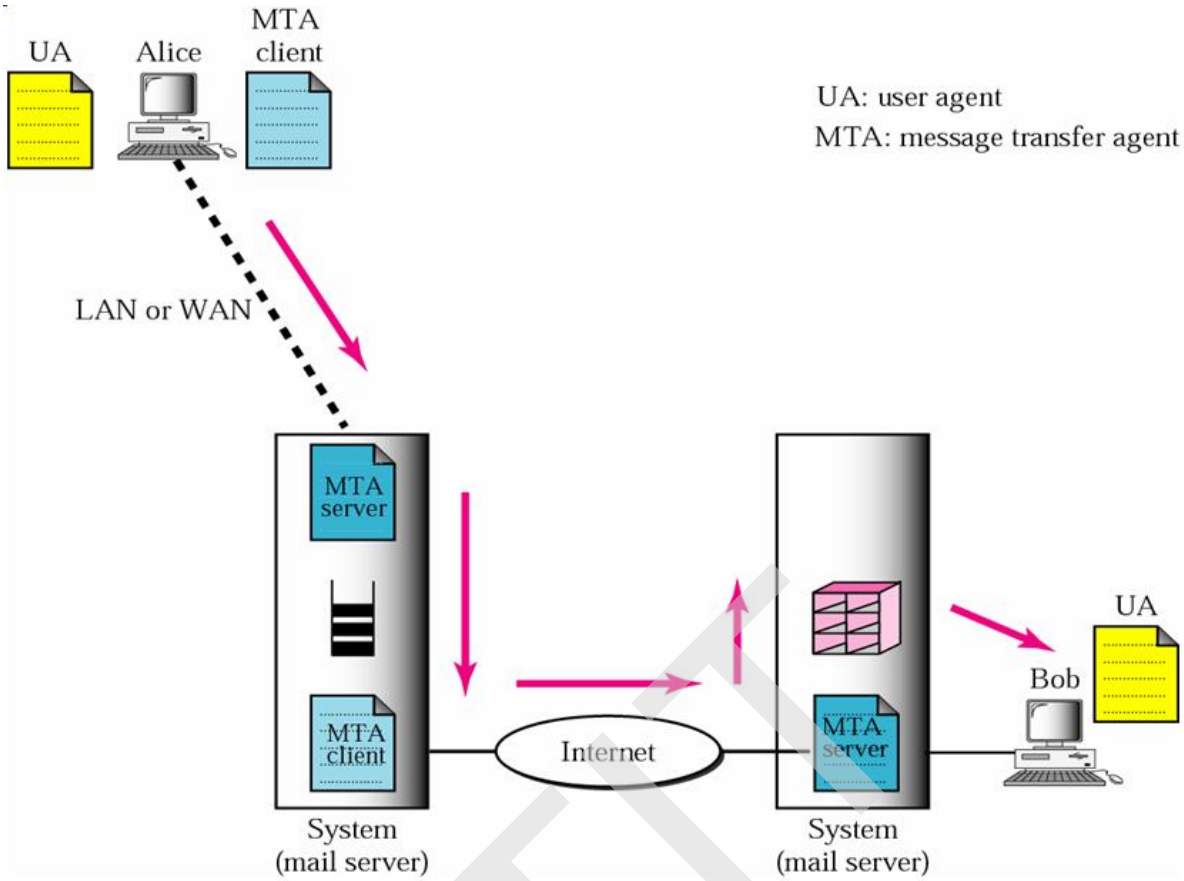
Giao thức SMTP được thiết kế dựa vào mô hình giao tiếp sau: khi có yêu cầu từ user về dịch vụ mail, bên gửi Sender-SMTP thiết lập một kênh truyền hai chiều tới bên nhận Receiver-SMTP và Receiver-SMTP gởi đáp ứng trở lại cho Sender-SMTP



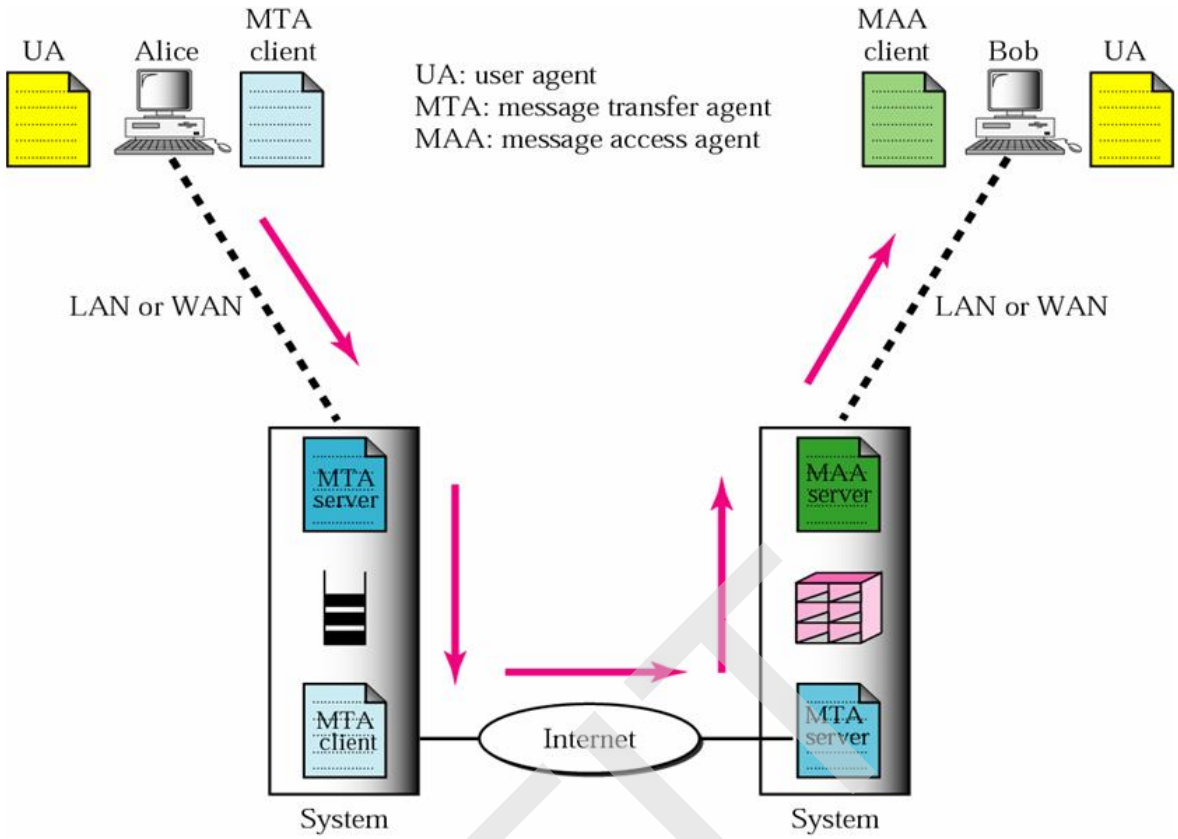
Hình 4.6. . Mô hình người gửi và nhận trên cùng hệ thống



Hình 4.7.Mô hình gửi thư qua hệ thống khác nhau



Hình 4.8.Mô hình gửi thư cả 2 phía qua mạng LAN/WAN



Hình 4.9. Mô hình người gửi/nhận kết nối mail server qua LAN/WAN

1.3. Tập lệnh và đáp ứng của SMTP

Những lệnh SMTP định nghĩa sự truyền mail hay chức năng của hệ thống mail được yêu cầu bởi user. Những lệnh SMTP là những chuỗi ký tự kết thúc bằng <CRLF>. Bản thân mã lệnh là những ký tự chữ (alphabetic) kết thúc bởi <SP> nếu có những tham số theo sau và nếu không có thì <CRLF>. Cú pháp của những mailbox phải tuân theo những quy ước của receiver.

Một phiên giao dịch mail chứa đựng một vài đối tượng dữ liệu, được truyền như là những đối số cho các lệnh khác nhau. Receiver-path là đối số của lệnh MAIL. Forward-path là đối số của những lệnh RCPT. Và mail data là đối số của lệnh DATA. Nhưng đối số hay những đối tượng dữ liệu này được truyền đi và duy trì cho đến khi truyền xong bởi sự chỉ định kết thúc của mail data. Mô hình hiện thực cho cách làm này là những buffer riêng biệt được cung cấp để lưu trữ kiểu của đối tượng dữ liệu, đó là các buffer: reverse-path, forward-path, và mail data buffer. Nhưng lệnh xác định tạo ra thông tin được gắn vào một buffer xác định , hoặc xóa bớt đi một hay một số buffer nào đó

Keyword	*Argument(s)*
HELO	Sender's host name
MAIL FROM	Sender of the message
RCPT TO	Intended recipient of the message
DATA	Body of the mail
QUIT	
RSET	
VRFY	Name of recipient to be verified
NOOP	
TURN	
EXPN	Mailing list to be expanded
HELP	Command name
SEND FROM	Intended recipient of the message
SMOL FROM	Intended recipient of the message
SMAL FROM	Intended recipient of the message

Còn các đáp ứng của SMTP tương tự gần giống như của FTP nhưng giá trị của x chỉ lấy từ 2 đến 5.

1.4. Cài đặt chương trình gửi thư với SMTP

Để gửi thư, chương trình ứng dụng phải thực hiện các thao tác cơ bản sau đây:

- Đầu tiên phải tạo đối tượng socket và kết nối tới mail server bằng cách chỉ ra tên miền hoặc địa chỉ IP của máy chủ mail server và sử dụng số cổng mặc định 25.
- Khai báo tạo luồng nhập xuất cho socket
- Thực hiện lần lượt gửi các lệnh và tham số của SMTP tới mail server theo trận tự sau:
  - ❖ HELLO

- MAIL FROM
- RCPT TO
- DATA
- QUIT

Sau mỗi lệnh gửi, phải thực hiện đọc các đáp ứng trả về.

Ví dụ về một giao dịch gửi thư của SMTP:

R: 220 BBN-UNIX.ARPA Simple Mail Transfer Service Ready.
S: HELO USC-ISIF.ARPA
R: 250 BBN-UNIX.ARPA
S: MAIL FROM:<Smith@USC-ISIF.ARPA>
R: 250 OK
S: RCPT TO:<Green@BBN-UNIX.ARPA>
R: 250 OK
S: DATA
R: 354 Start mail input; end with <CRLF>.<CRLF>
S: ...
S: ...
S: ...
...
R: 250 OK
S: QUIT
R: 221 BBN-UNIX.ARPA Service closing transmission channel.

Sau đây là mã cài đặt của chương trình ví dụ:

```
import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.InputStreamReader;
import java.io.PrintStream;
import java.net.Socket;
import java.util.StringTokenizer;

public class SendMail
{
Object mailLock             = null;  //In case we want a multi-threaded
mailer
public String mailServerHost = "";
public String from           = "";
public String to             = "";
public String replyTo        = "";
public String subject        = "Java is Fun";
public String mailData       =
   "HyperSendMail";
public String errorMsg = "";
public Socket mailSendSock = null;
```

```
public  BufferedReader inputStream = null;
public PrintStream outputStream    =  null;
public String serverReply          = "";
SendMail()
{
   // Doesn't do anything but we need this for extension purposes.
}

// Server, from,to,subject, data

SendMail(String server,String tFrom,String tTo,String sub,String sendData)
{
   mailServerHost = server;
   mailLock=this; //    from = tFrom;
   to   = tTo;
   if(sendData != null)
      mailData = sendData;
}

SendMail()
{
   if(mailLock != null)
   {
      if(mailLock instanceof Applet)
      {
         Applet app = (Applet)
      }
   }
}
*/

public void send()
{
   if(!open())          //Yikes! get out of here.
      return;
   try
   {
      outputStream.println("HELO sendMail");
      serverReply = inputStream.readLine();
   }
   catch(Exception e0)
   {
      e0.printStackTrace();
   }
   try
   {
      outputStream.println("MAIL FROM: "+from);
      serverReply = inputStream.readLine();
      if(serverReply.startsWith("5"))
      {
         close("FROM: Server error :"+serverReply);
         return;
      }

      if(replyTo == null)
         replyTo = from;
      outputStream.println("RCPT TO: <"+to+">");
      serverReply = inputStream.readLine();
      if(serverReply.startsWith("5"))
```

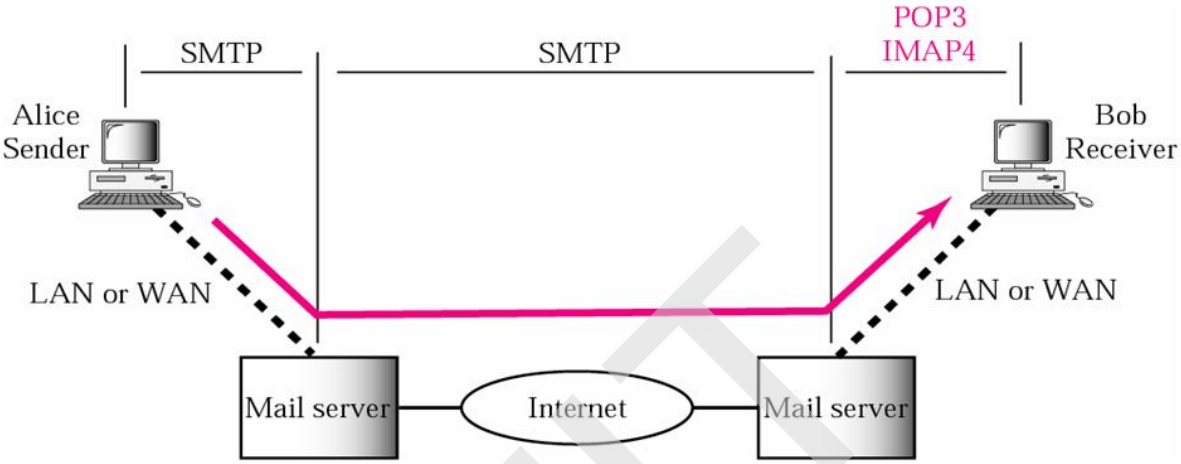
```
      {
         close("Reply error:"+serverReply);
         return;
      }
      outputStream.println("DATA");
      serverReply = inputStream.readLine();
      if(serverReply.startsWith("5"))
      {
         close("DATA Server error : "+serverReply);
         return;
      }
      outputStream.println("From: "+from);
      outputStream.println("To: "+to);
      if(subject != null)
         outputStream.println("Subject: "+subject);
      if(replyTo != null)
         outputStream.println("Reply-to: "+replyTo);
      outputStream.println("");
      outputStream.println(mailData);
      outputStream.print("\r\n.\r\n");
      outputStream.flush();
      serverReply = inputStream.readLine();
      if(serverReply.startsWith("5"))
      {
         close("DATA finish server error: "+serverReply);
         return;
      }
      outputStream.println("quit");
      serverReply = inputStream.readLine();
      if(serverReply.startsWith("5"))
      {
         close("Server error on QUIT: "+serverReply);
         return;
      }
      inputStream.close();
      outputStream.close();
      mailSendSock.close();
   }
   catch(Exception any)
   {
      any.printStackTrace();
      close("send() Exception");
   }
   close("Mail sent");
}

public boolean open()
{
   synchronized(mailLock)
   {
      try
      {
         mailSendSock = new Socket(mailServerHost, 25);
         outputStream = new PrintStream(mailSendSock.getOutputStream());
         inputStream = new BufferedReader(new InputStreamReader(
          mailSendSock.getInputStream()));
         serverReply = inputStream.readLine();
         if(serverReply.startsWith("4"))
         {
```

```
            errorMsg = "Server refused the connect message :
"+serverReply;
            return false;
         }
      }
      catch(Exception openError)
      {
         openError.printStackTrace();
         close("Mail Socket Error");
         return false;
      }
      System.out.println("Connected to "+mailServerHost);
      return true;
   }

}

public void close(String msg)
{
          //try to close the sockets
   System.out.println("Close("+msg+")");
   try
   {
      outputStream.println("quit");
      inputStream.close();
      outputStream.close();
      mailSendSock.close();
   }
   catch(Exception e)
   {
      System.out.println("Close() Exception");
     // We are closing so see ya later anyway
   }
}

// What do you know the damned thing works :)
/*
public static void main(String Args[])
{
SendMail sm = new
SendMail(
          "mail.hyperbyte.ab.ca",          //Mail Server
          "tswain@hyperbyte.ab.ca",        // sender
          "tswain@hyperbyte.ab.ca",        // Recipient
          "Java mail test",                // Subject
          "test test test!");              // Message Data
          sm.send();                       // Send it!
}
*/

/*
// Going a be an applet/thread safe version of readLine()
public void readLine(DataInputStream in,)
{
}
*/
}
```

2. Giao thức POP3

2.1. Giới thiệu

POP3 (Post Office Protocol Version 3) là một giao thức truy cập hộp thư. Nó gồm 2 phần mềm: POP3 Server cài trên máy chủ có chứa hộp thư; POP3 Client cài đặt trên máy cục bộ. Để truy cập được thư, người sử dụng dùng phần mềm truy cập hộp thư thiết lập kết nối tới POP3 Server tại số cổng mặc định là 110. POP3 server sẽ gửi trả về cho client một danh sách các mục thư chứa trong hộp thư người sử dụng. Giai đoạn sử dụng giao thức truy cập thư được thể hiện như hình vẽ.



2.2. Một số lệnh và đáp ứng của POP3

Một số lệnh quan trọng của POP3 được miêu tả sau đây. Còn các đáp ứng của POP3 tương tự như của giao thức FTP.

- **USER username:** đối số username là một chuỗi định danh mailbox, chỉ có ý nghĩa đối với server. Nó trả lời "+OK" nếu tên mailbox có hiệu lực và "-ERR" nếu không chấp nhận tên mailbox
- **PASS string:** đối số là một password cho mailbox hay server. Nó trả lời "+OK" đã khóa maildrop và sẵn sàng và "-ERR" nếu password không hiệu lực hoặc không được phép khóa maildrop.
- **QUIT:** Không có đối số và trả lời "+OK".
- **STAT:** không có đối số. Trả lời "+OK nn mm" với nn là số message, mm là kích thước maildrop tính bằng byte. Các message được đánh dấu xóa không được đếm theo tổng số.

- **LIST [msg]:** đối số là số thứ tự của message, có thể không liên quan tới các message đã được đánh dấu xóa. Trả lời "+OK scan listing flow" với scan listing là số thứ tự của message đó, theo sau là khoảng trống và kích thước chính xác của message đó tính theo byte; hoặc trả lời "-ERR no such message".
- **RETR msg:** đối số là số thứ tự message, có thể không liên quan tới các message đã được đánh dấu xóa. Trả lời "+OK message flows" hoặc "-ERR no such message".
- **DELE msg:** đối số là số thứ tự message, có thể không liên quan tới các message đã được đánh dấu xóa. Trả lời "+OK message deleted", POP3 sẽ đánh dấu xóa message này hoặc "-ERR no such message".
- **NOOP:** không có đối số và trả lời "+OK". POP3 server không làm gì hết, chỉ hồi âm lại cho client với trả lời "+OK".
- **RSET:** không có đối số, trả lời "+OK" để phục hồi lại các message đã bị đánh dấu xóa bởi POP3 server.

2.3. Các thao tác truy cập thư

Để thực hiện truy cập lấy thư, chương trình lấy thư phải thực hiện các thao tác cơ bản sau:

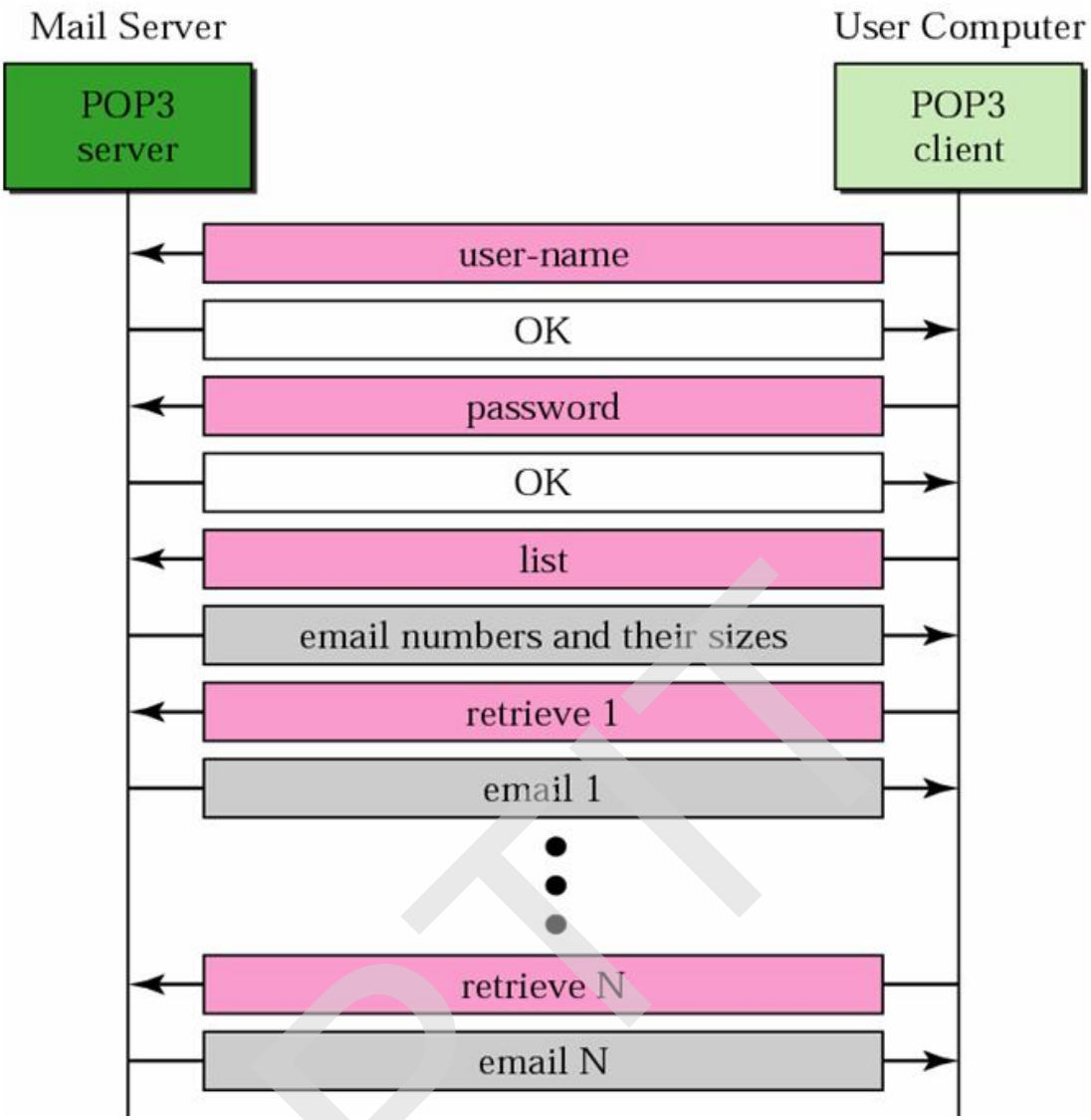
- Đăng nhập bằng lệnh USER, PASS với tài khoản hợp lệ
- Gửi lệnh thao tác với hộp thư.

Ví dụ quá trình thực hiện truy cập hộp thư lấy thư thể hiện như hình 4.10

2.4. Xây dựng chương trình truy cập hộp thư với giao thức POP3

Các thao tác cơ bản:

- Tạo đối tượng Socket và thiết lập với Mail Server tại số cổng 110.
- Tạo luồng nhập/xuất
- Thực hiện gửi lệnh tới mail server, sau mỗi lệnh gửi, nó thực hiện đọc đáp ứng trả về
- Kết thúc chương trình



Hình 4.10. Ví dụ quá trình lấy thư với giao thức POP3

Chương trình ví dụ sau minh hoạ cách cài đặt chương trình nhận thư với giao thức POP3.

```
//CheckMail.java
import java.net.*;
import java.io.*;
public class CheckMail {
public static void main(String s[]) {
   //  CheckMail [mailServer] [user] [password]
   try {
     CheckMail t = new CheckMail();
     int i = t.checkMyMail(s[0], s[1], s[2]);
     if (i==0) {
       System.out.println("No mail waiting.");
     }
     else {
       System.out.println
        ("There " + (i==1?"is " :"are ") + i +
         " message" +(i==1?"":"s")+ " waiting.");
     }
```

```
            }
            catch (Exception e) {
              e.printStackTrace();
            }
         }
         private void send(BufferedWriter out, String s) throws IOException {
            out.write(s+"\n");
            out.flush();
         }
         private String receive(BufferedReader in) throws IOException {
            return in.readLine();
         }
         private int checkMyMail
              (String server, String user, String pass) throws IOException {
            Socket s = new Socket(server, 110);
            BufferedReader in  = new BufferedReader(
                new InputStreamReader(s.getInputStream()));
            BufferedWriter out = new BufferedWriter(
                new OutputStreamWriter(s.getOutputStream()));
            receive(in);
            send(out, "USER " + user);
            receive(in);
            send(out, "PASS " + pass);
            receive(in);
            return getNumberOfMessages(in, out);
         }
         public int getNumberOfMessages
            (BufferedReader in, BufferedWriter out) throws IOException {
            int i = 0;
            String s;
            send(out, "LIST");
            receive(in);
            while((s = receive(in)) != null) {
               if (!(s.equals("."))) {
                 i++;
               }
               else
                 return i;
               }
            return 0;
         }
}
```

V. KẾT LUẬN

Như vậy trong chương này đã bước đầu cung cấp cho người lập trình cách lập trình với các giao thức truyền thông đã phát triển sẵn có thông qua kỹ thuật socket. Đây là chương quan trọng, nó vừa củng cố cho sinh viên kiến thức mạng, vừa trang bị cho sinh viên biết cách cài đặt các giao thức đó bằng một ngôn ngữ lập trình cụ thể. Trên cơ sở đó sinh viên có thể hoàn thiện một dịch vụ mạng hoàn chỉnh hoặc phát triển các modul chương trình để tích hợp vào các chương trình uúng dụng khác nhau. Ngoài các giao thức trên, sinh viên nên lập trình với một số giao thức Internet phổ biến khác như DNS, TFTP, HTTP, RTP hoặc cài đặt các giao thức, gói tin của các giao thức TCP, UDP, ICMP, ARP, IP, ICMP hoặc khảo sát phát triển các ứng dụng với họ giao thức Hxxx, SIP...Cuỗi cùng một điều nhấn mạnh với người học khi phát triển các ứng dụng mạng với các giao thức: Phải nắm chắc mô hình, cấu trúc, cơ chế truyền thông của các giao thức thì

mới lập trình được. Một vấn đề khác, thông qua chương này người lập trình có thể phát triển các giao thức truyền thông riêng của mình để giải quyết bài toán cụ thể.

PHẦN III. LẬP TRÌNH PHÂN TÁN

CHƯƠNG IV

KỸ THUẬT LẬP TRÌNH PHÂN TÁN ĐỐI TƯỢNG RMI

I. GIỚI THIỆU LẬP TRÌNH PHÂN TÁN VÀ RMI

(Remote Method Invocation)

1. Giới thiệu kỹ thuật lập trình phân tán

Kỹ thuật lập trình phân tán thực chất là kỹ thuật lập trình phân tán mã lệnh hay đối tượng. Nó cho phép phân bố tải lên toàn mạng để tận dụng tài nguyên mạng giải quyết bài toán lớn, phức tạp thay vì tập trung trên cùng một máy. Các thành phần mã lệnh phân tán "kết cặp" với nhau một cách chặt chẽ, khác với lập trình socket là "kết cặp" lỏng lẻo. Một điểm khác cơ bản nữa của lập trình phân tán so với lập trình socket là: Socket là giao diện, còn các kỹ thuật lập trình phân tán như RPC, RMI...là cơ chế truyền thông.

Hiện này có nhiều kỹ thuật lập trình phân tán khác nhau như:

- Kỹ thuật gọi thủ tục từ xa RPC(Remote Procedure Call)
- Kỹ thuật gọi phương thức từ xa RMI(Remote Method Invocation)
- Kỹ thuật mô hình đối tượng thành phần phân tán DCOM
- Kỹ thuật kiến trúc môi giới trung gian CORBA
- Kỹ thuật EJB, WebService, RPC-XML...

Các kỹ thuật lập trình phân tán hiện này đều hướng đến mô hình đa tầng với kỹ thuật lập trình hướng dịch vụ(SOP) mà tiêu biểu là WebService. Vì nó cho phép giải quyết các bài toán lớn, phức tạp hiệu quả và nhiều ưu điểm khác. Kỹ thuật lập trình RMI tương tự như kỹ thuật RPC nhưng khác ở chỗ: Trong RPC chương trình client gọi thủ tục phía Server , còn trong RMI client gọi phương thức từ xa ở phía server(hướng đối tượng).

2. Giới thiệu kỹ thuật lập trình RMI

2.1. Đặc trưng của kỹ thuật RMI

RMI là kỹ thuật lập trình phân tán đối tượng, nó cho phép gọi phương thức của đối tượng từ xa qua mạng và nhận kết quả trả về từ máy từ xa.

RMI là một cơ chế truyền thông và là kỹ thuật thuần Java. Điều đó nghĩa là, kỹ thuật RMI chỉ cho phép các đối tượng thuần Java mới gọi từ xa phương thức của nhau được. Còn các đối tượng viết bằng ngôn ngữ khác như Delphi, C^{++}... thì kỹ thuật RMI không cho phép.

Chương trình ứng dụng phân tán RMI cũng đượ tổ chức theo mô hình client/server:

- Phía server là phía máy tính từ xa chứa các đối tượng có phương thức cho phép gọi từ xa.

- Phía client là phía chứa các đối tượng phát sinh lời gọi phương thức từ xa.

Một chương trình Client có thể kích hoạt các phương thức ở xa trên một hay nhiều Server. Tức là sự thực thi của chương trình được trải rộng trên nhiều máy tính. Đây chính là đặc điểm của các ứng dụng phân tán. Nói cách khác, RMI là cơ chế để xây dựng các ứng dụng phân tán dưới ngôn ngữ Java.

Mỗi đối tượng có phương thức cho phép gọi từ xa, trước khi sử dụng được nó phải được đăng ký với máy ảo java thông qua bộ đăng ký của JDK hoặc do người sử dụng định nghĩa. Và mỗi đối tượng đó cũng phải được gán một chuỗi dùng làm tên để truy xuất tìm đối tượng trên mạng. Chuỗi tên đó có dạng như URL:

"rmi://<host>[:port]/ObjName"

Trong đó:

- rmi : chỉ phương thức truy cập
- host: địa chỉ của máy trạm chứa đối tượng từ xa cần tìm
- port: Chỉ ra số cổng được sử dụng để truy xuất tìm đối tượng, nó có thể có hoặc không. Trong trường hợp không khai báo thì nó mặc định lấy số cổng 1099.
- ObjName: Là chuỗi tên gán cho đối tượng có phương thức cho phép gọi từ xa.

RMI sử dụng giao thức truyền thông JRMI. Giao thức này cho phép tạo ra môi trường mạng truyền thông trong suốt mà từ đó lời gọi phương thức từ xa không khác gì lời gọi cục bộ. Và để truyền thông, java sử dụng 2 đối tượng trung gian để đóng gói truyền thông và khôi phục lại lời gọi, kết quả thi hành phương thức từ xa qua mạng từ các gói tin truyền qua mạng đó.. Đối tượng _Skel cài phía bên server và _Stub cài phía bên client.

Để hỗ trợ lập trình RMI, java hỗ trợ nhiều lớp và giao diện thư viện mà tập trung chủ yếu trong 2 gói: java.rmi và java.rmi.server.

Như vậy kỹ thuật lập trình phân tán đối tượng RMI trong java đã cho phép phân tán tải lên các máy tính trên mạng thay vì tập trung trên một máy. Điều này thật sự có ý nghĩa lớn đối với các ứng dụng mà có khối lượng tính toán lớn mà đòi hỏi thời gian thực. Vì một máy tính có mạnh đến mấy cũng vẫn hữu hạn. Nhất là đối với những bài toán thực tế như: Bài toán thị trường chứng khoán, ngân hàng, bài toán hàng không, dự bào thời tiết, bài toán nghiên cứu vũ trụ...Phần sau đây chúng ta sẽ đi sâu vào nghiên cứu các kỹ thuật lập trình của RMI và cơ chế hoạt động của chúng.

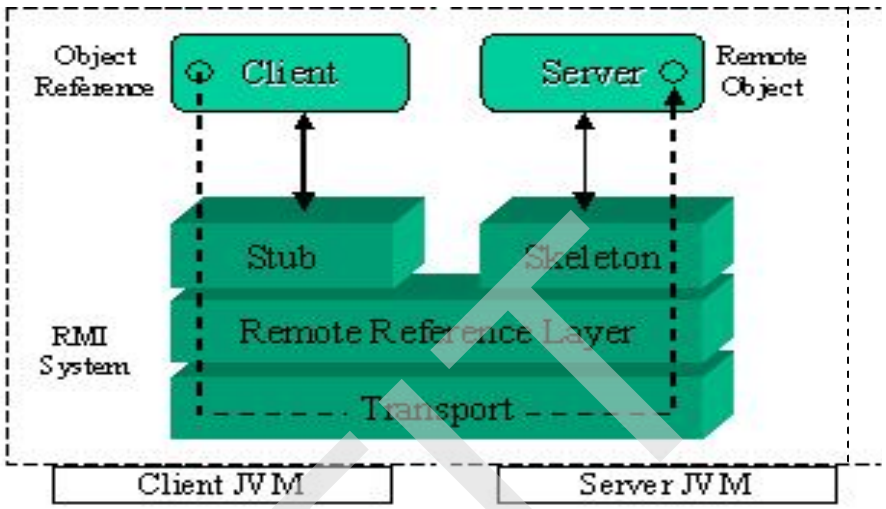
2.1.. Kiến trúc của chương trình Client – Server theo cơ chế RMI

Hình 5.1. là kiến trúc của một chương trình phân tán đối tượng RMI theo mô hình Client /Server:

Trong đó:

- Server là chương trình cung cấp các đối tượng có thể được gọi từ xa.

- Client là chương trình có tham chiếu đến các phương thức của các đối tượng có phương thức cho phép gọi từ xa trên Server.
- Stub là đối tượng môi giới trung gian phía client.
- Skeleton là đói tượng môi giới trung gian cài phía server.
- Remote Reference Layer là lớp tham chiều từ xa của RMI.



Hình5.1.. Kiến trúc Client/Server của chương trình RMI

2.2. Các cơ chế hoạt động RMI

Trong một ứng dụng phân tán cần có các cơ chế sau:

- Cơ chế định vị đối tượng ở xa
- Cơ chế giao tiếp với các đối tượng ở xa
- Tải các lớp dạng bytecodes cho các lớp mà nó được chuyển tải qua lại giữa JVM

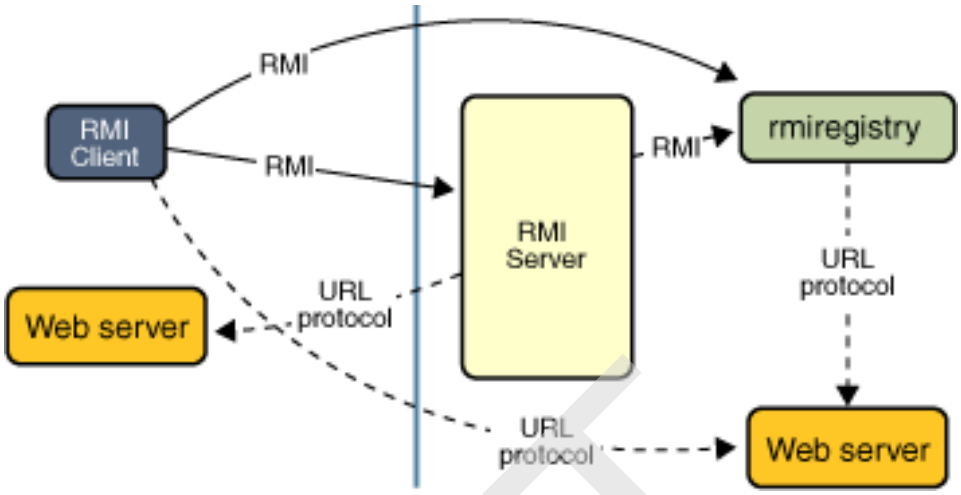
1.3.1.Cơ chế định vị đối tượng ở xa (Locate remote objects): Cơ chế này xác định cách thức mà chương trình Client có thể lấy được **tham chiếu** (Stub) đến các đối tượng ở xa. Thông thường người ta sử dụng một dịch vụ danh bạ (Naming Service) lưu giữ các tham chiếu đến các đối tượng cho phép gọi từ xa mà client sau đó có thể tìm kiếm.

1.3.2. Cơ chế giao tiếp với các đối tượng ở xa (Communicate with remote objects): cơ chế truyền thông với các đối tượng từ xa được cài đặt chi tiết bởi hệ thống RMI.

1.3.3. Tải các lớp dạng bytecodes cho các lớp mà thực hiện chuyển tải qua lại giữa Máy ảo (Load class bytecodes for objects that are passed around): Vì RMI cho phép các chương trình gọi phương thức từ xa trao đổi các đối tượng với các phương thức từ xa dưới dạng các tham số

hay giá trị trả về của phương thức, nên RMI cần có cơ chế cần thiết để tải mã Bytecodes của các đối tượng từ máy ảo này sang máy ảo khác.

Hình 5.2. mô tả một ứng dụng phân tán RMI sử dụng dịch vụ danh bạ để lấy các tham chiếu tới các đối tượng ở xa.



Hình 5.2. Vai trò của dịch vụ tên

Trong đó:

- Server đăng ký tên cho đối tượng có thể được gọi từ xa của mình với Dịch vụ danh bạ (Registry Server).
- Client tìm đối tượng ở xa thông qua tên đã được đăng ký trên Registry Server (looks up) và tiếp đó gọi các phương thức ở xa.

Hình 5.2. minh họa cũng cho thấy cách thức mà hệ thống RMI sử dụng một WebServer sẵn có để truyền tải mã bytecodes của các lớp qua lại giữa Client và Server

Tiến trình vận hành của một ứng dụng Client/Server theo kiểu RMI diễn ra như sau:

- Bước 1: Server tạo ra các đối tượng cho phép gọi từ xa cùng với các Stub và Skeleton của chúng.
- Bước 2: Server sử dụng lớp Naming để đăng ký tên cho một đối tượng từ xa (1).
- Bước 3: Naming đăng ký Stub của đối tượng từ xa với Registry Server (2).
- Bước 4: Registry Server sẵn sàng cung cấp tham thảo đến đối tượng từ xa khi có yêu cầu (3).
- Client yêu cầu Naming định vị đối tượng xa qua tên đã được đăng ký (phương thức lookup) với dịch vụ tên (4).
- Naming tải Stub của đối tượng xa từ dịch vụ tên mà đối tượng xa đã đăng ký về Client (5).
- Cài đặt đối tượng Stub và trả về tham khảo đối tượng xa cho Client (6).

- Client thực thi một lời gọi phương thức từ xa thông qua đối tượng Stub (7).

3. Các lớp hỗ trợ lập trình với RMI

Java hỗ trợ các lớp cần thiết để cài đặt các ứng dụng Client-Server theo kiểu RMI trong các gói: java.rmi. Trong số đó các lớp thường được dùng là:

- java.rmi.Naming:
- java.rmi.RMISecurityManager
- java.rmi.RemoteException;
- java.rmi.server.RemoteObject
- java.rmi.server.RemoteServer
- java.rmi. server.UnicastRemoteObject

II. XÂY DỰNG CHƯƠNG TRÌNH PHÂN TÁN RMI

Xây dụng một ứng dụng phân tán bằng cơ chế RMI gồm các bước sau:

1. Thiết kế và cài đặt các thành phần của ứng dụng.
2. Biên dịch các chương trình nguồn và tạo ra Stub và Skeleton.
3. Tạo các lớp có thể truy xuất từ mạng cần thiết.
4. Khởi tạo ứng dụng

1. Kỹ thuật lập trình RMI

Đầu tiên chúng ta phải xác định lớp nào là lớp cục bộ, lớp nào là lớp được gọi từ xa. Nó bao gồm các bước sau:

- *Định nghĩa các giao diện cho các phương thức ở xa (remote interfaces)*: Một remote interface mô tả các phương thức mà nó có thể được kích hoạt từ xa bởi các Client. Đi cùng với việc định nghĩa Remote Interface là việc xác định các lớp cục bộ làm tham số hay giá trị trả về của các phương thức được gọi từ xa.
- *Cài đặt các đối tượng từ xa (remote objects)*: Các Remote Object phải cài đặt cho một hoặc nhiều Remote Interfaces đã được định nghĩa. Các lớp của Remote Object class cài đặt cho các phương thức được gọi từ xa đã được khai báo trong Remote Interface và có thể định nghĩa và cài đặt cho cả các phương thức được sử dụng cục bộ. Nếu có các lớp làm đối số hay giá trị trả về cho các phương thức được gọi từ xa thì ta cũng định nghĩa và cài đặt chúng.
- *Cài đặt các chương trình Client*: Các chương trình Client có sử dụng các Remote Object có thể được cài đặt ở bất kỳ thời điểm nào sau khi các Remote Interface đã được định nghĩa.

Để nắm được kỹ thuật lập trình RMI cụ thể chúng ta xây dựng chương trình đơn giản sử dụng RMI như sau: Viết chương trình ứng dụng phân tán RMI theo mô hình client server. Chương trình có cấu trúc sau:

- Chương trình server: có đối tượng có phương thức cho phép gọi từ xa int add(int x,int y) để tính tổng của 2 số.
- Chương trình client: cho phép gọi phương thức thừ xa add() qua mạng để tính tổng của 2 số nguyên và hiển thị kết quả.

Quá trình xây dựng chương trình này có thể thực hiện qua các bước sau:

Bước 1: Khai báo giao diện để khai báo các phương thức cho phép gọi từ xa. Vì trong kỹ thuật RMI, bất kể phương thức nào cho phép gọi từ xa qua mạng đều phải được khai báo trong giao diện kế thừa từ giao diện Remote thuộc lớp java.rmi. Và phương thức đó phải đảm bảo 2 yêu cầu sau:

- Phải có thuộc tính public
- Phải ném trả về ngoại lệ RemoteException

Giả sử giao diện có tên TT, chúng ta có thể khai báo nó như sau:

```
//TT.java
import  java.rmi.*;
public interface   TT   extends     Remote
{
public int   add(int x,int  y) throws    RemoteException;
}
```

Bước 2: Khai báo lớp thực thi giao diện TT để cài dặt phương thức add(). Giả sử lớp có tên là TTImpl:

```
//TTImpl.java
import java.rmi.*;
class  TTImpl  implements   TT
{
public int add(int x,int y) throws   RemoteException
{
return  (x+y);
}
}
```

Bước 3: Xây dựng chương trình server. Chương trình server phải thực hiện 3 vấn đề cốt lõi sau đây:

- Tạo đối tượng có phương thức cho phép gọi từ xa và trả về tham chiếu đến giao diện của chúng. Ví dụ:

TT c=new TTImpl();

- Đăng ký đối tượng có phương thức cho phép gọi từ xa với máy ảo java, thông qua trình đăng ký của JDK hoặc do người lập trình tự định nghĩa, bằng cách sử dụng phương thức *exportObject()* của lớp UnicastRemoteObject thuộc gói java.rmi.server. Phương thưc *exportObject()* có thể khai báo như sau:

UnicastRemoteObject.exportObject(Obj);

- Gán cho đối tượng có phương thức cho phép gọi từ xa một tên dưới dạng chuỗi URL để thông qua chuỗi tên đó, các đối tượng client có thể truy xuất tìm thấy đối tượng trên mạng. Để thực hiện việc đó, lớp java.rmi.Naming cung cấp phương thức *bind()* hoặc *rebind()*. Phương thức *bind()* có dạng sau:

Naming.bind("rmi//<host>[:port]/ObjName", Obj);

Chương trình server được viết như sau:

```
//TTServer.java
import java.rmi.*;
import java.rmi.serrver.*;
class TTServer{
public static void main(String[] args)
{
try{
//Tao doi tuong
TT   c=new   TTImpl();
//dang ky voi mays ao java
UnicastRemoteObject.exportObject(c);
//Gan chuoi URL
Naming.bind("rmi://localhost/Obj", c);
System.out.println("Server RMI da san sang....");
}
catch(Exception   e)
{
System.out.println(e);
}
}}
```

Bước 4: Xây dựng chương trình client, giả sử chương trình là lớp TTClient.java.. Chương trình client phải có nhiệm vụ sau:

- Truy xuất tìm đối tượng có phương thức cho phép gọi từ xa thông qua chuỗi tên URL đa được chương trình server gán cho đối tượng. Bằng cách client sử dụng phương thức *lookup()* của lớp Naming hỏi bộ đăng ký thông qua số cổng cụ thể đã được định nghĩa trong chuỗi URL. Nếu tìm thấy, server sẽ trả về tham chiếu đến đối tượng từ xa có kiểu giao diện của đối tượng.
- Gọi thi hành phương thức từ xa thông qua biến tham chiếu tới đối tượng từ xa.

Chương trình client:

```
//TTClient.java
import java.rmi.*;
class TTClient{
public static void main(String[] args)
{
try{
TT   x=(TT)Naming.lookup("rmi://localhost/Obj");
int a=10, b=20;
System.out.println("Tong   cua      a="+a+"      voi    b="+b+"    la
s="+x.add(a,b));
}
catch(Exception e)
{
System.out.println(e);
}
}}
```

2. Biên dịch chương trình

Giai đoạn này gồm 2 bước:

Bước thứ nhất: Biên dịch các tệp chương trình thành dạng bytecode dùng trình javac.exe. Trong chương trình trên có 4 tệp:

javac.exe

TT.java ----------> *TT.class* (1)

TTImpl.java ----------> *TTImpl.class* (2)

TTServer.java ----------> *TTServer.class* (3)

TTClient.java ----------> *TTClient.class* (4)

Bước thứ 2: Phát sinh các tệp đối tượng trung gian _stub và _skel bằng cách sử dụng trình dịch rmic.exe của JDK để dịch tệp đối tượng có phương thức cho phép gọi từ xa TTImpl:

rmic TTImpl [Enter]

Sau khi dich, 2 tệp mới được tạo ra:

TTImpl_Stub.class (5)

TTImpl_Skel.class (6)

3. Thực thi chương trình ứng dụng

Bước 1: Phân bố các tệp chương trình phù hợp từ (1) đến (6) về máy client và server. Cụ thể:

- Phía client: (1), (4), (5)
- Phía server: (1), (2), (3), (5), (6)

Bước 2: Chạy chương trình

Thực hiện mở 3 của sổ lênh:

- Cửa sổ thứ nhất: Chạy trình đăng ký rmiregistry.exe với cú pháp sau:

  rmiregistry [porrt] [Enter]

- Của sổ thứ 2: Chạy chương trình server:

  java TTServer [Enter]

- Của sổ thứ 3: Chạy chương trình client:

  java TTClient [Enter]

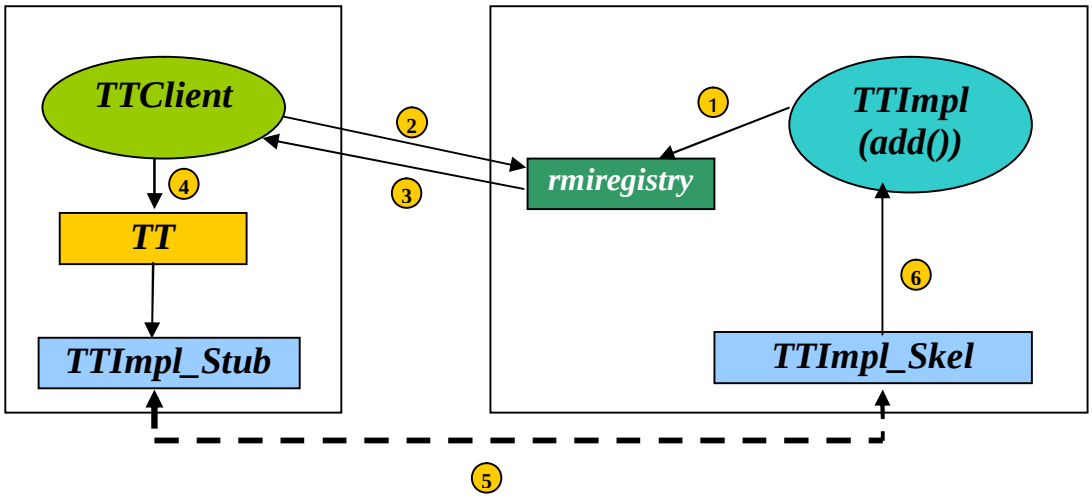
Sau khi thực hiện xong, sửa lại chương trình client phần địa chỉ host trong chuỗi URL, dịch lại và có thể chạy thử qua môi trường mạng. Khi đó của sổ 1, 2 chạy phia bên server, cửa sổ 3 chạy phía client.

III. CƠ CHẾ TRUYỀN THÔNG RMI

Cơ chế truyền thông RMI có thể giải thích theo hình 5.3 và nó thực hiện theo các bước sau:

Step 1: Đầu tiên đối tượng cài đặt các phương thức và gọi hàm *Naming.bind()* để đăng kỹ với bộ quản lý rmiregistry trên server thông qua quá trình 1.

Step 2: Đối tượng trên client gọi hàm Naming.lookup() để truy tìm đối tượng từ xa bằng cách hỏi bộ đăng ký thông qua chuỗi URL bằng quá trình 2.



Hình 5.3. Cơ chế RMI gọi phương thức của đối tượng từ xa

Step 3: Bộ đăng ký rmiregistry tìm đối tượng, nếu thấy nó trả về tham chiếu đến đối tượng từ xa cho client bằng quá trình 3 thông qua lớp giao diện(interface) mà đối tượng từ xa cung cấp.

Step 4: Dựa vào giao diện(TT) đối tượng TTClient sẽ gọi phương thức từ xa của đối tượng trên server(TTImpl) thông qua tham chiếu nhận được ở bước 3 bằng quá trình 4.

Step 5: Khi một phương thức được gọi, lời gọi sẽ được chuyển tới đối tượng trung gian _Stub và được đóng gói chuyển qua mạng theo giao thức JRMP tời đối tượng _Skel phía server.

Step 6: Đối tượng Skel phía server sẽ khôi phục lại lời gọi và gọi thi hành phương thức từ xa bằng quá trình 6.

Step 7: Sau khi phương thức tư xa thi hành xong, kết quả sẽ được đối tượng _Skel trả về cho đối tượng client bằng một quá trình truyền thông ngược với quá trình trên.

IV. VẤN ĐỀ TRUYỀN THAM SỐ CHO PHƯƠNG THỨC GỌI TỪ XA

1. Giới thiệu truyền tham số theo tham trị và tham chiếu cho phương thức từ xa

Trong kỹ thuật RMI, việc truyền tham số trong các lời gọi các phương thức từ xa qua mạng cũng có 2 cách: truyền tham trị và truyền tham chiếu. Với các tham số truyền có kiểu dữ liệu cơ bản thì đều là truyền tham trị. Còn đối với các đối tượng thì có 2 kiểu truyền: truyền đối tượng theo kiểu tham trị và truyền đối tượng theo kiểu tham chiếu. Trong cách truyền đối tượng theo kiểu tham trị thì bản thân đối tượng sẽ được truyền qua mạng. Cách truyền này có hạn chế là:

- Ảnh hưởng đến tốc độ của mạng nhất là khi truyền đối tượng có kích thước lớn.
- Chỉ truy xuất gọi phương thức từ xa theo một chiều từ client tới server.

Trong cách truyền đối tượng theo kiểu tham chiếu thì chỉ có tham chiếu đến đối tượng được truyền qua mạng nên khắc phục được hạn chế của truyền đối tượng theo kiểu tham trị. Và nó cho phép truy xuất gọi phương thức từ xa theo cả 2 chiều từ client đến server và ngược lại.

Để chỉ thị một đối tượng truyền cho phương thức từ xa qua mạng là truyền tham trị hay truyền tham chiếu thì trong Java sử dụng cách cài đặt như sau: Tất cả các đối tượng có kiểu lớp thực thi giao diện Serializable thì khi truyền cho phương thức đều được ấn định là truyền tham trị. Còn các đối tượng mà có kiểu lớp thực thi giao diện Remote thì khi truyền cho phương thức từ xa sẽ là truyền tham chiếu.

Sau đây chúng ta sẽ khảo sát kỹ 2 cách truyền đối tượng theo kiểu tham trị và kiểu tham chiếu.

2. Truyền đối tượng theo kiểu tham trị

Để nắm được kỹ thuật này, chúng ta xét ví dụ sau: Viết chương trình RMI có cấu trúc sau:

- Phía client cho phép tạo đối tượng BOX có các tham số w, h, d tương ứng là chiều rộng, chiều cao và chiều sâu của hình hộp chữ nhật. Sau đó gọi phương thức từ xa và truyền đối tượng BOX cho phương thức theo kiểu tham trị, nhận kết quả trả về và hiển thị.
- Phía server có đối tượng có phương thức cho phép gọi từ xa với tham số truyền là đối tượng BOX, thực hiện thay đổi w, h, d của đối tượng và trả đối tượng về cho client.

Quá trình xây dựng chương trình thực hiện các bước sau:

Bước 1: Xây dựng lớp BOX thực thi giao diện Serializable thuộc gói java.io để đối tượng có thể truyền theo kiểu tham trị.

```
//BOX.java
import java.io.*:
class  BOX  implements  Serializable
{
int   w,h,d;
BOX(){
w=10; h=20; d=15;
}}
```

Các bước 2 trở đi tương tự như kỹ thuật xây dựng chương trình RMI đã khảo sát ở phần trên.

Bước 2: Xây dựng giao diện để khai báo phương thức changeOBject() cho phép gọi từ xa

```
//BB.java
import java.rmi.*;
interface BB  extends   Remote
{
public  BOX   changeObject(BOX     obj) throws   RemoteException;
```

```
}
```

Bước 3: Khai báo lớp thực thi giao diện BB

```
//BBImpl.java
import java.rmi.*;
class  BBImpl   implements   BB
{
public  BOX  changeObject(BOX   obj) throws  RemoteException
{
obj.w+=10; obj.h+=5; obj.d+=15;
return obj;
}}
```

Bước 4: Xây dựng chương trình server

```
//BBServer.java
import java.rmi.*;
import java.rmi.server.*;
class BBServer{
public static boid main(String[] args)
{
try{
BB  c=new   BBImpl();
UnicastRemoteObject.exportObject(c);
Naming.bind("rmi://localhost/cObj",c);
}
catch(Exception  e)
{
System.out.println(e);
}}}
```

Bước 5: Xây dựng chương tình client

```
//BBClient.java
import  java.rmi.*;
class  BBClient{
public static void main(String[]  args)
{
try{
```

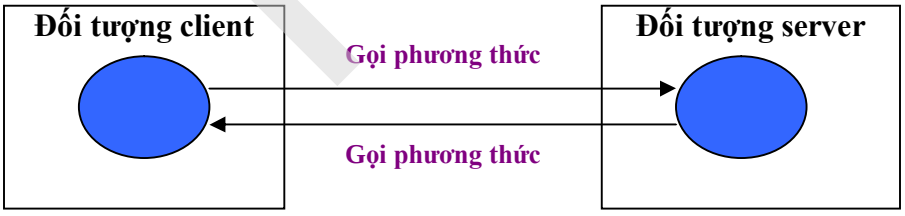
```
BB  c=(BB)Naming.lookup(“rmi://localhost/cObj”);
BOX  box=new  BOX();
System.out.println(“w=”+box.w+”,h=”+box.h+”,d=”+box.d);
box=c.changeObject(box);
System.out.println(“w=”+box.w+”,h=”+box.h+”,d=”+box.d);
}
catch(Exception e)
{
System.out.println(e);
}
}}
```

Sau khi tạo chương trình xong, thực hiện dịch và chạy chương trình theo kỹ thuật đã được trình bày ở phần trên. Để rõ hơn, các bạn có thể hiển thị kích cỡ của đối tượng BOX phía client, hiện thị kích cỡ đối tượng đó sau khi đã gọi phương thức từ xa phía server để so sánh.

3. Truyền đối tượng theo kiểu tham chiếu

Trong ký thuật này, đối tượng phía bên client và server đều có thể gọi phương thức từ xa của nhau và khác với kỹ thuật truền tham số trên là chỉ truyền tham chiếu đến đối tượng thay vì truyền bản thân đối tượng.

Nhưng có một vấn đề cần lưu ý trong kỹ thuật này là: Đối tượng phía client gọi phương thức từ xa của đối tượng phía server thì vẫn xẩy ra như đã trình bày. Nhưng khi đối tượng server gọi phương thức từ xa của client thí cơ chế có khác một chút. Cơ chế gọi ngược từ xa của đối tượng trên server đến đối tượng trên client thông quan tham chiếu gọi là cơ chế callback.



Hình 5.4. Đối tượng giữa client và server gọi phương thức của nhau

Để thể hiện kỹ thuật truyền đối tượng theo kiểu tham chiếu, chúng ta sẽ xét ví dụ sau[2]: Chúng ta sẽ tạo ra 2 đối tượng là AtClient chạy trên client và AtServer chạy trên server. Đầu tiên như kỹ thuật rmi bình thường, client sẽ hỏi trình rmiregistry để timg tham chiếu đến đối tượng AtServer, sau đó trình client sẽ tạo đối tượng AtClient phía client và gọi phương thức của AtServer để đăng ký đối tượng AtClient với server. Sau khi thực hiện những thao tác này, đối tượng AtClient và AtServer có thể tự do điều khiênt và gọi phương thức từ xa của nhau. Quá trình thực hiện này có thể thể hiện thông qua các bước sau:

Bước 1: Xây dựng giao diện phias client là AtClient.

```
//AtClient.java
import java.rmi.*;
public interface  AtClient  extends  Remote
{
public void callClientMethod(String msg) throws  RemoteException;
}
```

Bước 2: Xây dựng giao diện phía server là AtServer

```
//AtServer.java
import java.rmi.*;
public interface  AtServer  extends  Remote
{
public void registerClient(AtClient  c)throws  RemoteException;
public void callServerMethod(String  msg) throws RemoteException;
}
```

Bước 3: Cài đặt lớp thực thi cho các giao diện AtClient và AtServer

```
//AtClientImpl.java
import java.rmi.*;
class AtClientImpl  implements   AtClient
{
public void  callClientMethod(String msg) throws RemoteException
{
System.out.println(msg);
}}
//AtServerImpl.java
import java.rmi.*;
class AtServerImpl  implements   AtServer
{
AtClient client;
public void registerClient(AtClient c) throws RemoteException{
client=c;
}
public void  callServerMethod(String msg) throws RemoteException
{
```

```
System.out.println(msg);
for(int i=0;i<10;i++){
String msg="Server response  "+Math.random()*1000;
client.callClientMethod(msg);
}}}
```

Bước 4: Xây dựng chương trình server

```
//rServer.java
import java.rmi.*;
import java.rmi.server.*;
class  rServer{
public static void main(String[] args) throws Exception
{
AtServer  server=new  AtServerImpl();
UnicastRemoteObject.exportObject(server);
Naming.bind("rmi://localhost/serverObject",server);
System.out.println("Waiting for client request...");
}}
```

Bước 5: Xây dựng chương trình client

```
//rClient.java
import java.rmi.*;
import java.rmi.server.*;
class  rClient{
public static void main(String[] args) throws Exception
{
AtClient  cl=new  AtClientImpl();
UnicastRemoteObject.exportObject(cl);
Naming.bind("rmi://localhost/clObject", cl);
AtServer svr=(AtServer)Naming.lookup("rmi://localhost/serverObject");
svr.registerClient(cl);
svr.callServerMethod("Client contact server");
}}
```

Bước 6: Giả sử các tệp chương trình trên đều đặt trong thư mục D:\rmi, quá trình dịch và chạy chương trình có thể thực hiện bằng các câu lệnh sau:

- Dịch:

*D:\rmi\>javac *.java*

` *D:\rmi\>rmic AtServerImpl*

D:\rmi\>rmic AtClientImpl

- Mở của sổ lệnh chạy trình đăng ký

  D:\rmi\>rmiregistry

- Mở cửa sổ lệnh chạy trình rServer

  D:\rmi\>java rServer

- Mở cửa sổ lệnh chạy trình rClient

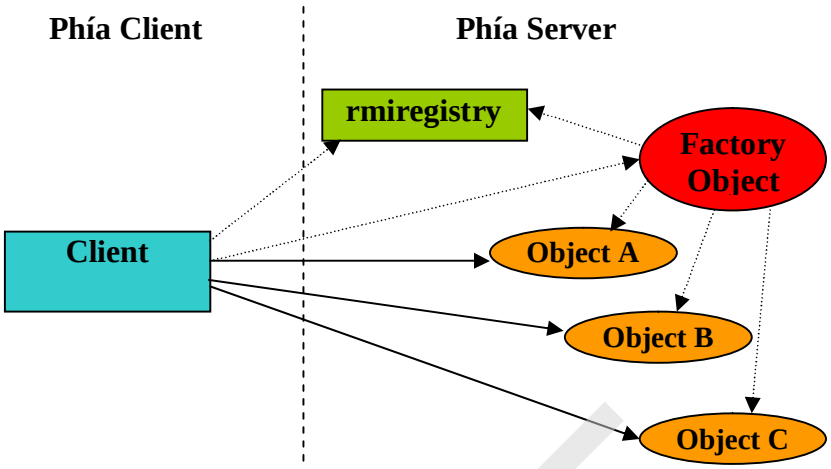
  D:\rmi\>java rClient

V. KỸ THUẬT SỬ DỤNG MỘT ĐỐI TƯỢNG SẢN SINH NHIỀU ĐỐI TƯỢNG

1. Giới thiệu

Chúng ta đã biết, một đối tượng có phương thức cho phép gọi từ xa truwcs khi sử dụng phải đăng ký đối tượng với máy ảo java thông qua trình rmiregistry, sau đó phải gán một chuỗi URL phía server và phía client phải có lệnh tìm kiếm thông qua URL đó. Đối với chương trình đơn giản, ít đối tượng thì không vẫn đề gì. Nhưng khi phía Server có hàng trăm, hàng nghìn và hơn nữa các đối tượng có phương thức cho phép gọi từ xa thi vẫn đề trở nên nghiêm trọng. Khi đó trình rmiregistry phải quản lý qua nhiều đối tượng, người lập trình chương trình client phải nhớ nhiều đối tượng, số câu lệnh đăng kỹ đối tượng, gán chuỗi URL phía server(bind()) và câu lệnh tìm kiếm phía client(lookup()) quá nhiều, tương tác qua mạng quá nhiều, từ đó chương trình trở nên phức tạp. Để giải quyết vấn đề này, rmi có một kỹ thuật cực kỳ hữu ích. Đó là, thay vì tạo truy xuất, nhớ nhiều đối tượng thì bầy giờ người lập trình chỉ cần nhớ một đối tượng, chỉ cần đăng ký với máy ảo java một đối tượng và chỉ cần truy tìm một đối tượng. Còn tất cả các đối tượng còn lại sẽ do đối tượng đại diện này tạo ra, đang ký và cho phép phía client chỉ cần gọi các phương thức từ xa của đối tượng đại diện để trả về tham chiếu đến các đối tượng do nó sản sinh ra. Đối tượng đó gọi là đối tượng sản sinh nhiều đối tượng(Factory Object). Hình 5.5. cho chung ta thấy cơ chế sử dụng một đối tượng(Factory Object) để sản sinh nhiều đối tượng khác(A, B, C). Cơ chế sử dụng đối tượng Factory để sản sinh nhiều đối tượng khác được thực hiện như sau:

- Đầu tiên đối tượng Factory Object đăng kỹ với bộ đăng ký rmiregistry
- Trình client muốn gọi đối tượng A, B, C; trước hết trình client phải liên hệ với rmiregistry để lấy về tham chiếu đến đối tượng Factory Object.
- Sau khi có tham chiếu đến đối tượng Factory Object, trình client thực hiện Factory Object để yêu cầu tạo ra các đối tượng A, B, C, đăng ký các đối tượng A, B, C với máy ảo java và trả về tham chiếu đến A, B, C cho client.
- Dựa vào tham chiếu nhận được, client thực hiện lời gọi phương thức từ xa của các đối tượng A, B, C.



Hình 5.5. Mô hình hoạt động của đối tượng Factory

2. Kỹ thuật cài đặt ứng dụng Factory

Kỹ thuật cài đặt đối tượng sản sinh ra nhiều đối tượng được thể hiện thông qua ví dụ sau: Giả sử mô hình bài toán như hình 5.5. Trong đó đối tượng Factory Object có các phương thức từ xa cho phép trả về tham chiếu đến đối tượng A, B, C là getRef_X() sau khi đối tượng này đã tạo A, B, C và đăng ký với rmiregistry. Sau đó client sẽ gọi các phương thức từ xa của các đối tượng A, B, C laf get_X(). Các phương thức này sẽ trả về cho client chuỗi "Day la doi tượng X". Với X là A, B hoặc C. Giả sử giao diện của đối tượng Factory Object là FF và lớp thực thi là FFImpl.

Bước 1: Xây dựng giao diện của A, B, C tương ứng là AA, BB, CC

```
//AA.java
import java.rmi.*;
public interface  AA  extends   Remote
{
public String   get_A()  throws  RemoteException;
}
//BB.java
import java.rmi.*;
public interface  BB  extends   Remote
{
public String   get_B()  throws  RemoteException;
}
//CC.java
```

```
import java.rmi.*;
public interface  CC  extends   Remote
{
public String   get_C()  throws  RemoteException;
}
```

Bước 2: Khai báo các lớp thực thi các giao diện AA, BB, CC

```
//AAImp.java
import java.rmi.*;
class  AAImpl  implements   AA{
public String   get_A()  throws  RemoteException{
return   “Day la doi tuong A.”;
}}
//BBImpl.java
import java.rmi.*;
class  BBImpl  implements   BB{
public String   get_B()  throws  RemoteException{
return   “Day la doi tuong B.”;
}}
//CCImpl.java
import java.rmi.*;
class  CCImpl  implements CC{
public String   get_C()  throws  RemoteException{
return   “Day la doi tuong C.”;
}}
```

Bước 3: Khai báo giao diện FF, trong giao diện này khai báo các phương thức getRef_X() để trả về các tham chiếu đến đối tượng A, B, C

```
//FF.java
import   java.rmi.*;
public interface   FF   extends    Remote{
public AA  getRef_A() throws   RemoteException;
public BB  getRef_B() throws   RemoteException;
public CC  getRef_C() throws   RemoteException;
}
```

Bước 5: Khai báo lớp FFImpl thực thi giao diện FF. Trong lớp này phải thực hiện các công việc sau:

- Tạo các đối tượng A, B, C tương ứng là AAImpl, BBImpl, CCImpl
- Đăng ký các đối tượng này với trình đăng ký rmiregistry
- Cài đặt các phương thức trả về tham chiếu đế các đối tượng đã được tạo ra

```
//FFImpl.java
import  java.rmi.*;
import  java.rmi.serrver.*;
class  FFImpl   implements   FF{
//Tao cac doi tuong
AA   a=new    AAImpl();
BB   b=new    BBImpl();
CC    c=new   CCImpl();
//Khai bao cau tu FFImpl, trong do thuc hien dang ky cac doi tuong
FFImpl()
{
try{
UnicastRemoteObject.exportObject(a);
UnicastRemoteObject.exportObject(b);
UnicastRemoteObject.exportObject(c);
}catch(Exception  e)
{}
}
//Cai dat cac phuong thu
public AA  getRef_A() throws  RemoteException{
return    a;
}
public BB  getRef_B() throws  RemoteException{
return    b;
}
public CC getRef_C() throws  RemoteException{
return    c;
}
}
```

Bước 6: Xây dựng chương trình phía server. Chương trình này phải thực hiện các nhiệm vụ sau(như kỹ thuật có bản):

- Tạo đối tượng FFImpl
- Đăng kỹ đối tượng FFImpl với máy ảo java
- Gán cho đối tượng một chuỗi URL để truy xuất đối tượng trên mạng

```
//FFServer.java
import  java.rmi.*;
import  java.rmi.server.*;
class  FFServer{
public static void main(String[] args) throws  Exception
{
FF    f=new    FFImpl();
UnicastRemoteObject.exportObject(f);
Naming.bind("rmi://localhost/ffObj",f);
System.out.println("Server da san sang...");
}}
```

Bước 7: Xây dựng chương trình client. Chương trình này có nhiệm vụ sau:

- Tìm đối tượng FFImpl và nhận tham chiếu đến đối tượng FFImpl
- Gọi các phương thức của FFImpl để lấy tham chiếu đến các đối tượng A, B, C
- Thông qua tham chiếu đến A, B, C gọi từ xa các phương thức của các đối tượng này.

```
//FFClient.java
import   java.rmi.*;
class   FFClient{
public static void main(String[] args ) throws Exception
{
//Lay tham chieu doi tuong Factorry
FF   f=(FF)Naming.lookup("rmi://localhost/ffObj");
//Goi phuong thuc cua Factory tra ve tham chieu toi A, B, C
AA  a=f.getRef_A();
BB  b=f.getRef_B();
CC  c=f.getRef_C();
//Goi cac phuong thuc cuar A, B, C thong qua tham chieu
System.out.println(a.get_A());
System.out.println(b.get_B());
System.out.println(c.get_C());
```

```
}}
```

Bước 8: Dịch và chạy chương trình

//Giả sử tất cả các tệp nguồn nằm trong thư mục d:\RMI\Factory

*D:\RMI\Factory>javac *.java*

//Phat sinh các tệp _Stub, _Skel của FFImp, AAImpl, BBImpl, CCImpl

D:\RMI\Factory>rmic AAImpl

D:\RMI\Factory>rmic BBImpl

D:\RMI\Factory>rmic CCImpl

D:\RMI\Factory>rmic FFImpl

//Chạy chương trình

//b1: Mở cửa sổ lệnh chạy trình đăng ký

D:\RMI\Factory>rmiregistry

//b2: Mở cửa sổ lệnh chạy chương trình server

D:\RMI\Factory>java FFServer

//b3: Mở cửa sổ lệnh chạy chương trình client

D:\RMI\Factory>java FFClient

Lưu ý: Nếu chạy chương trình server và client trên 2 máy tính nối mạng thì phải phân bố các tệp server và các tệp phía client tương tự như kỹ thuật có bản nhưng nhiều đối tượng. Và phải đổi địa chỉ trong chuỗi URL phía chương trình client là địa chỉ của server(trong lookup()).

VI. CASE STUDY 1: LOGIN TỪ XA DÙNG RMI

1. Bài toán

Bài toán login từ xa dùng RMI đặt ra như sau:

- Cở sở dữ liệu đọc lưu trữ và quản lí trên server RMI, trong đó có bảng users chứa ít nhất hai cột: cột username và cột password.
- Tại phía server, có khai báo, định nghĩa, và đăng kí một đối tượng từ xa có phương th kiểm tra đăng nhập, nó sẽ tiến hành kiểm tra trong cơ sở dữ liệu xem có tài khoản nào trùng với thông tin đăng nhập nhận được hay không.
- Chương trình phía client phải hiện giao diện đồ họa, trong đó có một ô text để nhập username, một ô text để nhập password, và một nút nhấn Login.
- Khi nút Login được click, chương trình client sẽ triệu gọi làm kiểm tra login từ server RMI, lấy thông tin đăng nhập (username/password) trên form giao diện để kiểm tra

- Sau khi có kết quả kiểm tra (đăng nhập đúng, hoặc sai), client sẽ hiển thị thông báo tương ứng với kết quả nhận được: nếu đăng nhập đúng thì thông báo login thành công. Nếu đăng nhập sai thì thông báo là username/password không đúng.
- Yêu cầu kiến trúc hệ thống ở cả hai phía client và server RMI đều được thiết kế theo mô hình MVC

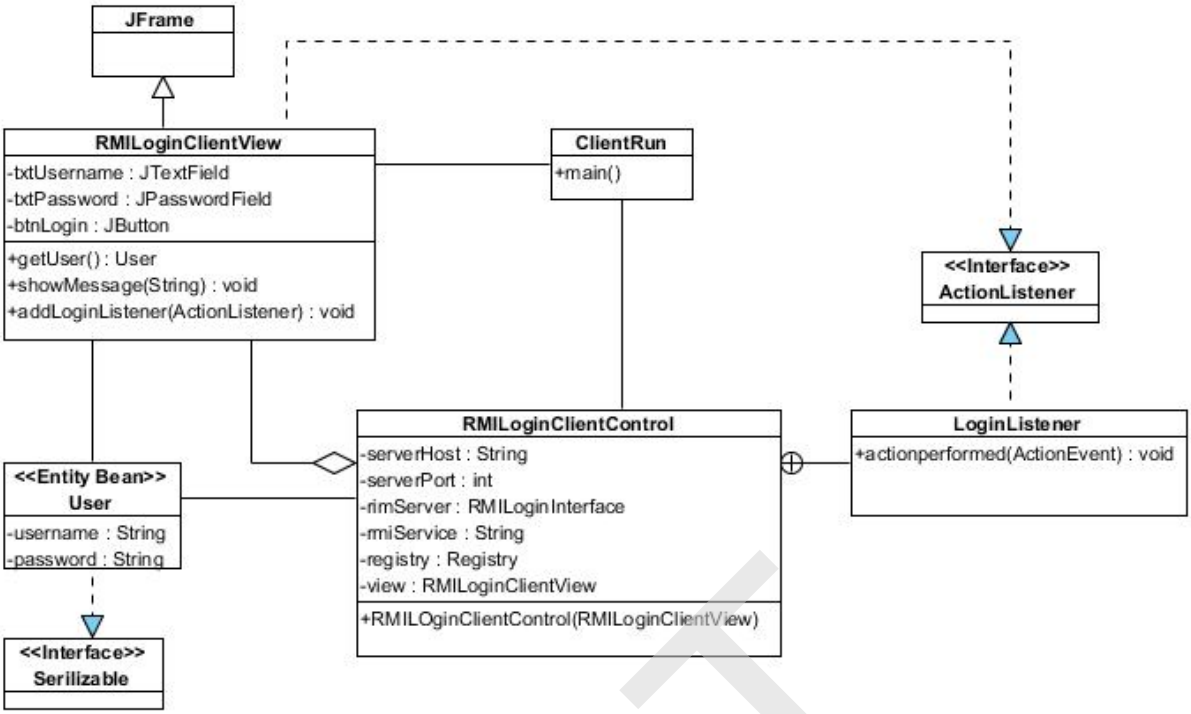
2. Thiết kế hệ thống

Vì hệ thống được thiết kế theo mô hình client/server RMI nên mỗi phía client, server sẽ có một sơ đồ lớp riêng, các sơ đồ này được thiết kế theo mô hình MVC.

2.1 Sơ đồ lớp phía client

Sơ đồ lớp của phía client được thiết kế theo mô hình MVC trong Hình 5.6, bao gồm 3 lớp chính tương ứng với sơ đồ M-V-C như sau:

- Lớp User: là lớp tương ứng với thành phần model (M), bao gồm hai thuộc tính username và password, các hàm khởi tạo và các cặp getter/setter tương ứng với các thuộc tính.
- Lớp RMILoginClientView: là lớp tương ứng với thành phần view (V), là lớp form nên phải kế thừa từ lớp JFrame của Java, nó chứa các thuộc tính là các thành phần đồ họa bao gồm ô text nhập username, ô text nhập password, nút nhất Login.
- Lớp RMILoginClientControl: là lớp tương ứng với thành phần control (C), nó chứa một lớp nội tại là LoginListener. Khi nút Login trên tầng view bị click thì nó sẽ chuyển tiếp sự kiện xuống lớp nội tại này để xử lí. Tất cả các xử lí đều gọi từ trong phương thức actionPerformed của lớp nội tại này, bao gồm: lấy thông tin trên form giao diện, triệu gọi thủ tục từ xa RMI về kiểm tra đăng nhập và yêu cầu form giao diện hiển thị.

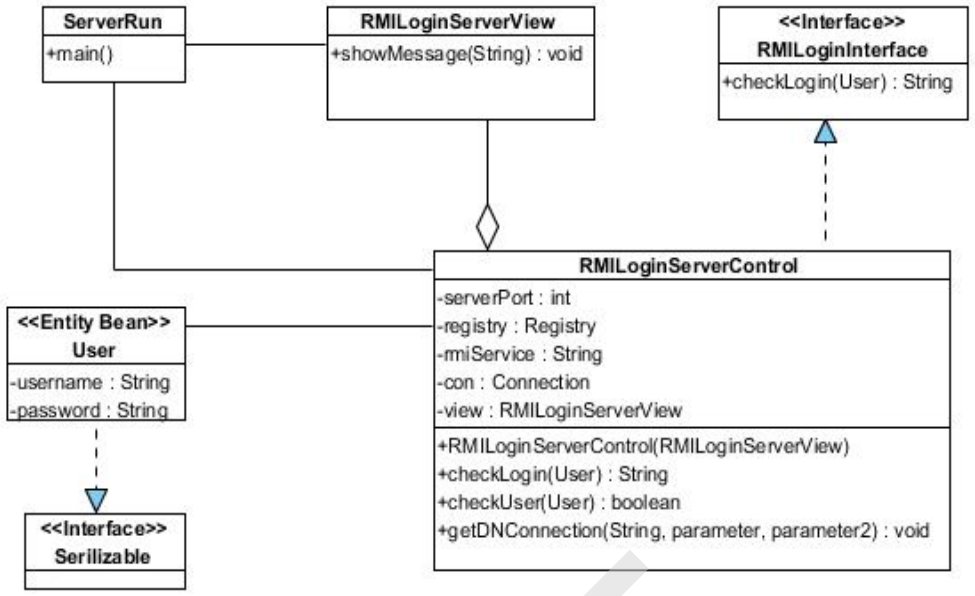


Hình 5.6: Sơ đồ lớp phía client RMI

2.2 Sơ đồ lớp phía server

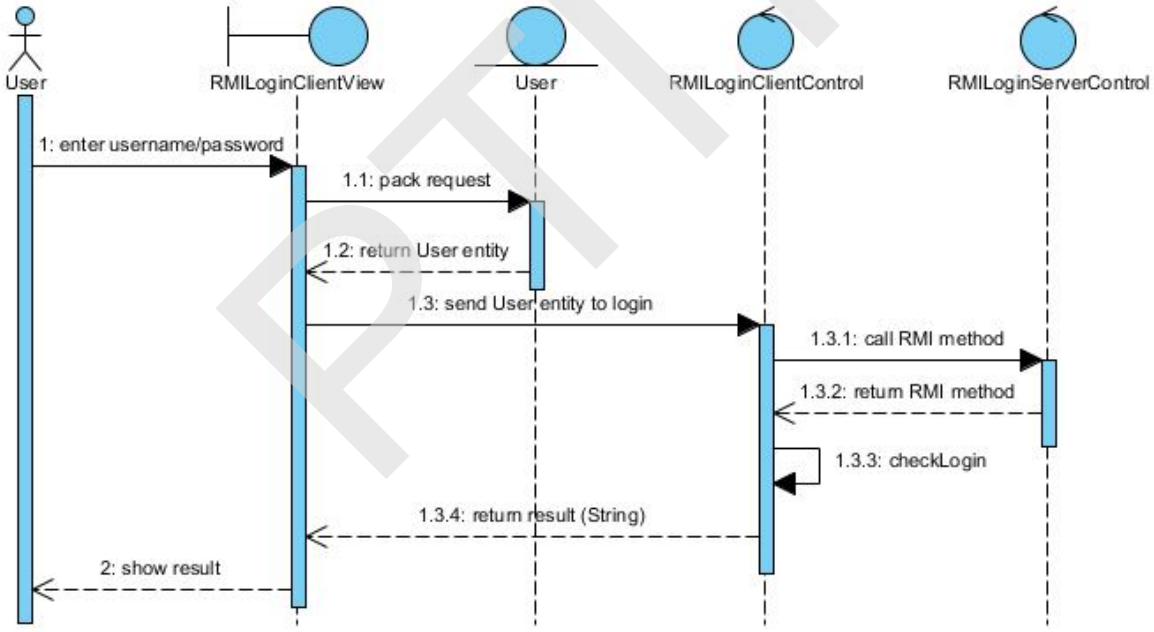
Sơ đồ lớp của phía server được thiết kế theo mô hình MVC trong Hình 5.7, bao gồm 3 lớp chính tương ứng với sơ đồ M-V-C như sau:

- Lớp User: là lớp thực thể, dùng chung thống nhất với lớp phía bên client.
- Lớp RMILoginServerView: là lớp tương ứng với thành phần view (V), là lớp dùng hiển thị các thông báo và trạng thái hoạt động bên server RMI.
- Giao diện RMILoginInterface: là giao diện (interface) khai báo đối tượng từ xa, trong đó nó khai báo thủ tục checkLogin(): thủ tục nhận vào một tham số kiểu User, trả kết quả về dạng String.
- Lớp RMILoginServerControl: là lớp tương ứng với thành phần control (C), nó đảm nhiệm vai trò xử lí của server RMI, trong đó nó định nghĩa cụ thể lại phương thức đã được khai báo trong RMILoginInterface, sau đó đăng kí bản thân nó vào server RMI để phục vụ các lời triệu gọi từ phía các client.



Hình 5.7: Sơ đồ lớp phía server RMI

2.3 Tuần tự các bước thực hiện



Hình 5.8: Tuần tự các bước thực hiện khi login từ xa với RMI

Tuần tự các bước xử lí như sau (Hình 5.8):

1. Ở phía client, người dùng nhập username/password và click vào giao diện của lớp RMILoginClientView
2. Lớp RMILoginClientView sẽ đóng gói thông tin username/password trên form vào một đối tượng model User bằng phương thức getUser() và chuyển xuống cho lớp RMILoginClientControl xử lí

3. Lớp RMILoginClientControl sẽ triệu gọi làm checkLogin() từ phía server RMI
4. Server trả về cho bên client một skeleton của phương thức checkLogin().
5. Bên phía client, khi nhận được skeleton, nó gọi phương thức checkLogin() để kểm tra thông tin đăng nhập.
6. Kết quả kiểm tra sẽ được lớp RMILoginClientControl sẽ chuyển cho lớp RMILoginClientView hiển thị bằng phương thức showMessage()
7. Lớp RMILoginClientView hiển thị kết quả đăng nhập lên cho người dùng

3. Cài đặt

3.1 Các lớp phía client RMI

Lớp User.java

```
package rmi.client;
import java.io.Serializable;

public class User implements Serializable{
	private String userName;
	private String password;

	public User(){
	}

	public User(String username, String password){
		this.userName = username;
		this.password = password;
	}

	public String getPassword() {
		return password;
	}

	public void setPassword(String password) {
		this.password = password;
	}

	public String getUserName() {
		return userName;
	}

	public void setUserName(String userName) {
		this.userName = userName;
	}
}
```

Lớp RMILoginClientView.java

```
package rmi.client;
import java.awt.FlowLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
```

```java
import java.awt.event.WindowEvent;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JPasswordField;
import javax.swing.JTextField;

public class RMILoginClientView extends JFrame implements ActionListener{
	private JTextField txtUsername;
	private JPasswordField txtPassword;
	private JButton btnLogin;

	public RMILoginClientView(){
		super("RMI Login MVC");

		txtUsername = new JTextField(15);
		txtPassword = new JPasswordField(15);
		txtPassword.setEchoChar('*');
		btnLogin = new JButton("Login");

		JPanel content = new JPanel();
		content.setLayout(new FlowLayout());
		content.add(new JLabel("Username:"));
		content.add(txtUsername);
		content.add(new JLabel("Password:"));
		content.add(txtPassword);
		content.add(btnLogin);

		this.setContentPane(content);
		this.pack();

		this.addWindowListener(new WindowAdapter(){
			public void windowClosing(WindowEvent e){
				System.exit(0);
			}
		});
	}

	public void actionPerformed(ActionEvent e) {
	}

	public User getUser(){
		User model = new User(txtUsername.getText(), txtPassword.getText());
		return model;
	}

	public void showMessage(String msg){
		JOptionPane.showMessageDialog(this, msg);
	}

	public void addLoginListener(ActionListener log) {
		btnLogin.addActionListener(log);
		}
}
```

Lớp RMILoginClientControl.java

```
package rmi.client;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.rmi.NotBoundException;
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import rmi.server.RMILoginInterface;

public class RMILoginClientControl {
	private RMILoginClientView view;
	private String serverHost = "localhost";
	private int serverPort = 3232;
	private RMILoginInterface rmiServer;
	private Registry registry;
	private String rmiService = "rmiLoginServer";

	public RMILoginClientControl(RMILoginClientView view){
		this.view = view;
		view.addLoginListener(new LoginListener());

		try{
			// lay the dang ki
			registry = LocateRegistry.getRegistry(serverHost, serverPort);
			// tim kiem RMI server
			rmiServer =  (RMILoginInterface)(registry.lookup(rmiService));
		}catch(RemoteException e){
			view.showMessage(e.getStackTrace().toString());
			e.printStackTrace();
		}catch(NotBoundException e){
			view.showMessage(e.getStackTrace().toString());
			e.printStackTrace();
		}
	}

	class LoginListener implements ActionListener {
	  public void actionPerformed(ActionEvent e) {
	      try {
	          User model = view.getUser();
	          if(rmiServer.checkLogin(model).equals("ok")){
	              view.showMessage("Login succesfully!");
	          }else{
	              view.showMessage("Invalid username and/or password!");
	          }
	      } catch (Exception ex) {
	          view.showMessage(ex.getStackTrace().toString());
	          ex.printStackTrace();
	      }
	  }
    }
}
```

Lớp ClientRun.java

```
package rmi.client;

public class ClientRun {
	public static void main(String[] args) {
		RMILoginClientView view       = new RMILoginClientView();
		RMILoginClientControl control = new RMILoginClientControl(view);
		view.setVisible(true);
	}

}
```

3.2 Các lớp phía server RMI

Lớp RMILoginServerView.java

```
package rmi.server;

public class RMILoginServerView {
	public RMILoginServerView(){
	}

	public void showMessage(String msg){
		System.out.println(msg);
	}

}
```

Interface RMILoginInterface.java

```
package rmi.server;
import java.rmi.Remote;
import java.rmi.RemoteException;
import rmi.client.User;

public interface RMILoginInterface extends Remote{
	public String checkLogin(User user) throws RemoteException;
}
```

Lớp RMILoginServerControl.java

```
package rmi.server;
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.rmi.server.UnicastRemoteObject;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
import rmi.client.User;

public class RMILoginServerControl extends UnicastRemoteObject implements
RMILoginInterface{
```

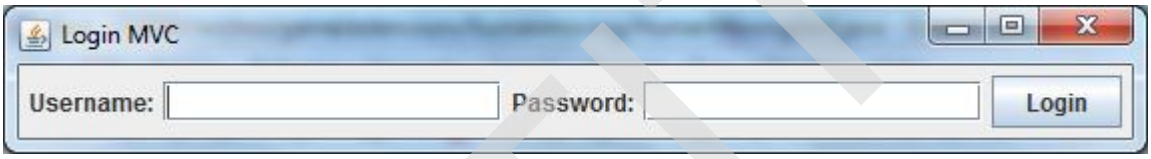
```
private int serverPort = 3232;
private Registry registry;
private Connection con;
private RMILoginServerView view;
private String rmiService = "rmiLoginServer";

public RMILoginServerControl(RMILoginServerView view) throws RemoteException{
	this.view = view;
	getDBConnection("usermanagement", "root", "12345678");
	view.showMessage("RMI server is running...");

	// dang ki RMI server
	try{
		registry = LocateRegistry.createRegistry(serverPort);
		registry.rebind(rmiService, this);
	}catch(RemoteException e){
		throw e;
	}
}

public String checkLogin(User user) throws RemoteException{
	String result = "";
	if(checkUser(user))
		result = "ok";
	return result;
}

private void getDBConnection(String dbName,
                             String username, String password){
    String dbUrl = "jdbc:mysql://localhost:3306/" + dbName;
    String dbClass = "com.mysql.jdbc.Driver";

    try {
      Class.forName(dbClass);
      con = DriverManager.getConnection (dbUrl, username, password);
    }catch(Exception e) {
      view.showMessage(e.getStackTrace().toString());
    }
}

private boolean checkUser(User user) {
    String query = "Select * FROM users WHERE username ='"
             + user.getUserName()
             + "' AND password ='" + user.getPassword() + "'";

    try {
      Statement stmt = con.createStatement();
      ResultSet rs = stmt.executeQuery(query);

      if (rs.next()) {
            return true;
      }
    }catch(Exception e) {
      view.showMessage(e.getStackTrace().toString());
    }
    return false;
  }
```

```
}
```

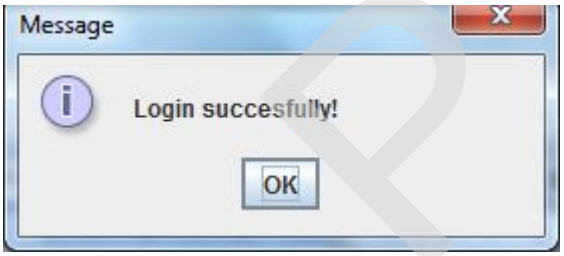
Lớp ServerRun.java

```
package rmi.server;

public class ServerRun {
	public static void main(String[] args) {
		RMILoginServerView view	= new RMILoginServerView();
		try{
			RMILoginServerControl
				control = new RMILoginServerControl(view);
		}catch(Exception e){
			e.printStackTrace();
		}
	}
}
```

4. Kết quả



Login thành công:



Login lỗi:

VII. CASE STUDY 2: KẾT HỢP RMI VÀ TCP/IP

1. Bài toán

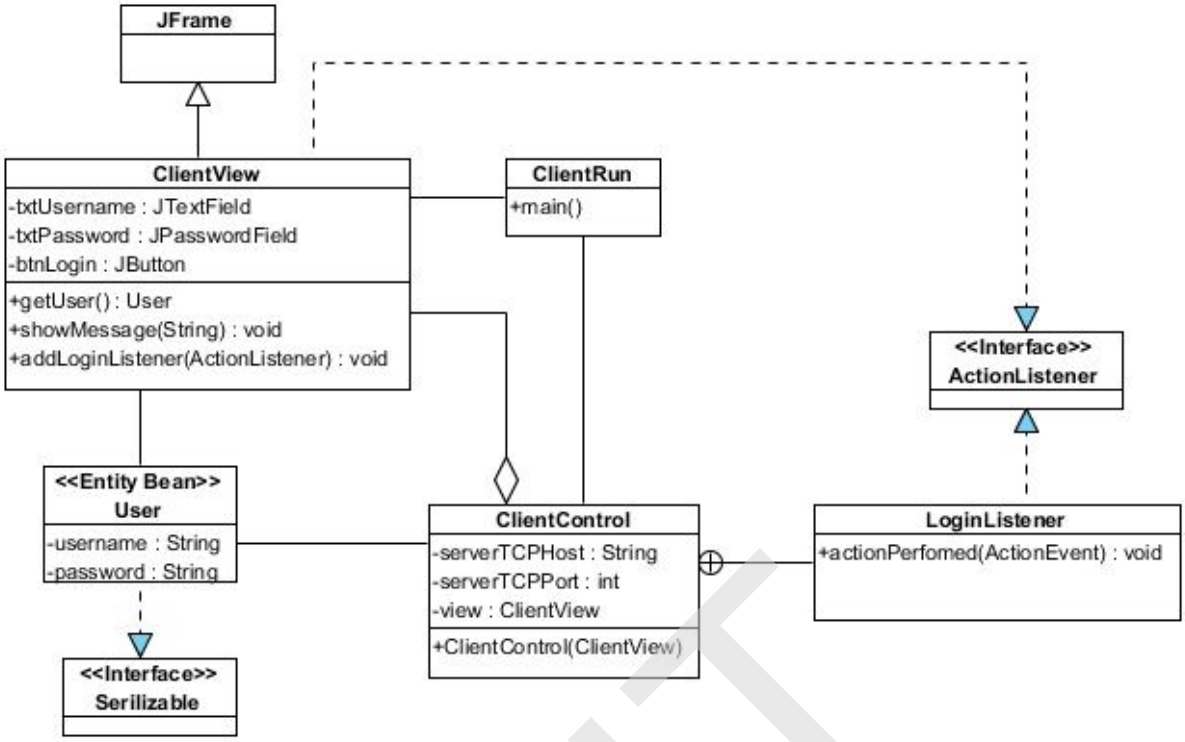
Bài toán login từ xa dùng kết hợp TCP/IP và RMI đặt ra như sau:

- Cở sở dữ liệu đợc lưu trữ và quản lí trên server RMI, trong đó có bảng users chứa ít nhất hai cột: cột username và cột password.
- Tại phía server RMI, có khai báo, định nghĩa, và đăng kí một đối tượng từ xa có phương th kiểm tra đăng nhập, nó sẽ tiến hành kiểm tra trong cơ sở dữ liệu xem có tài khoản nào trùng với thông tin đăng nhập nhận được hay không.
- Chương trình phía client TCP phải hiện giao diện đồ họa, trong đó có một ô text để nhập username, một ô text để nhập password, và một nút nhấn Login.
- Khi nút Login được click, chương trình client sẽ lấy thông tin đăng nhập (username/password) trên form giao diện để gửi snag server TCP kiểm tra
- Tại phía server TCP, khi nhận được yêu cầu kiểm tra đanưg nhập (kèm theo username/password), nó sẽ triệu gọi hàm kiểm tra đăng nhập từ xa của RMI
- Sau khi có kết quả kiểm tra (đăng nhập đúng, hoặc sai), server TCP sẽ gửi lại cho client TCP.
- Client TCP sẽ hiển thị thông báo tương ứng với kết quả nhận được: nếu đăng nhập đúng thì thông báo login thành công. Nếu đăng nhập sai thì thông báo là username/password không đúng.
- Yêu cầu kiến trúc hệ thống ở cả hai phía client TCP, server TCP và server RMI đều được thiết kế theo mô hình MVC

2. Thiết kế hệ thống

Hệ thống sẽ bao gồm ba phía: client TCP, server TCP, và server RMI. Mỗi phía đều được thiết kế theo mô hình MVC.

2.1 Sơ đồ lớp phía client TCP

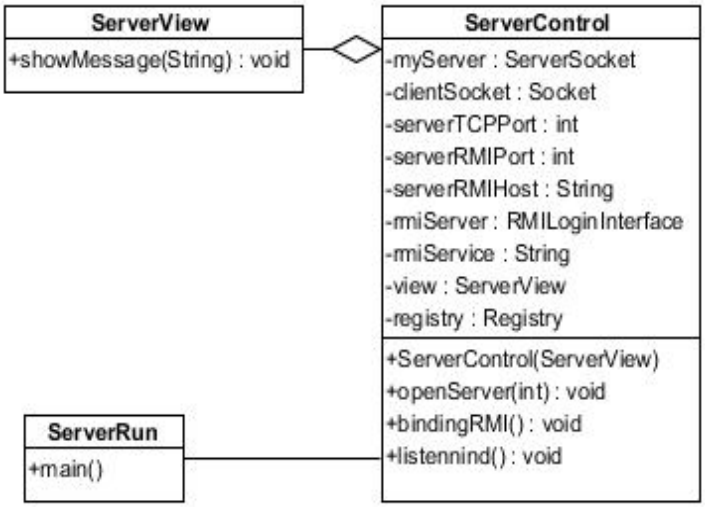


Hình 5.9 : Sơ đồ lớp phía client TCP

Sơ đồ lớp của phía client được thiết kế theo mô hình MVC trong Hình 5.9, bao gồm 3 lớp chính tương ứng với sơ đồ M-V-C như sau:

- Lớp User: là lớp tương ứng với thành phần model (M), bao gồm hai thuộc tính username và password, các hàm khởi tạo và các cặp getter/setter tương ứng với các thuộc tính.
- Lớp ClientView: là lớp tương ứng với thành phần view (V), là lớp form nên phải kế thừa từ lớp JFrame của Java, nó chứa các thuộc tính là các thành phần đồ họa bao gồm ô text nhập username, ô text nhập password, nút nhất Login.
- Lớp ClientControl: là lớp tương ứng với thành phần control (C), nó chứa một lớp nội tại là LoginListener. Khi nút Login trên tầng view bị click thì nó sẽ chuyển tiếp sự kiện xuống lớp nội tại này để xử lí. Tất cả các xử lí đều gọi từ trong phương thức actionPerformed của lớp nội tại này, bao gồm: lấy thông tin trên form giao diện, gửi sang server TCP theo giao thức TCP/IP, nhận kết quả về và yêu cầu form giao diện hiển thị.

2.2 Sơ đồ lớp phía server TCP



Hình 5.10: Sơ đồ lớp phía server TCP

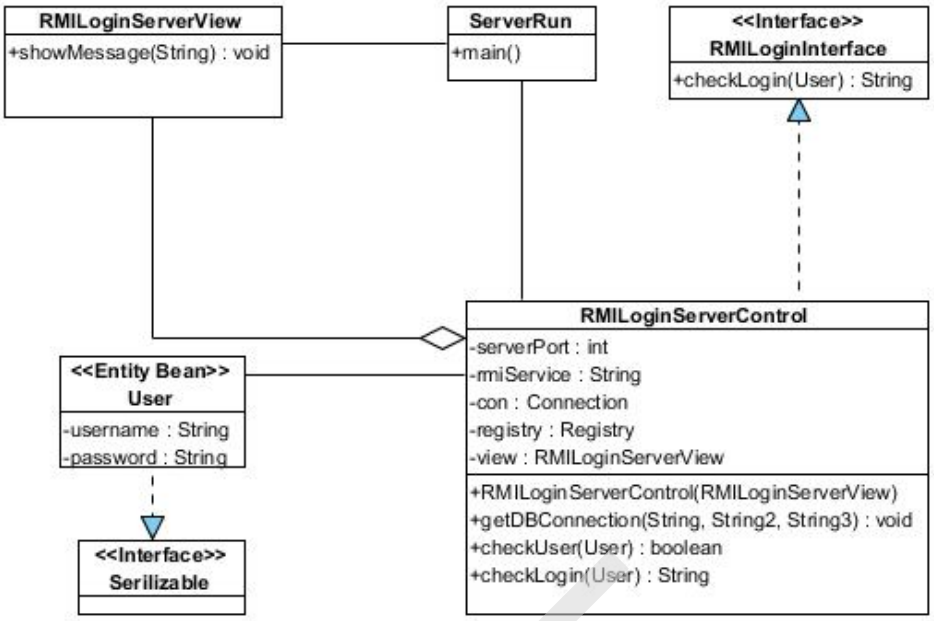
Sơ đồ lớp của phía server được thiết kế theo mô hình MVC trong Hình 5.10, bao gồm 3 lớp chính tương ứng với sơ đồ M-V-C như sau:

- Lớp User: là lớp thực thể, dùng chung thống nhất với lớp phía bên client.
- Lớp ServerView: là lớp tương ứng với thành phần view (V), là lớp dùng hiển thị các thông báo và trạng thái hoạt động bên server TCP.
- Lớp ServerControl: là lớp tương ứng với thành phần control (C), nó đảm nhiệm vai trò xử lí của server TCP: nhận thông tin đăng nhập từ các client TCP, triệu gọi thủ tục từ xa của server RMI về chạy để kiểm tra đăng nhập, sau đó gửi kết quả kiểm tra đăng nhập về lại phía client TCP.

2.3 Sơ đồ lớp phía server RMI

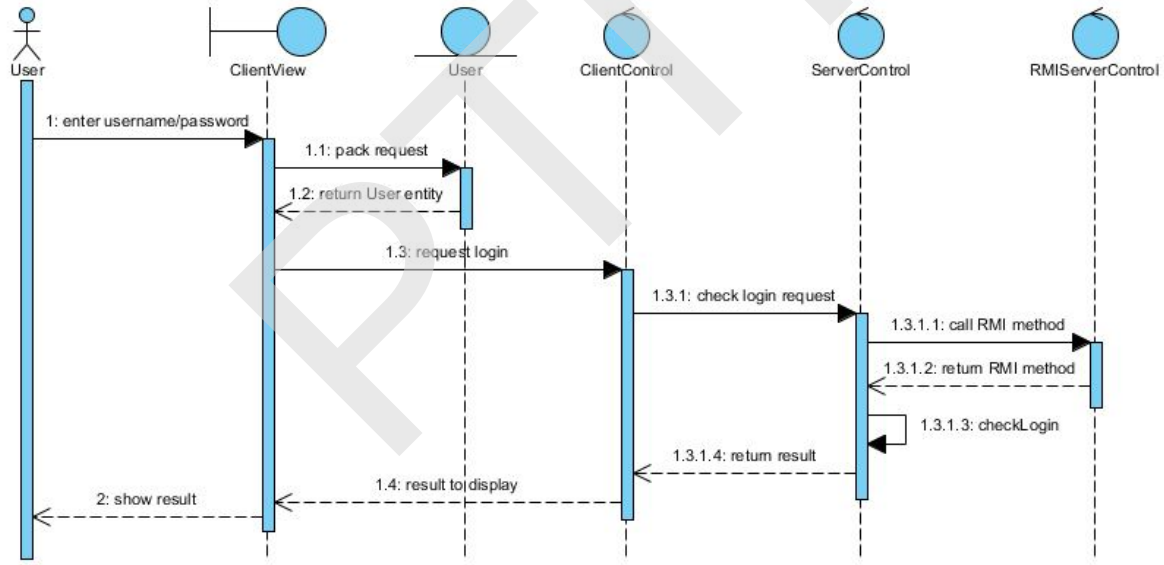
Sơ đồ lớp của phía server được thiết kế theo mô hình MVC trong Hình 5.11, bao gồm 3 lớp chính tương ứng với sơ đồ M-V-C như sau:

- Lớp User: là lớp thực thể, dùng chung thống nhất với lớp phía bên client.
- Lớp RMILoginServerView: là lớp tương ứng với thành phần view (V), là lớp dùng hiển thị các thông báo và trạng thái hoạt động bên server RMI.
- Giao diện RMILoginInterface: là giao diện (interface) khai báo đối tượng từ xa, trong đó nó khai báo thủ tục checkLogin(): thủ tục nhận vào một tham số kiểu User, trả kết quả về dạng String.
- Lớp RMILoginServerControl: là lớp tướng ứng với thành phần control (C), nó đảm nhiệm vai trò xử lí của server RMI, trong đó nó định nghĩa cụ thể lại phương thức đã được khai báo trong RMILoginInterface, sau đó đăng kí bản thân nó vào server RMI để phục vụ các lời triệu gọi từ phía các server TCP.



Hình 5.11 : Sơ đồ lớp phía server RMI

2.4 Tuần tự các bước thực hiện



Hình 5.12 : Tuần tự các bước thực hiện khi đăng nhập kết hợp TCP/IP - RMI

Tuần tự các bước xử lí như sau (Hình 5.12):

1. Ở phía client, người dùng nhập username/password và click vào giao diện của lớp ClientView
2. Lớp ClientView sẽ đóng gói thông tin username/password trên form vào một đối tượng model User bằng phương thức getUser() và chuyển xuống cho lớp ClientControl xử lí
3. Lớp ClientControl gửi thông tin đăng nhập sang server TCP

4. Lớp ServerControl triệu gọi làm checkLogin() từ phía server RMI ngay khi nhận được yêu cầu từ phía client TCP
5. Server RMI trả về cho bên server TCP một skeleton của phương thức checkLogin().
6. Bên phía server TCP, khi nhận được skeleton, nó gọi phương thức checkLogin() để kểm tra thông tin đăng nhập.
7. Kết quả kiểm tra sẽ được lớp ServerControl trả về cho lớp Client Control.
8. Lớp ClientControl sẽ chuyển cho lớp ClientView hiển thị bằng phương thức showMessage()
9. Lớp ClientView hiển thị kết quả đăng nhập lên cho người dùng

3. Cài đặt

3.1 Các lớp phía client TCP

Lớp User.java

```
package rmi_tcp.tcpClient;
import java.io.Serializable;

public class User implements Serializable{
	private String userName;
	private String password;

	public User(){
	}

	public User(String username, String password){
		this.userName = username;
		this.password = password;
	}

	public String getPassword() {
		return password;
	}

	public void setPassword(String password) {
		this.password = password;
	}

	public String getUserName() {
		return userName;
	}

	public void setUserName(String userName) {
		this.userName = userName;
	}
}
```

Lớp ClientView.java

```
package rmi_tcp.tcpClient;
```

```java
import java.awt.FlowLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JPasswordField;
import javax.swing.JTextField;

public class ClientView extends JFrame implements ActionListener{
	private JTextField txtUsername;
	private JPasswordField txtPassword;
	private JButton btnLogin;

	public ClientView(){
		super("TCP-RMI Login MVC");

		txtUsername = new JTextField(15);
		txtPassword = new JPasswordField(15);
		txtPassword.setEchoChar('*');
		btnLogin = new JButton("Login");

		JPanel content = new JPanel();
		content.setLayout(new FlowLayout());
		content.add(new JLabel("Username:"));
		content.add(txtUsername);
		content.add(new JLabel("Password:"));
		content.add(txtPassword);
		content.add(btnLogin);

		this.setContentPane(content);
		this.pack();

		this.addWindowListener(new WindowAdapter(){
			public void windowClosing(WindowEvent e){
				System.exit(0);
			}
		});
	}

	public void actionPerformed(ActionEvent e) {
	}

	public User getUser(){
		User model = new User(txtUsername.getText(), txtPassword.getText());
		return model;
	}

	public void showMessage(String msg){
		JOptionPane.showMessageDialog(this, msg);
	}

	public void addLoginListener(ActionListener log) {
```

```
            btnLogin.addActionListener(log);
        }
}
```

Lớp ClientControl.java

```
package rmi_tcp.tcpClient;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.Socket;

public class ClientControl {
	private ClientView view;
	private String serverHost = "localhost";
	private int serverPort = 8888;

	public ClientControl(ClientView view){
		this.view = view;
		this.view.addLoginListener(new LoginListener());
	}

	class LoginListener implements ActionListener {
	  public void actionPerformed(ActionEvent e) {
	      try {
	          User user = view.getUser();
	          Socket mySocket = new Socket(serverHost, serverPort);
	          ObjectOutputStream oos =
	              new ObjectOutputStream(mySocket.getOutputStream());
	          oos.writeObject(user);

	          ObjectInputStream ois =
	             new ObjectInputStream(mySocket.getInputStream());
	          Object o = ois.readObject();
	          if(o instanceof String){
	              String result = (String)o;
	              if(result.equals("ok"))
	                    view.showMessage("Login succesfully!");
	              else view.showMessage("Invalid username and/or password!");
	          }
	          mySocket.close();
	      } catch (Exception ex) {
	          view.showMessage(ex.getStackTrace().toString());
	      }
	   }
    }
}
```

Lớp ClientRun.java

```
package rmi_tcp.tcpClient;

public class ClientRun {
	public static void main(String[] args) {
		ClientView view = new ClientView();
```

```
            ClientControl control = new ClientControl(view);
            view.setVisible(true);
      }
}
```

3.2 Các lớp phía server TCP

Lớp ServerView.java

```
package rmi_tcp.tcpServer;

public class ServerView {
      public ServerView(){
      }

      public void showMessage(String msg){
            System.out.println(msg);
      }
}
```

Lớp ServerControl.java

```
package rmi_tcp.tcpServer;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.ServerSocket;
import java.net.Socket;
import java.rmi.NotBoundException;
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import rmi_tcp.rmiServer.RMILoginInterface;
import rmi_tcp.tcpClient.User;

public class ServerControl {
      private ServerView view;
      private ServerSocket myServer;
      private Socket clientSocket;
      private String serverRMIHost = "localhost";
      private int serverRMIPort = 3535;
      private int serverTCPPort = 8000;
      private RMILoginInterface rmiServer;
      private Registry registry;
      private String rmiService = "rmitcpLoginServer";

      public ServerControl(ServerView view){
            this.view = view;
            openServer(serverTCPPort);
            bindingRMI();
            view.showMessage("TCP server is running...");

            while(true){
                  listenning();
            }
```

```
	}


	private void openServer(int portNumber){
		try {
			myServer = new ServerSocket(portNumber);
		}catch(IOException e) {
			view.showMessage(e.toString());
			e.printStackTrace();
		}
	}

	private void bindingRMI(){
		try{
			// lay the dang ki
			registry = LocateRegistry.getRegistry(serverRMIHost,
					serverRMIPort);
			// tim kiem RMI server
			rmiServer =  (RMILoginInterface)(registry.lookup(rmiService));
		}catch(RemoteException e){
			view.showMessage(e.getStackTrace().toString());
			e.printStackTrace();
		}catch(NotBoundException e){
			view.showMessage(e.getStackTrace().toString());
			e.printStackTrace();
		}
	}

	private void listenning(){
		try {
			clientSocket = myServer.accept();
			ObjectInputStream ois =
				new ObjectInputStream(clientSocket.getInputStream());

			Object o = ois.readObject();
			if(o instanceof User){
				User user = (User)o;
				String result = rmiServer.checkLogin(user);
				ObjectOutputStream oos =
				new ObjectOutputStream(clientSocket.getOutputStream());
				oos.writeObject(result);
			}
		}catch (Exception e) {
			view.showMessage(e.toString());
			e.printStackTrace();
		}
	}
}
```

Lớp ServerRun.java

```
package rmi_tcp.tcpServer;

public class ServerRun {
	public static void main(String[] args) {
		ServerView view	= new ServerView();
```

```
			ServerControl control = new ServerControl(view);
		}
}
```

3.3 Các lớp phía server RMI

Lớp RMILoginServerView.java

```
package rmi_tcp.rmiServer;

public class RMILoginServerView {
	public RMILoginServerView(){
	}

	public void showMessage(String msg){
		System.out.println(msg);
	}

}
```

Interface RMILoginInterface.java

```
package rmi_tcp.rmiServer;
import java.rmi.Remote;
import java.rmi.RemoteException;
import rmi_tcp.tcpClient.User;

public interface RMILoginInterface extends Remote{
	public String checkLogin(User user) throws RemoteException;
}
```

Lớp RMILoginServerControl.java

```
package rmi_tcp.rmiServer;
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.rmi.server.UnicastRemoteObject;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
import rmi.client.User;

public class RMILoginServerControl extends UnicastRemoteObject implements
RMILoginInterface{
	private int serverPort = 3535;
	private Registry registry;
	private Connection con;
	private RMILoginServerView view;
	private String rmiService = "rmitcpLoginServer";

	public RMILoginServerControl(RMILoginServerView view) throws RemoteException{
```

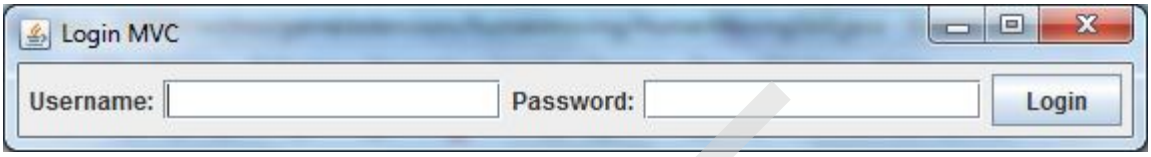
```
            this.view = view;
            getDBConnection("usermanagement", "root", "12345678");
            view.showMessage("RMI server is running...");

            // dang ki RMI server
            try{
                  registry = LocateRegistry.createRegistry(serverPort);
                  registry.rebind(rmiService, this);
            }catch(RemoteException e){
                  throw e;
            }
      }

      public String checkLogin(User user) throws RemoteException{
            String result = "";
            if(checkUser(user))
                  result = "ok";
            return result;
      }

      private void getDBConnection(String dbName,
                                   String username, String password){
          String dbUrl = "jdbc:mysql://localhost:3306/" + dbName;
          String dbClass = "com.mysql.jdbc.Driver";

          try {
             Class.forName(dbClass);
             con = DriverManager.getConnection (dbUrl, username, password);
          }catch(Exception e) {
             view.showMessage(e.getStackTrace().toString());
          }
      }

      private boolean checkUser(User user) {
          String query = "Select * FROM users WHERE username ='"
                   + user.getUserName()
                   + "' AND password ='" + user.getPassword() + "'";

          try {
             Statement stmt = con.createStatement();
             ResultSet rs = stmt.executeQuery(query);

             if (rs.next()) {
                   return true;
             }
          }catch(Exception e) {
             view.showMessage(e.getStackTrace().toString());
          }
          return false;
        }
}
```

Lớp ServerRun.java

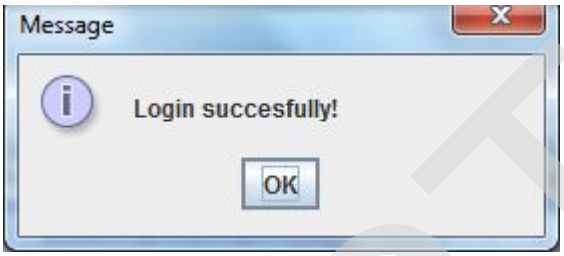
```
package rmi_tcp.rmiServer;
```

```
public class ServerRun {
	public static void main(String[] args) {
		RMILoginServerView view	= new RMILoginServerView();
		try{
			RMILoginServerControl
				control = new RMILoginServerControl(view);
		}catch(Exception e){
			e.printStackTrace();
		}
	}
}
```

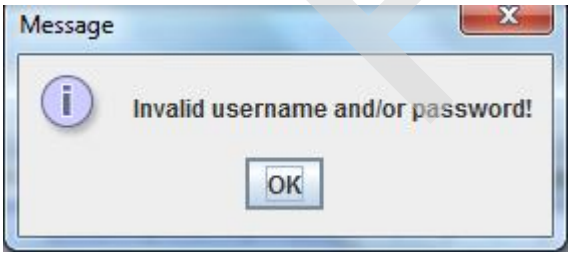
4. Kết quả



Login thành công:



Login lỗi:



VIII. KẾT LUẬN

Qua các mục của chương này, chúng ta đã làm sáng tỏ kỹ thuật lập trình, cơ chế truyền thông và bản chất của lập trình phân tán đối tượng của RMI. Thông qua đó, sinh viên có thể hiểu được các kỹ thuật lập trình khác như RPC, DCOM, CORBA, EJB, WebService... với các kỹ thuật lập trình OOP, SOP và kiến trúc nhiều tầng. Ngoài các vấn đề nêu trong chương, còn một số kỹ thuật khác của RMI không kém phần quan trọng mà sẽ được đề cập đến trong bài giảng và thông qua bài tập của sinh viên như: Vấn đề định nghĩa bộ đăng ký, vấn đề tuần tự hoá đối tượng, Kỹ thuật gọi đối

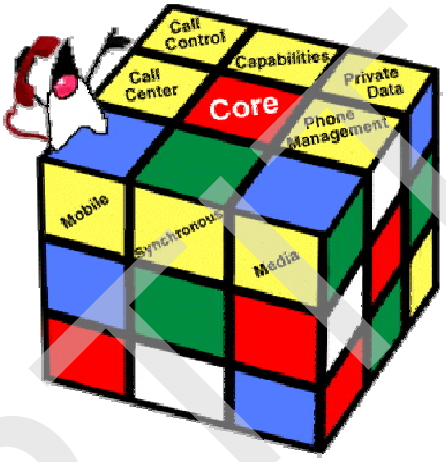
tượng từ xa bằng phương thức động, kỹ thuật kích hoạt đối tượng từ xa tụ động, chính sách bảo mật từ phía client.v.v..

PHẦN IV.
LẬP TRÌNH TRUYỀN THÔNG QUA MẠNG PSTN&ET
CHƯƠNG 5
LẬP TRÌNH ỨNG DỤNG TRUYỀN THÔNG QUA MẠNG ĐIỆN THOẠI CÔNG CỘNG (PSTN)

I. KỸ THUẬT LẬP TRÌNH VỚI JTAPI

1. Giới thiệu thư viện JTAPI

JTAPI là một giao diện lập trình ứng dụng hướng đối tượng cho những ứng dụng máy tính-điện thoại trên nền Java. Tương tự như những giao diện lập trình ứng dụng cho các nền tảng khác như TAPI (Telephony API) trên Microsoft Windows và TSAPI trên Novell Netware. Cấu trúc của thư viện JTAPI được thể hiện như hình sau: Nó gồm bộ cốt lõi và các gói mở rộng chuẩn.



Hình 6.1. Cấu trúc thư viện JTAPI

Tại trung tâm của JTAPI là gói "cốt lõi ". Gói cốt lõi cung cấp khung cơ bản cho mô hình gọi điện thoại và những đặc trưng điện thoại sơ khai ban đầu . Những đặc tính này bao gồm định vị một cuộc gọi, trả lời một gọi, và huỷ một cuộc gọi. Những ứng dụng kỹ thuật điện thoại đơn giản sẽ chỉ cần sử dụng lõi để thực hiện các tác vụ của chúng mà không cần quan tâm tới những chi tiết của những gói khác. Chẳng hạn, gói lõi cho phép người sử dụng dễ dàng thiết kế để thêm đặc tính điện thoại vào một trang Web.

Phân tầng xung quanh gói lõi JTAPI là một số gói "mở rộng chuẩn ". Những gói mở rộng này bổ sung thêm các chức năng điện thoại cho API. Các gói mở rộng chuẩn trong API bao gồm các gói sau:*callcontrol, callcenter, media, phone, privatepackages* và gói *capabilities.*

- *Gói điều khiển gọi – call control.*

Gói *javax.telephony.callcontrol*: Mở rộng lõi bằng việc cung cấp các cuộc gọi mức cao hơn bao gồm các đặc tính điều khiển điện thoại như giữ cuộc gọi, chuyển cuộc gọi... Gói này cũng cung cấp một mô hình trạng thái chi tiết hơn của những cuộc gọi. Các lớp tiêu biểu của gói gồm các giao diện sau:

- CallControlAddress
- CallControlAddressObserver

- CallControlCall
- CallControlCallObserver
- CallControlConnection
- CallControlTerminal
- CallControlTerminalConnection
- CallControlTerminalObserver

➢ *Gói callcenter*

Gói *javax.telephony.callcenter* cung cấp khả năng thực hiện quản lý các trung tâm cuộc gọi lớn ở mức độ cao. Ví dụ như: định tuyến, phân bố cuộc gọi tự động ACD, dự báo cuộc gọi và liên kết dữ liệu ứng dụng với đối tượng điện thoại. Gói này gồm các lớp sau:

- ACDAddress
- ACDAddressObserver
- ACDConnection
- ACDManagerAddress
- ACDManagerConnection
- AgentTerminal
- AgentTerminalObserver
- CallCenterAddress
- CallCenterCall
- CallCenterCallObserver
- CallCenterProvider
- RouteAddress
- RouteCallback
- RouteSession

➢ *Gói Media.*

Gói *javax.telephony.media* cho phép truy nhập tới các luồng(stream) phương tiện truyền thông liên quan đến cuộc gọi. Chúng cho phép đọc và viết dữ liệu từ những luồng phương tiện truyền thông này. Gói này gồm các lớp:

- MediaCallObserver
  - MediaTerminalConnection

➢ *Gói Phone:*

Gói *javax.telephony.phone* cho phép các ứng dụng điều khiển các đặc tính vật lý của phần cứng điện thoại.

Gói Phone gồm các lớp:

- Component
- ComponentGroup
- PhoneButton
- PhoneDisplay
- PhoneGraphicDisplay
- PhoneHookswitch
- PhoneLamp

- PhoneMicrophone
- PhoneRinger
- PhoneSpeaker
- PhoneTerminal
- PhoneTerminalObserver

➢ *Gói capabilities :*

Gói *javax. telephony.capabilities* là gói cung cấp cho các ứng dụng khả năng truy vấn tới hoạt động xác định một khi nó được thực hiện. Và nó gồm các lớp sau :

- AddressCapabilities
- CallCapabilities
- ConnectionCapabilities
- ProviderCapabilities
- TerminalCapabilities
- TerminalConnectionCapabilities

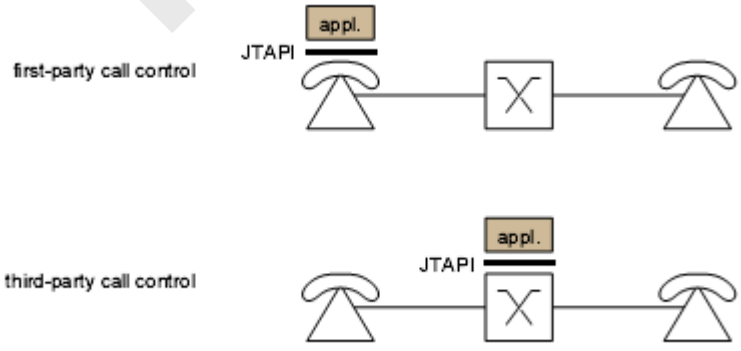
➢ *Gói Private Data*

Gói *javax.telephony.privatedata* cho phép các ứng dụng truyền trực tiếp dữ liệu trên các chuyển mạch cứng. Dữ liệu này được sử dụng để chỉ dẫn chuyển mạch thực hiện một thao tác chuyển mạch cụ.

2. Cơ sở của JTAPI.

Mục đích của thư viện JTAPI được xây dựng để tạo ra một giao diện cho phép trình ứng dụng Java giao tiếp với hệ thống điện thoại. Điểm giao tiếp này xác định mức độ điều khiển mà một ứng dụng phải có. JTAPI hỗ trợ cả 2 kiểu ứng dụng: first-party và third-party.

Trong ứng dụng first- party, giao diện được định vị tại thiết bị đầu cuối. Ứng dụng có cùng mức độ điều khiển như cuộc gọi điện thoại bình thường của người dùng. Trong kịch bản điều khiển third –party, giao diện được xác định bên trong hệ thống điện thoại và phụ thuộc vào hệ thống điện thoại. Sự truy cập bên trong này thường cung cấp cho ứng dụng nhiều khả năng điều khiển hơn kịch bản first- party.



Hình 6.2. Điều khiển cuộc gọi

JTAPI trong thực tế, thực chất là một tập API. Bộ cốt lõi của API cung cấp mô hình cuộc gọi cơ bản và những đặc trưng điện thoại cơ sở nhất như: định vị cuộc gọi và trả lời các cuộc gọi telephone.

Các đặc trưng của điện thoại Java là:

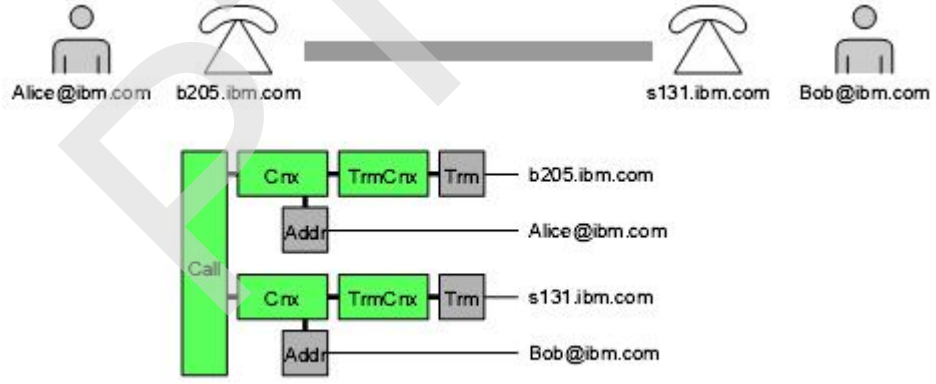
- Làm đơn giản hầu hết các ứng dụng điện thoại cơ bản
- Cung cấp một khung làm việc mà trải khắp các ứng dụng desktop đối với các ứng dụng điện thoại trung tâm gọi phân tán.
- Giao tiếp các ứng dụng trực tiếp với các nơi cung cấp dịch vụ hoặc thực hiện giao tiếp với các API điện thoại tồn tại sẵn như SunXTL, TSAPI, and TAPI.
- Dựa trên bộ lõi đơn giản, gia tăng thêm các gói mở rộng chuẩn.
- Chạy được trên một phạm vi rộng các cấu hình phần cứng một khi Java run-time được sử dụng.

3. Các cấu hình cuộc gọi tiêu biểu

Mục này trình bày những ví dụ cấu hình cuộc gọi được lựa chọn để giải thích mô hình gọi. Nó được bắt đầu với một cuộc gọi 2 phía cơ bản, sau đó mở rộng ví dụ với cuộc gọi, người sử dụng và các thiết bị đầu cuối khác.

Cuộc gọi 2 phía(two- party call):

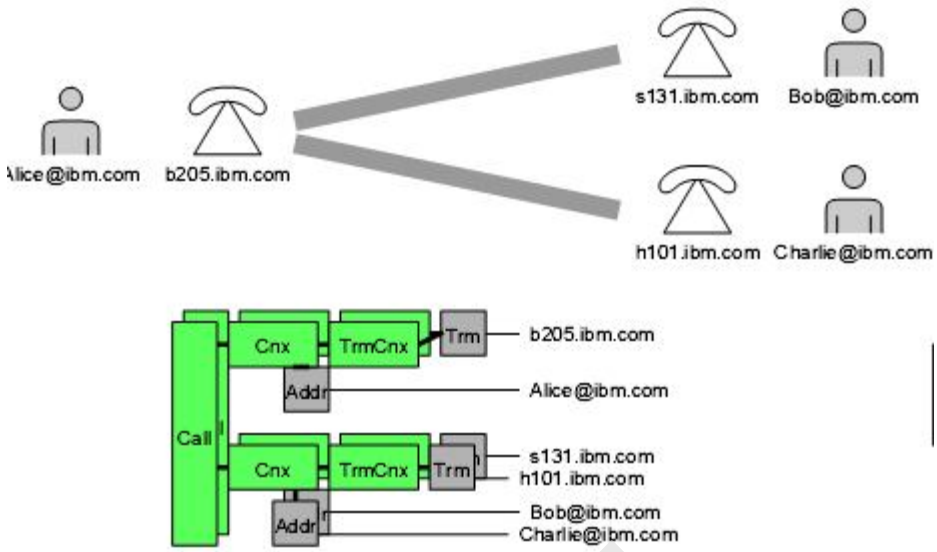
Một ví dụ cuộc gọi với hai người tham gia được biểu diễn trong hình 6.3. Những người chưa có kinh nghiệm có thể rất ngạc nhiên trong trường hợp đơn giản này: hai đối tượng kết nối (*Connection object*) gắn vào đối tượng cuộc gọi (*Call object*), mỗi đối tượng kết nối cho mỗi người tham gia. Cấu hình này cho phép mở rộng để thực hiện cho cuộc gọi hội thảo với ba hoặc nhiều người tham gia hơn. Cần chú ý rằng mô hình này hoàn toàn cân đối (Nó không phân biệt giữa thực thể cục bộ và thực thể ở xa) bởi vì nó cung cấp cách nhìn third-party



Hình 6.3.. Mô hình two- party call

Hai cuộc gọi đồng thời:

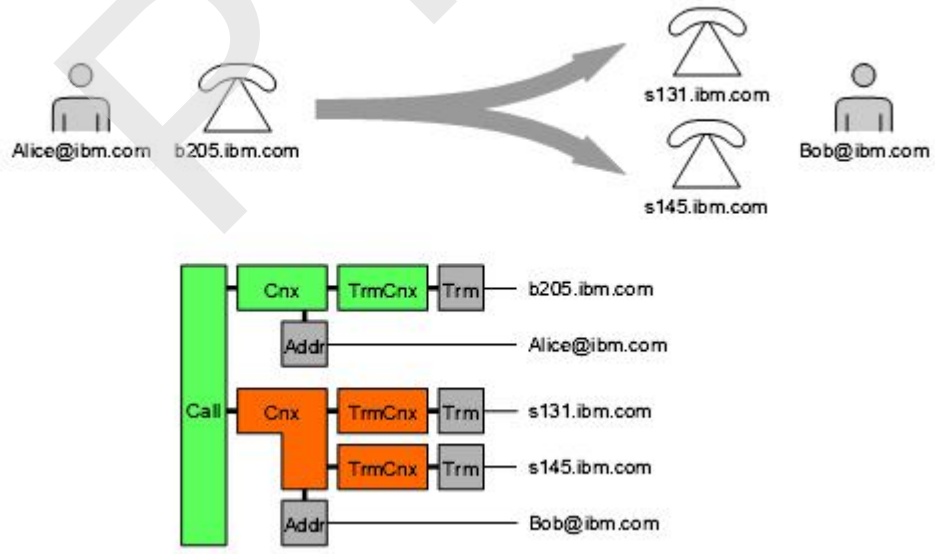
Một ví dụ về một người sử dụng mà có hai cuộc gọi đồng loạt trên cùng thiết bị đầu cuối được đưa vào hình 6.4. Mọi đối tượng liên quan cuộc gọi đã gấp đôi số của họ. Đối tượng địa chỉ (*Address object*) và đối tượng thiết bị đầu cuối (*Terminal object*) của người sử dụng có hai cuộc gọi chỉ sinh ra một lần nhưng được gán cho hai đối tượng kết nối (*Connection object*) và hai đối tượng kết nối đầu cuối (*TerminalConnection*).



Hình6.4. Mô hình Two simultaneous calls.

Cài đặt cuộc gọi với hai thiết bị đầu cuối:

Một ví dụ cuộc gọi hai người với thiết bị đầu cuối có chuông báo được trình bày trong hình 6.5. Trong ví dụ trên, Bob thực hiện nhiều luồng, có nghĩa rằng khi Bob được gọi thì vài thiết bị đầu cuối sẽ đổ chuôngự thể hiện nhiều luồng được đại diện bởi hai đối tượng kết nối đầu cuối gắn cho kết nối đối tượng của Bob, mỗi đối tượng cho mỗi thiết bị đầu cuối. Khi một trong những thiết bị đầu cuối trả lời cuộc gọi thì thiết bị đầu cuối khác sẽ bị loại ra(trong giới hạn của mô hình cuộc gọi này, đối tượng kết nối đầu cuối được đặt vào trong một trạng thái cấm hoạt động)

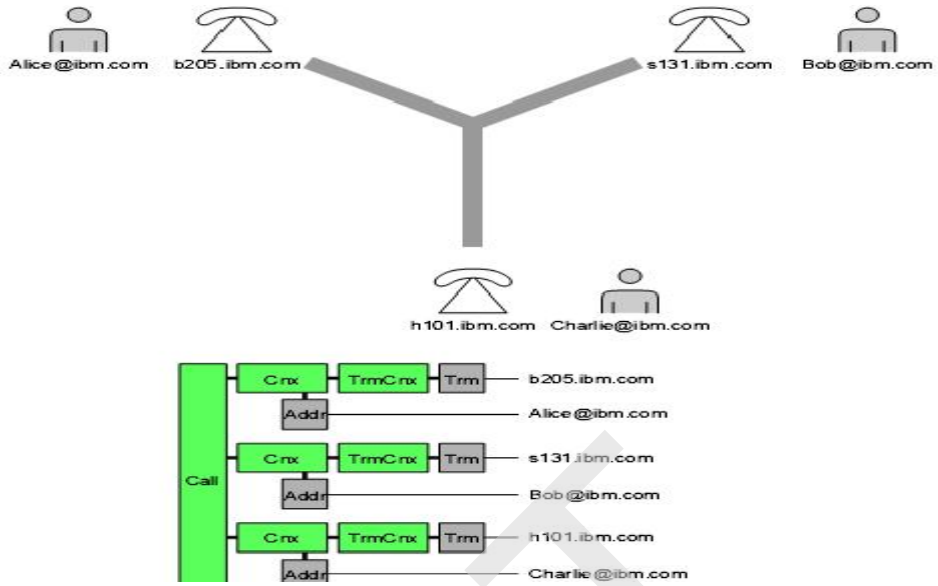


Hình 6.5. Mô hình Two alerting terminal calls.

Cuộc gọi 3 thành viên:

Một ví dụ tiêu biểu cho cuộc gọi ba thành viên là cuộc gọi hội nghị với ba người tham gia được thể hiện như hình 6.6. Mô hình cuộc gọi là một sự mở rộng trực tiếp từ mô hình cuộc gọi cơ bản

với hai người tham gia. Mô hình đơn giản thêm một thành viên thứ ba với các đối tượng kết nối, địa chỉ, kết nối đầu cuối và thiết bị đầu cuối cho người thứ ba tham gia.



Hình 6.6. Mô hình Third-party call.

4. Mô hình cuộc gọi Java

4.1. *Nguyên tắc*

JTAPI là một mô hình trừu tượng hóa mức độ cao và độc lập về công nghệ. Nó mô tả cuộc gọi như là một tập hữu hạn trạng thái máy mà phải trải qua trạng thái chuyển tiếp đó khi cuộc gọi được thực hiện.

Mô hình cuộc gọi được xây dựng tổng quát, bao trùm nhiều kịch bản cuộc gọi khác nhau. Nó có thể được mô tả bằng ví dụ chẳng hạn :

- Cuộc gọi giữa hai đối tác.
- Nhiều cuộc gọi đồng loạt xảy ra trên cùng thiết bị đầu cuối.
- Một cuộc hội thảo nhiều đối tác.
- Cài đặt cuộc gọi để thông báo nhiều thiết bị đầu cuối.

Mô hình cuộc gọi mô tả việc gọi cũng như những thành phần tham gia cuộc gọi. Tất cả nó định nghĩa trong 5 lớp cơ sở. Hai lớp mô tả những thành phần tham gia cuộc gọi. Những đối tượng duy trì và độc lập của cuộc gọi:

- Một người sử dụng (*user*) được đại diện bởi một đối tượng địa chỉ (*Address*). Thuộc tính chính của đối tượng địa chỉ là định danh người sử dụng (*user identifier*).
- Một điện thoại đầu cuối được đại diện cho bởi đối tượng đầu cuối (*Terminal*). Thuộc tính chính của đối tượng thiết bị đầu cuối là địa chỉ của thiết bị đó.

Ba lớp khác mô tả một cuộc gọi. Những đối tượng thể hiện của các lớp này không duy trì mà được tạo ra động trong khi cuộc gọi xảy ra. Mỗi đối tượng bao gồm một trạng thái máy hữu hạn:

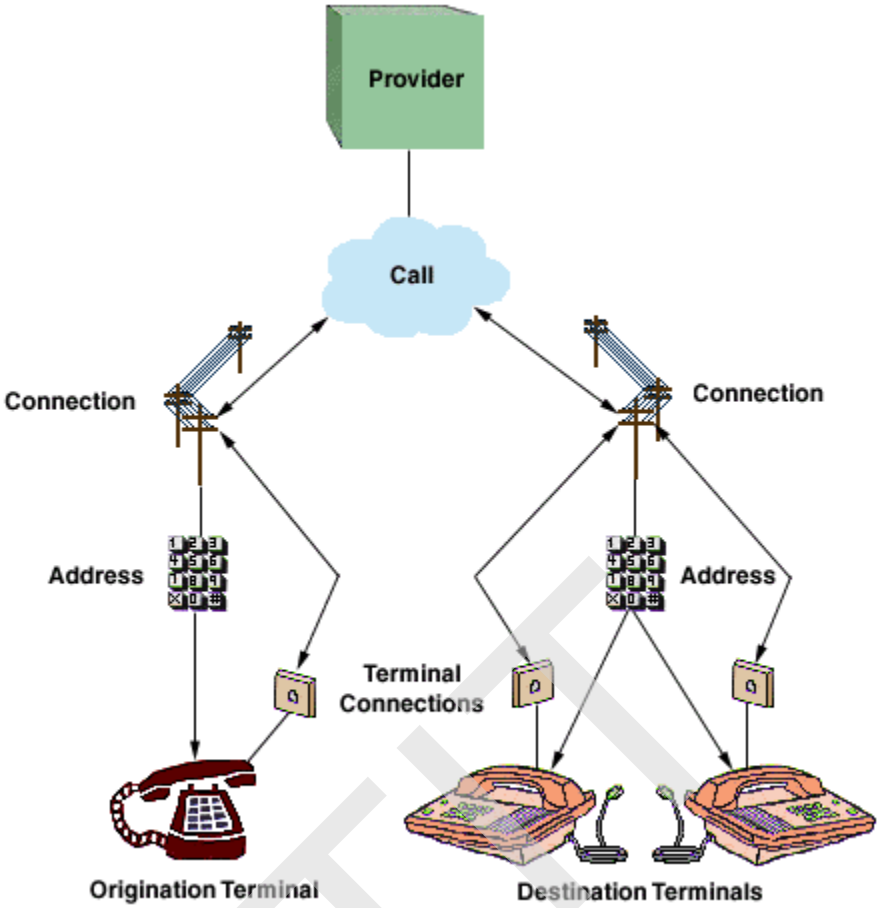
- Một đối tượng gọi (*Call*) được tạo ra cho mỗi cuộc gọi.

- Một đối tượng kết nối (*Connection*) được tạo ra cho mỗi người sử dụng tham gia vào cuộc gọi. Nó kết nối đối tượng địa chỉ của người sử dụng với đối tượng gọi.
- Một đối tượng kết nối đầu cuối (*TerminalConnection*) được tạo ra cho mỗi thiết bị đầu cuối tham gia vào cuộc gọi. Nó kết nối đối tượng (*Terminal*) thiết bị đầu cuối với đối tượng kết nối (*Connection*).

4.2. Các đối tượng trong mô hình gọi thoại java

Các đối tượng trong mô hình gọi thoại Java được thể hiện như hình 6.7.

- **Đối tượng Provider**: là một sự trừu tượng của phần mềm service-provider telephone. Provider có thể quản lý kết nối giữa PBX với server, một card telephony/fax trong máy desktop hoặc một công nghệ mạng máy tính như IP. Provider ẩn tất cả các chi tiết dịch vụ cụ thể của các hệ thống con telephone và cho phép ứng dụng Java hoặc Applet tương tác với các hệ thống con telephone trong cơ chế độc lập thiết bị.
- **Đối tượng Call**: Đối tượng này thể hiện một cuộc gọi điện thoại là luồng thông tin giữa người cung cấp dịch vụ và các thành viên của cuộc gọi. Một cuộc gọi điện thoại bao gồm một đối tượng Call và không hoặc nhiều kết nối. Trong kiểu gọi two-party gồm một đối tượng Call và 2 kết nối, còn trong kiểu hội thảo thì có 3 hoặc nhiều hơn số kết nối với một đối tượng Call.
- **Đối tượng Address**: Đối tượng này biểu diễn một số điện thoại. Nó là sự trừu tượng đối với một điểm cuối logic của một cuộc gọi điện thoại. Trong thực tế một số điện thoại có thể tương ứng với một số điểm cuối vật lý.
- **Đối tượng Connection**: Một đối tượng Connection mô hình hoá liên kết truyền thông giữa đối tượng Call và đối tượng Address. Đối tượng Connection có thể ở trong một trong các trạng thái khác nhau chỉ thị trạng thái quan hệ hiện thời giữa Call và Address.



Hình 6.7. Mô hình cuộc gọi thoại Java

- **Đối tượng Terminal**: Biểu diễn một thiết bị vật lý như điện thoại và các thuộc tính gắn với nó. Mỗi đối tượng Terminal có một hoặc nhiều đối tượng Address(số điện thoại) gắn kết với nó. Terminal cũng được xem như là điểm cuối vật lý của một cuộc gọi vì nó tương ứng với một phần cứng vật lý.
- *Đ□i t□□ng TerminalConnection: **Th□ hi□n m□i quan h□ gi□a m□t k□t n□i và m□t đi□m cu□i v□t lý c□a m□t cu□c g□i mà đ□□c bi□u di□n b□i đ□i t□□ng Terminal. Đ□i tu□ng này mô t□ tr□ng thái hi□n th□i c□a m□i quan h□ gi□a đ□i t□□ng Connection và Terminal c□ th□.***

4.3. Các ph□□ng th□c gói c□t lõi JTAPI

Gói c□t lõi c□a JTAPI đ□nh nghĩa 3 ph□□ng th□c h□ tr□ các đ□c tr□ng c□ b□n: Thi□t đ□t m□t cu□c g□i, tr□ l□i cu□c g□i và hu□ k□t n□i c□a m□t cu□c g□i. Các ph□□ng th□c t□□ng □ng v□i các tác v□ này là **Call.connect(), TerminalConnection.answer(), Connection.disconnect().**

- Ph□□ng th□c Call.connect():***Khi m□t □ng d□ng có đ□i t□□ng r□i(thu đ□□c thông qua ph□□ng th□c* Provider.createCall()), *nó có th□ thi□t l□p m□t cu□c g□i đi□n tho□i*

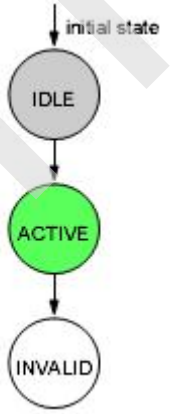
b□ng cách s□ d□ng ph□□ng th□c* Call.connect(). *□ng d□ng ph□i ch□ ra ddoois t□□ng Terminal ngu□n(đi□m cu□i v□t lý) và đ□i t□□ng Address ngu□n(đi□m cu□i logic) trên Terminal đó. Nó cũng cung c□p m□t chu□i s□ đi□n tho□i đích. Hai đ□i t□□ng Connection đ□□c tr□ v□ t□ ph□□ng th□c* Call.connect() *bi□u di□n các đ□u cu□i ngu□n và đích c□a m□t cu□c g□i đi□n tho□i.

- TerminalConnection.answer(): ***Khi m□t cu□c g□i đi t□i m□t Terminal, nó s□ đ□□c ch□ th□ b□i đ□i t□□ng TerminalConnection đ□i v□i Terminal đó trong tr□ng thái RINGING. T□i th□i đi□m đó, □ng d□ng s□ g□i ph□□ng th□c* TerminalConnection.answer() *đ□ tr□ l□i cu□c g□i t□i đó.***
- Connection.disconnect(): ***Ph□□ng th□c này đ□□c g□i đ□ lo□i b□ Address t□ m□t cu□c tho□i. Đ□i t□□ng Connection bi□u di□n quan h□ đ□i t□□ng Address v□i cu□c g□i đi□n tho□i. □ng d□ng s□ g□i ph□□ng th□c này khi đ□i t□□ng Connection đang □ tr□ng thái CONNECTED và tr□ v□ k□t qu□ là đ□i t□□ng Connetction chuy□n đ□n tr□ng thái DISCONNECTED.***

4.4. Những trạng thái máy hữu hạn

4.4.1.Đối tượng cuộc gọi

Mỗi đối tượng cuộc gọi được tạo ra mỗi khi thực hiện cuộc gọi. Trạng thái của đối tượng cuộc gọi phụ thuộc vào mã số của đối tượng kết nối và nó gồm các trạng thái thể hiện như hình 6.8.



Hình 6.8. Đối tượng gọi.

Trạng thái nhàn rỗi(IDLE): Đây là trạng thái khởi đầu cho mọi cuộc gọi. Trong trạng thái này, cuộc gọi không có kết nối nào.

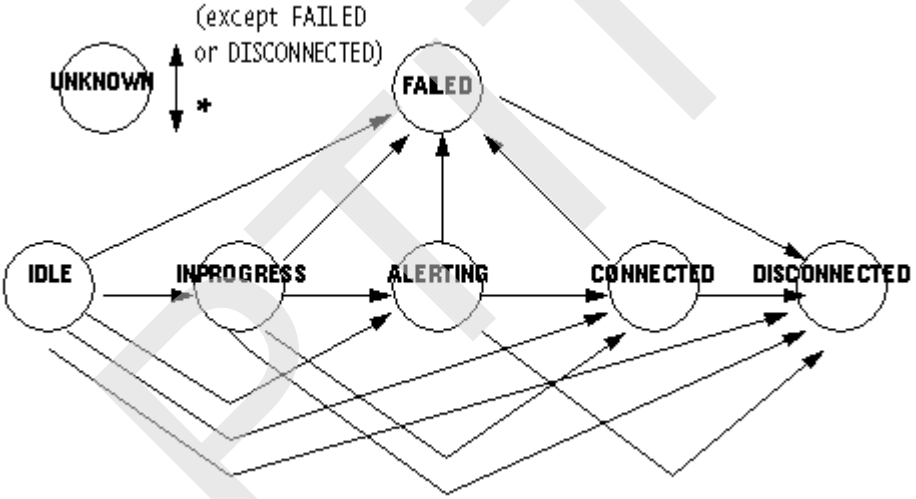
*Hoạt động(Active):*Đây trạng thái khi một cuộc gọi đang xẩy ra.Các cuộc gọi với một hoặc nhiều kết nối đều phải ở trong trạng thái này.

*Vô hiệu hóa(Invalid):*Đây là trạng thái cuối cùng cho mọi cuộc gọi. Cuộc gọi mà mất tất cả các đối tượng kết nối (thông qua một sự chuyển tiếp của đối tượng kết nối vào trong kết nối - trạng thái ngưng kết nối) sẽ chuyển vào trong trạng thái này. Các cuộc gọi khi ở trong trạng thái này sẽ không có kết nối nào và những đối tượng cuộc gọi này có thể không được sử dụng cho bất kỳ hoạt động nào trong tương lai. *2.4.1. Các trạng thái đối tượng Connection*

4.4.2. Các trng thái đi tng Connection và đi tng TerminalConnection

L□□c đ□ d□ch chuy□n tr□ng thái c□a đ□i t□□ng Connection có th□ đ□□c bi□u di□n nh□ hình 6.9. Nó g□m các tr□ng thái sau:

- *IDLE:* ***Đây là tr□ng thái kh□i t□o ban đ□u c□a t□t c□ các đ□i t□□ng Connection m□i.***
- *INPROGRESS:* ***Ch□ th□ cu□c g□i đi□n tho□i hi□n th□i đã thi□t đ□t t□i đi□m cu□i đích.***
- *ALERTING:* ***Ch□ th□ phía đích c□a cu□c g□i đã c□nh báo m□t cu□c g□i t□i.***
- *CONNECTED:* ***Ch□ th□ tr□ng thái đ□wcj k□t n□i c□a m□t cu□c đi□n tho□i***
- *DISCONNECTED:* ***Ch□ th□ tr□ng thái k□t thúc cu□c g□i.***
- *FAILED:* ***Ch□ th□ m□t cu□c g□i thi□t đ□t t□i đi□m cu□i b□ l□i, ví d□ k□t n□i t□i m□t phía đang b□n.***
- *UNKNOWN:* ***Ch□ th□ r□ng đ□i t□□ng Provider không th□ xác đ□nh đ□□c đ□i t□□ng Connection t□i th□i đi□m hi□n th□i.***

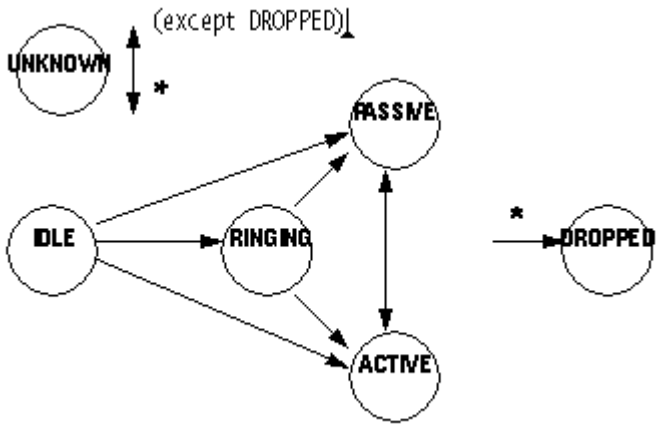


Hình 6.9. L□□c đ□ d□ch chuy□n tr□ng thái c□a Connection

***4.4.3.* Các trng thái đi tng TerminalConnection**

L□□c đ□ d□ch chuy□n tr□ng thái c□a đ□i t□□ng TerminalConnection th□ hi□n nh□ hình 6.10.

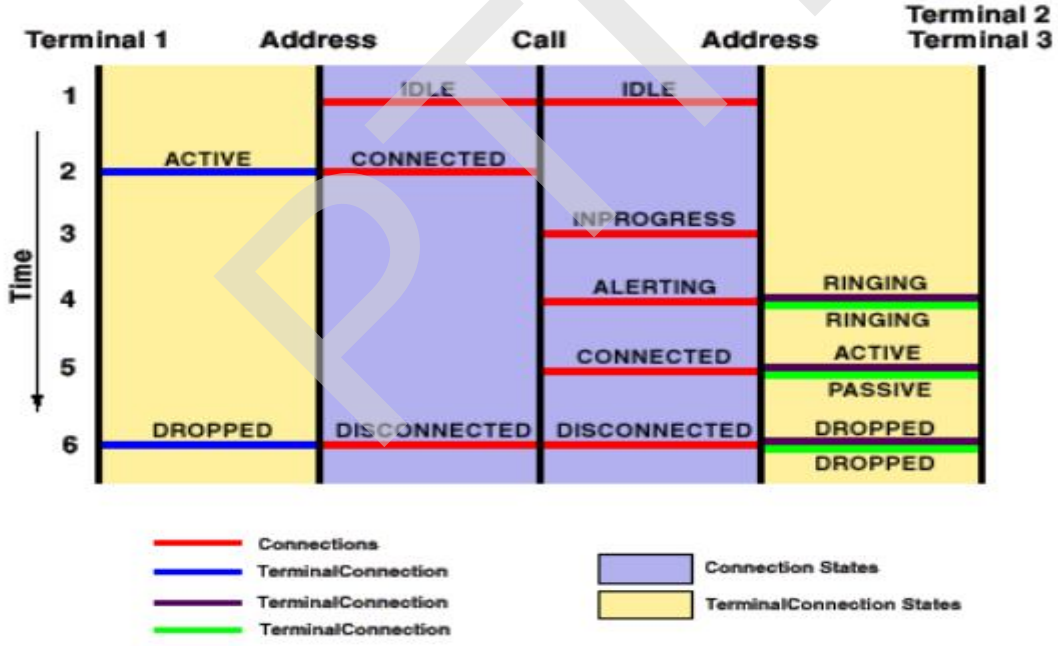
- *IDLE:* ***Tr□ng thái kh□i t□o ban đ□u c□a m□i đ□i t□□ng TerminalConnection***
- *ACTIVE:* ***Ch□ th□ Terminal là ph□n kích ho□t c□a m□t cu□c g□i đi□n tho□i.***
- *RINGING:* ***Ch□ th□ r□ng m□t Terminal báo tín hi□u cho ng□□i s□ d□ng có cu□c g□i t□i t□i Terminal hi□n th□i.***
- *DROPPED:* ***Ch□ th□ tr□ng thái b□ d□t cu□c g□i***
- *PASSIVE:* ***Ch□ th□ tr□ng thái không kích ho□t c□a Terminal.***
- *UNKNOWN:* ***Ch□ th□ provider không cho phép xác đ□nh tr□ng thái hi□n th□i c□a TerminalConnection.***



Hình 6.10. L□□c đ□ d□ch chuy□n tr□ng thái c□a TerminalConnection

4.5. Thi□t đ□t m□t cu□c g□i đi□n tho□i

Ph□n này s□ mô t□ s□ thay đ□i tr□ng thái c□a toàn b□ mô hình g□i ph□i tr□i qua khi thi□t đ□t m□t cu□c g□i đi□n tho□i đ□n gi□n. Quá trình này có th□ đ□□c th□ hi□n b□ng m□t l□□c đ□ đ□nh th□i mô hình g□i nh□ hình 6.11.



Hình 6.11. L□□c đ□ đ□nh th□i mô hình cu□c g□i

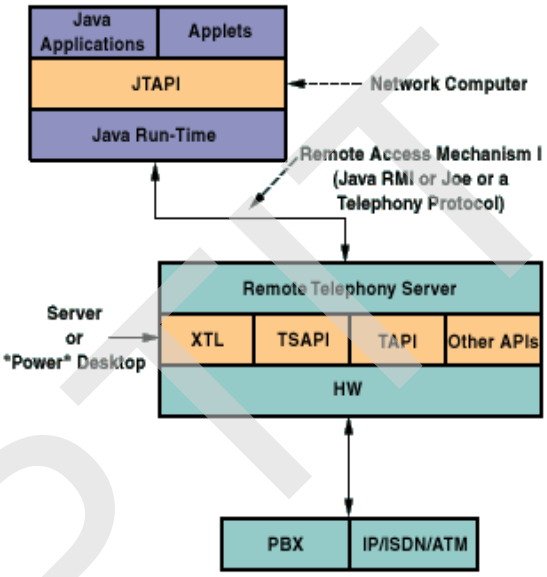
Trong l□□c đ□ trên, các b□□c th□i gian r□i r□c b□i các s□ nguyên h□□ng xu□ng theo tr□c tung. L□□c đ□ này bi□u di□n m□t cu□c g□i đ□n gi□n ki□u two-party. L□□c đ□ này chia làm 2 ph□n, n□a trái và n□a ph□i. N□a trái bi□u di□n đi□m cu□i ngu□n c□a cu□c g□i và n□a ph□i bi□u di□n đi□m cu□i đích c□a cu□c g□i.

II. CẤU HÌNH HỆ THỐNG

JTAPI chạy trên nhiều cấu hình hệ thống khác nhau, bao gồm trung tâm phục vụ và máy tính mạng từ xa truy nhập tài nguyên điện thoại qua mạng. Trong cấu hình đầu tiên, một máy tính mạng đang chạy ứng dụng JTAPI và đang truy nhập những tài nguyên điện thoại qua một mạng được minh họa trong hình 6.12. Cấu hình thứ hai ứng dụng đang chạy trên một máy tính với những tài nguyên điện thoại riêng được minh họa trong hình 6.13.

1. Cấu hình máy tính mạng

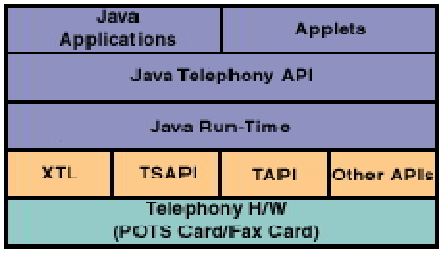
Trong cấu hình này, ứng dụng JTAPI hay Java applet chạy trên một trạm từ xa. Trạm làm việc này có thể là một máy tính nối mạng. Nó truy nhập tài nguyên mạng, sử dụng một trung tâm quản lý kỹ thuật điện thoại. JTAPI giao tiếp với bộ phận phục vụ này qua một cơ chế truyền thông từ xa, như RMI của Java, JOE hay một giao thức điện thoại nào đó. Cấu hình này đurwcj thể hiện như hình 6.10.



Hình 6.12. Cấu hình máy tính mạng

2. Cấu hình desktop

Trong cấu hình máy để bàn, ứng dụng JTAPI hay Java applet chạy trên cùng trạm làm việc. Cấu hình desktop thể hiện như hình 6.13.



Hình 6.13: Cấu hình Máy để bàn

III. MỘT SỐ VÍ DỤ LẬP TRÌNH VỚI JTAPI

1. Ví dụ thiết lập một cuộc gọi điện thoại sử dụng phương thức Call.connect()

```
import javax.telephony.*;
import javax.telephony.events.*;
/*
 * The MyOutCallObserver class implements the CallObserver
 * interface and receives all events associated with the Call.
 */

public class MyOutCallObserver implements CallObserver {

  public void callChangedEvent(CallEv[] evlist) {

    for (int i = 0; i < evlist.length; i++) {

      if (evlist[i] instanceof ConnEv) {

        String name = null;
        try {
          Connection connection = ((ConnEv)evlist[i]).getConnection();
          Address addr = connection.getAddress();
          name = addr.getName();
        } catch (Exception excp) {
          // Handle Exceptions
        }
        String msg = "Connection to Address: " + name + " is ";

        if (evlist[i].getID() == ConnAlertingEv.ID) {
          System.out.println(msg + "ALERTING");
        }
        else if (evlist[i].getID() == ConnInProgressEv.ID) {
          System.out.println(msg + "INPROGRESS");
        }
        else if (evlist[i].getID() == ConnConnectedEv.ID) {
          System.out.println(msg + "CONNECTED");
        }
        else if (evlist[i].getID() == ConnDisconnectedEv.ID) {
          System.out.println(msg + "DISCONNECTED");
        }
      }
    }
  }
}
```

2. Thực hiện cuộc gọi điện thoại từ một số tới một số

```
import javax.telephony.*;
import javax.telephony.events.*;
```

```
import MyOutCallObserver;
public class Outcall {
  public static final void main(String args[]) {
     /*
      * Create a provider by first obtaining the default implementation of
      * JTAPI and then the default provider of that implementation.
      */
    Provider myprovider = null;
    try {
      JtapiPeer peer = JtapiPeerFactory.getJtapiPeer(null);
      myprovider = peer.getProvider(null);
    } catch (Exception excp) {
      System.out.println("Can't get Provider: " + excp.toString());
      System.exit(0);
    }
   /*
    * We need to get the appropriate objects associated with the
    * originating side of the telephone call. We ask the Address for a list
    * of Terminals on it and arbitrarily choose one.
    */
    Address origaddr = null;
    Terminal origterm = null;
    try {
      origaddr = myprovider.getAddress("4761111");
      /* Just get some Terminal on this Address */
      Terminal[] terminals = origaddr.getTerminals();
      if (terminals == null) {
        System.out.println("No Terminals on Address.");
        System.exit(0);
      }
      origterm = terminals[0];
    } catch (Exception excp) {
      // Handle exceptions;
    }
    /*
     * Create the telephone call object and add an observer.
     */
    Call mycall = null;
    try {
      mycall = myprovider.createCall();
      mycall.addObserver(new MyOutCallObserver());
    } catch (Exception excp) {
      // Handle exceptions
    }
    /*
     * Place the telephone call.
     */
    try {
      Connection c[] = mycall.connect(origterm, origaddr, "5551212");
    } catch (Exception excp) {
```

```
      // Handle all Exceptions
    }
  }
}
```

3. Ví dụ minh hoạ cuộc gọi điện thoại tới

```
import javax.telephony.*;
import javax.telephony.events.*;
import javax.telephony.*;
import javax.telephony.events.*;
/*
 * The MyInCallObserver class implements the CallObserver and
 * recieves all Call-related events.
 */
public class MyInCallObserver implements CallObserver {

  public void callChangedEvent(CallEv[] evlist) {
    TerminalConnection termconn;
    String name;
    for (int i = 0; i < evlist.length; i++) {

      if (evlist[i] instanceof TermConnEv) {
        termconn = null;
        name = null;

        try {
          TermConnEv tcev = (TermConnEv)evlist[i];
          Terminal term = termconn.getTerminal();
          termconn = tcev.getTerminalConnection();
          name = term.getName();
        } catch (Exception excp) {
          // Handle exceptions.
        }

        String msg = "TerminalConnection to Terminal: " + name + " is ";

        if (evlist[i].getID() == TermConnActiveEv.ID) {
          System.out.println(msg + "ACTIVE");
        }
        else if (evlist[i].getID() == TermConnRingingEv.ID) {
          System.out.println(msg + "RINGING");

          /* Answer the telephone Call using "inner class" thread */
          try {
               final TerminalConnection _tc = termconn;
               Runnable r = new Runnable() {
                 public void run(){
                   try{
```

```
                            _tc.answer();
                    } catch (Exception excp){
                      // handle answer exceptions
                    }
                        };

                };
                Thread T = new Thread(r);
                T.start();
            } catch (Exception excp) {
              // Handle Exceptions;
            }
          } else if (evlist[i].getID() == TermConnDroppedEv.ID) {
            System.out.println(msg + "DROPPED");
          }
        }
      }
    }
}
import javax.telephony.*;
import javax.telephony.events.*;
import MyInCallObserver;

/*
 * Create a provider and monitor a particular terminal for an incoming call.
 */
public class Incall {

  public static final void main(String args[]) {

    /*
     * Create a provider by first obtaining the default implementation of
     * JTAPI and then the default provider of that implementation.
     */
    Provider myprovider = null;
    try {
      JtapiPeer peer = JtapiPeerFactory.getJtapiPeer(null);
      myprovider = peer.getProvider(null);
    } catch (Exception excp) {
      System.out.println("Can't get Provider: " + excp.toString());
      System.exit(0);
    }

    /*
     * Get the terminal we wish to monitor and add a call observer to that
     * Terminal. This will place a call observer on all call which come to
     * that terminal. We are assuming that Terminals are named after some
     * primary telephone number on them.
     */
    try {
      Terminal terminal = myprovider.getTerminal("4761111");
      terminal.addCallObserver(new MyInCallObserver());
    } catch (Exception excp) {
      System.out.println("Can't get Terminal: " + excp.toString());
      System.exit(0);
  }
```

```
  }
}
```

4. Ví dụ xây dựng dịch vụ RAS với thư viện JTAPI

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
import com.jpackages.jdun.*;
import javax.swing.border.*;
public class Do_an extends JFrame {
  public class DialNotify implements DialingNotification {
    // Phuong thuc goi lai
    public void notifyDialingState(String name, int state, int error) {
      // Hien thi y nghia cua trang thai quay so moi
      System.out.println("Tien trinh - " + name + ": (" + state + ") " +
DialingState.getDialingStateString(state));
      // Neu co loi thi hien thi y nghia cua ma loi
      if (error != 0) {
        System.out.println("Loi:"+error+""+
dum.getErrorMessageForCode(error));
      }
    }
  }
  // handle cho minh hoa ve quan ly quay so (DialUpManager)
  DialUpManager dum;
  // Minh hoa lop DialNotify (da dinh nghia o tren) ma co phuong thuc goi lai
  DialNotify dnot = new DialNotify();
  // Dinh nghia giao dien do hoa
  JPanel contentPane;
  BorderLayout borderLayout1 = new BorderLayout();
  JPanel jPanel1 = new JPanel();
  JPanel jPanel2 = new JPanel();
  JScrollPane jScrollPane1 = new JScrollPane();
  DefaultListModel lm = new DefaultListModel();
  JList jList1 = new JList(lm);
  JPanel jPanel3 = new JPanel();
  BorderLayout borderLayout2 = new BorderLayout();
  JButton jButtonConnect = new JButton();
  JButton jButtonDisconnect = new JButton();
  BorderLayout borderLayout3 = new BorderLayout();
  JPanel jPanel4 = new JPanel();
  JLabel jLabel1 = new JLabel();
  JButton jButtonRefresh = new JButton();
  JPanel jPanel5 = new JPanel();
  FlowLayout flowLayout1 = new FlowLayout();
  JButton jButtonDelete = new JButton();
  JButton jButtonRename = new JButton();
  JPanel jPanel6 = new JPanel();
```

```
JCheckBox jCheckBox1 = new JCheckBox();
BorderLayout borderLayout4 = new BorderLayout();
JPanel jPanel9 = new JPanel();
JPanel jPanel7 = new JPanel();
JTextField jTextFieldUsername = new JTextField();
JLabel jLabel2 = new JLabel();
JPasswordField jPasswordField1 = new JPasswordField();
JPanel jPanel8 = new JPanel();
JLabel jLabel3 = new JLabel();
BorderLayout borderLayout5 = new BorderLayout();
BorderLayout borderLayout6 = new BorderLayout();
BorderLayout borderLayout7 = new BorderLayout();
JPanel jPanel10 = new JPanel();
JRadioButton jRadioButtonOverride = new JRadioButton();
JRadioButton jRadioButtonDefault = new JRadioButton();
JTextField jTextFieldPhoneNumber = new JTextField();
JLabel jLabel4 = new JLabel();
//Constructor
public Do_an() {
  try {
    // minh hoa lop quan ly quay so (DialUpManager)
    dum = new DialUpManager(dnot);
  }
  catch (LibraryLoadFailedException e) {
    if (e instanceof JDUNLibraryLoadFailedException)
      System.out.println("Khong the tai duoc thu vien JDUN...");
    else if (e instanceof RASLibraryLoadFailedException)
      System.out.println("Khong the tai duoc thu vien RAS... ");
    System.exit(0);
  }
  enableEvents(AWTEvent.WINDOW_EVENT_MASK);
  try {
    jbInit();
    initialize();
  }
  catch(Exception e) {
    e.printStackTrace();
  }
}
// kich hoat kha nang ghi de username/password
public void enableParams(boolean b) {
  this.jTextFieldUsername.setEnabled(b);
  this.jTextFieldUsername.setEditable(b);
  this.jPasswordField1.setEnabled(b);
  this.jPasswordField1.setEditable(b);
}
// khoi tao he thong
public void initialize() {
  this.enableParams(false);
  System.out.println("Dang khoi tao...");
```

```
  ButtonGroup bg = new ButtonGroup();
  bg.add(this.jRadioButtonDefault);
  bg.add(this.jRadioButtonOverride);
  this.refreshList();
}
// Danh sach JList
public void refreshList() {
  lm.clear();
  try {
    // Tim nap ten
    String[] names = dum.getEntryNames();
    for (int i=0; i < names.length; i++) {
      lm.addElement(names[i]);
    }
    this.jList1.repaint();
  }
  catch (Exception e) {}
}
//Khoi tao cac thanh phan
private void jbInit() throws Exception  {
  contentPane = (JPanel) this.getContentPane();
  contentPane.setLayout(borderLayout1);
  this.setSize(new Dimension(500, 400));
  this.setTitle("Chuong trinh minh hoa JDUNPhamHienTN2008@yahoo.com");
  jPanel1.setLayout(borderLayout2);
  jScrollPane1.setPreferredSize(new Dimension(660, 80));
  jPanel1.setPreferredSize(new Dimension(260, 100));
  jButtonConnect.setPreferredSize(new Dimension(105, 24));
  jButtonConnect.setText("Ket noi");
  jButtonConnect.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(ActionEvent e) {
      jButtonConnect_actionPerformed(e);
    }
  });
  jButtonDisconnect.setPreferredSize(new Dimension(105, 24));
  jButtonDisconnect.setText("Ngat ket noi");
  jButtonDisconnect.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(ActionEvent e) {
      jButtonDisconnect_actionPerformed(e);
    }
  });
  jPanel2.setLayout(borderLayout3);
  jLabel1.setText("Nhap ten quay so");
  jButtonRefresh.setPreferredSize(new Dimension(79, 24));
  jButtonRefresh.setText("Lam lai");
  jButtonRefresh.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(ActionEvent e) {
      jButtonRefresh_actionPerformed(e);
    }
  });
```

```
    jPanel5.setPreferredSize(new Dimension(70, 28));
    jPanel5.setLayout(flowLayout1);
    flowLayout1.setVgap(2);
    jButtonDelete.setPreferredSize(new Dimension(60, 24));
    jButtonDelete.setMargin(new Insets(0, 0, 0, 0));
    jButtonDelete.setText("Xoa");
    jButtonDelete.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        jButtonDelete_actionPerformed(e);
      }
    });
    jButtonRename.setPreferredSize(new Dimension(70, 24));
    jButtonRename.setMargin(new Insets(0, 0, 0, 0));
    jButtonRename.setText("Doi ten");
    jButtonRename.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        jButtonRename_actionPerformed(e);
      }
    });
    jPanel6.setBorder(BorderFactory.createEtchedBorder());
    jPanel6.setLayout(borderLayout4);
    jCheckBox1.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        jCheckBox1_actionPerformed(e);
      }
    });
    jList1.addListSelectionListener(new
javax.swing.event.ListSelectionListener() {
      public void valueChanged(ListSelectionEvent e) {
        jList1_valueChanged(e);
      }
    });
    jTextFieldUsername.setPreferredSize(new Dimension(200, 24));
    jLabel2.setPreferredSize(new Dimension(65, 17));
    jLabel2.setText("Nguoi dung");
    jPasswordField1.setPreferredSize(new Dimension(200, 24));
    jLabel2.setPreferredSize(new Dimension(65, 17));
    jLabel2.setText("Mat khau");
    jPanel9.setLayout(borderLayout5);
    jPanel8.setLayout(borderLayout7);
    jPanel7.setLayout(borderLayout6);
    jRadioButtonOverride.setText("Ghi de");
    jRadioButtonOverride.addActionListener(new
java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        jRadioButtonOverride_actionPerformed(e);
      }
    });
    jRadioButtonDefault.setSelected(true);
    jRadioButtonDefault.setText("Mac dinh");
```

```
    jRadioButtonDefault.addActionListener(new  java.awt.event.ActionListener()
{
      public void actionPerformed(ActionEvent e) {
        jRadioButtonDefault_actionPerformed(e);
      }
    });
    jTextFieldPhoneNumber.setPreferredSize(new Dimension(120, 24));
    jTextFieldPhoneNumber.setEditable(false);
    jLabel3.setText("So dien thoai");
    contentPane.add(jPanel1, BorderLayout.CENTER);
    jPanel1.add(jScrollPane1, BorderLayout.CENTER);
    jPanel1.add(jPanel3, BorderLayout.NORTH);
    jPanel2.add(jPanel4, BorderLayout.CENTER);
    jPanel3.add(jLabel1, null);
    jPanel3.add(jButtonRefresh, null);
    jPanel2.add(jPanel5, BorderLayout.SOUTH);
    jPanel5.add(jButtonDelete, null);
    jPanel5.add(jButtonRename, null);
    jPanel1.add(jPanel6, BorderLayout.SOUTH);
    jPanel6.add(jCheckBox1, BorderLayout.WEST);
    jPanel6.add(jPanel9, BorderLayout.CENTER);
    jPanel8.add(jLabel3, BorderLayout.WEST);
    jPanel8.add(jPasswordField1, BorderLayout.CENTER);
    jPanel9.add(jPanel8, BorderLayout.SOUTH);
    jPanel9.add(jPanel7, BorderLayout.NORTH);
    jPanel7.add(jLabel2, BorderLayout.WEST);
    jPanel7.add(jTextFieldUsername, BorderLayout.CENTER);
    jPanel6.add(jPanel10, BorderLayout.SOUTH);
    jPanel10.add(jLabel4, null);
    jPanel10.add(jRadioButtonDefault, null);
    jPanel10.add(jRadioButtonOverride, null);
    jPanel10.add(jTextFieldPhoneNumber, null);
    jScrollPane1.getViewport().add(jList1, null);
    contentPane.add(jPanel2, BorderLayout.SOUTH);
    jPanel2.add(jButtonConnect, null);
    jPanel2.add(jButtonDisconnect, null);
  }
  //Kha nang ghi de nho the ta co the thoat khi cua so duoc dong
  protected void processWindowEvent(WindowEvent e) {
    super.processWindowEvent(e);
    if (e.getID() == WindowEvent.WINDOW_CLOSING) {
      System.exit(0);
    }
  }
  // Cap nhat lai JList
  void jButtonRefresh_actionPerformed(ActionEvent e) {
    this.refreshList();
  }
  // Ket noi toi doi tuong da chon
  void jButtonConnect_actionPerformed(ActionEvent e) {
```

```
    String entryName = (String) this.jList1.getSelectedValue();
    if (entryName == null)
      return;
    // Tim nap so dien thoai ghi de
    String phonenum = null;
    if (this.jRadioButtonOverride.isSelected()) {
      phonenum = this.jTextFieldPhoneNumber.getText();
    }
    if (!jCheckBox1.isSelected()) {
      // Quay so voi username/password mac dinh
      if (phonenum == null) {
        dum.dialEntryAsynchronous(entryName);
      }
      else {
        dum.dialEntryAsynchronous(entryName, phonenum);
      }
    }
    else {
      // Lay username/password ghi de va su dung chung de quay so
      String username = this.jTextFieldUsername.getText();
      String password = new String(this.jPasswordField1.getPassword());
      if (phonenum == null) {
        dum.dialEntryAsynchronous(entryName, username, password);
      }
      else {
       dum.dialEntryAsynchronous(entryName,username,password,"",phonenum);
      }
    }
  }
  // Ket thuc cuoc goi da chon
  void jButtonDisconnect_actionPerformed(ActionEvent e) {
    final String entryName = (String) this.jList1.getSelectedValue();
    if (entryName == null)
      return;
    dum.hangUpEntry(entryName);
  }
  // Xoa doi tuong duoc chon
  void jButtonDelete_actionPerformed(ActionEvent e) {
    String entryName = (String) this.jList1.getSelectedValue();
    if (entryName == null)
      return;
    // Xac nhan xoa
    Int    eply=JOptionPane.showConfirmDialog(this,"Ban    co    chac    chan
muonxoa"+entryName+"khong?"Chuy...",JOptionPane.YES_NO_OPTION,JOptionPane.PLAI
N_MESSAGE);
    if (reply == JOptionPane.NO_OPTION) {
      return;
    }
    // Da xac nhan vi the xoa doi tuong
    dum.deleteEntry(entryName);
```

```
    // Cap nhat danh sach(JList) sau khi doi tuong da duoc xoa
    this.refreshList();
  }
  // Doi ten doi tuong duoc chon
  void jButtonRename_actionPerformed(ActionEvent e) {
    String entryName = (String) this.jList1.getSelectedValue();
    if (entryName == null)
      return;
    // Doi ten moi
    String message = "Nhap ten moi '" + entryName + "'";
    String newname = (String) JOptionPane.showInputDialog(this, message, "Doi
ten", JOptionPane.PLAIN_MESSAGE, null, null, entryName);
    if (newname == null)
      return;
    if (newname.equals(entryName))
      return;
    // DOi ten bat ky doi tuong nao sang ten moi
    dum.renameEntry(entryName, newname);
    // Cap nhat lai danh sac (JList) sau khi doi tuong duoc chon da duoc doi
ten
    this.refreshList();
  }
  void jCheckBox1_actionPerformed(ActionEvent e) {
    if (jCheckBox1.isSelected())
      this.enableParams(true);
    else
      this.enableParams(false);
  }
  // Khi mot danh sach cac lua chon duoc tao ra thi nap username/password
thich hop.
  void jList1_valueChanged(ListSelectionEvent e) {
    String entryName = (String) this.jList1.getSelectedValue();
    if (entryName == null)
      return;
    // Tim nap username/password
    String password = dum.getPassword(entryName);
    String username = dum.getUsername(entryName);
    // Hien thi username/password
    this.jTextFieldUsername.setText(username);
    this.jPasswordField1.setText(password);
    // Tim nap cac thuoc tinh doi tuong
    DialUpEntryProperties props = dum.getDialUpEntryProperties(entryName);
    // Hien thi so dien thoai
    if (props.getUseCountryAndAreaCodes()) {
      String areacode = props.getAreaCode();
      String phonenum = props.getLocalPhoneNumber();
      this.jTextFieldPhoneNumber.setText(areacode + phonenum);
    }
    else {
      this.jTextFieldPhoneNumber.setText(props.getLocalPhoneNumber());
```

```
    }
  }
  void phoneButtonChange() {
    if (this.jRadioButtonDefault.isSelected()) {
      this.jTextFieldPhoneNumber.setEditable(false);
    }
    else {
      this.jTextFieldPhoneNumber.setEditable(true);
    }
  }
  void jRadioButtonDefault_actionPerformed(ActionEvent e) {
    phoneButtonChange();
  }
  void jRadioButtonOverride_actionPerformed(ActionEvent e) {
    phoneButtonChange();
  }
  //Phuong thuc chinh
  public static void main(String[] args) {
    Do_an frame = new Do_an();
    //Can giua cho cua so
    Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
    Dimension frameSize = frame.getSize();
    if (frameSize.height > screenSize.height) {
      frameSize.height = screenSize.height;
    }
    if (frameSize.width > screenSize.width) {
      frameSize.width = screenSize.width;
    }
    frame.setLocation((screenSize.width      -frameSize.width)      /      2,
(screenSize.height - frameSize.height) / 2);
    frame.setVisible(true);
  }
}
```

V. KẾT LUẬN

Trong chương nay chúng ta đã khảo sát gói thư viện JTAPI và kỹ thuật lập trình với nó. Qua chương này sinh viên nắm được cấu trúc của thư viện JTAPI, các khái niệm, mô hình và cách cài đặt chương trình với các cuộc gọi điện thoại đơn giản. Trên cơ sở đó sinh viên có thể phát triển các chương trình ứng dụng thực tế như dịch vụ truy cập từ xa RAS, hội thảo trực tuyến và các công nghệ liên qua đến IP khác, nhất là các dịch vụ trên hệ thống điện thoại doanh nghiệp(ET: Enterprise Telephony).

PHẦN IV. LẬP TRÌNH MẠNG AN TOÀN BẢO MẬT
CHƯƠNG VII
LẬP TRÌNH MẠNG AN TOÀN BẢO MẬT VỚI SSL

I. GIỚI THIỆU SSL VÀ MỘT SỐ KHÁI NIỆM

1. Giới thiệu SSL

SSL (Secure Socket Layer) là giao thức đa mục đích được thiết kế để tạo ra các giao tiếp giữa hai chương trình ứng dụng trên một cổng định trước (socket 443) nhằm mã hoá toàn bộ thông tin đi/đến, được sử dụng trong giao dịch điện tử như truyền số liệu thẻ tín dụng, mật khẩu, số bí mật cá nhân (PIN) trên Internet.

Trong các giao dịch điện tử trên mạng và trong các giao dịch thanh toán trực tuyến, thông tin/dữ liệu trên môi trường mạng Internet không an toàn thường được bảo đảm bởi cơ chế bảo mật thực hiện trên tầng vận tải có tên Lớp cổng bảo mật SSL (Secure Socket Layer) - một giải pháp kỹ thuật hiện nay được sử dụng khá phổ biến trong các hệ điều hành mạng máy tính trên Internet. Giao thức SSL được hình thành và phát triển đầu tiên năm 1994 bởi nhóm nghiên cứu Netscape dẫn dắt bởi Elgammal, và ngày nay đã trở thành chuẩn bảo mật thực hành trên mạng Internet. Phiên bản SSL hiện nay là 3.0 và vẫn đang tiếp tục được bổ sung và hoàn thiện. Tương tự như SSL, một giao thức khác có tên là Công nghệ truyền thông riêng tư PCT (Private Communication Technology) được đề xướng bởi Microsoft, hiện nay cũng được sử dụng rộng rãi trong các mạng máy tính chạy trên hệ điều hành Windows NT. Ngoài ra, một chuẩn của Nhóm đặc trách kỹ thuật Internet IETF (Internet Engineering Task Force) có tên là Bảo mật lớp giao vận TLS (Transport Layer Security) dựa trên SSL cũng được hình thành và xuất bản dưới khuôn khổ nghiên cứu của IETF Internet Draft được tích hợp và hỗ trợ trong sản phẩm của Netscape.

2. Khóa – Key

Định nghĩa khóa

Khóa (key) là một thông tin quan trọng dùng để mã hóa thông tin hoặc giải mã thông tin đã bị mã hóa. Có thể hiểu nôm na khóa giống như là mật khẩu(password).

Độ dài khóa – Key Length

Độ dài khóa được tính theo bit: 128 bits, 1024 bits hay 2048 bits,... Khóa càng dài thì càng khó phá. Chẳng hạn như khóa RSA 1024 bits đồng nghĩa với việc chọn 1 trong 2^{1024} khả năng.

Password và PassParse

Password và passparse gần giống nhau. Password không bao giờ hết hạn(expire). Passparse chỉ có hiệu lực trong một khoảng thời gian nhất định có thể là 5 năm, 10 năm hay chỉ là vài ba ngày. Sau thời gian đó, phải thay đổi lại mật khẩu mới. Nói chung, mọi thứ trong SSL như passparse, khóa, giấy chứng nhận, chữ kí số (sẽ nói sau), ... đều chỉ có thời hạn sử dụng nhất định. Passparse được dùng để mở (mã hóa/giải mã) khóa riêng.

3. Thuật toán mã hóa

Mã hóa (encrypt) và giải mã (decrypt) thông tin dùng các hàm toán học đặt biệt. Được biết đến với cái tên là thuật toán mã hóa (cryptographic algorithm) và thường được gọi tắt là cipher. Các thuật toán mã hoá và xác thực của SSL được sử dụng bao gồm (phiên bản 3.0):

(1) DES - Chuẩn mã hoá dữ liệu (ra đời năm 1977), phát minh và sử dụng của chính phủ Mỹ.

(2) DSA - Thuật toán chữ ký điện tử, chuẩn xác thực điện tử, phát minh và sử dụng của chính phủ Mỹ.

(3) KEA - Thuật toán trao đổi khoá, phát minh và sử dụng của chính phủ Mỹ.

(4) MD5 - Thuật toán tạo giá trị "băm" (message digest), phát minh bởi Rivest.

(5) RC2, RC4 - Mã hoá Rivest, phát triển bởi công ty RSA Data Security.

(6) RSA - Thuật toán khoá công khai, cho mã hoá và xác thực, phát triển bởi Rivest, Shamir và Adleman.

(7) RSA key exchange - Thuật toán trao đổi khoá cho SSL dựa trên thuật toán RSA.

(8) SHA-1 - Thuật toán hàm băm an toàn, phát triển và sử dụng bởi chính phủ Mỹ.

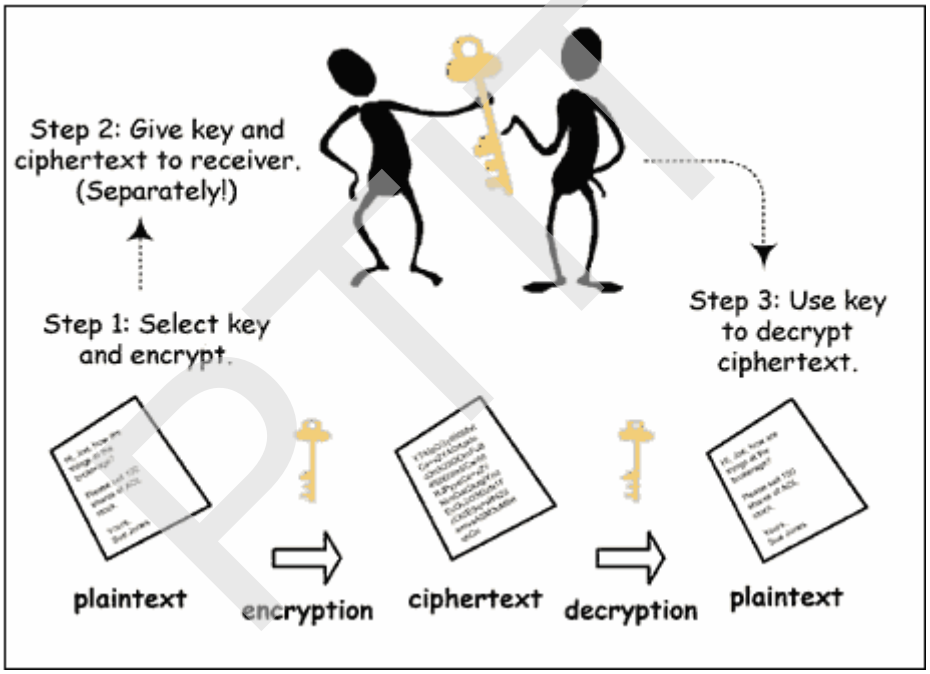
(9) SKIPJACK - Thuật toán khoá đối xứng phân loại được thực hiện trong phần cứng Fortezza, sử dụng bởi chính phủ Mỹ;

(10) Triple-DES - Mã hoá DES ba lần.

Các phương pháp mã hóa

Có hai phương pháp mã hóa được sử dụng phổ biến hiện nay là mã hóa bằng khóa đối xứng và mã hóa dùng cặp khóa chung - khóa riêng..

Mã hóa bằng khóa đối xứng (symmetric-key)



Hình 7.1. Mã hoá bằng khoá đối xứng

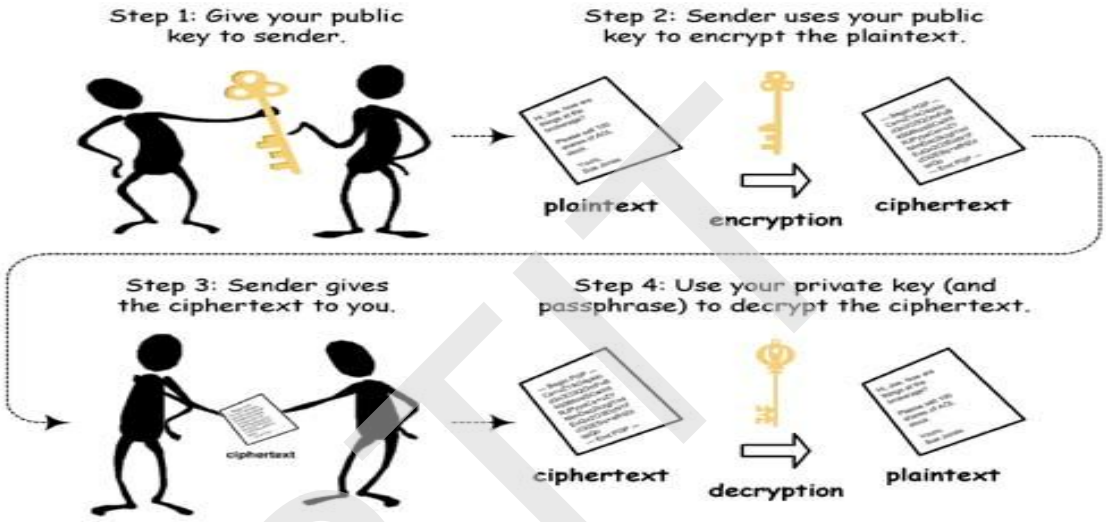
Khóa dùng để mã hóa cũng là khóa dùng để giải mã.

Một khe hở trong mã hóa đối xứng là bạn phải chuyển khóa cho người nhận để họ có thể giải mã. Việc chuyển khóa không được mã hóa qua mạng là một điều cực kì mạo hiểm. Nhỡ như khóa này rơi vào tay người khác thế là họ có thể giải mã được thông tin mà đã chuyển đi. Phương pháp mã hóa bằng khóa chung - khóa riêng ra đời nhằm giải quyết vấn đề này.

Thay vì chỉ có một khóa duy nhất dùng chung cho mã hóa và giải mã, sẽ có một cặp khóa gồm khóa chung chỉ dùng để mã hóa và khóa riêng chỉ dùng để giải mã.

Khi người A muốn gửi thông điệp cho người B thì người B cần biết khóa chung của người A. (Khóa này được người A công bố công khai). Người B mã hóa các thông tin gởi đến người A bằng khóa chung của người A. Chỉ có người A mới có khóa riêng để giải mã các thông tin này. Nhỡ như thông tin này có rơi vào tay người khác thì họ cũng không thể giải mã được vì chỉ có người A mới có khóa riêng dành cho việc giải mã đúng thông điệp trên.

Mã hóa dùng c□p khóa chung – khóa riêng



Hình 7.2. Mã hoá công khai

4. Cơ chế làm việc của SSL – SSL Protocol

Điểm cơ bản của SSL là được thiết kế độc lập với tầng ứng dụng để đảm bảo tính bí mật, an toàn và chống giả mạo luồng thông tin qua Internet giữa hai ứng dụng bất kỳ, thí dụ như webserver và các trình duyệt (browser), do đó được sử dụng rộng rãi trong nhiều ứng dụng khác nhau trên môi trường Internet. Toàn bộ cơ chế hoạt động và hệ thống thuật toán mã hoá sử dụng trong SSL được phổ biến công khai, trừ khóa chia sẻ tạm thời được sinh ra tại thời điểm trao đổi giữa hai ứng dụng là tạo ngẫu nhiên và bí mật đối với người quan sát trên mạng máy tính. Ngoài ra, giao thức SSL còn đòi hỏi ứng dụng chủ phải được chứng thực bởi một đối tượng lớp thứ ba (CA) thông qua chứng chỉ điện tử (digital certificate) dựa trên mật mã công khai (thí dụ RSA).

Sau đây ta xem xét một cách khái quát cơ chế hoạt động của SSL để phân tích cấp độ an toàn của nó và các khả năng áp dụng trong các ứng dụng nhạy cảm, đặc biệt là các ứng dụng về thương mại và thanh toán điện tử.

Giao thức SSL dựa trên hai nhóm con giao thức là giao thức "bắt tay" (handshake protocol) và giao thức "bản ghi" (record protocol). Giao thức bắt tay xác định các tham số giao dịch giữa hai đối tượng có nhu cầu trao đổi thông tin hoặc dữ liệu, còn giao thức bản ghi xác định khuôn dạng cho tiến hành mã hoá và truyền tin hai chiều giữa hai đối tượng đó. Khi hai ứng dụng máy tính, thí dụ giữa một trình duyệt web và máy chủ web, làm việc với nhau, máy chủ và máy khách sẽ trao đổi "lời chào" (hello) dưới dạng các thông điệp cho nhau với xuất phát đầu tiên chủ động từ máy chủ, đồng thời xác định các chuẩn về thuật toán mã hoá và nén số liệu có thể được áp dụng giữa hai ứng dụng. Ngoài ra, các ứng dụng còn trao đổi "số nhận dạng/khoá theo phiên" (session ID, session key) duy nhất cho lần làm việc đó. Sau đó ứng dụng khách (trình duyệt) yêu cầu có chứng chỉ điện tử (digital certificate) xác thực của ứng dụng chủ (web server).

Chứng chỉ điện tử thường được xác nhận rộng rãi bởi một cơ quan trung gian (Thẩm quyền xác nhận CA - Certificate Authority) như RSA Data Sercurity hay VeriSign Inc., một dạng tổ chức độc lập, trung lập và có uy tín. Các tổ chức này cung cấp dịch vụ "xác nhận" số nhận dạng của một công ty và phát hành chứng chỉ duy nhất cho công ty đó như là bằng chứng nhận dạng (identity) cho các giao dịch trên mạng, ở đây là các máy chủ webserver.

Sau khi kiểm tra chứng chỉ điện tử của máy chủ (sử dụng thuật toán mật mã công khai, như RSA tại trình máy trạm), ứng dụng máy trạm sử dụng các thông tin trong chứng chỉ điện tử để mã hoá thông điệp gửi lại máy chủ mà chỉ có máy chủ đó có thể giải mã. Trên cơ sở đó, hai ứng dụng trao đổi khoá chính (master key) - khoá bí mật hay khoá đối xứng - để làm cơ sở cho việc mã hoá luồng thông tin/dữ liệu qua lại giữa hai ứng dụng chủ khách. Toàn bộ cấp độ bảo mật và an toàn của thông tin/dữ liệu phụ thuộc vào một số tham số:

(i) Số nhận dạng theo phiên làm việc ngẫu nhiên.

(ii) Cấp độ bảo mật của các thuật toán bảo mật áp dụng cho SSL.

(iii) Độ dài của khoá chính (key length) sử dụng cho lược đồ mã hoá thông tin.

5. Bảo mật của giao thức SSL

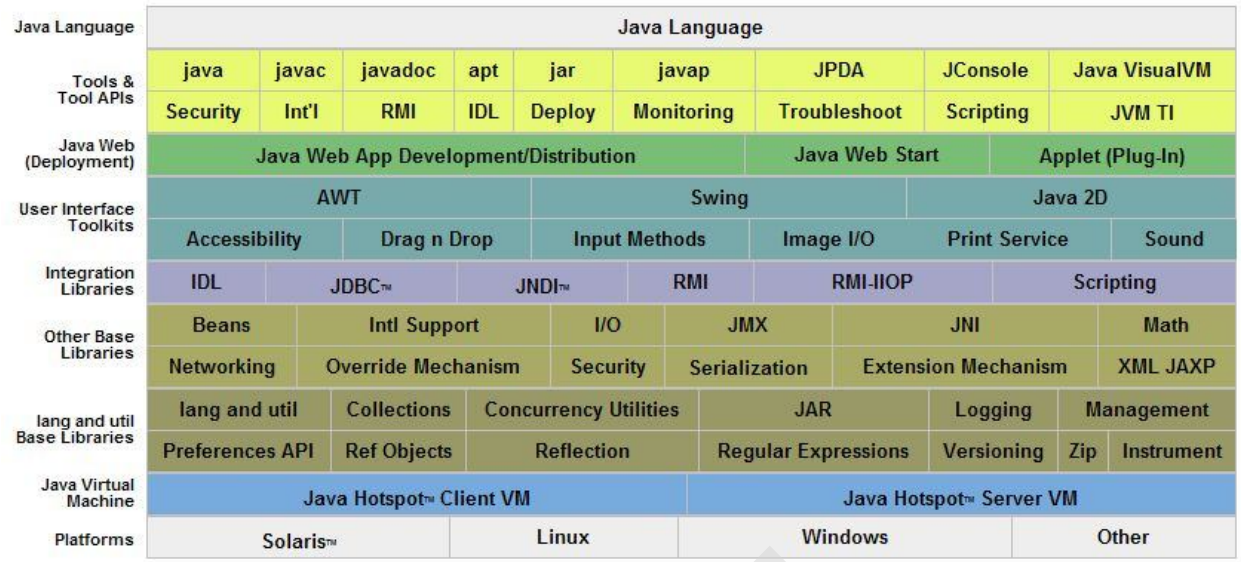
Mức độ bảo mật của SSL như trên mô tả phụ thuộc chính vào độ dài khoá hay phụ thuộc vào việc sử dụng phiên bản mã hoá 40 bits và 128bits. Phương pháp mã hoá 40 bits được sử dụng rộng rãi không hạn chế ngoài nước Mỹ và phiên bản mã hoá 128 bits chỉ được sử dụng trong nước Mỹ và Canada. Theo luật pháp Mỹ, các mật mã "mạnh" được phân loại vào nhóm "vũ khí" (weapon) và do đó khi sử dụng ngoài Mỹ (coi như là xuất khẩu vũ khí) phải được phép của chính phủ Mỹ hay phải được cấp giấy phép của Bộ Quốc phòng Mỹ (DoD). Đây là một lợi điểm cho quá trình thực hiện các dịch vụ thương mại và thanh toán điện tử trong Mỹ và các nước đồng minh phương Tây và là điểm bất lợi cho việc sử dụng các sản phẩm cần có cơ chế bảo mật và an toàn trong giao dịch điện tử nói chung và thương mại điện tử nói riêng trong các nước khác.

Các phương thức tấn công (hay bẻ khoá) của các thuật toán bảo mật thường dùng dựa trên phương pháp "tấn công vét cạn" (brute-force attack) bằng cách thử-sai miền không gian các giá trị có thể của khoá. Số phép thử-sai tăng lên khi độ dài khoá tăng và dẫn đến vượt quá khả năng và công suất tính toán, kể cả các siêu máy tính hiện đại nhất. Thí dụ, với độ dài khoá là 40 bits, thì số phép thử sẽ là 2^{40}=1,099,511,627,776 tổ hợp. Tuy nhiên độ dài khoá lớn kéo theo tốc độ tính toán giảm (theo luỹ thừa nghịch đảo) và dẫn đến khó có khả năng áp dụng trong thực tiễn. Một khi khoá bị phá, toàn bộ thông tin giao dịch trên mạng sẽ bị kiểm soát toàn bộ. Tuy nhiên do độ dài khoá lớn (thí dụ 128 bits, 256 bits), số phép thử-sai trở nên "không thể thực hiện" vì phải mất hàng năm hoặc thậm chí hàng nghìn năm với công suất và năng lực tính toán của máy tính mạnh nhất hiện nay.

Ngay từ năm 1995, bản mã hoá 40 bits đã bị phá bởi sử dụng thuật toán vét cạn. Ngoài ra, một số thuật toán bảo mật (như DES 56 bits, RC4, MD4,...) hiện nay cũng bị coi là không an toàn khi áp dụng một số phương pháp và thuật toán tấn công đặc biệt. Đã có một số đề nghị thay đổi trong luật pháp Mỹ nhằm cho phép sử dụng rộng rãi các phần mềm mã hoá sử dụng mã hoá 56 bits song hiện nay vẫn chưa được chấp thuận.

II. LẬP TRÌNH MẠNG AN TOÀN BẢO MẬT VỚI SSL

1. Thư viện java hỗ trợ lập trình SSL



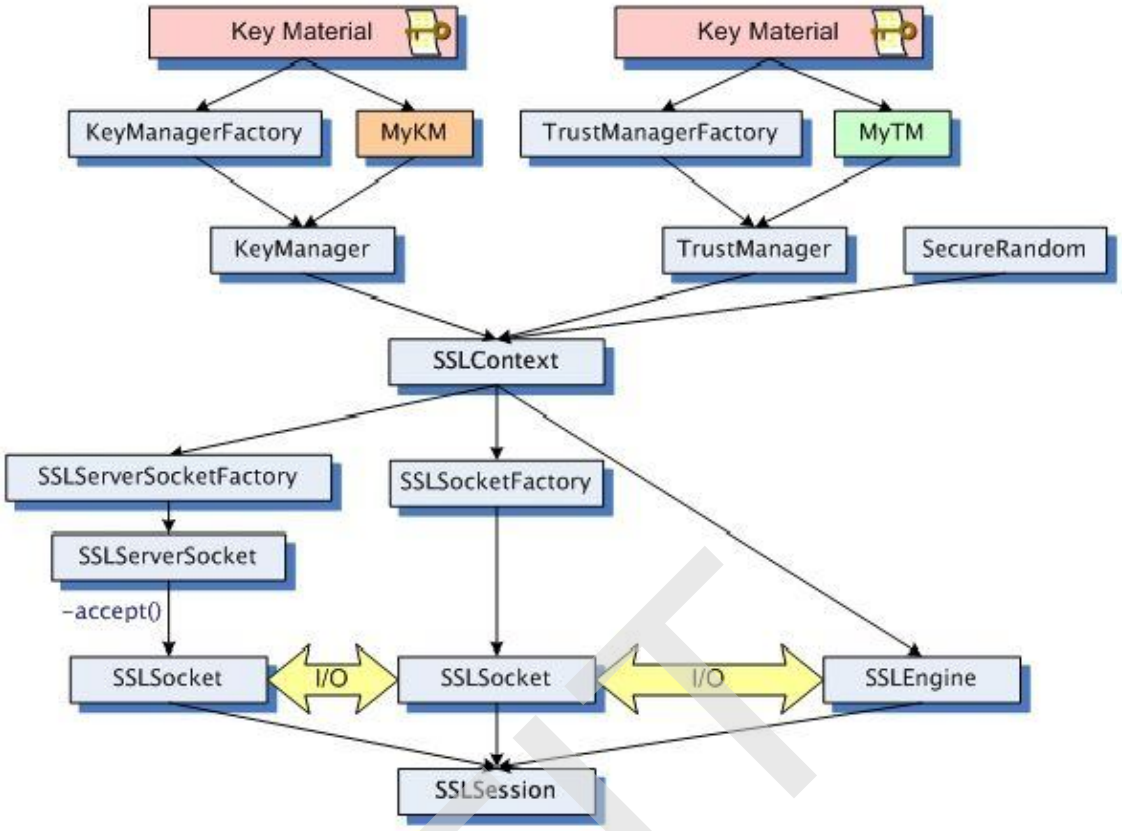
Hình 7.3. Kiến trúc JDK

Java™ security bao gồm tập hợp rất nhiều APIs, công cụ, và cài đặt của các thuật toán bảo mật thông dụng (commonly-used security algorithms), các cơ chế (mechanisms) và các giao thức (protocols). Java security APIs được sử dụng rộng rãi. Bao gồm mã hóa (cryptography), hạ tầng khóa chung (public key infrastructure), trao đổi bảo mật (secure communication), xác thực (authentication), và điều khiển truy cập (access control). Bao gồm rất nhiều lớp thư viện như Java Authentication and Authorization Service (JAAS), Java Cryptography Extension (JCE), Java Secure Socket Extension (JSSE)… Tuy nhiên trong báo cáo này chỉ tập trung vào JSSE. Và cụ thể hơn là JSSE hỗ trợ SSL. Các gói thư viện hỗ trợ lập trình với SSL:

- Gói javax.net.ssl (JSSE)
- Gói javax.rmi.ssl (SSL/TLS-based RMI Socket Factories)

1.1. Lớp SSL

Để truyền thông an toàn, cả 2 phía của kết nối đều phải sử dụng SSL. Trong java, các lớp điểm cuối của kết nối là SSLSocket và SSLEngine. Hình 7.4. cho thấy các lớp chính được sử dụng để tạo ra SSLSocket/SSLEngines.



Hình 7.4. Các lớp java SSL

2. Ví dụ về sử dụng các lớp SSL

Chương trình ví dụ có mã lệnh cho phép server và client có thể xác thực nhau. Muốn vậy thì client phải có chứng chỉ của server (thực tế là một tập (chain) chứng chỉ). Trường hợp ví dụ chứng chỉ của server là chứng chỉ tự ký (self-certificate). Sau đó khi chạy chương trình thì trỏ tới nó.

Sau khi đánh lệnh trên thì sẽ hiện ra các thông tin để điền vào như mật khẩu, tên cá nhân, tên tổ chức, thành phố,…

Server source code (EchoServer.java)

```
import javax.net.ssl.SSLServerSocket;
import javax.net.ssl.SSLServerSocketFactory;
import javax.net.ssl.SSLSocket;
import java.io.BufferedReader;
import java.io.InputStream;
import java.io.InputStreamReader;
public class EchoServer {
    public      static    void           main(String[] arstring) {
        try {
```

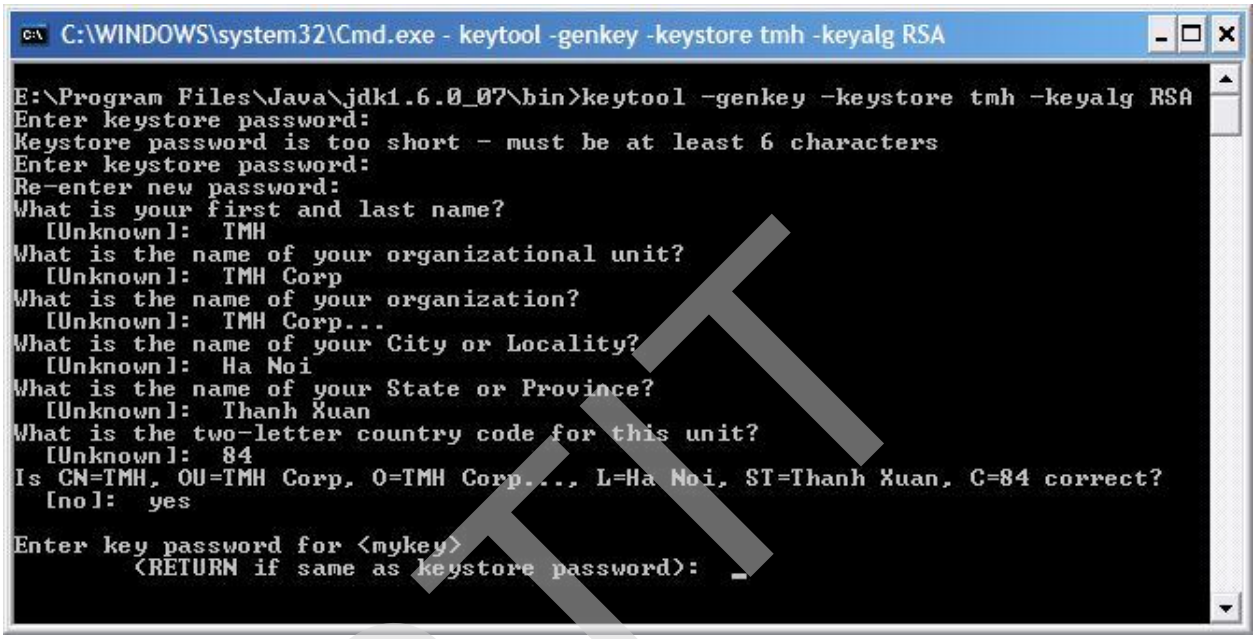
```
                SSLServerSocketFactory sslserversocketfactory =
                        (SSLServerSocketFactory)
        SSLServerSocketFactory.getDefault();
                SSLServerSocket sslserversocket =
                        (SSLServerSocket)
        sslserversocketfactory.createServerSocket(9999);
                SSLSocket sslsocket = (SSLSocket) sslserversocket.accept();
                InputStream inputstream = sslsocket.getInputStream();
                InputStreamReader       inputstreamreader       =       new
        InputStreamReader(inputstream);
                BufferedReader         bufferedreader         =         new
        BufferedReader(inputstreamreader);
                String string = null;
                while ((string = bufferedreader.readLine()) != null) {
                    System.out.println(string);
                    System.out.flush();
                }
            } catch (Exception exception) {
                exception.printStackTrace();
            }
        }
    }
```

Client source code (EchoClient.java)

```
import javax.net.ssl.SSLSocket;
import javax.net.ssl.SSLSocketFactory;
import java.io.*;
public class EchoClient {
    public      static    void     main(String[] arstring) {
        try {
 SSLSocketFactory       sslsocketfactory       =       (SSLSocketFactory)
SSLSocketFactory.getDefault();
 SSLSocket              sslsocket              =              (SSLSocket)
sslsocketfactory.createSocket("localhost", 9999);
           InputStream inputstream = System.in;
            InputStreamReader        inputstreamreader        =        new
InputStreamReader(inputstream);
            BufferedReader          bufferedreader          =          new
BufferedReader(inputstreamreader);
            OutputStream outputstream = sslsocket.getOutputStream();
            OutputStreamWriter       outputstreamwriter       =       new
OutputStreamWriter(outputstream);
            BufferedWriter          bufferedwriter          =          new
BufferedWriter(outputstreamwriter);
            String string = null;
            while ((string = bufferedreader.readLine()) != null) {
                bufferedwriter.write(string + '\n');
```

```
                    bufferedwriter.flush();
                }
            } catch (Exception exception) {
                exception.printStackTrace();
            }
        }
    }
```

Sau khi dịch chạy chương trình , đực kết quả sau:



Giải thích:

-genkey: Lệnh tạo key

-keystore mySrvKeystore: Tên key là mySrvKeystore

-keyalg RSA: Thuật toán dùng để mã hóa là RSA

Về phần ứng dụng qua giao diện dòng lệnh thì sử dụng chương trình mẫu giống như ở trên. Chạy như sau:

Tạo chứng nhận:

keytool -genkey -keystore mySrvKeystore -keyalg RSA

Mật khẩu sẽ điền là 123456

Sau khi tạo xong chứng chỉ thì copy file key vào trong thư mục chứa file

Phía server thì chứng chỉ được lưu trong keyStore

Chạy chương trình:

java -Djavax.net.ssl.keyStore=mySrvKeystore -Djavax.net.ssl.keyStorePassword=123456 EchoServer

Phía Client thì chứng chỉ được lưu trong trustStore

Chạy chương trình:

java -Djavax.net.ssl.trustStore=mySrvKeystore -Djavax.net.ssl.trustStorePassword=123456 EchoClient

III. KẾT LUẬN

Chương này bước đầu đề cập đến vấn đề lập trình mạng an toàn bảo mật mà chủ yếu với SSL, là giao thức được sử dụng rộng rãi nhất cho việc cài đặt mã hoá trong Web. Với cách tiếp cận này, sinh viên có thể tự nghiên cứu khai thác các kỹ thuật lập trình mạng an toàn bảo mật khác nhau, khai thác các hỗ trợ khác nhau của các môi trường Java, .NET...(bảo mật trong java, phương thức SOCKS, JCA, JCE...).

TÀI LIỆU THAM KHẢO

[1] Behrouz A Forouzan, ***TCP/IP protocol suite***, McGraw-Hill , 2 edition (June 27, 2002)

[2] Elliotte Rusty Harold, ***Java Network Programming*, 3rd Edition,** Publisher: O'Reilly, 2004.

[3] Nguyễn Hồng Sơn at. al., ***Kỹ thuật điện thoại qua IP và Internet***, Nhà XBXH, năm 2002.

[4] Nguyễn Phương Lan at al., ***Giáo trình Java 1,2,3,*** NXB Minh Khai, Năm 2001

[5] Nguyễn Hồng Sơn...***Kỹ thuật điện thoại qua IP & Internet***, NXB Lao động Xã hội, 2002

[6] jtapi12-doc-html, Copyright © 1996 Sun Microsystems, Inc