

BỘ GIÁO DỤC VÀ ĐÀO TẠO
ĐẠI HỌC CÔNG NGHỆ TP.HCM

LẬP TRÌNH MẠNG

Biên soạn:

TS. Văn Thiên Hoàng

www.hutech.edu.vn

Lập Trình Mạng

Ấn bản 2021

*Các ý kiến đóng góp về tài liệu học tập này, xin gửi về e-mail của ban biên tập :
tailieuhoclap@hutech.edu.vn*

MỤC LỤC

MỤC LỤC	I
HƯỚNG DẪN	V
BÀI 1. TỔNG QUAN VỀ LẬP TRÌNH MẠNG	1
1.1 VAI TRÒ CỦA ỨNG DỤNG MẠNG.....	1
1.1 CÁC KHÁI NIỆM MẠNG CƠ BẢN.....	2
1.1.1 Mạng máy tính	2
1.1.2 Mô hình phân tầng mạng	2
1.2 CÁC GIAO THỨC MẠNG	5
1.2.1 TCP-Transmission Control Protocol	5
1.2.2 UDP-User Datagram Protocol	7
1.2.3 IP-Internet Protocol.....	9
1.2.4 Địa chỉ IP.....	11
1.2.5 Dịch vụ tên miền	12
CÂU HỎI ÔN TẬP	14
BÀI 2. QUẢN LÝ CÁC LUỒNG NHẬP XUẤT	15
2.1 GIỚI THIỆU.....	15
2.2 CÁC LUỒNG BYTE	16
2.2.1 Các luồng byte tổng quát	17
2.2.2 Các luồng đọc byte hiện thực.....	19
2.3 CÁC LUỒNG KÝ TỰ.....	21
2.3.1 Các luồng ký tự tổng quát	22
2.3.2 Các luồng ký tự hiện thực.....	24
2.4 CÁC LUỒNG LỌC DỮ LIỆU	27
2.4.1 Các luồng lọc tổng quát	28
2.4.2 Các luồng lọc hiện thực.....	28
2.5 CÁC LUỒNG ĐỆM DỮ LIỆU.....	29
2.6 CÁC LỚP NHẬP/XUẤT ĐỊNH KIỂU DỮ LIỆU	31
CÂU HỎI ÔN TẬP	32
BÀI 3. LẬP TRÌNH ĐA TUYẾN	34
3.1 GIỚI THIỆU.....	34
3.1.1 Đơn tiến trình.....	34
3.1.2 Đa tiến trình.....	34
3.1.3 Tiến trình.....	35

3.2 LỚP THREAD.....	35
3.2.1 Tạo Thread.....	36
3.2.2 Chinh độ ưu tiên	37
3.2.3 Thực thi thread	37
3.2.4 Dừng thread	37
3.3 GIAO DIỆN RUNNABLE.....	38
3.4 ĐỒNG BỘ.....	39
3.4.1 Đồng bộ hóa sử dụng cho phương thức.....	39
3.4.2 Lệnh synchronized.....	41
3.5 TRAO ĐỔI DỮ LIỆU GIỮA CÁC THREAD	41
CÂU HỎI ÔN TẬP	42
BÀI 4. QUẢN LÝ ĐỊA CHỈ KẾT NỐI MẠNG	43
4.1 LỚP INETADDRESS	43
4.1.1 Tạo các đối tượng InetAddress	43
4.1.2 Các phương thức lấy dữ liệu của InetAddress	44
4.2 LỚP URL.....	45
4.2.1 Tạo các URL	45
4.2.2 Nhận thông tin các thành phần của URL	46
4.2.3 Nhận dữ liệu từ máy đích trong URL.....	47
4.3 LỚP URLCONNECTION	48
CÂU HỎI ÔN TẬP	49
BÀI 5. LẬP TRÌNH SOCKET CHO GIAO THỨC TCP	50
5.1 MÔ HÌNH KHÁCH CHỦ (CLIENT/SERVER).....	50
5.2 MÔ HÌNH TRUYỀN TIN SOCKET	51
5.3 SOCKET	52
5.3.1 Các hàm khởi tạo Socket	53
5.3.2 Các phương thức giao tiếp giữa các Socket.....	54
5.3.3 Các phương thức đóng Socket.....	55
5.3.4 Các phương thức thiết lập các tùy chọn cho Socket	56
5.4 SERVERSOCKET	57
5.4.1 Các hàm khởi tạo	57
5.4.2 Chấp nhận và ngắt liên kết	59
5.5 VÍ DỤ	60
CÂU HỎI ÔN TẬP	61
BÀI 6. KỸ THUẬT ĐA TIẾN TRÌNH VÀ TUẦN TỰ HÓA ĐỐI TƯỢNG TRONG ỨNG DỤNG MẠNG	63
6.1 ỨNG DỤNG LẬP TRÌNH ĐA TIẾN TRÌNH.....	63

6.1.1 Chương trình phía server.....	63
6.1.2 Chương trình phía client.....	65
6.2 TUẦN TỰ HÓA ĐỐI TƯỢNG	66
6.2.1 Luồng viết đối tượng.....	66
6.2.2 Truyền các đối tượng thông qua Socket.....	67
6.2.3 Ví dụ minh họa	68
CÂU HỎI ÔN TẬP	71
BÀI 7. LẬP TRÌNH SOCKET CHO GIAO THỨC UDP	72
7.1 TỔNG QUAN UDP	72
7.1.1 Một số thuật ngữ UDP.....	72
7.1.2 Hoạt động của giao thức UDP.....	74
7.1.3 Các nhược điểm của giao thức UDP.....	74
7.1.4 Các ưu điểm của UDP	75
7.1.5 Khi nào thì nên sử dụng UDP	76
7.2 DATAGRAMPACKET.....	77
7.2.1 Các hàm khởi tạo để nhận datagram.....	77
7.2.2 Các hàm khởi tạo để gửi các datagram	78
7.2.3 Các phương thức nhận dữ liệu từ DatagramPacket	78
7.3 DATAGRAMSOCKET	80
7.4 GỬI/NHẬN GÓI TIN	82
7.4.1 Nhận gói tin	82
7.4.2 Gửi gói tin.....	83
7.4.3 Ví dụ minh họa giao thức UDP.....	84
CÂU HỎI ÔN TẬP	86
BÀI 8. LẬP TRÌNH MULTICAST	87
8.1 TỔNG QUAN MULTICAST	87
8.1.1 Mục đích của multicast.....	88
8.1.2 Địa chỉ multicast	88
8.1.3 Định tuyến multicast	90
8.2 KIỂU DỮ LIỆU MULTICASTSOCKET.....	90
8.3 CÁC BƯỚC LẬP TRÌNH MULTICAST	91
8.3.1 Các bước để gửi gói dữ liệu multicast	91
8.3.2 Các bước để nhận gói multicast.....	92
CÂU HỎI ÔN TẬP	93
BÀI 9. PHÂN TÁN ĐỐI TƯỢNG TRONG JAVA BẰNG RMI	94
9.1 TỔNG QUAN	94
9.1.1 Mục đích của RMI.....	95

9.1.2 Một số thuật ngữ.....	95
9.1.3 Các lớp trung gian Stub và Skeleton.....	96
9.1.4 Kiến trúc RMI.....	99
9.2 LẬP TRÌNH RMI	101
9.2.1 Gói java.rmi	101
9.2.2 Giao Diện Remote	101
9.2.3 Lớp Naming	102
9.2.4 Gói java.rmi.registry.....	103
9.2.5 Gói java.rmi.server.....	105
9.3 CÀI ĐẶT CHƯƠNG TRÌNH RMI.....	105
CÂU HỎI ÔN TẬP	108
TÀI LIỆU THAM KHẢO.....	1

HƯỚNG DẪN

MÔ TẢ MÔN HỌC

Hiện nay, mạng máy tính là công nghệ của của thời đại. Các ứng dụng mạng đóng vai trò không thể thiếu để khai thác tiềm năng của mạng máy tính, đặt biệt là mạng Internet. Do vậy, Lập trình mạng là môn học không thể thiếu của sinh viên ngành Công nghệ thông tin nói chung và sinh viên chuyên ngành mạng nói riêng. Mục đích của môn học Lập trình mạng là cung cấp cho sinh viên biết kiến thức mạng liên quan cũng như cơ chế hoạt động và kiến trúc của các phần mềm mạng. Từ đó, sinh viên hiểu và biết cách viết các chương trình ứng dụng trong một hệ thống mạng quy mô nhỏ cũng như mạng Internet.

NỘI DUNG MÔN HỌC

- Bài 1: Giới thiệu vai trò của chương trình mạng, những khái niệm căn bản về mạng máy tính, cũng như kiến thức liên quan để người đọc có thể tiếp cận với các chương tiếp theo.
- Bài 2: Các luồng nhập xuất. Bài này giới thiệu khái niệm nhập xuất bằng các luồng dữ liệu, các kiểu luồng, ý nghĩa sử dụng của luồng trong chương trình Java. Luồng được chia thành các nhóm như luồng byte và luồng ký tự. Việc nắm vững kiến thức ở bài này cũng giúp cho việc lập trình ứng dụng mạng trở nên đơn giản hơn vì thực chất của việc truyền và nhận dữ liệu giữa các ứng dụng mạng là việc đọc và ghi các luồng.
- Bài 3: Lập trình mạng với các lớp InetAddress, URL và URLConnection. Lớp InetAddress là lớp căn bản đầu tiên trong lập trình mạng. Nó chỉ ra cách một chương trình Java tương tác với hệ thống tên miền. Sau đó, bài học giới thiệu các khái niệm về URI, URL, URN và lớp biểu diễn URL trong Java, cách sử dụng URL để tải về thông tin và tệp tin từ các server.
- Bài 4: Lập trình đa tiến trình. Bài học này trình bày các khái niệm về lập trình đơn tiến trình và lập trình đa tiến trình, thư viện các kiểu dữ liệu java hỗ trợ cho lập trình đa tuyến. Thư viện này có 2 kiểu dữ liệu quan trọng là Thread

và Runnable. Bài học sẽ hướng dẫn cách cài đặt chương trình đa tiến trình sử dụng lớp Thread hoặc sử dụng giao diện Runnable.

- Bài 5: Lập trình Socket cho giao thức TCP. Bài học này trình bày cách lập trình cho mô hình client/server và các kiểu kiến trúc client/server. Sau đó, các kiểu dữ liệu trong thư viện java.net hỗ trợ cho lập trình ứng dụng theo giao thức TCP được trình bày chi tiết. Hai kiểu dữ liệu quan trọng là: lớp Socket và ServerSocket.
- Bài 6: Phát triển ứng dụng mạng khách/chủ. Bài học này trình bày vai trò của giao thức mạng trong lập trình ứng dụng mạng và các thành phần cơ bản cần có khi thiết kế giao thức. Hơn nữa, các ứng dụng server phải cung cấp dịch vụ cho nhiều client cùng một thời điểm. Do vậy, để tăng tốc độ xử lý đồng thời cho các máy khác và khai thác tối đa tài nguyên vi xử lý, thì các ứng dụng server cần phải hỗ trợ đa tiến trình. Hơn nữa, các ứng dụng mạng thường có nhu cầu trao đổi các đối tượng xử lý qua mạng, nên việc tuần tự hóa đối tượng là cần thiết. Bài học này trình bày các bước áp dụng lập trình đa tiến trình và tuần tự hóa đối tượng để xây dựng ứng dụng mạng.
- Bài 7: Lập trình Socket cho giao thức UDP. Chương này giới thiệu giao thức UDP và các đặc trưng của giao thức này. Tiếp đến, các kiểu dữ liệu như DatagramPacket và DatagramSocket được mô tả rõ và cách áp dụng để viết các chương trình ứng dụng mạng cho giao thức UDP.
- Bài 8: Lập trình UDP Multicast. Bài học này trình bày các khái niệm liên quan đến multicast, mục đích của nó, thư viện java hỗ trợ lập trình ứng dụng UDP multicast và cách áp dụng phát triển ứng dụng hỗ trợ multicast.
- Bài 9: Phân tán đối tượng bằng Java RMI. Bài học này trình bày cách lập trình phân tán đối tượng bằng kỹ thuật triệu gọi phương thức từ xa, gọi tắt là RMI (Remote Method Invocation). Các ưu điểm của RMI trong việc phát triển các ứng dụng phân tán, kiến trúc hạ tầng của công nghệ RMI, và các bước xây dựng phần mềm sử dụng công nghệ này được trình bày chi tiết.

Ngoài ra, môn học còn trang bị cho sinh viên cách phân tích thiết kế các kiểu giao thức của các ứng dụng mạng hiện có và cách áp dụng cho việc phát triển ứng dụng mới.

KIẾN THỨC TIỀN ĐỀ

Môn học lập trình mạng đòi hỏi sinh viên phải có kiến thức về mạng cơ bản và lập trình hướng đối tượng với java.

YÊU CẦU MÔN HỌC

Người học phải dự học đầy đủ các buổi lên lớp, làm đầy đủ các bài tập thực hành và viết phần mềm của đồ án môn học.

CÁCH TIẾP CẬN NỘI DUNG MÔN HỌC

Để học tốt môn này, người học cần ôn tập các bài đã học, trả lời các câu hỏi và làm đầy đủ bài tập; đọc trước bài mới và tìm thêm các thông tin liên quan đến bài học. Đối với mỗi bài học, người học cần làm bài tập thực hành lập trình trên máy vi tính để hiểu được tính năng công nghệ hỗ trợ và lý thuyết quy trình xử lý cần thiết kế.

PHƯƠNG PHÁP ĐÁNH GIÁ MÔN HỌC

- Điểm quá trình: 50%. Điểm kiểm tra thường xuyên trong quá trình học tập. Điểm kiểm tra giữa học phần, điểm làm bài tập trên lớp, hoặc điểm chuyên cần.
- Điểm thi: 50%. Hình thức bài thi tự luận trong 90 phút, không được mang tài liệu vào phòng thi. Nội dung gồm các câu hỏi và bài tập tương tự như các câu hỏi và bài tập về nhà.

BÀI 1. TỔNG QUAN VỀ LẬP TRÌNH MẠNG

1.1 VAI TRÒ CỦA ỨNG DỤNG MẠNG

Các chương trình mạng cho phép người sử dụng khai thác thông tin được lưu trữ trên các máy tính trong hệ thống mạng (đặt biệt các chương trình chạy trên mạng Internet với hàng triệu máy vi tính được định vị khắp nơi trên thế giới.), trao đổi thông tin giữa các máy tính trong hệ thống mạng, cho phép huy động nhiều máy tính cùng thực hiện để giải quyết một bài toán hoặc giám sát hoạt động mạng.

Ví dụ như các chương trình cho phép truy xuất dữ liệu. Các chương trình này cho phép máy khách (client) truy xuất dữ liệu lưu trên máy chủ (server), định dạng dữ liệu và trình bày người sử dụng. Các chương trình này sử dụng các giao thức chuẩn như HTTP (gửi dữ liệu Web), FTP (gửi/nhận tập tin), SMTP (gửi mail) hoặc thiết kế những giao thức mới phục vụ cho việc truy xuất dữ liệu. Các chương trình này hoạt động theo mô hình Client/Server, hoặc cho phép người dùng tương tác với nhau như Game online, Chat, ...

Mô hình khách/chủ (client/server): Hầu hết các chương trình mạng hiện nay sử dụng theo mô hình client/server. Chương trình chạy ở máy chủ quản lý một lượng dữ liệu lớn và cung cấp dịch vụ cho máy khách. Chương trình chạy ở máy khách thực hiện khai thác dữ liệu ở máy chủ. Trong hầu hết trường hợp, máy chủ gửi dữ liệu, máy khách nhận dữ liệu. Thông thường, client sẽ thiết lập cuộc giao tiếp, và server sẽ đợi yêu cầu thiết lập từ client và giao tiếp với nó.

Socket là biểu diễn trừu tượng hóa một cơ chế kết nối giữa hai ứng dụng trên hai máy bằng cách sử dụng kết hợp địa chỉ IP và số hiệu cổng. Nó cho phép các ứng dụng gửi và nhận dữ liệu cho nhau. Cả client và server đều sử dụng socket để giao tiếp với

n nhau. Trong ngôn ngữ Java, một socket được biểu diễn bằng một đối tượng của một trong thư viện java.net như Socket, ServerSocket, DatagramSocket hoặc MulticastSocket. Các ứng dụng mạng máy tính hiện nay được xây dựng dựa vào giao diện Socket này.

1.1 CÁC KHÁI NIỆM MẠNG CƠ BẢN

1.1.1 Mạng máy tính

Mạng máy tính là tập hợp các máy tính hoặc các thiết bị được nối với nhau bởi các đường truyền vật lý và theo một kiến trúc nào đó. Mạng máy tính có thể phân loại theo qui mô như sau:

Mạng LAN (Local Area Network)-mạng cục bộ: kết nối các nút trên một phạm vi giới hạn. Phạm vi này có thể là một công ty, hay một tòa nhà.

Mạng WAN (Wide Area Network): nhiều mạng LAN kết nối với nhau tạo thành mạng WAN.

MAN (Metropolitan Area Network), tương tự như WAN, nó cũng kết nối nhiều mạng LAN. Tuy nhiên, một mạng MAN có phạm vi là một thành phố hay một đô thị nhỏ. MAN sử dụng các mạng tốc độ cao để kết nối các mạng LAN của trường học, chính phủ, công ty, ..., bằng cách sử dụng các liên kết nhanh tới từng điểm như cáp quang.

Trong một hệ thống mạng, mạng xương sống (Backbone) đóng vai trò quan trọng. Mạng Backbone là một mạng tốc độ cao kết nối các mạng có tốc độ thấp hơn. Một công ty sử dụng mạng Backbone để kết nối các mạng LAN có tốc độ thấp hơn. Mạng Backbone Internet được xây dựng bởi các mạng tốc độ cao kết nối các mạng tốc độ cao. Nhà cung cấp Internet hoặc kết nối trực tiếp với mạng backbone Internet, hoặc một nhà cung cấp lớn hơn.

1.1.2 Mô hình phân tầng mạng

Hoạt động gửi dữ liệu đi trong một mạng từ máy trạm đến máy đích là hết sức phức tạp cả mức vật lý lẫn logic. Các vấn đề đặt ra như biến đổi tín hiệu số sang tín hiệu tương tự, tránh xung đột giữa các gói tin (phân biệt các gói tin), phát hiện và

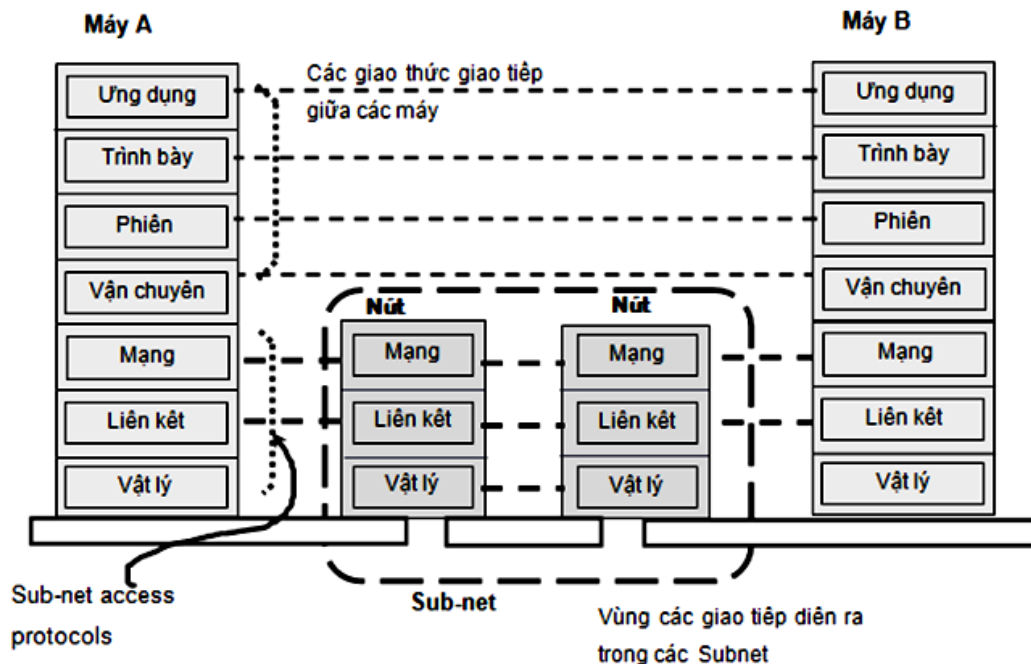
kiểm tra lỗi gói tin, định tuyến gói tin đi tới đích, ... Nhằm quản lý được hoạt động này, các hoạt động Giao tiếp mạng liên quan được phân tách vào các khâu xử lý, hình thành mô hình phân tầng. Có hai dạng mô hình phân tầng phổ biến: OSI, TCP/IP.

Mô OSI trình bày bảy tầng đã rất thành công và nó hiện đang được sử dụng như là một mô hình tham chiếu để mô tả các giao thức mạng khác nhau và chức năng của chúng. Các tầng của mô hình OSI phân chia các nhiệm vụ cơ bản mà các giao thức mạng phải thực hiện, và mô tả các ứng dụng mạng có thể truyền tin như thế nào. Mỗi tầng có một mục đích cụ thể và được kết nối với các tầng ở ngay dưới và trên nó.

Tầng ứng dụng (Application): định nghĩa một giao diện lập trình Giao diện với mạng cho các ứng dụng người dùng, bao gồm các ứng dụng sử dụng các tiện ích mạng. Các ứng dụng này có thể thực hiện các tác vụ như truyền tệp tin, in ấn, e-mail, duyệt web,...

Tầng trình diễn (Presentation): có trách nhiệm mã hóa/giải mã), nén/giải nén dữ liệu từ tầng ứng dụng để truyền đi trên mạng và ngược lại.

Tầng phiên (Session): tạo ra một liên kết ảo giữa các ứng dụng. Ví dụ các giao thức FTP, HTTP, SMTP, ...



Hình 1.1: Mô hình OSI

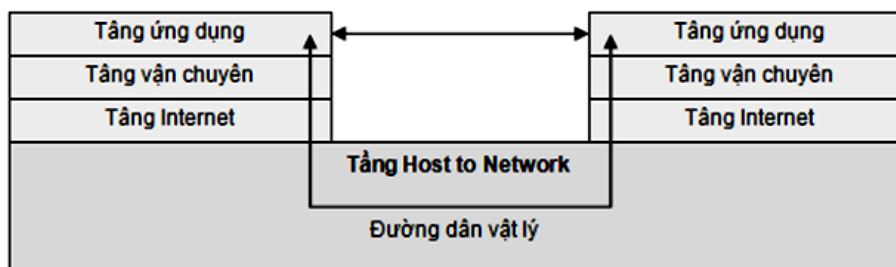
Tầng giao vận (Transport): cho phép truyền dữ liệu với độ tin cậy cao chẳng hạn gói tin gửi đi phải được xác thực hay có thông điệp truyền lại nếu dữ liệu bị hỏng hay bị thất lạc, hay dữ liệu bị trùng lặp.

Tầng mạng (Network): cho phép truy xuất tới các nút trong mạng bằng cách sử dụng địa chỉ logic được sử dụng để kết nối tới các nút khác. Các địa chỉ MAC của tầng 2 chỉ có thể được sử dụng trong một mạng LAN, nên địa chỉ IP được sử dụng để đánh địa chỉ của tầng 3 khi truy xuất tới các nút trong mạng WAN. Các router ở tầng 3 được sử dụng để định đường đi trong mạng.

Tầng liên kết dữ liệu (Data Link): truy xuất tới một mạng vật lý bằng các địa chỉ vật lý. Địa chỉ MAC là địa chỉ của tầng 2. Các nút trên LAN gửi thông điệp cho nhau bằng cách sử dụng các địa chỉ IP, và các địa chỉ này phải được chuyển đổi sang các địa chỉ MAC tương ứng. Giao thức phân giải địa chỉ (ARP: Address Resolution Protocol) chuyển đổi địa chỉ IP thành địa chỉ MAC. Một vùng nhớ cache lưu trữ các địa chỉ MAC để tăng tốc độ xử lý này, và có thể kiểm tra bằng tiện ích `arp -a`,

Cuối cùng, tầng vật lý (Physical): bao gồm môi trường vật lý như các yêu cầu về cáp nối, các thiết bị kết nối, các đặc tả Giao diện, hub và các repeater,...

Mô hình TCP/IP: Mô hình ISO phức tạp nên ít được sử dụng nhiều trong thực tế. Mô hình TCP/IP đơn giản và thích hợp được sử dụng cho mạng Internet. Mô hình này gồm có bốn tầng.



Hình 1.2: Mô hình TCP/IP

Tầng ứng dụng: các phần mềm ứng dụng mạng như trình duyệt Web (IE, Firefox, ..), game online, chat, ... sử dụng các giao thức như HTTP (web), SMTP, POP, IMAP (email), FTP, FSP, TFTP (chuyển tải tập tin), NFS (truy cập tập tin), ...

Tầng vận chuyển: đảm bảo vận chuyển gói tin tin cậy bằng cách sử dụng giao thức TCP hoặc UDP.

Tầng Internet: cho phép định tuyến để gửi gói tin tới đích trên mạng bằng cách sử dụng giao thức IP.

Tầng truy cập mạng: vận chuyển dữ liệu thông qua thiết bị vật lý như dây cáp quan đến tầng này của hệ thống ở xa.

Mô hình TCP/IP đơn giản hơn mô hình OSI là nó nhóm các khâu xử lý thuộc về lập trình ứng dụng thành một nhóm (tầng ứng dụng, tầng trình bày, tầng phiên (OSI) → tầng ứng dụng (TCP/IP), tầng liên kết dữ liệu và tầng vật lý (OSI) → tầng truy cập mạng (TCP/IP)).

1.2 CÁC GIAO THỨC MẠNG

Các giao thức biểu diễn khuôn dạng thông tin dữ liệu tại mỗi mức Giao tiếp trong mạng. Giao thức thường được phân loại theo mức độ áp dụng tại mỗi tầng trong các mô hình mạng. Tại mỗi tầng xử lý, các giao thức lại được phân loại dựa vào chức năng. Ví dụ giao thức ở tầng phiên trong mô hình OSI, gồm HTTP (web), SMTP, POP, IMAP (email), FTP, FSP, TFTP (chuyển tải tập tin), NFS (truy cập tập tin), ...

1.2.1 TCP-Transmission Control Protocol

Giao thức TCP được sử dụng ở tầng vận chuyển (OSI) đảm bảo cho dữ liệu gửi đi tin cậy và xác thực giữa các nút mạng. Giao thức TCP phân chia dữ liệu thành các gói tin gọi là datagram. TCP gắn thêm phần header vào datagram. Phần head được mô tả trong hình vẽ bên dưới.

Source port			Destination port		
Sequence Number					
Acknowledge Number					
Offset	Reserved		Flags	Window	
Checksum			Urgent pointer		
Options				Padding	
Start of Data					

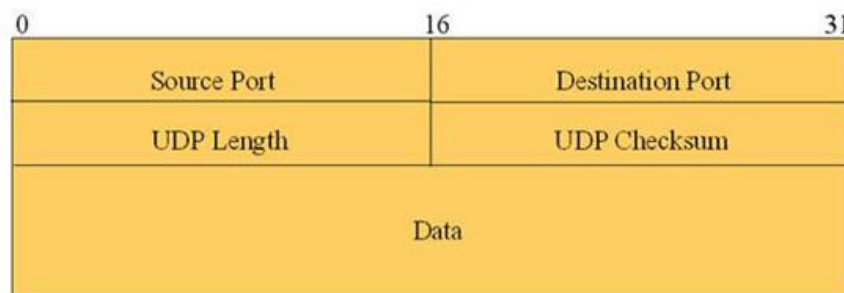
Hình 1.3: Khuôn dạng Header của gói tin TCP**Bảng 1.1: Mô tả chi tiết các thuộc tính của TCP Header**

Trường	Mô tả
Cổng nguồn (source port)	Số hiệu cổng của nguồn
Cổng đích (destination port)	Số hiệu cổng đích
Số thứ tự (Sequence Number)	Số thứ tự được tạo ra bởi nguồn và được sử dụng bởi đích để sắp xếp lại các gói tin để tạo ra thông điệp ban đầu, và gửi xác thực tới nguồn.
Acknowledge Number	Cho biết dữ liệu được nhận thành công.
Data offset	Các chi tiết về nơi dữ liệu gói tin bắt đầu
Reserved	Dự phòng
Flags	chỉ ra rằng gói tin cuối cùng hoặc gói khẩn cấp
Window	chỉ ra kích thước của vùng đệm nhận. Phía nhận có thể thông báo cho phía gửi kích thước dữ liệu tối đa mà có thể được gửi đi bằng cách sử dụng các thông điệp xác thực

Checksum	xác định xem gói tin có bị hỏng không
Urgent Pointer	thông báo cho phía nhận biết có dữ liệu khẩn
Options	vùng dự phòng cho việc thiết lập trong tương lai
Padding	chỉ ra rằng dữ liệu kết thúc trong vòng 32 bit.

1.2.2 UDP-User Datagram Protocol

Giao thức UDP này được sử dụng ở tầng giao vận (OSI) thực hiện cơ chế không liên kết. Nó không cung cấp dịch vụ tin cậy như TCP. UDP sử dụng IP để phát tán các gói tin này.



Hình 1.4: Khuôn dạng UDP Header.

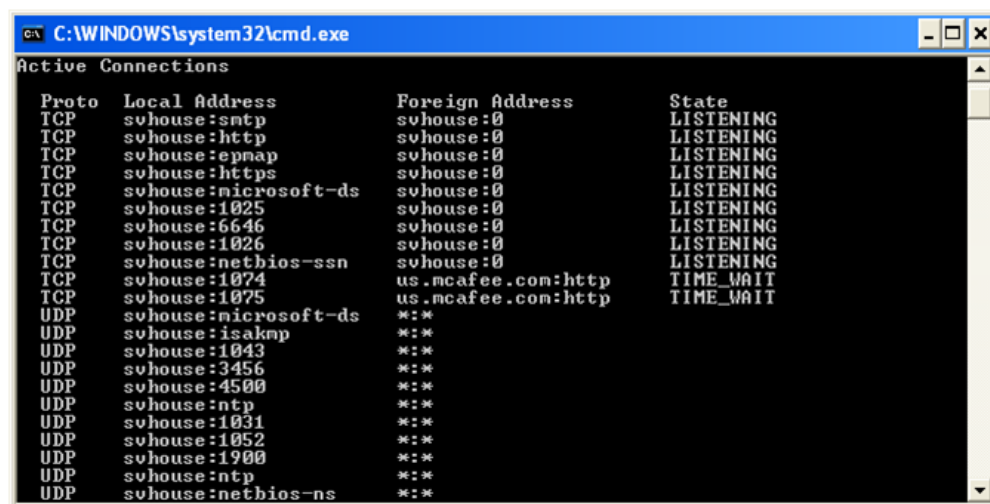
Nhược điểm của UDP là: Các thông điệp có thể được nhận theo bất kỳ thứ tự nào. Thông điệp được gửi đầu tiên có thể được nhận sau cùng. Không có gì đảm bảo là các gói tin sẽ đến đích, và các thông điệp có thể bị thất lạc, hoặc thậm chí có thể nhận được hai bản sao của cùng một thông điệp.

Ưu điểm của UDP là: UDP một giao thức có tốc độ truyền tin nhanh vì nó chỉ xác định cơ chế tối thiểu để truyền dữ liệu. Cụ thể là trong cách thức truyền tin unicast, broadcast và multicast. Một thông điệp unicast được gửi từ nút này tới nút khác. Kiểu truyền tin là truyền tin điểm-điểm. Giao thức TCP chỉ hỗ trợ truyền tin unicast. Truyền tin broadcast nghĩa là một thông điệp có thể được gửi tới tất cả các nút trong một

mạng. Multicast cho phép các thông điệp được truyền tới một nhóm các nút được lựa chọn. UDP có thể được sử dụng cho truyền tin unicast nếu cần tới tốc độ truyền tin nhanh, như truyền tin đa phương tiện, nhưng ưu điểm chính của UDP là truyền tin broadcast và truyền tin multicast. Vì nếu dùng giao thức TCP thì tất cả các nút gửi về các xác thực cho server sẽ làm cho server quá tải.

Bảng 1.2: Mô tả chi tiết các thuộc tính của UDP Header

Trường thông tin	Mô tả
Source port (Cổng nguồn)	Xác định cổng nguồn là một tùy chọn với UDP. Nếu trường này được sử dụng, phía nhận thông điệp có thể gửi một phúc đáp tới cổng này
Destination Port	Số hiệu cổng đích
Length	Chiều dài của thông điệp bao gồm header và dữ liệu
Checksum	Để kiểm tra tính đúng đắn



Hình 1.5: Minh họa lệnh netstat để xem thông tin các cổng đang sử dụng

Số hiệu cổng: Các số hiệu cổng của TCP và UDP được phân thành ba loại: Các số hiệu cổng hệ thống; Các số hiệu cổng người dung; Các số hiệu cổng riêng và động. Các số hiệu cổng hệ thống nằm trong khoảng từ 0 đến 1023. Các cổng hệ thống chỉ

được sử dụng bởi các tiến trình được quyền ưu tiên của hệ thống. Các giao thức nổi tiếng có các số hiệu cổng nằm trong khoảng này. Các số hiệu cổng người dùng nằm trong khoảng từ 1024 đến 49151. Các ứng dụng server của bạn sẽ nhận một trong các số này làm cổng, hoặc bạn có thể đăng ký số hiệu cổng với IANA .

Các cổng động nằm trong khoảng từ 49152 đến 65535. Khi không cần thiết phải biết số hiệu cổng trước khi khởi động một ứng dụng, một số hiệu cổng trong khoảng này sẽ là thích hợp. Các ứng dụng client kết nối tới server có thể sử dụng một cổng như vậy.

Nếu chúng ta sử dụng tiện ích netstat với tùy chọn -a, chúng ta sẽ thấy một danh sách tất cả các cổng hiện đang được sử dụng, nó cũng chỉ ra trạng thái của liên kết- nó đang nằm trong trạng thái lắng nghe hay liên kết đã được thiết lập.

1.2.3 IP-Internet Protocol

Giao thức IP được thiết kế để định tuyến truyền gói tin trong mạng từ nút nguồn tới nút đích. Mỗi nút được định danh bởi một địa chỉ IP (32 bit). Khi nhận gói dữ liệu ở tầng trên (như theo khuôn dạng TCP hoặc UDP), giao thức IP sẽ thêm vào trường header chứa thông tin của nút đích.

Version	IHL	TOS	Total length	
Identification			Flags	Fragmentation Offset
Time to Live	Protocol		Header Checksum	
TCP Header				
Start of Data				

Hình 1.6: Khuôn dạng Header của gói tin IP

Bảng 1.3: Mô tả chi tiết các thuộc tính của IP Header

Trường	Mô tả
Version (Phiên bản IP)	Phiên bản IP. (Phiên bản giao thức hiện nay là IPv4)

IP Header Length (Chiều dài Header)	Chiều dài của header.
Type of Service (Kiểu dịch vụ)	Kiểu dịch vụ cho phép một thông điệp được đặt ở chế độ thông lượng cao hay bình thường, thời gian trễ là bình thường hay lâu, độ tin cậy bình thường hay cao. Điều này có lợi cho các gói được gửi đi trên mạng. Một số kiểu mạng sử dụng thông tin này để xác định độ ưu tiên
Total Length (Tổng chiều dài)	Hai byte xác định tổng chiều dài của thông điệp-header và dữ liệu. Kích thước tối đa của một gói tin IP là 65,535, nhưng điều này là không thực tế đối với các mạng hiện nay. Kích thước lớn nhất được chấp nhận bởi các host là 576 bytes. Các thông điệp lớn có thể phân thành các đoạn-quá trình này được gọi là quá trình phân đoạn
Identification (Định danh)	Nếu thông điệp được phân đoạn, trường định danh trợ giúp cho việc lắp ráp các đoạn thành một thông điệp. Nếu một thông điệp được phân thành nhiều đoạn, tất cả các đoạn của một thông điệp có cùng một số định danh.
Flags	Các cờ này chỉ ra rằng thông điệp có được phân đoạn hay không, và liệu gói tin hiện thời có phải là đoạn cuối cùng của thông điệp hay không.
Fragment Offset	13 bit này xác định offset của một thông điệp. Các đoạn có thể đến theo một thứ tự khác với khi gửi, vì vậy trường offset là cần thiết để xây dựng lại dữ liệu ban đầu. Đoạn đầu tiên của một thông điệp có offset là 0

Time to Live	Xác định số giây mà một thông điệp tồn tại trước khi nó bị loại bỏ.
Protocol	Byte này chỉ ra giao thức được sử dụng ở mức tiếp theo cho thông điệp này. Các số giao thức
Header Checksum	Đây là chỉ là checksum của header. Bởi vì header thay đổi với từng thông điệp mà nó chuyển tới, checksum cũng thay đổi.
Source Address	Cho biết địa chỉ IP 32 bit của phía gửi
Destination Address	Địa chỉ IP 32 bit của phía nhận
Options	
Padding	

1.2.4 Địa chỉ IP

IPv4-32 bit được dùng để định danh mỗi nút trên mạng TCP/IP. Thông thường một địa chỉ IP được biểu diễn bởi bốn phần x.x.x.x, chẳng hạn 192.168.0.1. Mỗi phần là một số có giá trị từ 0 đến 255. Một địa chỉ IP gồm hai phần: phần mạng và phần host.

Bảng 1.4: Phân lớp dải địa chỉ IP

Lớp	Byte 1	Byte 2	Byte 3	Byte 4
A (0)	Networks (1-126)	Host (0-255)	Host (0-255)	Host (0-255)
B (10)	Networks(128-191)	Networks (0-255)	Host (0-255)	Host (0-255)
C (110)	Networks(192-223)	Networks(0-255)	Networks(0-255)	Host (0-255)

Bảng 1.5: Dải địa chỉ IP dành riêng cho mạng nội bộ

Lớp	Khoảng địa chỉ riêng
A	10
B	172.16-172.31
C	192.168.0-192.168.255

Trong đó địa chỉ lớp D(1110) được sử dụng cho địa chỉ multicast. Địa chỉ dự phòng (01111111). Địa chỉ 127.0.0.1 là địa chỉ của localhost, và địa chỉ 127.0.0.0 là địa chỉ loopback.

Để tránh cạn kiệt các địa chỉ IP, các host không được kết nối trực tiếp với Internet có thể sử dụng một địa chỉ trong các khoảng địa chỉ riêng. Các địa chỉ IP riêng không duy nhất về tổng thể, mà chỉ duy nhất về mặt cục bộ trong phạm vi mạng đó. Tất cả các lớp mạng dự trữ các khoảng nhất định để sử dụng như là các địa chỉ riêng cho các host không cần truy cập trực tiếp tới Internet. Các host như vậy vẫn có thể truy cập Internet thông qua một gateway mà không cần chuyển tiếp các địa chỉ IP riêng.

IPv6 được sử dụng 128 bit để biểu diễn địa chỉ nhằm biểu diễn nhiều hơn số địa chỉ của nút trên mạng.

1.2.5 Dịch vụ tên miền

Địa chỉ IP được viết dưới dạng 4 nhóm con số nên người sử dụng rất khó nhớ. Vì vậy hệ thống tên miền được sử dụng để hỗ trợ cho người sử dụng. Các máy tính chuyên dụng để lưu trữ và phân giải tên miền bằng cách lưu danh sách địa chỉ IP và tên miền được gọi là Máy chủ DNS. Ví dụ www.microsoft.com, www.bbc.co.uk. Các tên này không bắt buộc phải có ba phần, nhưng việc đọc bắt đầu từ phải sang trái, tên bắt đầu với miền mức cao. Các miền mức cao là các tên nước cụ thể hoặc tên các tổ chức và được định nghĩa bởi tổ chức IANA.

Các server tên miền

Các tên miền được phân giải bằng cách sử dụng các máy chủ tên miền DNS (Domain Name Service). Các máy chủ này có một cơ sở dữ liệu các tên miền và các bí

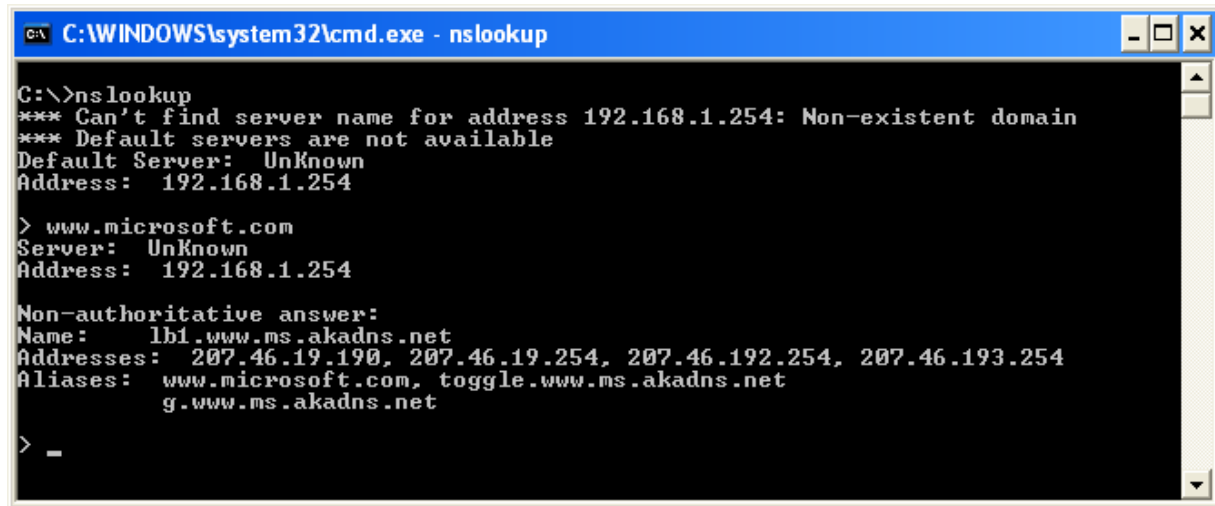
danh ánh xạ các tên thành địa chỉ IP. Ngoài ra, các DNS cũng đăng ký thông tin cho các Mail Server, các số ISDN, các tên hòm thư, và các dịch vụ.

Trong Windows, chính các thiết lập TCP/IP xác định máy chủ DNS được sử dụng để truy vấn. Lệnh `ipconfig/all` chỉ ra các máy chủ DNS đã được thiết lập và các thiết lập cấu hình khác. Khi kết nối với một hệ thống ở xa sử dụng tên miền, trước tiên, máy chủ DNS cục bộ được truy vấn để tìm địa chỉ IP. Nếu thất bại trong việc phân giải tên, máy chủ DNS sẽ truy vấn máy chủ DNS gốc.

Lệnh `Nslookup`: Dịch vụ tên miền (Domain Name Service) là tập hợp nhiều máy tính được liên kết với nhau và phân bố rộng trên mạng Internet. Các máy tính này được gọi là máy chủ tên miền. Chúng cung cấp cho máy khách tên, địa chỉ IP của bất kỳ máy tính nào nối vào mạng Internet hoặc tìm ra những máy chủ tên miền có khả năng cung cấp thông tin này. Cơ chế truy tìm địa chỉ IP thông qua dịch vụ DNS:

Giả sử trình duyệt cần tìm tập tin hay trang Web của một máy chủ nào đó, khi đó cơ chế truy tìm địa chỉ sẽ diễn ra như sau:

- Trình duyệt yêu cầu hệ điều hành trên client chuyển hostname thành địa chỉ IP.
- Client truy tìm xem hostname có được ánh xạ trong tập tin `localhost`, `hosts` hay không?
- Nếu có client chuyển đổi hostname thành địa chỉ IP và gửi về cho trình duyệt.
- Nếu không client sẽ tìm cách liên lạc với máy chủ DNS.
- Nếu tìm thấy địa chỉ IP của hostname máy chủ DNS sẽ gửi địa chỉ IP cho client.
- Client gửi địa chỉ IP cho trình duyệt.
- Trình duyệt sử dụng địa chỉ IP để liên lạc với Server.
- Quá trình kết nối thành công. Máy chủ gửi thông tin cho client.



```
C:\WINDOWS\system32\cmd.exe - nslookup

C:\>nslookup
*** Can't find server name for address 192.168.1.254: Non-existent domain
*** Default servers are not available
Default Server: UnKnown
Address: 192.168.1.254

> www.microsoft.com
Server: UnKnown
Address: 192.168.1.254

Non-authoritative answer:
Name: lb1.www.ms.akadns.net
Addresses: 207.46.19.190, 207.46.19.254, 207.46.192.254, 207.46.193.254
Aliases: www.microsoft.com, toggle.www.ms.akadns.net
          g.www.ms.akadns.net

> _
```

Hình 1.7: Minh họa dùng lệnh nslookup để tìm địa chỉ IP

CÂU HỎI ÔN TẬP

1. Trình bày vai trò của phần mềm ứng dụng mạng. Mô hình khách/chủ là gì? Lấy ví dụ minh họa?
2. Cho biết nhiệm vụ của mỗi khâu xử lý trong mô hình OSI?
3. Tìm hiểu các giao thức chuẩn hóa phổ biến khác ở các tầng khác nhau của mô hình OSI ?
4. Tìm hiểu các giao thức được sử dụng phổ biến trong các ứng dụng trên mạng Internet. Cho biết số hiệu cổng của các ứng dụng này. Liệt kê các ứng dụng thực tế sử dụng giao thức này. Liệt kê các phần mềm nguồn mở đang phát triển sử dụng các giao thức này.

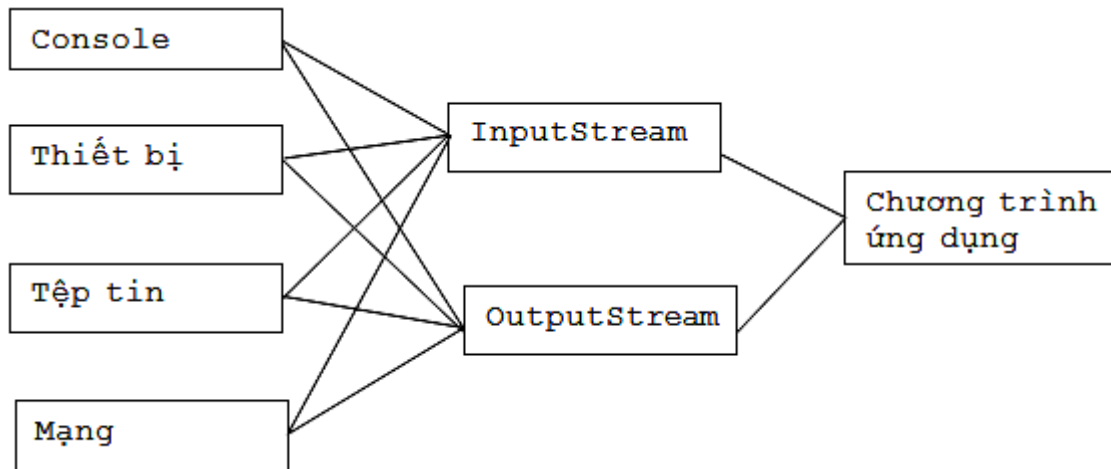
BÀI 2. QUẢN LÝ CÁC LUỒNG NHẬP XUẤT

2.1 GIỚI THIỆU

Một chương trình máy tính cần phải lấy dữ liệu từ bên ngoài vào chương trình để xử lý. Sau khi chương trình xử lý xong sẽ xuất kết quả ra ngoài cho thành phần khác để tiếp tục xử lý hoặc lưu trữ. Dữ liệu được truyền từ bộ nhớ của chương trình này sang bộ nhớ của chương trình khác bằng các luồng dữ liệu. Để cài đặt việc truyền dữ liệu này, Java cung cấp package `java.io` với rất nhiều lớp và hỗ trợ nhiều tính năng. Xét theo loại dữ liệu, Java chia các luồng thành hai loại là luồng byte (`byte stream`) và luồng ký tự (`character stream`). Giữa hai loại dữ liệu này có các lớp trung gian để phục vụ cho việc chuyển đổi. Xét theo thao tác, Java chia các luồng thành hai loại là `input` (đọc byte) và `output` (ghi byte) hoặc `reader` (đọc character) và `writer` (ghi character). Ngoài ra, Java còn hỗ trợ rất nhiều luồng hỗ trợ cho việc đọc ghi khác.

Bảng 2.1: các kiểu luồng dữ liệu trong java

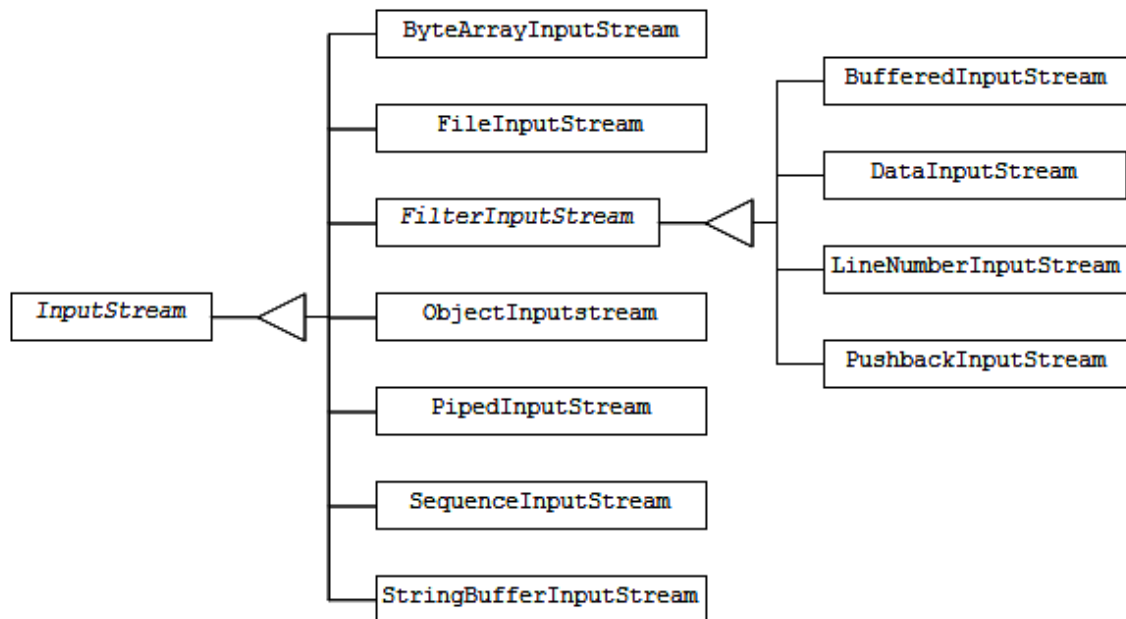
	Byte streams	Character streams
Source streams	InputStream	Reader
Sink streams	OutputStream	Writer



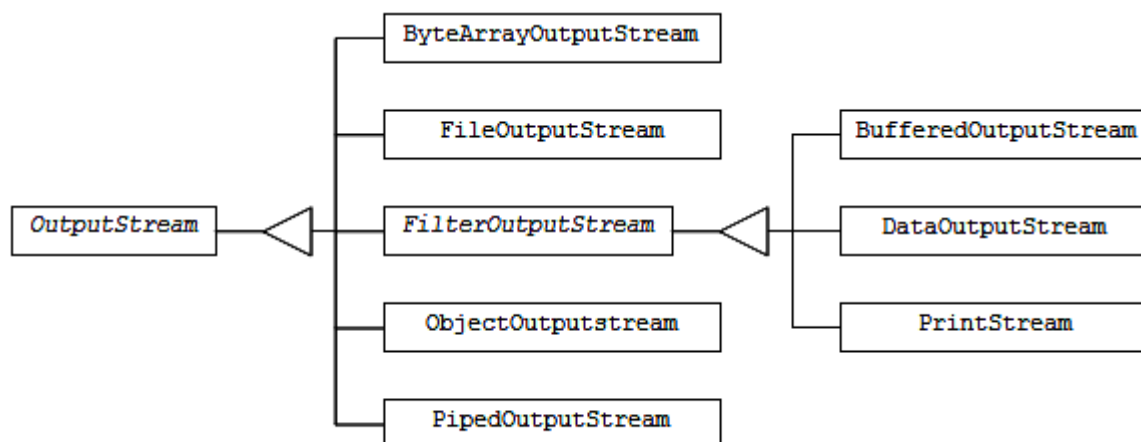
Hình 2.1: Minh họa hoạt động của luồng nhập/xuất cơ bản

2.2 CÁC LƯỒNG BYTE

Các luồng byte có chức năng đọc hoặc ghi dữ liệu dạng byte. Dựa vào thao tác, các luồng byte được chia ra hai loại chính: nhóm input được đại diện bởi lớp `InputStream` và nhóm output được đại diện bởi `OutputStream`. Các lớp xử lý trên luồng byte đều được kế thừa từ hai lớp này.



Hình 2.2: Mô hình kế thừa của các kiểu luồng byte nhập dữ liệu.



Hình 2.3: Mô hình kế thừa của các kiểu luồng byte xuất dữ liệu.

2.2.1 Các luồng byte tổng quát

Gồm các lớp InputStream và OutputStream. Đây hai là lớp trừu tượng, nó quy định các phương thức đọc và ghi dữ liệu dạng byte và một số phương thức hỗ trợ khác:

Bảng 2.2: Các phương thức cơ bản của các byte stream tổng quát

Phương thức	Ý nghĩa
Lớp InputStream	
int available()	Trả về số byte còn lại trong stream.
void close()	Đóng stream.
void mark(int readlimit)	Đánh dấu vị trí hiện tại. Nếu sau khi mark ta đọc quá readlimit byte thì chỗ đánh dấu không còn hiệu lực.
boolean markSupported()	Trả về true nếu stream hỗ trợ mark và reset.
abstract int read()	<p>Đọc byte kế tiếp trong stream.</p> <p>Quy ước: Trả về -1 nếu hết stream.</p> <p>Đây là phương thức trừu tượng.</p>

<code>int read(byte[] buffer)</code>	<p>Đọc một dãy byte và lưu kết quả trong mảng buffer. Số byte đọc được tối đa là sức chứa (length) của buffer. Nếu buffer có sức chứa là 0 thì sẽ không đọc được gì.</p> <p>Sau khi đọc xong sẽ trả về số byte đọc được thật sự (nếu đang đọc mà hết stream thì số byte đọc được thật sự sẽ nhỏ hơn sức chứa của buffer).</p> <p>Nếu stream đã hết mà vẫn đọc nữa thì kết quả trả về là -1.</p>
<code>int read(byte[] buffer, int offset, int len)</code>	<p>Đọc một dãy byte và lưu kết quả trong mảng buffer kể từ byte thứ offset. Số byte đọc được tối đa là len. Nếu len là 0 thì sẽ không đọc được gì.</p> <p>Sau khi đọc xong sẽ trả về số byte đọc được thật sự (nếu đang đọc mà hết stream thì số byte đọc được thật sự sẽ nhỏ hơn len).</p> <p>Nếu stream đã hết mà vẫn đọc nữa thì kết quả trả về là -1.</p>
<code>void reset()</code>	Trở lại chỗ đã đánh dấu bằng phương thức mark().
<code>long skip(long n)</code>	Bỏ qua n byte (để đọc các byte tiếp sau n byte đó).
Lớp OutputStream	
<code>void close()</code>	Đóng stream.
<code>void flush()</code>	Buộc stream ghi hết dữ liệu trong vùng đệm ra ngoài.
<code>void write(byte[] buffer)</code>	Ghi một dãy byte vào stream. Số byte được ghi sẽ là buffer.length.
<code>int write(byte[] buffer, int offset, int len)</code>	Ghi một dãy byte vào stream. Bắt đầu ghi từ byte thứ offset, và ghi tổng cộng len byte.

abstract void write(int b)	Ghi một byte vào stream. Đây là phương thức trừu tượng.
----------------------------	--

Lưu ý: Các phương thức đọc/ghi sẽ phát sinh IOException nếu có lỗi xảy ra.

2.2.2 Các luồng đọc byte hiện thực

Để sử dụng đọc/ghi luồng dữ liệu dạng byte, các lớp con hiện thực của InputStream và OutputStream thường được sử dụng gồm:

- FileInputStream: đọc (các) byte từ tập tin.
- FileOutputStream: ghi (các) byte vào tập tin.
- ByteArrayInputStream: chứa bộ đệm là mảng byte để đọc dữ liệu.
- ByteArrayOutputStream: chứa bộ đệm là mảng byte để ghi dữ liệu.
- PipedInputStream: đọc (các) byte từ một piped output stream.
- PipedOutputStream: ghi (các) byte vào một piped input stream.

Nhìn chung các lớp này đều có những chức năng chính tương tự nhau. Dưới đây chỉ trình bày những phương thức đặc thù của chúng.

Bảng 2.3: Những phương thức đặc trưng của các byte stream cụ thể

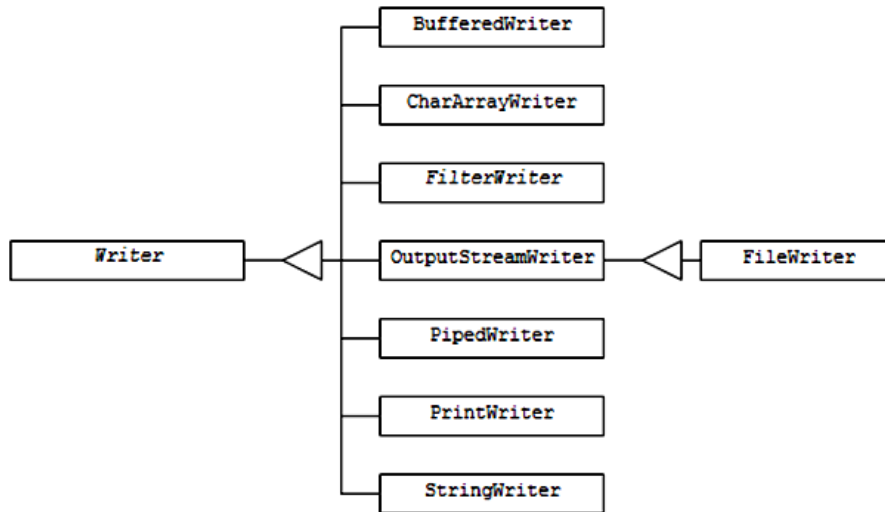
Stream	Phương thức	Ý nghĩa
File Input Stream	FileInputStream(File file)	Tạo FileInputStream để đọc dữ liệu từ một tập tin liên kết tới đối tượng File.
	FileInputStream(String name)	Tạo FileInputStream để đọc dữ liệu từ một tập tin có tên là name.
File Output	FileOutputStream(File file)	Tạo FileOutputStream để ghi dữ liệu vào một tập tin liên kết tới đối tượng File.
	FileOutputStream(File file, ...)	Tạo FileOutputStream để ghi dữ liệu tập tin

Stream	boolean append)	liên kết tới đối tượng File. Nếu append là true thì sẽ ghi tiếp vào cuối tập tin, ngược lại thì ghi đè lên tập tin.
	FileOutputStream(String name)	Tạo FileOutputStream để ghi dữ liệu vào một tập tin có tên là name.
	FileOutputStream(String name, boolean append)	Tạo FileOutputStream để ghi dữ liệu vào một tập tin có tên là name. Nếu append là true thì sẽ ghi tiếp vào cuối tập tin, ngược lại thì ghi đè lên tập tin.
Byte Array	ByteArrayInputStream(byte[] buf)	Tạo ra ByteArrayInputStream và dùng buf để làm vùng đệm.
Input Stream	ByteArrayInputStream(byte[] buf, int off, int len)	Tạo ra ByteArrayInputStream và dùng một phần của buf để làm vùng đệm.
Byte Array Output Stream	ByteArrayOutputStream()	Tạo ra một ByteArrayOutputStream với vùng đệm 32 byte và có thể tăng nếu cần.
	ByteArrayOutputStream(int size)	Tạo ra một ByteArrayOutputStream với vùng đệm size byte.
	byte[]toByteArray()	Tạo ra mảng byte là bản sao của vùng đệm của this.
	String toString()	Tạo ra chuỗi là bản sao của vùng đệm của this với các byte được đổi thành ký tự tương ứng.
	void writeTo(OutputStream out)	Ghi dữ liệu trong vùng đệm vào một output stream khác.

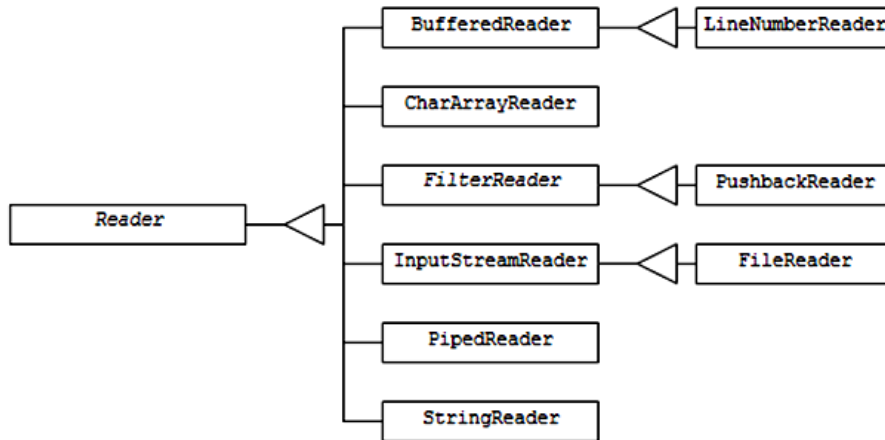
Piped Input Stream	PipedInputStream()	Tạo ra một piped input stream. Stream này chưa được kết nối với piped output stream nào.
	PipedInputStream (PipedOutputStream src)	Tạo ra một piped input stream kết nối với piped output stream src.
	connect(PipedOutputStream src)	Kết nối tới piped output stream src.
Piped Output Stream	PipedOutputStream()	Tạo ra một piped output stream. Stream này chưa được kết nối với piped input stream nào.
	PipedOutputStream (PipedInputStream snk)	Tạo ra một piped output stream kết nối với piped input stream snk.
	connect(PipedInputStream snk)	Kết nối tới piped input stream snk.

2.3 CÁC LUỒNG KÝ TỰ

Các luồng ký tự có chức năng đọc hoặc ghi dữ liệu dạng ký tự. Dựa vào thao tác, các luồng ký tự được chia ra hai loại chính: nhóm nhập được đại diện bởi lớp Reader và nhóm xuất được đại diện bởi Writer.



Hình 2.4: Mô hình kiến trúc kế thừa của các luồng ký tự nhập dữ liệu.



Hình 2.5: Mô hình kiến trúc kế thừa của các luồng ký tự nhập dữ liệu.

2.3.1 Các luồng ký tự tổng quát

Gồm các lớp Reader và Writer. Đây hai là lớp trừu tượng, nó quy định các phương thức đọc và ghi dữ liệu dạng character và một số phương thức hỗ trợ khác:

Bảng 2.4: Các phương thức cơ bản của character stream tổng quát

Phương thức	Ý nghĩa
Lớp Reader	
void close()	Đóng stream.

<code>void mark(int readlimit)</code>	Đánh dấu vị trí hiện tại. Nếu sau khi đánh dấu ta đọc quá <code>readlimit</code> ký tự thì chỗ đánh dấu không còn hiệu lực.
<code>boolean markSupported()</code>	Trả về <code>true</code> nếu stream hỗ trợ <code>mark</code> và <code>reset</code> .
<code>abstract int read()</code>	Đọc ký tự kế tiếp trong stream. Quy ước: Trả về -1 nếu hết stream. Đây là phương thức trừu tượng.
<code>int read(char[]cbuf)</code>	Đọc một dãy ký tự và lưu kết quả trong mảng <code>cbuf</code> . Sau khi đọc xong sẽ trả về số ký tự đọc được thật sự (nếu đang đọc mà hết stream thì số ký tự đọc được thật sự sẽ nhỏ hơn sức chứa của <code>cbuf</code>). Nếu stream đã hết mà vẫn đọc nữa thì kết quả trả về là -1.
<code>int read(char[]cbuf, int offset, int len)</code>	Đọc một dãy ký tự và lưu kết quả trong mảng <code>cbuf</code> kể từ ký tự thứ <code>offset</code> . Sau khi đọc xong sẽ trả về số ký tự đọc được thật sự (nếu đang đọc mà hết stream thì số ký tự đọc được thật sự sẽ nhỏ hơn <code>len</code>). Nếu stream đã hết mà vẫn đọc nữa thì kết quả trả về là -1.
<code>void read(CharBuffer target)</code>	Đọc một dãy ký tự và lưu kết quả trong <code>target</code> .
<code>boolean ready()</code>	Trả về <code>true</code> nếu stream sẵn sàng để đọc.
<code>void reset()</code>	Trở lại chỗ đã đánh dấu bằng phương thức <code>mark()</code> .

long skip(long n)	Bỏ qua n ký tự (để đọc các ký tự tiếp sau).
Lớp Writer	
Writer append(char c)	Nối đuôi ký tự c vào stream.
Writer append(CharSequence csq)	Nối đuôi dãy ký tự csq vào stream.
Writer append(CharSequence csq, int start, int end)	Nối đuôi một phần của dãy ký tự csq vào stream.
void close()	Đóng stream.
void flush()	Buộc stream ghi hết dữ liệu trong vùng đệm ra ngoài.
void write(char[] cbuf)	Ghi một mảng ký tự vào stream. Số ký tự được ghi sẽ là cbuffer.length.
abstract void write(int c)	Ghi một ký tự vào stream. Đây là phương thức trừu tượng.
void write(String c)	Ghi một chuỗi vào stream.
void write(String str, int off, int len)	Ghi một phần của chuỗi vào stream.

Lưu ý: Các phương thức đọc/ghi sẽ phát sinh IOException nếu có lỗi xảy ra.

2.3.2 Các luồng ký tự hiện thực

Để sử dụng đọc/ghi ký tự, ta phải dùng các lớp con của Reader và Writer. Các lớp thường dùng gồm:

FileReader: đọc (các) ký tự từ tập tin.

FileWriter: ghi (các) ký tự vào tập tin.

CharArrayReader: chứa bộ đệm là mảng ký tự để đọc dữ liệu.

CharArrayWriter: chứa bộ đệm là mảng ký tự để ghi dữ liệu.

PipedReader: đọc (các) ký tự từ một piped writer.

PipedWriter: ghi (các) ký tự vào một piped reader.

StringReader: đọc chuỗi ký tự.

StringWriter: ghi chuỗi ký tự.

Nhìn chung các lớp này đều có những chức năng chính tương tự như nhau. Dưới đây chỉ trình bày những phương thức đặc thù của chúng.

Bảng 2.5: Những phương thức đặc trưng của các character stream cụ thể

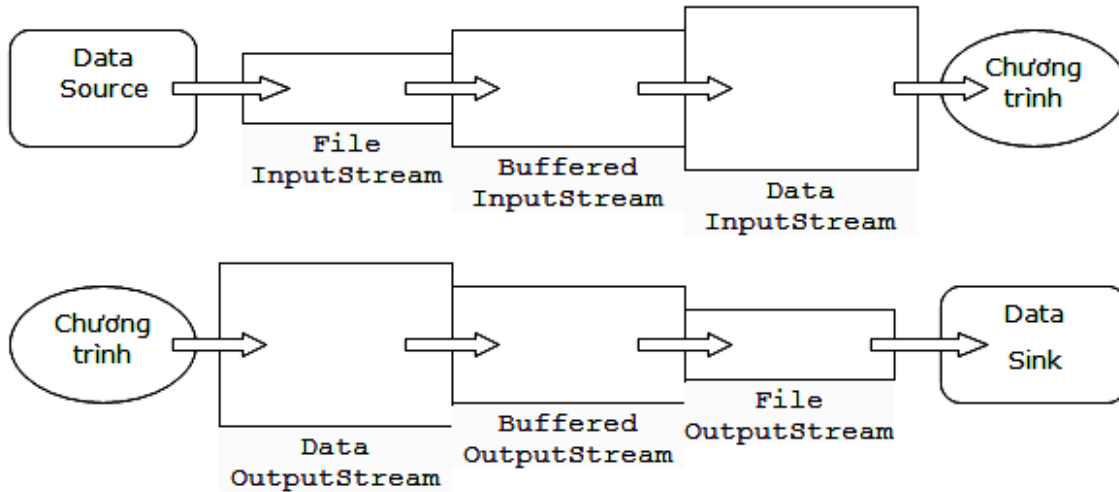
Stream	Phương thức	Ý nghĩa
File Reader	FileReader(File file)	Tạo FileReader để đọc dữ liệu từ một tập tin liên kết tới đối tượng File.
	FileReader(String name)	Tạo FileReader để đọc dữ liệu từ một tập tin có tên là name.
File Writer	FileWriter (File file)	Tạo FileWriter để ghi dữ liệu vào một tập tin liên kết tới đối tượng File.
	FileWriter(File file, boolean append)	Tạo FileWriter để ghi dữ liệu tập tin liên kết tới đối tượng File. Nếu append là true thì sẽ ghi tiếp vào cuối tập tin, ngược lại thì ghi đè lên tập tin.
	FileWriter(String name)	Tạo FileWriter để ghi dữ liệu vào một tập tin có tên là name.

	<code>FileWriter(String name, boolean append)</code>	<p>Tạo <code>FileWriter</code> để ghi dữ liệu vào một tập tin có tên là <code>name</code>.</p> <p>Nếu <code>append</code> là <code>true</code> thì sẽ ghi tiếp vào cuối tập tin, ngược lại thì ghi đè lên tập tin.</p>
Char Array Reader	<code>CharArrayReader(char[] buf)</code>	Tạo ra <code>CharArrayReader</code> và dùng buf để làm vùng đệm.
	<code>CharArrayReader(char[] buf, int off, int len)</code>	Tạo ra <code>CharArrayReader</code> và dùng một phần của buf để làm vùng đệm.
Char Array Writer	<code>CharArrayWriter()</code>	Tạo ra một <code>CharArrayWriter</code> .
	<code>CharArrayWriter(int size)</code>	Tạo ra một <code>CharArrayWriter</code> với vùng đệm size ký tự.
	<code>char[]toCharArray()</code>	Tạo ra mảng ký tự là bản sao của vùng đệm của <code>this</code> .
	<code>String toString()</code>	Tạo ra chuỗi là bản sao của vùng đệm của <code>this</code> với các byte được đổi thành ký tự tương ứng.
	<code>void writeTo(Writer out)</code>	Ghi dữ liệu trong vùng đệm vào một writer khác.
Piped Reader	<code>PipedReader()</code>	Tạo ra một piped reader. Stream này chưa được kết nối với piped output stream nào.
	<code>PipedReader(PipedWriter src)</code>	Tạo ra một piped reader kết nối với piped output stream <code>src</code> .

	<code>connect(PipedWriter src)</code>	Kết nối tới piped writer src.
Piped Writer	<code>PipedWriter()</code>	Tạo ra một piped writer. Stream này chưa được kết nối với piped input stream nào.
	<code>PipedWriter (PipedReader snk)</code>	Tạo ra một piped writer kết nối với piped reader snk.
	<code>connect(PipedReader snk)</code>	Kết nối tới piped reader snk.
String Reader	<code>StringReader(String s)</code>	Tạo ra một StringReader.
String Writer	<code>StringWriter()</code>	Tạo ra một StringWriter, dùng vùng đệm có kích thước mặc định.
	<code>StringWriter(int initialSize)</code>	Tạo ra một StringWriter, dùng vùng đệm có kích thước là initialSize.
	<code>StringBuffer getBuffer()</code>	Trả về vùng đệm của stream.

2.4 CÁC LUỒNG LỌC DỮ LIỆU

Bởi vì các luồng dữ liệu được chuyển tải dạng byte hoặc ký tự, nên cần có một bộ biến đổi các luồng dữ liệu này sang những giá trị có định kiểu khác nhau. Java cung cấp thêm các luồng lọc dữ liệu. Khi các luồng này được tạo ra, nó phải được gắn với một luồng dữ liệu cụ thể để lọc nó. Do vậy, việc đọc dữ liệu từ một nguồn là sự phối hợp giữa các luồng tạo thành một dây chuyền xử lý trên các luồng khi có thao tác đọc/ghi xảy ra. Dây chuyền đọc bắt đầu bởi một luồng đọc cơ bản và kết thúc bởi một luồng lọc dữ liệu. Ở giữa có thể có thêm các luồng xử lý đọc trung gian. Dây chuyền ghi bắt đầu bởi một luồng xử lý ghi và kết thúc bởi một luồng ghi cơ bản. Ở giữa có thể có các luồng xử lý ghi trung gian.



Hình 2.6: Ví dụ về dây chuyền đọc/ghi dữ liệu

2.4.1 Các luồng lọc tổng quát

Trong Java, các luồng xử lý lọc tổng quát là các lớp trừu tượng sau:

`FileInputStream`: luồng xử lý cho thao tác đọc dữ liệu dạng byte.

`FileOutputStream`: luồng xử lý cho thao tác ghi dữ liệu dạng byte.

`FileReader`: luồng xử lý cho thao tác đọc dữ liệu dạng ký tự.

`FileWriter`: luồng xử lý cho thao tác ghi dữ liệu dạng ký tự.

2.4.2 Các luồng lọc hiện thực

Các luồng lọc hiện thực các luồng lọc tổng quát chịu trách nhiệm hiện thực các thao tác biến đổi dữ liệu có định kiểu thành luồng byte/ký tự hoặc ngược lại từ luồng byte/ký tự thành giá trị có định kiểu. Cụ thể, các luồng lọc này là `DataInputStream`, `DataOutputStream`. Các lớp này có những phương thức đặc trưng như trong bảng 2-6.

Bảng 2.6: Các phương thức của các luồng định kiểu

Stream	Phương thức	Ý nghĩa
Data	<code>DataInputStream</code>	Tạo stream để đọc dữ liệu theo định

Input Stream	(InputStream in)	dạng.
	boolean readBoolean()	Đọc một giá trị kiểu boolean. Các kiểu byte, char, double, float,... cũng có những phương thức tương ứng.
Data Output Stream	DataOutputStream (OutputStream out)	Tạo stream để ghi dữ liệu theo định dạng.
	int size()	Trả về số byte mà stream đã ghi.
	Void writeBoolean (boolean v)	Ghi một biến kiểu boolean. Các kiểu byte, char, double, float,... cũng có những phương thức tương ứng.
	void writeBytes(String s)	Ghi chuỗi thành một dãy các byte.
	void writeChars(String s)	Ghi chuỗi thành một dãy các ký tự.

Lưu ý:

Khi đọc các biến, phải đọc đúng theo thứ tự đã ghi.

Để ghi chuỗi hoặc các đối tượng khác, ta phải dùng chức năng đọc/ghi object.

Ta có thể kết hợp đọc có định dạng với vùng đệm.

2.5 CÁC LUỒNG ĐỆM DỮ LIỆU

Để tăng tốc độ đọc/ghi, nhất là đối với thao tác đọc/ghi trên bộ nhớ phụ, người ta dùng kỹ thuật vùng nhớ đệm. Vùng đệm đọc: dữ liệu sẽ được đọc vào vùng đệm thành từng khối, sau đó lấy ra dùng từ từ, khối hết thì sẽ đọc tiếp khối kế → giảm số thao tác đọc. Vùng đệm ghi: dữ liệu cần ghi sẽ được gom lại đến khi đủ số lượng cần thiết thì sẽ được ghi một lần → giảm số thao tác ghi. Như vậy, vùng đệm càng lớn thì càng tăng tốc độ, nhưng sẽ càng dễ xảy ra mất dữ liệu khi chương trình bị chấm dứt đột ngột. Ví dụ: ta yêu cầu ghi nhưng do chưa đủ dữ liệu nên chương trình chưa ghi thật sự vào đĩa, sau đó bị cúp điện → các dữ liệu được ta yêu cầu ghi đó sẽ mất.

Trong Java, các lớp luồng đệm hỗ trợ thao tác đọc/ghi như:

BufferedInputStream: đọc dữ liệu dạng byte, có dùng vùng đệm

BufferedOutputStream: ghi dữ liệu dạng byte, có dùng vùng đệm

BufferedReader: đọc dữ liệu dạng ký tự, có dùng vùng đệm

BufferedWriter: ghi dữ liệu dạng ký tự, có dùng vùng đệm

Khi tạo ra một đối tượng luồng đệm, nó phải gắn với một đối tượng luồng dữ liệu khác. Nhìn chung các luồng đệm có những phương thức giống như luồng thông thường, nhưng có khác biệt về cách khởi tạo.

Bảng 2.7: Các hàm khởi tạo các luồng đệm dữ liệu

Stream	Phương thức	Ý nghĩa
Buffered Input Stream	BufferedInputStream (InputStream in)	Tạo một stream vùng đệm để đọc dữ liệu dạng byte.
	BufferedInputStream (InputStream in, int size)	Tạo một stream với vùng đệm kích thước <code>size</code> để đọc dữ liệu dạng byte.
Buffered Output Stream	BufferedOutputStream (OutputStream out)	Tạo một stream vùng đệm để ghi dữ liệu dạng byte.
	BufferedOutputStream (OutputStream out, int sz)	Tạo một stream với vùng đệm kích thước <code>sz</code> byte để ghi dữ liệu dạng byte.
Buffered Reader	BufferedReader(Reader in)	Tạo một stream vùng đệm để đọc dữ liệu dạng ký tự.
	BufferedReader (Reader in, int sz)	Tạo một stream với vùng đệm kích thước <code>sz</code> để đọc dữ liệu dạng ký tự.
Buffered	BufferedWriter(Writer out)	Tạo một stream vùng đệm để ghi dữ liệu dạng ký tự.

Writer	BufferedWriter (Writer out, int sz)	Tạo một stream với vùng đệm kích thước <code>sz</code> byte để ghi dữ liệu dạng byte.
	void newLine()	Ghi ký tự xuống dòng vào stream.

2.6 CÁC LỚP NHẬP/XUẤT ĐỊNH KIỂU DỮ LIỆU

Java hỗ trợ hai lớp đối tượng Scanner (thuộc gói java.util.*) và PrintWriter (thuộc gói java.io.*) rất hiệu quả cho việc định dạng dữ liệu của một luồng nhập/xuất. Bảng dưới đây trình bày một số phương thức thông dụng của hai lớp đối tượng này.

Phương thức	Ý nghĩa
Lớp Scanner	
Scanner(InputStream source)	Khởi tạo đối tượng Scanner lấy dữ liệu từ một luồng nhập.
Scanner(ReadableByteChannel source)	Khởi tạo đối tượng Scanner lấy dữ liệu từ một kênh nhập.
void close()	Đóng đối tượng Scanner.
bool nextBoolean()	Đọc một khối dữ liệu trong luồng và ép nó sang kiểu logic bool.
int nextInt()	Đọc một khối dữ liệu trong luồng và ép nó sang kiểu số nguyên int.
long nextLong()	Đọc một khối dữ liệu trong luồng và ép nó sang kiểu số nguyên long.
float nextFloat()	Đọc một khối dữ liệu trong luồng và ép nó sang kiểu số thực float.
double nextDouble()	Đọc một khối dữ liệu trong luồng và ép nó sang

	kiểu số thực double.
String nextLine()	Đọc các khối dữ liệu trong luồng và ép nó sang kiểu chuỗi String.
bool hasNext()	kiểm tra luồng còn khối dữ liệu nào không.
Lớp PrintWriter	
PrintWriter(OutputStream out)	Khởi tạo đối tượng PrintWriter để viết dữ liệu ra luồng xuất byte out.
PrintWriter(Writer out)	Khởi tạo đối tượng PrintWriter để viết dữ liệu ra luồng xuất ký tự out.
void close()	Đóng đối tượng PrintWriter.
void print(String s)	Ghi một chuỗi ra luồng.
void print(bool b)	Ghi giá trị bool ra luồng.
void println()	Ghi ký hiệu kết thúc dòng ra luồng
void println(int i)	Ghi một số nguyên i và ký hiệu kết thúc dòng ra luồng.
void println(String s)	Ghi một chuỗi s và ký hiệu kết thúc dòng ra luồng.
void flush()	Ghi toàn bộ dữ liệu trong PrintWriter ra luồng.

CÂU HỎI ÔN TẬP

1. Định nghĩa luồng nhập/xuất? Nhu cầu sử dụng luồng?
2. Phân loại các kiểu luồng? Cách áp dụng ? lấy ví dụ minh họa?

3. Viết chương trình nhập vào tên đường dẫn của một thư mục. Hiển thị tên tất cả tập tin trong thư mục lên màn hình (sử dụng phương thức `fileList` của lớp đối tượng `File` và lớp `FileFilter`).
4. Viết chương trình nhập tên một tập tin. Tách tập tin thành từng đoạn. Mỗi đoạn lưu vào mảng số nguyên có kích thước tối đa là 100 phần tử. Sau đó, đọc nội dung các mảng này ghi vào tập tin mới có tên là "Mang.txt".

BÀI 3. LẬP TRÌNH ĐA TUYẾN

3.1 GIỚI THIỆU

Thông thường, một số chương trình ứng dụng cần xử lý nhiều yêu cầu cùng một lúc (ví dụ chương trình Webserver phải đáp trả nhiều yêu cầu của nhiều máy khách cùng một lúc hoặc một cửa sổ vừa có hình ảnh động vừa cho phép người sử dụng nhập liệu bình thường, chương trình soạn thảo văn bản sẽ vừa cho phép bạn gõ văn bản vừa chạy chức năng kiểm lỗi chính tả ...). Ngôn ngữ Java cho phép lập trình đa tuyến trình. Do vậy, các ứng dụng mạng thực hiện theo cơ chế khách/chủ được cài đặt dễ dàng.

3.1.1 Đơn tiến trình

Một chương trình thực hiện đơn tiến trình thì các câu lệnh của chương trình sẽ được thực hiện tuần tự. Nếu một lệnh không hoàn thành thì lệnh tiếp theo sẽ không được xử lý. Do vậy, trạng thái của chương trình tại thời điểm bất kỳ có thể dự đoán được đang ở thời điểm nào cho trước.

3.1.2 Đa tiến trình

Một chương trình có thực hiện đa tiến trình cho phép nhiều tiến trình thực hiện cùng một lúc. Mỗi tiến trình được xử lý bởi một CPU riêng. Với máy có nhiều CPU (hoặc CPU đa nhân) thì việc xử lý đồng thời là thật nhưng dĩ nhiên số lệnh chạy cùng lúc không thể vượt quá số CPU. Khi đó, việc điều phối xem tiến trình chạy trên CPU nào thông thường sẽ do HĐH quy định. Với CPU (đơn nhân) thì chỉ có thể thực thi lệnh một cách tuần tự, nghĩa là một CPU chỉ chạy một lệnh trong một thời điểm. Vậy làm sao chương trình đa tiến trình có thể chạy trên máy chỉ có 1 CPU ? Câu trả lời chính là phương pháp xoay vòng. Theo đó, hệ điều hành sẽ cấp cho mỗi tiến trình một CPU ảo và một vùng nhớ ảo, khi đó mỗi tiến trình đều "tưởng" rằng chỉ có một mình nó dùng

CPU và bộ nhớ. Còn về thực chất, mỗi tiến trình sẽ dùng CPU trong một khoảng thời gian ngắn rồi trả CPU lại cho tiến trình khác sử dụng. Cách làm này thực chất không phải là đồng thời nhưng do việc "mượn" và "trả" CPU diễn ra rất nhanh và hoàn toàn tự động nên người dùng sẽ không cảm thấy điều gì bất thường.

3.1.3 Tiến trình

Một tuyến trình đoạn có thể ở một trong bốn trạng thái sau trong suốt vòng đời của nó:

- Running: tiến trình đang thực thi.
- Suspended: việc thực thi tiến trình bị tạm dừng và có thể phục hồi để chạy tiếp.
- Blocked: tiến trình cần dùng tài nguyên nhưng tài nguyên đang được tiến trình khác dùng nên nó phải chờ.
- Terminated: việc thực thi của tiến trình bị ngừng hẳn và không thể phục hồi.

3.2 LỚP THREAD

Để lập trình đa tiến trình, lớp dẫn xuất từ lớp Thread được cài đặt sau đó override phương thức run(). Khi tiến trình cần chạy thì phương thức start() của tiến trình được gọi thực hiện.

Xem ví dụ sau (file Main01.java):

```
public class Main01{
    public static void main(String[] args){
        MyThread t1=new MyThread();// Tạo tiến trình
        MyThread t2=new MyThread();
        t1.start();// Gọi thực thi tiến trình
        t2.start();
    }
}
class MyThread extends Thread{
    public void run(){
        int i=0;
        while(i<200){
            System.out.println(getName()+" "+i);
            i++;
        }
    }
}
```

```

    }
}

```

Quản lý thread gồm: tạo thread, chỉnh độ ưu tiên, chạy thread, và dừng thread. Ngoài ra, thread còn hỗ trợ một số phương thức khác trong bảng 3-2.

3.2.1 Tạo Thread

Dùng các constructor trong bảng 3-1. Nếu không dùng Runnable, ta buộc phải tạo class kế thừa từ class Thread có sẵn rồi override phương thức run(). Khi thread được kích hoạt, nó sẽ chạy các lệnh trong phương thức run() này. Do đó trong thực tế, ta dùng các constructor của lớp dẫn xuất chứ không dùng các constructor của thread một cách trực tiếp.

Bảng 3.1: các hàm khởi tạo lớp Thread

Constructor	Ý nghĩa
Thread()	Tạo Thread với giá trị mặc định
Thread(String name)	Tạo Thread với tên cho trước
Thread(Runnable target)	Tạo thread liên kết với đối tượng Runnable
Thread(Runnable tar, String name)	Tạo Thread liên kết với đối tượng Runnable với tên cho trước

Ví dụ:

```

class MyThread extends Thread{// Kế thừa lớp Thread
// ...
public void run() { // Override phương thức run
// ...
}
// ...
}

```

Rồi dùng như sau:

```
MyThread t=new MyThread();// Tạo tiểu trình
t.start();// Gọi thực thi tiểu trình
```

3.2.2 Chính độ ưu tiên

Trong Java, thread có 10 mức ưu tiên, đánh số từ 1 (độ ưu tiên thấp nhất) tới 10 (độ ưu tiên cao nhất). Khi thread được tạo ra, nó sẽ có độ ưu tiên mặc định là 5. ta có thể chỉnh lại độ ưu tiên của thread bằng phương thức void setPriority(int newPriority).

Ví dụ:

```
t.setPriority(8);
```

Lưu ý: Nếu truyền vào độ ưu tiên không thuộc giá trị từ 1 tới 10, phương thức sẽ phát sinh IllegalArgumentException. Độ ưu tiên thấp nhất và cao nhất được lưu trong hai hằng (kiểu số nguyên) là Thread.MIN_PRIORITY và Thread.MAX_PRIORITY.

3.2.3 Thực thi thread

Để thực thi thread, ta gọi phương thức void start(). Khi được thực thi, thread sẽ chạy các lệnh trong phương thức run().

Lưu ý: Nếu thread đã start() rồi, việc gọi start() lần nữa sẽ bị phát sinh exception IllegalStateException. Để start() lại thread, ta cần tạo mới thread.

3.2.4 Dừng thread

Để dừng thread đang chạy, ta dùng phương thức void interrupt(). Nếu thread đang ở trạng thái sleep, việc dừng nó sẽ làm phát sinh InterruptedException.

Bảng 3.2: Một số phương thức khác của lớp Thread

Phương thức	Ý nghĩa
public final void wait(long timeout) throws InterruptedException	Tuyến trình hiện thời chờ cho tới khi được cảnh báo hoặc một khoảng thời gian timeout nhất định. Nếu timeout bằng 0 thì phương thức sẽ chỉ chờ cho tới khi có

	cảnh báo về sự kiện.
<code>public final void notify()</code>	Cảnh báo ít nhất một tuyến trình đang chờ một điều kiện nào đó thay đổi. Các tuyến trình phải chờ một điều kiện thay đổi trước khi có thể gọi phương thức <code>wait</code> nào đó.
<code>public final void notifyAll()</code>	Phương thức này cảnh báo tất cả các tuyến trình đang chờ một điều kiện thay đổi. Các tuyến trình đang chờ thường chờ một tuyến trình khác thay đổi điều kiện nào đó. Trong số các tuyến trình đã được cảnh báo, tuyến trình nào có độ ưu tiên cao nhất thì sẽ chạy trước tiên.
<code>public static void sleep(long ms)</code> <code>throws InterruptedException</code>	Phương thức này đưa tiến trình hiện hành vào trạng thái nghỉ tối thiểu là <code>ms</code> (mili giây).
<code>public static void yield()</code>	Phương thức này giành lấy quyền thực thi của tuyến trình hiện hành cho một trong các tuyến trình khác.

3.3 GIAO DIỆN RUNNABLE

Nếu không thể tạo class kế thừa từ `Thread`, ta có thể cho nó implement interface `Runnable`. Khi đó, class phải hiện thực phương thức `void run()` của interface này. Sau đó, ta tạo thread liên kết với `Runnable` thông qua constructor tương ứng của lớp `Thread`. Khi thread được start, nó sẽ gọi thực thi lệnh trong phương thức `run()` của `Runnable` liên kết với nó.

Ví dụ:

```
public class Main09{
```



```
public static void main(String[] args) {
    Runnable r1=new MyRunnable();// Tạo runnable
    Runnable r2=new MyRunnable();
    Thread t1=new Thread(r1);// Tạo thread liên kết
    Thread t2=new Thread(r2);// với runnable
    t1.start();
    t2.start();
}
}
class MyRunnable implements Runnable{
    public void run() {
        int i=0;
        while(i<200){
            System.out.println(getName()+" "+i);
            i++;
        }
    }
}
```

Kết quả thực thi tương tự khi dùng cách kế thừa lớp Thread. Việc quản lý Runnable, được tiến hành thông qua Thread liên kết với nó như đã trình bày trong phần trước.

3.4 ĐỒNG BỘ

3.4.1 Đồng bộ hóa sử dụng cho phương thức

Nếu nhiều thread cùng truy cập vào một vùng nhớ, lỗi có thể xảy ra. Để chỉ cho phép trong một lúc chỉ có tối đa một thread truy cập vào vùng nhớ, ta cần đồng bộ hóa bằng cách dùng từ khóa synchronized.

Xem ví dụ sau:

```
public class Main08 extends Thread
{
    public static void main(String[] args)
    {
        MyThread[] T=new MyThread[3];
        for(int i=0;i<T.length;i++){
            T[i]=new MyThread();
            T[i].start();
        }
    }
}
```

```

    }
}
class MyThread extends Thread
{
    public static int x=5;
    public static boolean stop=false;

    public static /*synchronized*/ void Trans() {
        if (x>0) {
            try{
                sleep(100);
            } catch (Exception E) {
                return;
            }
            x--;
        }
    }
    public void run() {
        do{
            Trans();
            System.out.println("x = "+x);
        } while (x>0);
    }
}

```

Theo đoạn code thì các tiểu trình sẽ chỉ chạy khi $x > 0$. Kết quả thực thi khi không đồng bộ:

```

x = 4
x = 3
x = 2
x = 1
x = 0
x = -1
x = -2
Press any key to continue...

```

Kết quả thực thi khi có đồng bộ:

```

x = 4
x = 3
x = 2
x = 1
x = 0
x = 0
x = 0
Press any key to continue...

```

3.4.2 Lệnh synchronized

Lệnh synchronized cho phép đồng bộ hóa một đối tượng mà không cần yêu cầu bạn tác động một phương thức synchronized trên đối tượng đó. Cú pháp:

```
synchronized (expr)
{statement}
```

Khi có được khóa, statement được xử lý giống như nó là một phương thức synchronized trên đối tượng đó. Ví dụ: Chuyển các phần tử trong mảng thành các số không âm

```
public static void abs(int[] v){
    synchronized(v){
        for(int i=0;i<v.length;i++){
            if(v[i]<0) v[i]=-v[i];
        }
    }
}
```

3.5 TRAO ĐỔI DỮ LIỆU GIỮA CÁC THREAD

Để trao đổi dữ liệu giữa các thread, ta dùng các lớp PipedOutputStream (xuất dữ liệu) và PipedInputStream (nhận dữ liệu). Lưu ý: cần import package java.io. Để kết nối stream xuất và nhận, ta dùng phương thức connect() của các pipe. Sau khi đã kết nối, ta dùng các phương thức read() và write() để đọc / ghi dữ liệu. Ví dụ:

```
import java.io.*;
public class Multi10
{
    public static void main(String[]args) throws Exception
    {
        Teacher t=new Teacher();
        Student s=new Student();
        t.pout.connect(s.pin);// Kết nối hai Piped Stream
        //s.pin.connect(t.pout);// Tương đương với lệnh trên
        t.start();
        s.start();
    }
}
class Teacher extends Thread
{

```

```
public PipedOutputStream pout=new PipedOutputStream();
// ..
public void run(){
    // ..
}
// ..
}
class Student extends Thread{
    public PipedInputStream pin=new PipedInputStream();
    // ..
    public void run(){
        // ...
    }
    // ..
}
```

Lưu ý: Khi thread đang đọc dữ liệu từ stream mà stream không có dữ liệu, nó sẽ bị lock và không hoạt động cho đến khi đọc được dữ liệu hoặc stream đóng. Để tránh bị lock khi đang đọc, ta dùng phương thức `available()` của `PipedInputStream` để kiểm tra số byte dữ liệu đang có trong các stream. Nếu số byte thỏa yêu cầu thì mới tiến hành đọc. Để trao đổi dữ liệu dạng text, ta dùng `PipedReader` và `PipedWrite`. Cách dùng cũng tương tự `PipedInputStream` và `PipedOutputStream`.

CÂU HỎI ÔN TẬP

1. Cho chương trình minh hoạt cơ chế deadlock có thể xảy ra khi lập trình nhiều tuyến trình khi sử dụng dữ liệu chia sẻ.
2. Tìm hiểu vai trò của lập trình đa tuyến trình trong các ứng dụng mạng.
3. Sử dụng tuyến trình trong việc viết chương trình hiển thị đồng hồ đếm giờ lên một cửa sổ.

BÀI 4. QUẢN LÝ ĐỊA CHỈ KẾT NỐI MẠNG

4.1 LỚP INETADDRESS

Lớp `InetAddress` nằm trong gói `java.net` và biểu diễn một địa chỉ Internet bao gồm hai thuộc tính cơ bản: `hostname` (`String`) và `address` (`int[]`).

4.1.1 Tạo các đối tượng `InetAddress`

Lớp `InetAddress` không giống với các lớp khác, không có các constructor cho lớp `InetAddress`. Tuy nhiên, lớp `InetAddress` có ba phương thức tĩnh trả về các đối tượng `InetAddress`. Các phương thức trong lớp `InetAddress` được trình bày trong bảng 4-1.

Bảng 4.1: các phương thức tạo đối tượng `InetAddress`

Phương thức	Ý nghĩa
<code>public static InetAddress InetAddress.getByName(String hostname)</code>	Cung cấp tên miền, và hàm trả về đối tượng <code>InetAddress</code> .
<code>public static InetAddress[] InetAddress.getAllByName(String hostname)</code>	Cung cấp tên miền và trả về tất cả địa chỉ <code>InetAddress</code> có thể có.
<code>public static InetAddress InetAddress.getLocalHost()</code>	trả về đối tượng <code>InetAddress</code> biểu diễn địa chỉ Internet của máy đang chạy chương trình.

4.1.2 Các phương thức lấy dữ liệu của InetAddress

Chỉ có các lớp trong gói java.net có quyền truy xuất tới các trường của lớp InetAddress. Các lớp trong gói này có thể đọc dữ liệu của một đối tượng InetAddress bằng cách gọi phương thức getHostName và getAddress().

Bảng 4.2: Các phương thức cơ bản của InetAddress.

Phương thức	Ý nghĩa
public String getHostName()	Phương thức này trả về một chuỗi biểu diễn hostname của một đối tượng InetAddress. Nếu máy không có hostname, thì nó sẽ trả về địa chỉ IP của máy này dưới dạng một chuỗi ký tự.
public byte[] getAddress()	Trả về một địa chỉ IP dưới dạng một mảng các byte.

Bảng 4.3: Các phương thức kiểm tra địa chỉ IP

Phương thức	Ý nghĩa
public boolean isReachable(int timeout) throws IOException	Kiểm tra xem một liên kết mạng đã được thiết lập hay chưa. Nếu host đáp ứng trong khoảng thời gian timeout mili giây, các phương thức này trả về giá trị true nếu đến được, ngược lại nó trả về giá trị false. <i>Chú ý. Các liên kết có thể bị phong tỏa vì nhiều nguyên nhân như firewall, các server ủy quyền, các router hoạt động sai chức năng, dây cáp bị đứt, hoặc host ở xa không bật.</i>
public boolean isReachable(NetworkInterface netif, int ttl, int timeout) throws	Kiểm tra có thể tạo kết nối tới địa chỉ mạng này hay không bằng cách thực hiện một ICMP ECHO REQUEST hoặc thiết lập kết nối TCP tới cổng 7

IOException		(Echo) của máy đích. netif – null hoặc giao diện bất kỳ. ttl – số lượng tối đa các hops, mặc định là 0. timeout – thời gian xem xét tối đa, mili giây.
public isMulticastAddress()	boolean	Kiểm tra có là địa chỉ Multicast hay không, true nếu đúng, ngược lại là false.
public isLoopbackAddress()	boolean	Kiểm tra địa chỉ này có là địa chỉ loopback không.
public isSiteLocalAddress()	boolean	Kiểm tra địa chỉ này có là địa chỉ cục bộ không. Tiềm lợi trong việc định tuyến.

4.2 LỚP URL

Ngôn ngữ Java hỗ trợ việc định vị và lấy dữ liệu thông qua địa chỉ URL (Uniform Resource Locator) bằng lớp đối tượng URL. Lớp đối tượng này cho phép chương trình giao tiếp với Server để lấy dữ liệu về dựa trên các giao thức chuẩn được hỗ trợ bởi Server. Việc xác định Server, giao thức, cũng như cổng dịch vụ, đường dẫn là dựa vào URL được cung cấp cho lớp đối tượng URL. Do vậy, lớp URL quản lý các thông tin liên quan đến việc định vị tài nguyên như sau: giao thức (protocol), tên miền (hostname), cổng (port), đường dẫn (path), tên tập tin (filename), mục tài liệu (document section). Các thông tin này có thể thiết lập độc lập.

4.2.1 Tạo các URL

Trong việc tạo đối tượng URL, có bốn hàm khởi tạo hỗ trợ ứng với các trường hợp khác nhau của việc cung cấp tham số về định vị nguồn tài nguyên URL. Tất cả các hàm khởi tạo này đều ném ra ngoại lệ MalformedURLException nếu URL không đúng khuôn dạng.

Bảng 4.4: Các phương thức tạo đối tượng URL

Hàm khởi tạo	Ý nghĩa
<code>public URL(String url) throws MalformedURLException</code>	Tạo đối tượng URL với chuỗi url.
<code>public URL(String protocol, String host, String file) throws MalformedURLException</code>	Tạo đối tượng URL với từng phần tách biệt: giao thức, tên miền, tên tập tin, (cổng sẽ thiết lập là -1, sử dụng dụng cổng mặc định của giao thức đó) .
<code>public URL(String protocol, String host, int port, String file) throws MalformedURLException</code>	Tạo ra đối tượng URL với đầy đủ các thông tin như giao thức, tên miền, số hiệu cổng, tên tập tin.
<code>public URL(URL u, String s) throws MalformedURLException</code>	Tạo ra đối tượng URL từ một đối tượng URL khác.

4.2.2 Nhận thông tin các thành phần của URL

URL có sáu trường thông tin trong lớp URL: tên miền, địa chỉ IP, giao thức, cổng, tập tin, mục tham chiếu tài liệu.

Bảng 4.5: Các phương thức nhận thông tin các thành phần của URL

Các phương thức	Ý nghĩa
<code>public String getProtocol()</code>	trả về một xâu ký tự biểu diễn giao thức
<code>public String getHost()</code>	trả về một xâu ký tự biểu diễn tên miền
<code>public int getPort()</code>	trả về một số nguyên kiểu int biểu diễn số hiệu cổng.
<code>public int getDefaultPort()</code>	trả về số hiệu cổng mặc định cho giao thức.
<code>public String getFile()</code>	trả về một xâu ký tự chứa đường dẫn tập tin

4.2.3 Nhận dữ liệu từ máy đích trong URL

Sau khi xác định được thông tin tài nguyên của dịch vụ máy đích, đối tượng URL hỗ trợ các phương thức để lấy dữ liệu từ máy đích về.

Bảng 4.6: Các phương thức hỗ trợ nhận dữ liệu từ phần mềm máy chủ

Các phương thức	Ý nghĩa
public final InputStream openStream() throws java.io.IOException	Phương thức này kết nối tới một tài nguyên được tham chiếu bởi một URL, thực hiện cơ chế bắt tay cần thiết giữa client và server, và trả về một luồng nhập InputStream. Ta sử dụng luồng này để đọc dữ liệu. Dữ liệu nhận từ luồng này là dữ liệu thô của một tệp tin mà URL tham chiếu (mã ASCII nếu đọc một tệp văn bản, mã HTML nếu đọc một tài liệu HTML, một ảnh nhị phân nếu ta đọc một file ảnh). Nó không có các thông tin header và các thông tin có liên quan đến giao thức
public URLConnection openConnection() throws java.io.IOException	Phương thức openConnection() mở một socket tới một URL xác định và trả về một đối tượng URL. Một đối tượng URLConnection biểu diễn một liên kết mở tới một tài nguyên mạng. Nếu lời gọi phương thức thất bại nó đưa ra ngoại lệ IOException.
public final Object getContent() throws java.io.IOException	Phương thức getContent() tìm kiếm dữ liệu được tham chiếu bởi một URL và chuyển nó thành một kiểu đối tượng nào đó. Nếu đối tượng tham chiếu tới một kiểu đối tượng văn bản nào đó như tệp tin ASCII hoặc tệp HTML, đối tượng được trả về thông thường sẽ là một kiểu luồng nhập InputStream nào đó. Nếu URL tham chiếu tới một đối tượng ảnh như ảnh GIF hoặc JPEG thì phương thức getContent() trả về đối tượng java.awt.ImageProducer

4.3 LỚP URLCONNECTION

URLConnection là một lớp trừu tượng biểu diễn một liên kết tới một tài nguyên được xác định bởi một url. Lớp URLConnection cung cấp nhiều khả năng điều khiển hơn so với lớp URL thông qua việc tương tác với một server. Hơn nữa, lớp đối tượng này cho phép gửi dữ liệu đến Server và đọc thông tin đáp trả từ Server (tùy thuộc giao thức mà Server đó có hỗ trợ). Việc tạo ra đối tượng hiện thực lớp URLConnection bằng phương thức `openConnection` của lớp URL. Để sử dụng lớp URLConnection, thực hiện các bước cơ bản sau:

- Xây dựng một đối tượng URL.
- Gọi phương thức `openConnection()` của đối tượng URL để tìm kiếm một đối tượng URLConnection cho URL đó.
- Cấu hình đối tượng URL.
- Đọc các trường header.
- Nhận một luồng nhập và đọc dữ liệu.
- Nhận một luồng xuất và ghi dữ liệu.
- Đóng liên kết.

Một số phương thức hỗ trợ cho việc thao tác với lớp URLConnection.

Bảng 4.7: Các phương thức hỗ trợ nhập xuất dữ liệu của URLConnection

Các phương thức	Ý nghĩa
<code>public</code> <code>getInputStream()</code> InputStream	trả lại một luồng đọc dữ liệu từ máy đích.
<code>public</code> <code>getOutputStream()</code> OutputStream	trả về một luồng OutputStream trên đó bạn có thể ghi dữ liệu để truyền tới máy đích.

Ngoài ra lớp URLConnection còn hỗ trợ một số phương thức xem thông tin về tập tin và các phương thức cho phép nhận và phân tích thông tin Header của giao thức HTTP.

Bảng 4.8: Các phương thức hỗ trợ xem thông tin dữ liệu

Các phương thức	Ý nghĩa
<code>public int getLength()</code>	trả về độ dài của trường header.
<code>public long getDate()</code>	trả về một số nguyên kiểu long cho biết tài liệu đã được gửi khi nào.
<code>public String getEncoding()</code>	Phương thức này trả về String cho ta biết cách thức mã hóa. Nếu nội dung được gửi không được mã hóa (như trong trường hợp của HTTP server), phương thức này trả về giá trị null.
<code>public long getLastModified()</code>	trả về ngày mà tài liệu được sửa đổi lần cuối
<code>public Map getHeaderFields()</code>	trả lại thông tin trường Header đáp trả bởi Server ở máy đích.
<code>public String getHeaderField(String name)</code>	trả về giá trị của thành phần trường header ứng với tên name.

CÂU HỎI ÔN TẬP

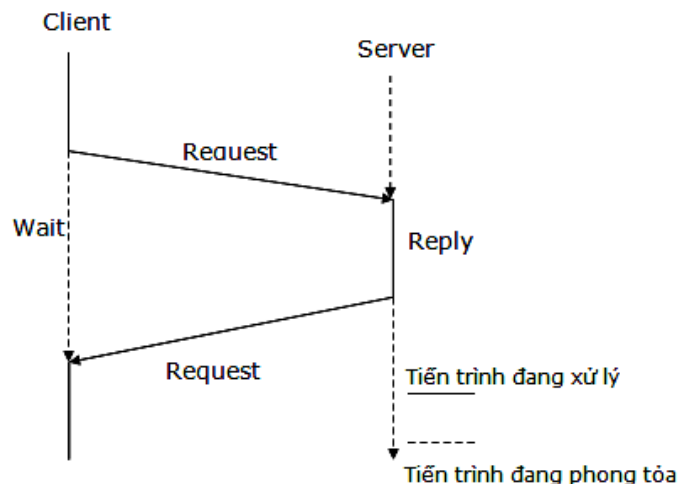
Vì sao cần phân giải tên miền ? Trình bày vai trò của kiểu dữ liệu InetAddress? Trình bày các hàm phân giải tên miền của bản của InetAddress?

Trình bày cú pháp và ý nghĩa của bộ định vị tài nguyên URL? Cho biết các hàm cơ bản của URL và ý nghĩa sử dụng? ví dụ minh họa?

BÀI 5. LẬP TRÌNH SOCKET CHO GIAO THỨC TCP

5.1 MÔ HÌNH KHÁCH CHỦ (CLIENT/SERVER)

Các ứng dụng mạng thường hoạt động theo mô hình client/server như thư điện tử, truyền nhận tập tin, game trên mạng, ... Mô hình này gồm có một chương trình đóng vai trò là client và một chương trình đóng vai trò là server. Hai chương trình này sẽ giao tiếp với nhau thông qua mạng. Chương trình server đóng vai trò cung cấp dịch vụ. Chương trình này luôn luôn lắng nghe các yêu cầu từ phía client, rồi tính toán và đáp trả kết quả tương ứng. Chương trình client cần một dịch vụ và gửi yêu cầu dịch vụ tới chương trình server và đợi đáp trả từ server. Như vậy, quá trình trao đổi dữ liệu giữa client/server bao gồm: Truyền một yêu cầu từ chương trình client tới chương trình server, yêu cầu được server xử lý, truyền đáp ứng cho client



Hình 5.1: Mô hình xử lý hỏi đáp giữa client và server

Mô hình truyền tin này thực hiện truyền hai thông điệp qua lại giữa client và server một cách đồng bộ hóa. Chương trình server nhận được thông điệp từ client thì nó

phát ra yêu cầu client chuyển sang trạng thái chờ (tạm dừng) cho tới khi client nhận được thông điệp đáp ứng do server gửi về. Mô hình client/server thường được cài đặt dựa trên các thao tác cơ bản là gửi (send) và nhận (receive).

5.2 MÔ HÌNH TRUYỀN TIN SOCKET

Chương trình client và server sử dụng giao thức vận chuyển để gửi và nhận dữ liệu. Một ví dụ là giao thức TCP/IP được sử dụng để giao tiếp qua mạng Internet.

TCP và UDP là các giao thức tầng giao vận để truyền dữ liệu. Mỗi giao thức có những ưu và nhược điểm riêng. Chẳng hạn, giao thức TCP có độ tin cậy truyền tin cao, nhưng tốc độ truyền tin bị hạn chế do phải có giai đoạn thiết lập và giải phóng liên kết khi truyền tin, khi gói tin có lỗi hay bị thất lạc thì giao thức TCP phải có trách nhiệm truyền lại,...Ngược lại, giao thức UDP có tốc độ truyền tin rất nhanh vì nó chỉ có một cơ chế truyền tin rất đơn giản: không cần phải thiết lập và giải phóng liên kết.

Dữ liệu được truyền trên mạng Internet dưới dạng các gói (packet) có kích thước hữu hạn được gọi là datagram. Mỗi datagram chứa một header và một payload. Header chứa địa chỉ và cổng cần truyền gói tin đến, cũng như địa chỉ và cổng xuất phát của gói tin, và các thông tin khác được sử dụng để đảm bảo độ tin cậy truyền tin, payload chứa dữ liệu. Tuy nhiên do các datagram có chiều dài hữu hạn nên thường phải phân chia dữ liệu thành nhiều gói và khôi phục lại dữ liệu ban đầu từ các gói ở nơi nhận. Trong quá trình truyền tin có thể có một hay nhiều gói bị mất hay bị hỏng và cần phải truyền lại hoặc các gói tin đến không theo đúng trình tự. Để tránh những điều này, việc phân chia dữ liệu thành các gói, tạo các header, phân tích header của các gói đến, quản lý danh sách các gói đã nhận được và các gói chưa nhận được, ...

Để giải quyết bài toán này, Đại học UC Berkeley đưa ra khái niệm Socket. Chúng cho phép người lập trình xem một liên kết mạng như là một luồng mà có thể đọc dữ liệu ra hay ghi dữ liệu vào từ luồng này. Các Socket che dấu người lập trình khỏi các chi tiết mức thấp của mạng như kiểu đường truyền, các kích thước gói, yêu cầu truyền lại gói, các địa chỉ mạng...

Một socket có thể thực hiện bảy thao tác cơ bản:

- Kết nối với một máy ở xa
- Gửi dữ liệu
- Nhận dữ liệu
- Ngắt liên kết
- Gán cổng
- Nghe dữ liệu đến
- Chấp nhận liên kết từ các máy ở xa trên cổng đã được gán

Có nhiều kiểu Socket khác nhau tương ứng với mỗi kiểu giao thức được sử dụng để giao tiếp giữa hai máy trên mạng Internet. Đối với chồng giao thức TCP/IP, có hai kiểu Socket chính được sử dụng là stream socket và datagram socket. Stream socket sử dụng giao thức TCP để cung cấp dịch vụ gửi dữ liệu tin cậy. Datagram socket sử dụng giao thức UDP để cung cấp dịch vụ gửi gói tin đơn giản. Ngôn ngữ lập trình Java hỗ trợ lớp Socket và ServerSocket cho kiểu stream socket và lớp DatagramPacket và DatagramSocket cho kiểu datagram socket.

Đối với kiểu stream socket, lớp Socket được sử dụng bởi cả client và server. Lớp Socket này có các phương thức tương ứng với bốn thao tác đầu tiên của thao tác cơ bản của socket trình bày ở trên. Ba thao tác cuối được hỗ trợ bởi server để chờ các client liên kết tới. Các thao tác này được cài đặt bởi lớp ServerSocket.

5.3 SOCKET

Client thiết lập giao tiếp với server và đợi sự đáp trả từ Server. Các socket cho client thường được sử dụng theo các bước sau:

Tạo ra một Socket mới sử dụng hàm khởi tạo với địa chỉ IP và số hiệu cổng dịch vụ của máy đích. Socket cố gắng liên kết với socket server của máy đích.

Sau khi liên kết được thiết lập, lấy luồng nhập và luồng xuất được tạo ra giữa socket ở máy gửi - client và socket ở máy đích - server. Các luồng này được sử dụng để gửi dữ liệu cho nhau. Kiểu liên kết này được gọi là song công (full-duplex) các host có thể nhận và gửi dữ liệu đồng thời. Ý nghĩa của dữ liệu phụ thuộc vào giao thức.

Khi việc truyền dữ liệu hoàn thành, một hoặc cả hai phía ngắt liên kết. Một số giao thức, như HTTP, đòi hỏi mỗi liên kết phải bị đóng sau mỗi khi yêu cầu được phục vụ. Các giao thức khác, chẳng hạn FTP, cho phép nhiều yêu cầu được xử lý trong một liên kết đơn.

5.3.1 Các hàm khởi tạo Socket

Bảng 5.1: Các hàm khởi tạo kiểu dữ liệu Socket

Hàm khởi tạo	Ý nghĩa
<pre>public Socket(String host, int port) throws UnknownHostException, IOException</pre>	Hàm này tạo một socket TCP với host và cổng xác định, và thực hiện liên kết với host ở xa.
<pre>public Socket(InetAddress host, int port) throws IOException</pre>	tạo một socket TCP với thông tin là địa chỉ của một host được xác định bởi một đối tượng <code>InetAddress</code> và số hiệu cổng port, sau đó nó thực hiện kết nối tới host. Nó đưa ra ngoại lệ <code>IOException</code> nhưng không đưa ra ngoại lệ <code>UnknownHostException</code> . Constructor đưa ra ngoại lệ trong trường hợp không kết nối được tới host.
<pre>public Socket (String host, int port, InetAddress interface, int localPort) throws IOException, UnknownHostException</pre>	Tạo ra một socket với thông tin là địa chỉ IP được biểu diễn bởi một đối tượng <code>String</code> và một số hiệu cổng và thực hiện kết nối tới host đó. Socket kết nối tới host ở xa thông qua một giao tiếp mạng và số hiệu cổng cục bộ được xác định bởi hai tham số sau. Nếu <code>localPort</code> bằng 0 thì Java sẽ lựa chọn một cổng ngẫu nhiên có sẵn nằm trong khoảng từ 1024 đến 65535.
<pre>public Socket</pre>	Constructor chỉ khác constructor trên ở chỗ địa chỉ

(InetAddress host, int port, InetAddress interface, int localPort) throws IOException, UnknownHostException	của host lúc này được biểu diễn bởi một đối tượng InetAddress.
---	--

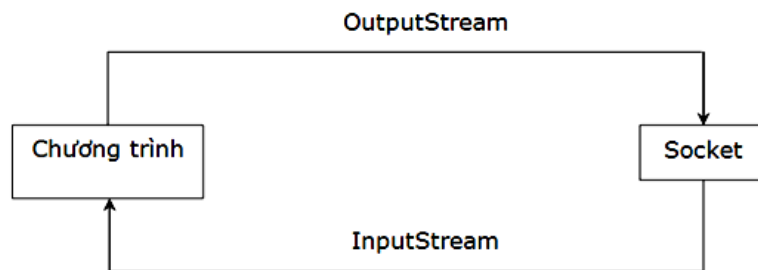
5.3.2 Các phương thức giao tiếp giữa các Socket

Bảng 5.2: Các phương thức phục vụ giao tiếp giữa các Socket

Các phương thức	Ý nghĩa
public InetAddress getInetAddress()	Cho trước một đối tượng Socket, phương thức getInetAddress() cho trả lại địa chỉ máy đích bằng đối tượng InetAddress.
public int getPort()	Lấy số hiệu cổng dịch vụ của máy đích.
public int getLocalPort()	trả lại địa chỉ cổng sử dụng ở máy cục bộ.
public InetAddress getLocalAddress()	trả lại địa chỉ mạng của máy cục bộ dùng để giao tiếp với máy ở xa.
public InputStream getInputStream() throws IOException	trả về một luồng nhập để đọc dữ liệu từ một socket vào chương trình. Thông thường ta có thể gắn kết luồng nhập thô InputStream tới một luồng lọc hoặc một luồng ký tự nhằm đưa các chức năng tiện ích (chẳng hạn như các luồng InputStream, hoặc InputStreamReader). Để tăng cao hiệu năng, ta có thể đệm dữ liệu bằng cách gắn kết nó với luồng lọc BufferedInputStream hoặc BufferedReader.

Public OutputStream getOutputStream() throws IOException	trả về một luồng xuất thô để ghi dữ liệu từ ứng dụng ra đầu cuối của một socket. Thông thường, ta sẽ gắn kết luồng này với một luồng tiện lợi hơn như lớp DataOutputStream hoặc OutputStreamWriter trước khi sử dụng nó. Để tăng hiệu quả ghi.
---	--

Hai phương thức `getInputStream()` và `getOutputStream()` là các phương thức cho phép lấy về các luồng dữ liệu nhập và xuất. Để nhận dữ liệu từ một máy ở xa, một luồng nhập từ socket được nhận và đọc dữ liệu từ luồng đó. Để ghi dữ liệu lên một máy ở xa, một luồng xuất được nhận từ socket và ghi dữ liệu lên luồng. Dưới đây là hình vẽ minh họa trực quan.



Hình 5.2: Trao đổi dữ liệu qua Socket.

5.3.3 Các phương thức đóng Socket

Khi giao tiếp giữa client và server được thực hiện xong, cần thiết phải đóng kênh kết nối lại. Dưới đây là các phương thức hỗ trợ đóng kết nối.

Các phương thức	Ý nghĩa
<code>public void close() throws IOException</code>	Đóng luồng kết nối lại.
<code>public void shutdownInput() throws IOException</code>	đóng một luồng nhập.
<code>public void shutdownOutput() throws IOException</code>	đóng luồng xuất.

<code>public boolean isOutputShutdown()</code>	kiểm tra luồng xuất đóng chưa.
<code>public boolean isInputShutdown()</code>	kiểm tra luồng nhập đóng chưa.

Các socket được đóng một cách tự động khi một trong hai luồng đóng lại, hoặc khi chương trình kết thúc, hoặc khi socket được thu hồi bởi garbage collector. Tuy nhiên, thực tế cho thấy việc cho rằng hệ thống sẽ tự đóng socket là không tốt, đặc biệt là khi các chương trình chạy trong khoảng thời gian vô hạn. Mỗi khi một Socket đã bị đóng lại, các trường thông tin `InetAddress`, địa chỉ cục bộ, và số hiệu cổng cục bộ vẫn có thể truy xuất tới được thông qua các phương thức `getInetAddress()`, `getPort()`, `getLocalHost()`, và `getLocalPort()`. Tuy nhiên khi ta gọi các phương thức `getInputStream()` hoặc `getOutputStream()` để đọc dữ liệu từ luồng đọc `InputStream` hoặc ghi dữ liệu `OutputStream` thì ngoại lệ `IOException` được đưa ra.

5.3.4 Các phương thức thiết lập các tùy chọn cho Socket

TCP_NODELAY: Thiết lập giá trị `TCP_NODELAY` là `true` để đảm bảo rằng các gói tin được gửi đi nhanh nhất có thể mà không quan tâm đến kích thước của chúng. Thông thường, các gói tin nhỏ được kết hợp lại thành các gói tin lớn hơn trước khi được gửi đi. Trước khi gửi đi một gói tin khác, host cục bộ đợi để nhận các xác thực của gói tin trước đó từ hệ thống ở xa.

```
public void setTcpNoDelay(boolean on) throws SocketException
public boolean getTcpNoDelay() throws SocketException
```

SO_LINGER: Tùy chọn `SO_LINGER` xác định phải thực hiện công việc gì với datagram vẫn chưa được gửi đi khi một socket đã bị đóng lại. Ở chế độ mặc định, phương thức `close()` sẽ có hiệu lực ngay lập tức; nhưng hệ thống vẫn cố gắng để gửi phần dữ liệu còn lại. Nếu `SO_LINGER` được thiết lập bằng 0, các gói tin chưa được gửi đi bị phá hủy khi socket bị đóng lại. Nếu `SO_LINGER` lớn hơn 0, thì phương thức `close()` phong tỏa để chờ cho dữ liệu được gửi đi và nhận được xác thực từ phía nhận. Khi hết thời gian qui định, socket sẽ bị đóng lại và bất kỳ phần dữ liệu còn lại sẽ không được gửi đi.

```
public void setSoLinger(boolean on, int seconds) throws SocketException
public int getSoLinger() throws SocketException
```

SO_TIMEOUT: Thông thường khi ta đọc dữ liệu từ một socket, lời gọi phương thức phong tỏa cho tới khi nhận đủ số byte. Bằng cách thiết lập phương thức SO_TIMEOUT, ta sẽ đảm bảo rằng lời gọi phương thức sẽ không phong tỏa trong khoảng thời gian quá số giây quy định.

```
public void setSoTimeout(int milliseconds) throws SocketException
public int getSoTimeout() throws SocketException
```

5.4 SERVERSOCKET

Lớp ServerSocket có đủ mọi thứ ta cần để viết các server bằng Java. Nó có các constructor để tạo các đối tượng ServerSocket mới, các phương thức để lắng nghe các liên kết trên một cổng xác định, và các phương thức trả về một Socket khi liên kết được thiết lập, vì vậy ta có thể gửi và nhận dữ liệu. Vòng đời của một server:

- Một ServerSocket mới được tạo ra trên một cổng xác định bằng cách sử dụng một constructor ServerSocket.
- ServerSocket lắng nghe liên kết đến trên cổng đó bằng cách sử dụng phương thức accept(). Phương thức accept() phong tỏa cho tới khi một client thực hiện một liên kết, phương thức accept() trả về một đối tượng Socket mà liên kết giữa client và server.
- Tùy thuộc vào kiểu server, hoặc getInputStream(), getOutputStream() hoặc cả hai được gọi để nhận các luồng vào ra để truyền tin với client.
- Server và client tương tác theo một giao thức có sẵn cho tới khi ngắt liên kết.
- Server, client hoặc cả hai ngắt liên kết.
- Server trở về bước hai và đợi liên kết tiếp theo.

5.4.1 Các hàm khởi tạo

Bảng 5.3: Các hàm khởi tạo lớp ServerSocket

Các hàm khởi tạo	Ý nghĩa
<pre>public ServerSocket(int port) throws IOException, BindException</pre>	<p>Constructor này tạo một socket cho server trên cổng xác định. Nếu port bằng 0, hệ thống chọn một cổng ngẫu nhiên cho ta. Cổng do hệ thống chọn đôi khi được gọi là cổng vô danh vì ta không biết số hiệu cổng. Với các server, các cổng vô danh không hữu ích lắm vì các client cần phải biết trước cổng nào mà nó nối tới (giống như người gọi điện thoại ngoài việc xác định cần gọi cho ai cần phải biết số điện thoại để liên lạc với người đó).</p>
<pre>public ServerSocket(int port, int queuelength, InetAddress bindAddress)throws IOException</pre>	<p>Constructor này tạo một đối tượng ServerSocket trên cổng xác định với chiều dài hàng đợi xác định. ServerSocket chỉ gán cho địa chỉ IP cục bộ xác định. Constructor này hữu ích cho các server chạy trên các hệ thống có nhiều địa chỉ IP.</p>

Ví dụ: Để tạo một server socket cho cổng 80

```
try{
    ServerSocket httpd = new ServerSocket(80);
}catch(IOException e){
    System.err.println(e);
}
```

Constructor đưa ra ngoại lệ IOException nếu ta không thể tạo và gán Socket cho cổng được yêu cầu. Ngoại lệ IOException phát sinh khi:

- Cổng đã được sử dụng
- Không có quyền hoặc cố liên kết với một cổng nằm giữa 0 và 1023.

5.4.2 Chấp nhận và ngắt liên kết

Một đối tượng `ServerSocket` hoạt động trong một vòng lặp chấp nhận các liên kết. Mỗi lần lặp nó gọi phương thức `accept()`. Phương thức này trả về một đối tượng `Socket` biểu diễn liên kết giữa client và server. Tương tác giữa client và server được tiến hành thông qua socket này. Khi giao tác hoàn thành, server gọi phương thức `close()` của đối tượng socket. Nếu client ngắt liên kết trong khi server vẫn đang hoạt động, các luồng vào ra kết nối server với client sẽ đưa ra ngoại lệ `InterruptedException` trong lần lặp tiếp theo.

Bảng 5.4: Các hàm chấp nhận và ngắt kết nối.

Các phương thức	Ý nghĩa
<code>public Socket accept() throws IOException</code>	Phương thức này phong tỏa; nó dừng quá trình xử lý và đợi cho tới khi client được kết nối. Khi client thực sự kết nối, phương thức <code>accept()</code> trả về đối tượng <code>Socket</code> .
<code>public int getLocalHost()</code>	Các constructor <code>ServerSocket</code> cho phép ta nghe dữ liệu trên cổng không định trước bằng cách gán số 0 cho cổng. Phương thức này cho phép ta tìm ra cổng mà server đang nghe.
<code>public InetAddress getInetAddress()</code>	Phương thức này trả về địa chỉ được sử dụng bởi server (localhost).
<code>public void close() throws IOException</code>	Nếu đã hoàn thành công việc với một <code>ServerSocket</code> , ta cần phải đóng nó lại, đặc biệt nếu chương trình của ta tiếp tục chạy. Điều này nhằm tạo điều kiện cho các chương trình khác muốn sử dụng nó. Đóng một <code>ServerSocket</code> không đồng nhất với việc đóng một <code>Socket</code> .

Khi bước thiết lập liên kết hoàn thành, và ta sẵn sàng để chấp nhận liên kết, cần gọi phương thức `accept()` của lớp `ServerSocket`. Ta sử dụng các phương thức `getInputStream()` và `getOutputStream()` để truyền tin với client.

5.5 VÍ DỤ

Viết chương trình client liên kết với một server. Người sử dụng nhập vào một dòng ký tự từ bàn phím và gửi dữ liệu cho server.

```
import java.net.*;
import java.io.*;
public class EchoClient1{
    public static void main(String[] args){
        String hostname="localhost";
        if(args.length>0){
            hostname=args[0];
        }
        PrintWriter pw=null;
        BufferedReader br=null;
        try{
            Socket s=new Socket(hostname,2007);
            br=newBufferedReader(new InputStreamReader(s.getInputStream()));
            BufferedReader user;
            user=new BufferedReader(new InputStreamReader(System.in));
            pw=new PrintWriter(s.getOutputStream());
            System.out.println("Da ket noi duoc voi server...");
            while(true){
                String st=user.readLine();
                if(st.equals("exit")){
                    break;
                }
                pw.println(st);
                pw.flush();
                System.out.println(br.readLine());
            }
        }catch(IOException e){
            System.err.println(e);
        }finally{
            try{
                if(br!=null)br.close();
                if(pw!=null)pw.close();
            }catch(IOException e){
                System.err.println(e);
            }
        }
    }
}
```

```

    }
}}
}

```

Viết chương trình server EchoServer để phục vụ chương trình EchoClient1

```

import java.net.*;
import java.io.*;
public class EchoServer1
{
    public final static int DEFAULT_PORT=2007;
    public static void main(String[] args){
        int port=DEFAULT_PORT;
        try{
            ServerSocket ss=new ServerSocket(port);
            Socket s=null;
            while(true){
                try{
                    s=ss.accept();
                    PrintWriter pw; BufferedReader br;
                    pw=new PrintWriter(new
OutputStreamWriter(s.getOutputStream()));
                    br=newBufferedReader(new InputStreamReader(s.getInputStream()));
                    while(true){
                        String line=br.readLine();
                        if(line.equals("exit"))break;
                        String upper=line.toUpperCase();
                        pw.println(upper);
                        pw.flush();
                    }
                }catch(IOException e){}
                finally{
                    try{
                        if(s!=null){s.close();}
                    }catch(IOException e){}
                }
            }}catch(IOException e){}
        }
    }
}

```

CÂU HỎI ÔN TẬP

1. Tại sao lập trình mạng thông qua giao diện Socket?
2. Trình bày các hàm hỗ trợ bởi lớp Socket và ServerSocket?
3. Viết chương trình đọc thông tin Socket tại máy cục bộ và thông tin Socket của máy kết nối trên cổng 80.

4. Viết chương trình thực hiện mô phỏng giao thức FTP như sau: Chương trình client: thực hiện chức năng gửi yêu cầu đường dẫn tên tập tin tới Server sau đó đợi đáp trả từ server dữ liệu liên quan đến tập tin và hiển thị thông tin dữ liệu tập tin lên màn hình và lưu thành tập tin phía máy client nếu tìm thấy ngoài ra hiển thị chuỗi lỗi. Chương trình server: có nhiệm vụ lắng nghe kết nối từ client, tìm và mở tập tin liên quan theo yêu cầu của client viết về cho client sau đó đóng luồng lại.

BÀI 6. KỸ THUẬT ĐA TIẾN TRÌNH VÀ TUẦN TỰ HÓA ĐỐI TƯỢNG TRONG ỨNG DỤNG MẠNG

6.1 ỨNG DỤNG LẬP TRÌNH ĐA TIẾN TRÌNH

Các server như đã viết ở trên rất đơn giản nhưng nhược điểm của nó là bị hạn chế về mặt hiệu năng vì nó chỉ quản lý được một client tại một thời điểm. Khi khối lượng công việc mà server cần xử lý một yêu cầu của client là quá lớn và không biết trước được thời điểm hoàn thành công việc xử lý thì các server này là không thể chấp nhận được. Để khắc phục điều này, người ta quản lý mỗi phiên của client bằng một tuyến đoạn riêng, cho phép các server làm việc với nhiều client đồng thời. Server này được gọi là server tương tranh (concurrent server)-server tạo ra một tuyến đoạn để quản lý từng yêu cầu, sau đó tiếp tục lắng nghe các client khác. Chương trình client/server ở bài 5 là chương trình client/server đơn tuyến đoạn. Các server đơn tuyến đoạn chỉ quản lý được một liên kết tại một thời điểm. Trong thực tế một server có thể phải quản lý nhiều liên kết cùng một lúc. Để thực hiện điều này server chấp nhận các liên kết và chuyển các liên kết này cho từng tuyến đoạn xử lý. Trong phần dưới đây chúng ta sẽ xem xét cách tiến hành cài đặt một chương trình client/server đa tuyến đoạn.

6.1.1 Chương trình phía server

```
import java.io.*;
import java.net.*;
class EchoServe extends Thread
{
    private Socket socket;
    private BufferedReader in;
    private PrintWriter out;
    public EchoServe (Socket s) throws IOException
    {
```

```

        socket = s;
        System.out.println("Serving: "+socket);
        in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
        // Cho phép auto-flush:
        out = new PrintWriter(new BufferedWriter(new OutputStreamWriter(
            socket.getOutputStream())) , true);
        // Nếu bất kỳ lời gọi nào ở trên đưa ra ngoại lệ
        // thì chương trình gọi có trách nhiệm đóng socket. Ngược lại tuyến đoạn sẽ
        // sẽ đóng socket
        start();
    }
    public void run()
    {
        try{
            while (true){
                System.out.println("....Server is waiting...");
                String str = in.readLine();
                if (str.equals("exit") ) break;
                System.out.println("Received: " + str);
                System.out.println("From: "+ socket);
                String upper=str.toUpperCase();
                out.println(upper);      // gửi lại cho client
            }
            System.out.println("Disconnected with.." +socket);
        }catch (IOException e) {}
        finally{
            try{        socket.close();    }catch(IOException e) {}
        }
    }
}
}

public class TCPServer1{
    static int PORT=0;
    public static void main(String[] args) throws IOException
    {
        if (args.length == 1){
            PORT=Integer.parseInt(args[0]); // Nhập số hiệu cổng từ đối dòng lệnh
        }
        ServerSocket s = new ServerSocket(PORT); // Tạo một đối tượng Server Socket
        InetAddress  addr= InetAddress.getLocalHost();
        System.out.println("TCP/Server : "+ addr + " ,Port"+s.getLocalPort());
        try {
            while(true) {
                Socket socket = s.accept();// Phong tỏa cho tới khi có một liên kết đến
                try {
                    new EchoServe(socket); // Tạo một tuyến đoạn quản lý riêng từng liên
kết
                } catch(IOException e) {
                    socket.close();
                }
            }
        }
        }finally {
            s.close();
        }
    }
}

```

```
}  
}}
```

6.1.2 Chương trình phía client

```
import java.net.*;  
import java.io.*;  
public class TCPClient1  
{  
    public static void main(String[] args) throws IOException  
    {  
        if (args.length != 2) {  
            System.out.println("Sử dụng: java TCPClient hostid port#");  
            System.exit(0);  
        }  
        try{  
            InetAddress addr = InetAddress.getByName(args[0]);  
            Socket socket = new Socket(addr, Integer.parseInt(args[1]));  
            try {  
                System.out.println("socket = " + socket);  
                BufferedReader in = new BufferedReader(new InputStreamReader(  
  
                    socket.getInputStream()));  
                PrintWriter out =new PrintWriter(new BufferedWriter(  
                    new OutputStreamWriter(socket.getOutputStream()),true);  
                // Đọc dòng ký tự từ bàn phím  
                DataInputStream myinput =  
                    new DataInputStream(new BufferedInputStream(System.in));  
                try{  
                    for(;;){  
                        System.out.println("Exit to terminate the program.");  
                        String strin=myinput.readLine();  
                        // Quit if the user typed ctrl+D  
                        if (strin.equals("exit")) break;  
                        else  
                            out.println(strin);           // Send the message  
                        String strout = in.readLine();      // Recive it back  
                        if ( strin.length()==strout.length()){ // Compare Both Strings  
                            System.out.println("Received: "+strout);  
                        } else  
                            System.out.println("Echo bad -- string unequal"+ strout);  
                        } // of for ;;  
                    }catch (IOException e){  
                        e.printStackTrace();  
                    }// User is exiting  
                }finally {  
                    System.out.println("EOF...exit");  
                    socket.close();  
                }  
            }catch (UnknownHostException e){
```

```
        System.err.println("Can't find host");
        System.exit(1);
    } catch (SocketException e) {
        System.err.println("Can't open socket");
        e.printStackTrace();
        System.exit(1);
    }
}
```

6.2 TUẦN TỰ HÓA ĐỐI TƯỢNG

Object serialization là khả năng biến đổi một đối tượng thành một dãy byte để lưu trữ trên bộ nhớ phụ hoặc truyền đi nơi khác. Trong Java, object serialization đã được tự động hóa hoàn toàn và hầu như lập trình viên không phải làm gì thêm. Để một đối tượng có thể được "serialize", ta chỉ cần cho nó implement interface `Serializable`. Mọi việc sau đó như đọc/ghi/truyền/nhận đều do Java thực hiện.

Lưu ý: Chỉ có các thuộc tính (dữ liệu) của đối tượng mới được serialize. Các thuộc tính được đánh dấu bằng từ khóa `transient` (nghĩa là có tính tạm thời) sẽ không được serialize. Sau khi serialize, trạng thái của đối tượng được lưu trữ trên bộ nhớ ngoài được gọi là persistence (nghĩa là được giữ lại một cách bền vững).

6.2.1 Luồng viết đối tượng

Trong Java, các lớp đảm nhận việc đọc/ghi đối tượng gồm:

- `ObjectInputStream`: đọc dãy byte và chuyển thành đối tượng phù hợp.
- `ObjectOutputStream`: chuyển đối tượng thành dãy byte và ghi.

Các đối tượng đọc/ghi này cũng không thể hoạt động độc lập mà phải được liên kết với stream khác. Chúng có các phương thức chính trong bảng 6-1.

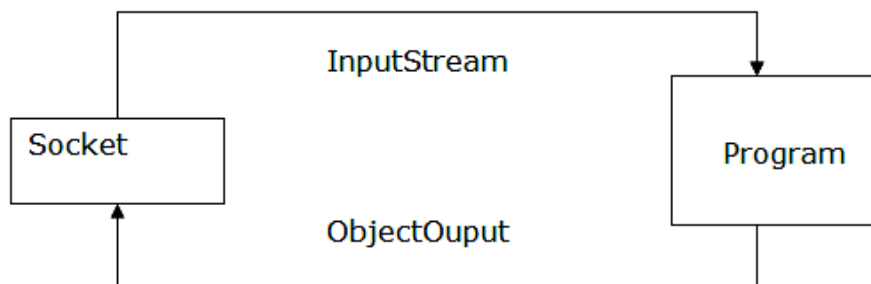
Bảng 6.1: Các phương thức hỗ trợ của luồng nhập/xuất đối tượng

Stream	Phương thức	Ý nghĩa
Object Input	<code>ObjectInputStream</code> (<code>InputStream in</code>)	Tạo stream để đọc đối tượng.

Stream	Object readObject()	Đọc một đối tượng. Nếu đối tượng không được khai báo trong chương trình thì sẽ phát sinh <code>ClassNotFoundException</code> .
Object Output Stream	ObjectOutputStream (OutputStream out)	Tạo stream để ghi đối tượng.
	void reset()	Phục hồi lại stream và hủy bỏ thông tin về các đối tượng mà stream đã ghi.
	writeObject(Object obj)	Ghi một đối tượng.

6.2.2 Truyền các đối tượng thông qua Socket

Việc truyền và nhận dữ liệu thực chất là các thao tác đọc và ghi dữ trên *Socket*. Ta có thể thấy điều này qua sơ đồ dưới đây:



Hình 6.1: Truyền/nhận dữ liệu qua Socket.

Giả sử *s* là một đối tượng *Socket*. Nếu chương trình nhận dữ liệu thì ta sẽ lấy dữ liệu từ luồng nhập đến từ *Socket*:

```
InputStream is=s.getInputStream()
```

Để phục hồi trạng thái đối tượng ta gắn kết luồng nhập thô lấy được từ *Socket* với luồng đọc đối tượng *ObjectInputStream*:

```
ObjectInputStream ois=new ObjectInputStream(is);
```

Khi đó đối tượng được phục hồi lại trạng thái bởi câu lệnh:

```
Object obj=(Object)ois.readObject();
```

Nếu chương trình gửi dữ liệu thì ta sẽ lấy dữ liệu từ luồng xuất đến từ *Socket*:

```
ObjectOutput os=s.getObjectOutput();
```

Để tiến hành ghi đối tượng ta gắn kết luồng xuất thô lấy được từ *Socket* với luồng xuất đối tượng *ObjectOutputStream*:

```
ObjectOutputStream oos=new ObjectOutputStream(os);
```

Việc truyền đối tượng lúc này trở thành một công việc rất đơn giản:

```
oos.writeObject(obj);  
oos.flush();
```

6.2.3 Ví dụ minh họa

Để minh họa kỹ thuật chúng ta viết một server thực hiện phép nhân hai mảng số nguyên với nhau. Client gửi hai đối tượng, mỗi đối tượng biểu diễn một mảng nguyên; server nhận các đối tượng này, thực hiện lời gọi phương nhân hai mảng số nguyên với nhau và gửi kết quả trả về cho client.

Trước tiên chúng ta định nghĩa đối tượng để có thể sử dụng trong việc truyền các đối tượng.

```
public class ArrayObject implements java.io.Serializable{  
    private int[] a=null;  
    public ArrayObject(){ }  
    public void setArray(int a[]){  
        this.a=a;  
    }  
    public int[] getArray(){  
        return a;  
    }  
}
```

Lớp *ArrayObject* là lớp được xây dựng để đóng gói các mảng số nguyên và có khả năng truyền đi qua lại trên mạng. Cấu trúc lớp như sau: trường thông tin là một mảng số nguyên *a[]*; phương thức *setArray()* thiết lập giá trị cho mảng. Phương thức *getArray()* trả về một mảng số nguyên từ đối tượng *ArrayObject*.

Mô hình client/server tối thiểu phải có hai modul client và server. Trong ví dụ này cũng vậy ta sẽ xây dựng một số modul chương trình như sau:

Đầu tiên chúng ta phát triển client. Client tạo ra hai thể hiện của các đối tượng *ArrayObject* và ghi chúng ra luồng xuất (thực chất là gửi tới server).

```
public class ArrayClient{
    public static void main(String[] args)throws Exception{
        ObjectOutputStream oos=null;
        ObjectInputStream ois=null;
        int dat1[]={3,3,3,3,3,3,3};
        int dat2[]={5,5,5,5,5,5,5};
        Socket s=new Socket("localhost",1234);
        oos=new ObjectOutputStream(s.getOutputStream());
        ois=new ObjectInputStream(s.getInputStream());
        ArrayObject a1=new ArrayObject();
        a1.setArray(dat1);
        ArrayObject a2=new ArrayObject();
        a2.setArray(dat2);
        ArrayObject res=null;
        int r[]=new int[7];
        oos.writeObject(a1);
        oos.writeObject(a2);
        oos.flush();
        res=(ArrayObject)ois.readObject();
        r=res.getArray();
        System.out.println("The result received from server...");
        System.out.println();
        for(int i=0;i<r.length;i++)System.out.print(r[i]+" ");
    }
}
```

Bước tiếp theo chúng ta phát triển server. Server là một chương trình cung cấp dịch vụ phục vụ các yêu cầu của client. Server nhận hai đối tượng *ArrayObject* và nhận về hai mảng từ hai đối tượng này và sau đó đem nhân chúng với nhau và gửi kết quả trở lại cho client.

```
public class ArrayServer extends Thread {
    private ServerSocket ss;
    public static void main(String args[])throws Exception{
        new ArrayServer();
    }
    public ArrayServer()throws Exception{
```

```

ss=new ServerSocket(1234);
System.out.println("Server running on port "+1234);
this.start();
}
public void run(){
    while(true){
        try{
            System.out.println("Waiting for client...");
            Socket s=ss.accept();
            System.out.println("Accepting a connection
                                from:"+s.getInetAddress());

            Connect c=new Connect(s);
        }catch(Exception e){
            System.out.println(e);
        }
    }
}
}
}

```

Trong mô hình client/server tại một thời điểm server có thể phục vụ các yêu cầu đến từ nhiều client, điều này có thể dẫn đến các vấn đề tương tranh. Chính vì lý do này mà lớp *ArrayServer* thừa kế lớp *Thread* để giải quyết vấn đề trên. Ngoài ra để nâng cao hiệu suất của chương trình thì sau khi đã chấp nhận liên kết từ một client nào đó, việc xử lý dữ liệu sẽ được dành riêng cho một tuyến đoạn để server có thể tiếp tục chấp nhận các yêu cầu khác. Hay nói cách khác, mỗi một yêu cầu của client được xử lý trong một tuyến đoạn riêng biệt.

```

class Connect extends Thread{
    private Socket client=null;
    private ObjectInputStream ois;
    private ObjectOutputStream oos;
    public Connect(){ }
    public Connect(Socket client){
        this.client=client;
        try{
            ois=new ObjectInputStream(client.getInputStream());
            oos=new ObjectOutputStream(client.getOutputStream());
        }catch(Exception e){
            System.err.println(e);
        }
        this.start();
    }
}

```



```
public void run() {
    ArrayObject x=null;
    ArrayObject y=null;
    int a1[]=new int[7];
    int a2[]=new int[7];
    int r[]=new int[7];
    try{
        x=(ArrayObject)ois.readObject();
        y=(ArrayObject)ois.readObject();
        a1=x.getArray();
        a2=y.getArray();
        for(int i=0;i<a1.length;i++)r[i]=a1[i]*a2[i];
        ArrayObject res=new ArrayObject();
        res.setArray(r);
        oos.writeObject(res);
        oos.flush();
        ois.close();
        client.close();
    }catch(Exception e){}
}
```

CÂU HỎI ÔN TẬP

1. Vì sao áp dụng đa tiến trình cho ứng dụng mạng?
2. Tuần tự hóa đối tượng là gì? Khi nào áp dụng tuần tự hóa đối tượng?
3. Viết chương trình thực hiện mô phỏng giao thức FTP như sau: Chương trình client: thực hiện chức năng gửi yêu cầu đường dẫn tên tập tin tới Server sau đó đợi đáp trả từ server dữ liệu liên quan đến tập tin và hiển thị thông tin dữ liệu tập tin lên màn hình và lưu thành tập tin phía máy client nếu tìm thấy ngoài ra hiển thị chuỗi lỗi. Chương trình server: có nhiệm vụ lắng nghe kết nối từ client, tìm và mở tập tin liên quan theo yêu cầu của client viết về cho client sau đó đóng luồng lại.

BÀI 7. LẬP TRÌNH SOCKET CHO GIAO THỨC UDP

7.1 TỔNG QUAN UDP

UDP là giao thức nằm ở tầng giao vận, phía trên giao thức IP. Tầng giao vận cung cấp khả năng truyền tin giữa các mạng thông qua các gateway. Nó sử dụng các địa chỉ IP để gửi các gói tin trên Internet hoặc trên mạng thông qua các trình điều khiển thiết bị khác nhau. Giao thức UDP là giao thức đơn giản, phi liên kết và cung cấp dịch vụ trên tầng giao vận với tốc độ nhanh. Nó hỗ trợ liên kết một-nhiều và thường được sử dụng thường xuyên trong liên kết một-nhiều bằng cách sử dụng các datagram multicast và unicast.

7.1.1 Một số thuật ngữ UDP

Trước khi kiểm tra xem giao thức UDP hoạt động như thế nào, chúng ta cần làm quen với một số thuật ngữ. Trong phần dưới đây, chúng ta sẽ định nghĩa một số thuật ngữ cơ bản có liên quan đến giao thức UDP.

Packet: Trong truyền số liệu, một packet là một dãy các số nhị phân, biểu diễn dữ liệu và các tín hiệu điều khiển, các gói tin này được chuyển đi và chuyển tới tới host. Trong gói tin, thông tin được sắp xếp theo một khuôn dạng cụ thể.

Datagram: Một datagram là một gói tin độc lập, tự chứa, mang đầy đủ dữ liệu để định tuyến từ nguồn tới đích mà không cần thông tin thêm.

MTU: MTU là viết tắt của Maximum Transmission Unit. MTU là một đặc trưng của tầng liên kết mô tả số byte dữ liệu tối đa có thể truyền trong một gói tin. Mặt khác,

MTU là gói dữ liệu lớn nhất mà môi trường mạng cho trước có thể truyền. Ví dụ, Ethernet có MTU cố định là 1500 byte. Trong UDP, nếu kích thước của một datagram lớn hơn MTU, IP sẽ thực hiện phân đoạn, chia datagram thành các phần nhỏ hơn (các đoạn), vì vậy mỗi đoạn nhỏ có kích thước nhỏ hơn MTU.

Port: UDP sử dụng các cổng để ánh xạ dữ liệu đến vào một tiến trình cụ thể đang chạy trên một máy tính. UDP định đường đi cho packet tại vị trí xác định bằng cách sử dụng số hiệu cổng được xác định trong header của datagram. Các cổng được biểu diễn bởi các số 16-bit, vì thế các cổng nằm trong dải từ 0 đến 65535. Các cổng cũng được xem như là các điểm cuối của các liên kết logic, và được chia thành ba loại sau: Các cổng phổ biến: Từ 0 đến 1023; Các cổng đã đăng ký: 1024 đến 49151; Các cổng động/dành riêng 49152 đến 65535.

Chú ý rằng các cổng UDP có thể nhận nhiều hơn một thông điệp ở một thời điểm. Trong một số trường hợp, các dịch vụ TCP và UDP có thể sử dụng cùng một số hiệu cổng, như 7 (Echo) hoặc trên cổng 23 (Telnet). UDP có các cổng thông dụng trong bảng 7-1.

Bảng 7.1: một số cổng thông dụng của giao thức UDP

Cổng UDP	Mô tả
15	Netstat- Network Status-Tình trạng mạng
53	DNS-Domain Name Server
69	TFTP-Trivial File Transfer Protocol Giao thức truyền tệp thông thường
137	NetBIOS Name Service
138	Dịch vụ Datagram NetBIOS
161	SNMP

TTL (Time To Live): Giá trị TTL cho phép chúng ta thiết lập một giới hạn trên của các router mà một datagram có thể đi qua. Giá trị TTL ngăn ngừa các gói tin khỏi bị

kết trong các vòng lặp định tuyến vô hạn. TTL được khởi tạo bởi phía gửi và giá trị được giảm đi bởi mỗi router quản lý datagram. Khi TTL bằng 0, datagram bị loại bỏ.

Multicasting: Multicasting là phương pháp dựa trên chuẩn có tính chất mở để phân phối các thông tin giống nhau đến nhiều người dùng. Multicasting là một đặc trưng chính của giao thức UDP. Multicasting cho phép chúng ta truyền tin theo kiểu một nhiều, ví dụ gửi tin hoặc thư điện tử tới nhiều người nhận, đài phát thanh trên Internet, hoặc các chương trình demo trực tuyến.

7.1.2 Hoạt động của giao thức UDP

Khi một ứng dụng dựa trên giao thức UDP gửi dữ liệu tới một host khác trên mạng, UDP thêm vào một header có độ dài 8 byte chứa các số hiệu cổng nguồn và đích, cùng với tổng chiều dài dữ liệu và thông tin checksum. IP thêm vào header của riêng nó vào đầu mỗi datagram UDP để tạo lên một datagram IP.

7.1.3 Các nhược điểm của giao thức UDP

So với giao thức TCP, UDP có những nhược điểm sau:

- Thiếu các tín hiệu bắt tay. Trước khi gửi một đoạn, UDP không gửi các tín hiệu bắt tay giữa bên gửi và bên nhận. Vì thế phía gửi không có cách nào để biết datagram đã đến đích hay chưa. Do vậy, UDP không đảm bảo việc dữ liệu đã đến đích hay chưa.
- Sử dụng các phiên. Để TCP là hướng liên kết, các phiên được duy trì giữa các host. TCP sử dụng các chỉ số phiên (session ID) để duy trì các liên kết giữa hai host. UDP không hỗ trợ bất kỳ phiên nào do bản chất phi liên kết của nó.
- Độ tin cậy. UDP không đảm bảo rằng chỉ có một bản sao dữ liệu tới đích. Để gửi dữ liệu tới các hệ thống cuối, UDP phân chia dữ liệu thành các đoạn nhỏ. UDP không đảm bảo rằng các đoạn này sẽ đến đích đúng thứ tự như chúng đã được tạo ra ở nguồn. Ngược lại, TCP sử dụng các số thứ tự cùng với số hiệu cổng và các gói tin xác thực thường xuyên, điều này đảm bảo rằng các gói tin đến đích đúng thứ tự mà nó đã được tạo ra.

- Bảo mật. TCP có tính bảo mật cao hơn UDP. Trong nhiều tổ chức, firewall và router cấm các gói tin UDP, điều này là vì các hacker thường sử dụng các cổng UDP.
- Kiểm soát luồng. UDP không có kiểm soát luồng; kết quả là, một ứng dụng UDP được thiết kế tồi có thể làm giảm băng thông của mạng.

7.1.4 Các ưu điểm của UDP

Không cần thiết lập liên kết. UDP là giao thức phi liên kết, vì thế không cần phải thiết lập liên kết. Vì UDP không sử dụng các tín hiệu handshaking, nên có thể tránh được thời gian trễ. Đó chính là lý do tại sao DNS thường sử dụng giao thức UDP hơn là TCP-DNS sẽ chậm hơn rất nhiều khi dùng TCP. Tốc độ. UDP nhanh hơn so với TCP. Bởi vì điều này, nhiều ứng dụng thường được cài đặt trên giao thức UDP hơn so với giao thức TCP. Hỗ trợ hình trạng (Topology). UDP hỗ trợ các liên kết 1-1, 1-n, ngược lại TCP chỉ hỗ trợ liên kết 1-1. Kích thước header. UDP chỉ có 8 byte header cho mỗi đoạn, ngược lại TCP cần các header 20 byte, vì vậy sử dụng băng thông ít hơn.

Bảng 7.2: tổng kết những sự khác nhau giữa hai giao thức TCP và UDP

Các đặc trưng	UDP	TCP
Hướng liên kết	Không	Có
Sử dụng phiên	Không	Có
Độ tin cậy	Không	Có
Xác thực	Không	Có
Đánh thứ tự	Không	Có
Điều khiển luồng	Không	Có
Bảo mật	Ít	Nhiều hơn

7.1.5 Khi nào thì nên sử dụng UDP

Rất nhiều ứng dụng trên Internet sử dụng UDP. Dựa trên các ưu và nhược điểm của UDP chúng ta có thể kết luận UDP có ích khi:

- Sử dụng cho các phương thức truyền broadcasting và multicasting khi chúng ta muốn truyền tin với nhiều host.
- Kích thước datagram nhỏ và trình tự đoạn là không quan trọng
- Không cần thiết lập liên kết
- Ứng dụng không gửi các dữ liệu quan trọng
- Không cần truyền lại các gói tin
- Băng thông của mạng đóng vai trò quan trọng

Việc cài đặt ứng dụng UDP trong Java cần có hai lớp là DatagramPacket và DatagramSocket. DatagramPacket đóng gói các byte dữ liệu vào các gói tin UDP được gọi là datagram và cho phép ta mở các datagram khi nhận được. Một DatagramSocket đồng thời thực hiện cả hai nhiệm vụ nhận và gửi gói tin. Để gửi dữ liệu, ta đặt dữ liệu trong một DatagramPacket và gửi gói tin bằng cách sử dụng DatagramSocket. Để nhận dữ liệu, ta nhận một đối tượng DatagramPacket từ DatagramSocket và sau đó đọc nội dung của gói tin.

UDP không có bất kỳ khái niệm nào về liên kết giữa hai host. Một socket gửi tất cả dữ liệu tới một cổng hoặc nhận tất cả dữ liệu từ một cổng mà không cần quan tâm host nào gửi. Một DatagramSocket có thể gửi dữ liệu tới nhiều host độc lập hoặc nhận dữ liệu từ nhiều host độc lập. Socket không dành riêng cho một liên kết cụ thể thể nào cả như trong giao thức TCP. Các socket TCP xem liên kết mạng như là một luồng: ta gửi và nhận dữ liệu với các luồng nhập và luồng xuất nhận được từ socket. UDP không cho phép điều này; ta phải làm việc với từng gói tin. Tất cả dữ liệu được đặt trong datagram được gửi đi dưới dạng một gói tin. Gói tin này cũng có thể nhận được bởi một nhóm hoặc cũng có thể bị mất. Một gói tin không nhất thiết phải liên quan đến gói tin tiếp theo. Cho trước hai gói tin, không có cách nào để biết được gói tin nào được gửi trước và gói tin nào được gửi sau.

7.2 DATAGRAMPACKET

Các datagram UDP đưa rất ít thông tin vào datagram IP. Header UDP chỉ đưa tám byte vào header IP. Header UDP bao gồm số hiệu cổng nguồn và đích, chiều dài của dữ liệu và header UDP, tiếp đến là một checksum tùy chọn. Vì mỗi cổng được biểu diễn bằng hai byte nên tổng số cổng UDP trên một host sẽ là 65536. Chiều dài cũng được biểu diễn bằng hai byte nên số byte trong datagram tối đa sẽ là 65536 trừ đi tám 8 byte dành cho phần thông tin header. Trong Java, một datagram UDP được biểu diễn bởi lớp DatagramPacket: `public final class DatagramPacket extends Object`

Lớp này cung cấp các phương thức để nhận và thiết lập các địa chỉ nguồn, đích từ header IP, nhận và thiết lập các thông tin về cổng nguồn và đích, nhận và thiết lập độ dài dữ liệu. Các trường thông tin còn lại không thể truy nhập được từ mã Java thuần túy. DatagramPacket sử dụng các constructor khác nhau tùy thuộc vào gói tin được sử dụng để gửi hay nhận dữ liệu.

7.2.1 Các hàm khởi tạo để nhận datagram

Hai constructor tạo ra các đối tượng DatagramSocket mới để nhận dữ liệu từ mạng:

```
public DatagramPacket(byte[] b, int length)
public DatagramPacket(byte[] b, int offset, int length)
```

Khi một socket nhận một datagram, nó lưu trữ phần dữ liệu của datagram ở trong vùng đệm b bắt đầu tại vị trí b[0] và tiếp tục cho tới khi gói tin được lưu trữ hoàn toàn hoặc cho tới khi lưu trữ hết length byte. Nếu sử dụng constructor thứ hai, thì dữ liệu được lưu trữ bắt đầu từ vị trí b[offset]. Chiều dài của b phải nhỏ hơn hoặc bằng b.length-offset. Nếu ta xây dựng một DatagramPacket có chiều dài vượt quá chiều dài của vùng đệm thì constructor sẽ đưa ra ngoại lệ IllegalArgumentException. Đây là kiểu ngoại lệ RuntimeException nên chương trình của ta không cần thiết phải đón bắt ngoại lệ này. Ví dụ, xây dựng một DatagramPacket để nhận dữ liệu có kích thước lên tới 8912 byte

```
byte b[]=new byte[8912];
DatagramPacket dp=new DatagramPacket(b,b.length);
```

7.2.2 Các hàm khởi tạo để gửi các datagram

Bốn constructor tạo các đối tượng DatagramPacket mới để gửi dữ liệu trên mạng:

```
public DatagramPacket(byte[] b, int length, InetAddress dc, int port)
public DatagramPacket(byte[] b, int offset, int length, InetAddress dc, int port)
public DatagramPacket(byte[] b, int length, SocketAddress dc, int port)
public DatagramPacket(byte[] b, int offset, int length, SocketAddress dc, int port)
```

Mỗi constructor tạo ra một DatagramPacket mới để được gửi đi tới một host khác. Gói tin được điền đầy dữ liệu với chiều dài là length byte bắt đầu từ vị trí offset hoặc vị trí 0 nếu offset không được sử dụng. Ví dụ để gửi đi một chuỗi ký tự đến một host khác như sau:

```
String s="This is an example of UDP Programming";
byte[] b= s.getBytes();
try{
    InetAddress dc=InetAddress.getByName("www.vnn.vn");
    int port =7;
    DatagramPacket dp=new DatagramPacket(b,b.length,dc,port);
    //Gửi gói tin
}
catch(IOException e){
    System.err.println(e);
}
```

Công việc khó khăn nhất trong việc tạo ra một đối tượng DatagramPacket chính là việc chuyển đổi dữ liệu thành một mảng byte. Đoạn mã trên chuyển đổi một chuỗi ký tự thành một mảng byte để gửi dữ liệu đi.

7.2.3 Các phương thức nhận dữ liệu từ DatagramPacket

DatagramPacket có sáu phương thức để tìm các phần khác nhau của một datagram: dữ liệu thực sự cộng với một số trường header. Các phương thức này thường được sử dụng cho các datagram nhận được từ mạng.

Bảng 7.3: các phương thức nhận dữ liệu trong DatagramPacke

Phương thức	Ý nghĩa
<pre>public InetAddress getAddress()</pre>	<p>Phương thức getAddress() trả về một đối tượng InetAddress chứa địa chỉ IP của host ở xa. Nếu datagram được nhận từ Internet, địa chỉ trả về chính là địa chỉ của máy đã gửi datagram (địa chỉ nguồn). Mặt khác nếu datagram được tạo cục bộ để được gửi tới máy ở xa, phương thức này trả về địa chỉ của host mà datagram được đánh địa chỉ.</p>
<pre>public int getPort()</pre>	<p>Phương thức getPort() trả về một số nguyên xác định cổng trên host ở xa. Nếu datagram được nhận từ Internet thì cổng này là cổng trên host đã gửi gói tin đi.</p>
<pre>public SocketAddress()</pre>	<p>Phương thức này trả về một đối tượng SocketAddress chứa địa chỉ IP và số hiệu cổng của host ở xa.</p>
<pre>public byte[] getData()InterruptedException</pre>	<p>Phương thức getData() trả về một mảng byte chứa dữ liệu từ datagram. Thông thường cần phải chuyển các byte này thành một dạng dữ liệu khác trước khi chương trình xử lý dữ liệu.</p>
<pre>public void setAddress(InetAddress dc)</pre>	<p>Phương thức setAddress() thay đổi địa chỉ của máy mà ta sẽ gửi gói tin tới. Điều này sẽ cho phép ta gửi cùng một datagram đến nhiều nơi nhận.</p>
<pre>public void setPort(int port)</pre>	<p>Phương thức này thay đổi số hiệu cổng gửi tới của gói tin.</p>
<pre>public void setLength(int)</pre>	<p>Phương thức này thay đổi số byte dữ liệu có</p>

length)

thể đặt trong vùng đệm.

Ví dụ đoạn mã sau đây sẽ gửi dữ liệu theo từng đoạn 512 byte:

```
int offset=0;
DatagramPacket dp=new DatagramPacket(b,offset,512);
int bytesSent=0;
while(bytesSent<b.length){
    ds.send(dp);
    bytesSent+=dp.getLength();
    int bytesToSend=b.length-bytesSent;
    int size=(bytesToSend>512):512:bytesToSend;
    dp.setData(b,byteSent,512);
}
```

7.3 DATAGRAMSOCKET

Để gửi hoặc nhận một DatagramPacket, bạn phải mở một DatagramSocket. Trong Java, một datagram socket được tạo ra và được truy xuất thông qua đối tượng DatagramSocket

```
public class DatagramSocket extends Object
```

Tất cả các datagram được gắn với một cổng cục bộ, cổng này được sử dụng để lắng nghe các datagram đến hoặc được đặt trên các header của các datagram sẽ gửi đi. Nếu ta viết một client thì không cần phải quan tâm đến số hiệu cổng cục bộ là bao nhiêu

DatagramSocket được sử dụng để gửi và nhận các gói tin UDP. Nó cung cấp các phương thức để gửi và nhận các gói tin, cũng như xác định một giá trị timeout khi sử dụng phương pháp vào ra không phong tỏa (non blocking I/O), kiểm tra và sửa đổi kích thước tối đa của gói tin UDP, đóng socket.

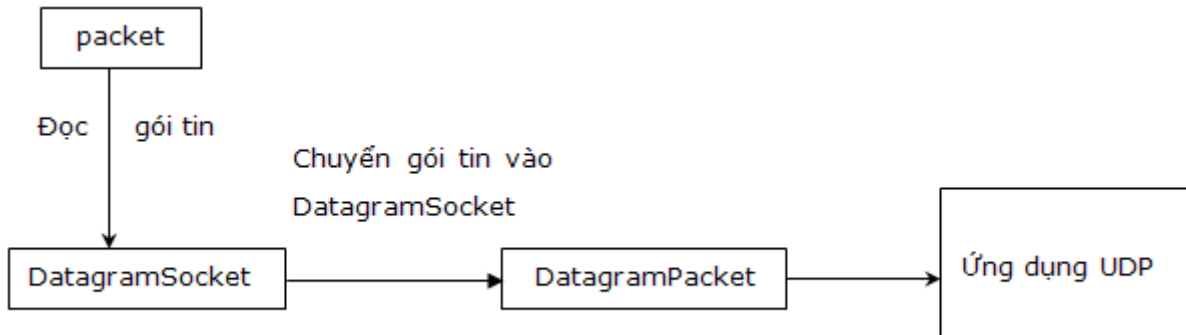
Bảng 7.4: Các phương thức hỗ trợ của DatagramSocket

Phương thức	Ý nghĩa
void close()	Đóng một liên kết và giải phóng nó khỏi cổng cục

	bộ.
InetAddress getInetAddress()	Phương thức này trả về địa chỉ remote mà socket kết nối tới, hoặc giá trị null nếu không tồn tại liên kết.
InetAddress getLocalAddress()	trả về địa chỉ cục bộ
Int getSoTimeout()	trả về giá trị tùy chọn timeout của socket. Giá trị này xác định thời gian mà thao tác đọc sẽ phong tỏa trước khi nó đưa ra ngoại lệ InterruptedException. Ở chế độ mặc định, giá trị này bằng 0, chỉ ra rằng vào ra không phong tỏa được sử dụng.
void receive(DatagramPacket dp) throws IOException	phương thức đọc một gói tin UDP và lưu nội dung trong packet xác định.
void send(DatagramSocket dp) throws IOException	phương thức gửi một gói tin
void setSoTimeout(int timeout)	thiết lập giá trị tùy chọn của socket.

7.4 GỬI/NHẬN GÓI TIN

7.4.1 Nhận gói tin



Hình 7.1: minh họa quá trình nhận gói tin UDP.

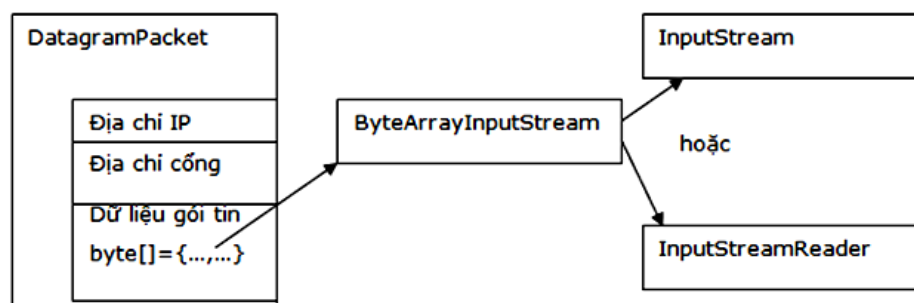
Trước khi một ứng dụng có thể đọc các gói tin UDP được gửi bởi các máy ở xa, nó phải gán một socket với một cổng UDP bằng cách sử dụng `DatagramSocket`, và tạo ra một `DatagramPacket` sẽ đóng vai trò như là một bộ chứa cho dữ liệu của gói tin UDP. Hình vẽ dưới đây chỉ ra mối quan hệ giữa một gói tin UDP với các lớp Java khác nhau được sử dụng để xử lý nó và các ứng dụng thực tế.

Khi một ứng dụng muốn đọc các gói tin UDP, nó gọi phương thức `DatagramSocket.receive()`, phương thức này sao chép gói tin UDP vào một `DatagramPacket` xác định. Xử lý nội dung nói tin và tiến trình lặp lại khi cần

```
DatagramPacket dp=new DatagramPacket(new byte[256],256);
DatagramSocket ds=new DatagramSocket(2000);
boolean finished=false;
while(!finished){
    ds.receive(dp); //Xử lý gói tin
}
ds.close();
```

Khi xử lý gói tin ứng dụng phải làm việc trực tiếp với một mảng byte. Tuy nhiên nếu ứng dụng là đọc văn bản thì ta có thể sử dụng các lớp từ gói vào ra để chuyển đổi giữa mảng byte và luồng stream và reader. Bằng cách gắn kết luồng nhập `ByteArrayInputStream` với nội dung của một datagram và sau đó kết nối với một kiểu luồng khác, khi đó bạn có thể truy xuất tới nội dung của gói UDP một cách dễ dàng. Rất nhiều người lập trình thích dùng các luồng vào ra I/O để xử lý dữ liệu, bằng cách

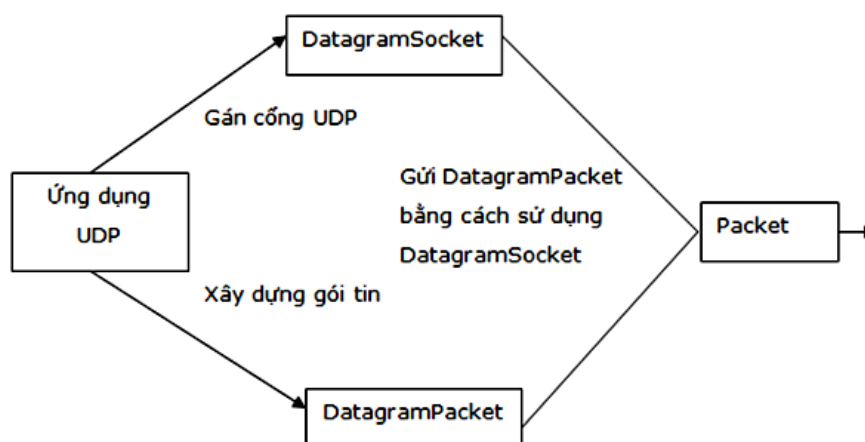
sử dụng luồng `DataInputStream` hoặc `BufferedReader` để truy xuất tới nội dung của các mảng byte.



Hình 7.2: Minh họa ứng dụng luồng mảng trong ứng dụng UDP

7.4.2 Gửi gói tin

Lớp `DatagramSocket` cũng được sử dụng để gửi các gói tin. Khi gửi gói tin, ứng dụng phải tạo ra một `DatagramPacket`, thiết lập địa chỉ và thông tin cổng, và ghi dữ liệu cần truyền vào mảng byte. Nếu muốn gửi thông tin phức tạp thì ta cũng đã biết địa chỉ và số hiệu cổng của gói tin nhận được. Mỗi khi gói tin sẵn sàng để gửi, ta sử dụng phương thức `send()` của lớp `DatagramSocket` để gửi gói tin đi.



Hình 7.3: Minh hoạt quá trình xử lý gói tin của ứng dụng UDP.

```

//Socket lắng nghe các gói tin đến trên cổng 2000
DatagramSocket socket = new DatagramSocket(2000);
DatagramPacket packet = new DatagramPacket (new byte[256], 256);
packet.setAddress ( InetAddress.getByName ( somehost ) );
packet.setPort ( 2000 );
boolean finished = false;
while !finished ){
  
```

```
// Ghi dữ liệu vào vùng đệm buffer
.....
socket.send (packet);
// Thực hiện hành động nào đó còn gói tin nào cần gửi đi hay không
.....
}
socket.close();
```

7.4.3 Ví dụ minh họa giao thức UDP

Để minh họa các gói tin UDP được gửi và nhận như thế nào, chúng ta sẽ viết, biên dịch và chạy ứng dụng sau. Viết chương trình theo mô hình Client/Server để:

Client thực hiện các thao tác sau đây: Client gửi một chuỗi ký tự do người dùng nhập từ bàn phím cho server; Client nhận thông tin phản hồi trở lại từ Server và hiển thị thông tin đó trên màn hình.

Server thực hiện các thao tác sau: Server nhận chuỗi ký tự do client gửi tới và in lên màn hình; Server biến đổi chuỗi ký tự thành chữ hoa và gửi trở lại cho Client.

```
import java.net.*;
import java.io.*;
public class UDPClient
{
    public final static int CONG_MAC_DINH=9;
    public static void main(String args[]){
        String hostname;
        int port=CONG_MAC_DINH;
        hostname="127.0.0.1";
        try{
            InetAddress dc=InetAddress.getByName(hostname);
            BufferedReader userInput=new BufferedReader(new
            InputStreamReader(System.in));
            DatagramSocket ds =new DatagramSocket(port);
            while(true){
                String line=userInput.readLine();
                if(line.equals("exit"))break;
                byte[] data=line.getBytes();
                DatagramPacket dp=new DatagramPacket(data,data.length,dc,port);
                ds.send(dp);
                dp.setLength(65507);
                ds.receive(dp);
                ByteArrayInputStream bis =new
                ByteArrayInputStream(dp.getData());
                BufferedReader dis =new BufferedReader(new InputStreamReader(bis));
                System.out.println(dis.readLine());
            }
        }
```

```

    } catch (Exception e) {
        System.err.println(e);
    }
}
}

```

```

import java.net.*;
import java.io.*;
public class UDPServer
{
    public final static int CONG_MAC_DINH=9;
    public static void main(String args[]) {
        int port=CONG_MAC_DINH;
        try{
            DatagramSocket ds =new DatagramSocket(port);
            DatagramPacket dp=new DatagramPacket(new byte[65507],65507);
            while(true){
                ds.receive(dp);
                ByteArrayInputStream bis=new ByteArrayInputStream(dp.getData());
                BufferedReader dis;
                dis =new BufferedReader(new InputStreamReader(bis));
                String s=dis.readLine();
                System.out.println(s);
                s.toUpperCase();
                dp.setData(s.getBytes());
                dp.setLength(s.length());
                dp.setAddress(dp.getAddress());
                dp.setPort(dp.getPort());
                ds.send(dp);
            }
        } catch (Exception e) {
            System.err.println(e);
        }
    }
}

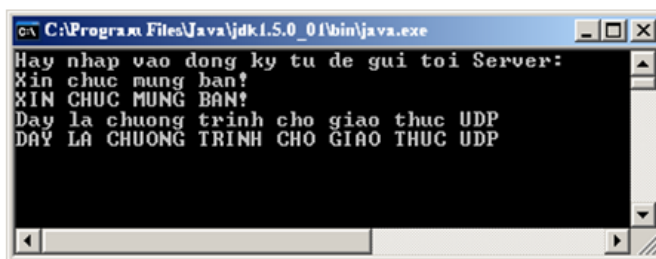
```

Chạy chương trình

```

C:\>start java UDPServer
C:\>start java UDPClient

```



Hình 7.4: Minh họa hoạt động phần mềm máy chủ

CÂU HỎI ÔN TẬP

1. Giao thức UDP là gì? Cấu trúc Header của UDP? Ưu/nhược điểm của giao thức UDP? Khi nào sử dụng giao thức UDP cho việc phát triển các ứng dụng mạng?
2. Trình bày các kiểu dữ liệu liên quan đến lập trình UDP? Ý nghĩa sử dụng từng kiểu dữ liệu?
3. Viết chương trình minh hoạt giao thức FTP cho phép hai máy gửi tập tin cho nhau sử dụng UDP Socket.

BÀI 8. LẬP TRÌNH MULTICAST

8.1 TỔNG QUAN MULTICAST

Có 3 kiểu truyền dữ liệu trên router và switch:

Unicast: Các gói tin được gửi từ một địa chỉ nguồn đến một địa chỉ đích. Một router hoặc một thiết bị lớp 3 sẽ chuyển các gói tin bằng cách tìm địa chỉ đích trong bảng định tuyến để chuyển ra cổng vật lý tương ứng. Nếu một thiết bị là L2, nó chỉ cần dựa vào địa chỉ MAC.

Broadcast: Các gói tin được gửi từ một máy nguồn đến một địa chỉ đích broadcast. Địa chỉ đích có thể là địa chỉ tất cả các hosts (255.255.255.255) hoặc là một phần của địa chỉ subnet. Một router hoặc một L3 switch sẽ không cho phép chuyển các dữ liệu broadcast này. Một thiết bị L2 sẽ cho phép phát tán broadcast traffic ra tất cả các cổng của nó.

Multicast: Các gói được gửi từ một địa chỉ nguồn đến một nhóm các máy tính. Địa chỉ đích tương trưng bằng các hosts muốn nhận traffic này. Mặc định, một router hoặc một L3 switch sẽ không chuyển các gói tin này trừ khi phải cấu hình multicast routing. Một thiết bị L2 switch không thể nhận biết được vị trí của địa chỉ multicast đích. Tất cả các gói sẽ được phát tán ra tất cả các port ở chế độ mặc định.

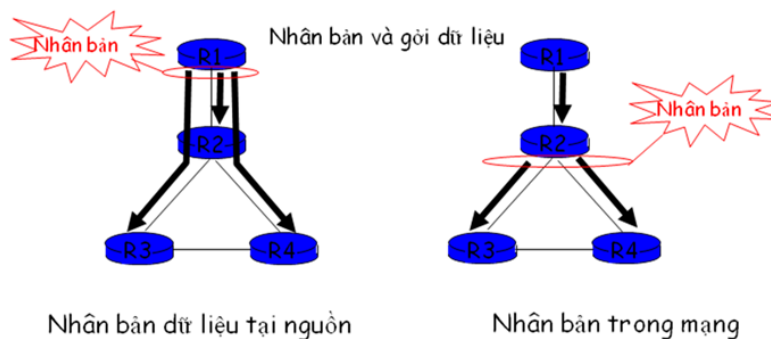
Có hai thái cực được mô tả ở đây. Cơ chế dùng unicast thì dữ liệu sẽ đi từ host đến host; broadcast thì traffic sẽ đi đến tất cả các host trên phân đoạn mạng đó. Cơ chế multicast sẽ nằm giữa hai thái cực này, trong đó máy nguồn chỉ gửi những gói tin từ một host đến các người dùng muốn nhận loại traffic đó. Nhóm này gọi là nhóm multicast. Các máy nhận multicast traffic có thể nằm ở bất cứ nơi nào chứ không chỉ trên phân đoạn mạng cục bộ. Các traffic dạng multicast thường là một chiều (unidirectional). Do có nhiều host nhận cùng một dữ liệu, nên thông thường các gói tin không được phép gửi ngược về máy nguồn trên cơ chế multicast. Một host đích sẽ

tra traffic ngược về nguồn theo cơ chế unicast. Cơ chế multicast cũng sẽ được truyền theo kiểu phi-kết-nối (connectionless). Multicast dùng UDP chứ không dùng TCP.

Các host muốn nhận dữ liệu từ một nguồn multicast có thể tham gia hoặc rời khỏi một nhóm multicast ở bất kỳ thời điểm nào. Hơn nữa, một host sẽ quyết định có trở thành thành viên của một hay nhiều nhóm multicast hay không. Nguyên tắc cần quan tâm là sẽ hoạch định làm thế nào để phân phối các multicast traffic đến các thành viên của nhóm mà không ảnh hưởng đến các thành viên ngoài nhóm.

8.1.1 Mục đích của multicast

Bởi vì nhiều máy trạm cùng nhận dữ liệu, nên có nhiều gói tin giống nhau tại một thời điểm trên đường truyền. Điều này dẫn đến tình trạng dư thừa gói dữ liệu gây hao tổn tài nguyên mạng và tắc nghẽn. Giải pháp multicast sẽ loại bỏ tình trạng này bằng cách sử dụng cây định tuyến multicast. Tại mỗi thời điểm trên đường truyền multicast chỉ có một gói tin được gửi đi.



Hình 8.1: Cơ chế nhân bản gói tin khi truyền trong cây multicast

8.1.2 Địa chỉ multicast

Các router và switch phải có phương thức để phân biệt traffic dạng multicast với dạng unicast hay broadcast. Điều này thực hiện thông qua việc gán địa chỉ IP, bằng cách dùng địa chỉ lớp D từ 224.0.0.0 đến 239.255.255.255 chỉ cho multicast. Các thiết bị mạng có thể nhanh chóng lọc ra các địa chỉ multicast bằng cách đọc 4 bit bên trái của một địa chỉ. Bốn bit này của một địa chỉ multicast luôn luôn bằng 1110. Làm thế nào mà một router và switch kết hợp một địa chỉ multicast của IP với một địa chỉ MAC. Do không có cơ chế tương đương với cơ chế ARP, một dạng giá trị đặc biệt dành

riêng cho địa chỉ MAC của multicast sẽ được dùng. Các địa chỉ này bắt đầu bằng 0100.5e. Phần 28 bit sau của địa chỉ multicast Ip sẽ được ánh xạ vào 23 bit thấp của địa chỉ MAC bằng một giải thuật đơn giản.

Tuy nhiên chú ý rằng có 5 bit của địa chỉ IP không được chuyển sang địa chỉ MAC. Khả năng này làm cho nảy sinh một vấn đề là có thể có 32 địa chỉ MAC khác nhau có thể ánh xạ vào cùng một địa chỉ MAC. Do sự nhập nhằng này, một host multicast có một vấn đề nhỏ khi nó nhận một Ethernet frame của một địa chỉ multicast. Một MAC có thể tương ứng với 32 địa chỉ multicast. Một vài không gian địa chỉ được dành riêng khác nhau. Vì vậy, khi một host phải nhận và kiểm tra tất cả các frame có MAC mà nó quan tâm. Sau đó host này phải kiểm tra phần địa chỉ IP bên trong mỗi frame để nhận ra phần địa chỉ của từng nhóm multicast.

Một vài không gian địa chỉ được dành riêng

- Toàn bộ không gian địa chỉ multicast: 224.0.0.0-239.255.255.255
- Địa chỉ link-local: 224.0.0.0 - 224.0.0.255 được dùng bởi các giao thức định tuyến.
- Router sẽ không chuyển các gói tin có địa chỉ này. Các địa chỉ bao gồm địa chỉ tất cả các host all-hosts 224.0.0.1, tất cả các router 224.0.0.2, tất cả các OSPF routers 224.0.0.5... Đây là địa chỉ các nhóm cố định vì các địa chỉ này được định nghĩa trước.
- Tầm địa chỉ dành cho quản trị (239.0.0.0-239.255.255.255) được dùng trong các vùng multicast riêng, giống như dãy địa chỉ dành riêng trong RFC1918. Địa chỉ này không được route giữa các domain nên nó có thể được dùng lại nhiều lần.
- Địa chỉ toàn cục (224.0.1.0-238.255.255.255) được dùng bởi bất cứ đối tượng nào. Các địa chỉ này có thể được route trên Internet, vì vậy địa chỉ này phải duy nhất.
- 224.0.1.7 multicast tin tức Audio
- 224.0.1.12 Video của IETF meetings

8.1.3 Định tuyến multicast

Các traffic IP phải được định tuyến giống như bất cứ một gói tin L3 nào. Sự khác nhau là ở điểm cần phải biết để chuyển gói tin về đâu. Các gói tin L3 dạng unicast chỉ có một cổng ra duy nhất trên router (ngay cả khi có quá trình load-balancing xảy ra), trong khi multicast traffic có thể được chuyển mạch ra nhiều cổng, tùy thuộc vào các máy nhận nằm ở đâu.

Cây multicast: Các router hoặc các multilayer switch trong một mạng phải xác định một tuyến đường để phân phối các gói tin multicast từ máy nguồn đến từng máy nhận. Khi đó, toàn bộ mạng giống như một cấu trúc cây, trong đó gốc của cây là nguồn của luồng traffic đó. Mỗi router dọc theo đường đi sẽ là một nhánh rẽ của cây. Nếu một router biết tất cả các địa chỉ multicast, router cũng phải biết cần phải nhân bản luồng multicast đó ra những nhánh nào của cây. Một vài router không có các máy nhận trong các phân đoạn mạng của nó thì các router đó sẽ không chuyển traffic. Các router khác sẽ có thể có các máy nhận multicast traffic. Cấu trúc cây này tương tự như cấu trúc cây Spanning Tree vì nó có một root và các lá. Cấu trúc cây này cũng đảm bảo là không bị vòng lặp sao cho traffic multicast không bị chuyển ngược về cây.

Giao thức IGMP: Làm thế nào một router biết được các máy cần nghe multicast traffic? Để nhận multicast traffic từ một nguồn, cả nguồn và các máy nhận đầu tiên phải gia nhập (join) vào một nhóm multicast. Nhóm này được xác định thông qua địa chỉ multicast. Một host có thể tham gia vào một nhóm multicast bằng cách gửi các yêu cầu đến router gần nhất. Tác vụ này được thực hiện thông qua giao thức IGMP. IGMPv1 được định nghĩa trong RFC1112 và bản cải tiến của nó, IGMPv2 được định nghĩa trong RFC2236. Khi có vài host muốn tham gia vào nhóm, giao thức PIM sẽ thông báo cho nhau giữa các router và hình thành nên cây multicast giữa các routers.

8.2 KIỂU DỮ LIỆU MULTICASTSOCKET

Thuộc gói thư viện `java.net.*` và là lớp con của lớp `java.net.DatagramSocket`. Các hàm khởi tạo tương tự lớp cha. Một số phương thức bổ sung:

Phương thức	Ý nghĩa
<code>void joinGroup(InetAddress mcastGroup)</code>	Nhập vào địa chỉ của ip multicast của nhóm.
<code>void leaveGroup(InetAddress mcastGroup)</code>	Rời nhóm địa chỉ.
<code>void setTimeToLive(int ttl)</code>	Thiết lập cách thức gói dữ liệu tồn tại trên đường truyền (qua số router).
<code>int getTimeToLive()</code>	Lấy số lượng router tối đa gói tin sẽ đi trên đường truyền.

8.3 CÁC BƯỚC LẬP TRÌNH MULTICAST

8.3.1 Các bước để gửi gói dữ liệu multicast

Việc gửi gói dữ liệu gồm Các bước như sau:

- Tạo đối tượng `MulticastSocket`.
- Tham gia nhóm multicast.
- Tạo đối tượng `DatagramPacket`.
- Gửi gói thông qua socket.
- Rời nhóm multicast.

Ví dụ:

```
InetAddress multicastGroup = InetAddress.getByName(multicastGroupAddr);
//tạo socket để gửi dữ liệu
MulticastSocket socket = new MulticastSocket();
socket.joinGroup(multicastGroup); //tham gia vào nhóm multicast
//thiết lập thời gian sống của gói dữ liệu (số router)
socket.setTimeToLive(5);
//dữ liệu gửi đi
byte[] data = "This is the message".getBytes();
```

```
//đóng gói dữ liệu để gửi đi
DatagramPacket datagram = new DatagramPacket(data, data.length);
//thiết lập địa chỉ multicast cho gói tới
datagram.setAddress(multicastGroup);
//thiết lập số hiệu cổng
datagram.setPort(9876);
//gửi gói dữ liệu đi
socket.send(datagram);
//thoát khỏi nhóm
socket.leaveGroup(multicastGroup);
```

8.3.2 Các bước để nhận gói multicast

Các bước như sau:

- Tạo đối tượng socket lắng nghe ở port tương ứng.
- Tham gia nhóm multicast.
- Tạo gói datagram rỗng.
- Đợi để nhận gói (datagram to) được phân tán tới socket.
- Sử dụng dữ liệu trong gói.
- Rời nhóm multicast.

Ví dụ:

```
//tạo địa chỉ nhóm multicast
InetAddress multicastGroup = InetAddress.getByName(multicastGroupAddr);
//tạo đối tượng socket để nhận gói dữ liệu
MulticastSocket socket = new MulticastSocket(9876);
//gia nhập vào nhóm multicast để đọc dữ liệu
socket.joinGroup(multicastGroup);
//tạo bộ đệm để nhận dữ liệu
byte[] data = new byte[1000];
//tạo gói rỗng dựa trên bộ đệm
DatagramPacket packet = new DatagramPacket(data, data.length);
//nhận dữ liệu vào gói
socket.receive(packet);
//lấy dữ liệu trong gói để hiển thị
String message = new String(packet.getData(), 0, packet.getLength());
//rời nhóm multicast
```

```
socket.leaveGroup(multicastGroup);
```

CÂU HỎI ÔN TẬP

1. Truyền tin Multicast là gì? Tại sao cần giải pháp Multicast?
2. Các hàm cơ bản của lớp MulticastSocket là gì? Ý nghĩa?
3. Viết chương trình gửi video cho một nhóm máy trên mạng.

BÀI 9. PHÂN TÁN ĐỐI TƯỢNG TRONG JAVA BẰNG RMI

9.1 TỔNG QUAN

RMI là một cơ chế cho phép một đối tượng đang chạy trên một máy ảo Java này (Java Virtual Machine) gọi các phương thức của một đối tượng đang tồn tại trên một máy ảo Java khác (JVM).

Thực chất RMI là một cơ chế gọi phương thức từ xa đã được thực hiện và tích hợp trong ngôn ngữ Java. Vì Java là một ngôn ngữ lập trình hướng đối tượng, nên phương pháp lập trình trong RMI là phương pháp hướng đối tượng do đó các thao tác hay các lời gọi phương thức đều liên quan đến đối tượng. Ngoài ra, RMI còn cho phép một Client có thể gửi tới một đối tượng đến cho Server xử lý, và đối tượng này cũng có thể được xem là tham số cho lời gọi hàm từ xa, đối tượng này cũng có những dữ liệu bên trong và các hành vi như một đối tượng thực sự.

So sánh giữa gọi phương thức từ xa với các lời gọi thủ tục từ xa: Gọi phương thức từ xa không phải là một khái niệm mới. Thậm chí trước khi ra đời lập trình hướng đối tượng phần mềm đã có thể gọi các hàm và các thủ tục từ xa. Các hệ thống như RPC đã được sử dụng trong nhiều năm và hiện nay vẫn được sử dụng.

Trước hết, Java là một ngôn ngữ độc lập với nền và cho phép các ứng dụng Java truyền tin với các ứng dụng Java đang chạy trên bất kỳ phần cứng và hệ điều hành nào có hỗ trợ JVM. Sự khác biệt chính giữa hai mục tiêu là RPC hỗ trợ đa ngôn ngữ, ngược lại RMI chỉ hỗ trợ các ứng dụng được viết bằng Java.

Ngoài vấn đề về ngôn ngữ và hệ thống, có một số sự khác biệt căn bản giữa RPC và RMI. Gọi phương thức từ xa làm việc với các đối tượng, cho phép các phương thức chấp nhận và trả về các đối tượng Java cũng như các kiểu dữ liệu nguyên tố

(primitive type). Ngược lại gọi thủ tục từ xa không hỗ trợ khái niệm đối tượng. Các thông điệp gửi cho một dịch vụ RPC (Remote Procedure Calling) được biểu diễn bởi ngôn ngữ XDR (External Data Representation): dạng thức biểu diễn dữ liệu ngoài. Chỉ có các kiểu dữ liệu có thể được định nghĩa bởi XDR mới có thể truyền đi.

9.1.1 Mục đích của RMI

Hỗ trợ gọi phương thức từ xa trên các đối tượng trong các máy ảo khác nhau. Hỗ trợ gọi ngược phương thức ngược từ server tới các applet. Tích hợp mô hình đối tượng phân tán vào ngôn ngữ lập trình Java theo một cách tự nhiên trong khi vẫn duy trì các ngữ cảnh đối tượng của ngôn ngữ lập trình Java. Làm cho sự khác biệt giữa mô hình đối tượng phân tán và mô hình đối tượng cục bộ không có sự khác biệt. Tạo ra các ứng dụng phân tán có độ tin cậy một cách dễ dàng. Duy trì sự an toàn kiểu được cung cấp bởi môi trường thời gian chạy của nền tảng Java. Hỗ trợ các ngữ cảnh tham chiếu khác nhau cho các đối tượng từ xa. Duy trì môi trường an toàn của Java bằng các trình bảo an và các trình nạp lớp.

9.1.2 Một số thuật ngữ

Cũng như tất cả các chương trình khác trong Java, chương trình RMI cũng được xây dựng bởi các Giao diện và lớp. Giao diện định nghĩa các phương thức và các lớp thực thi các phương thức đó. Ngoài ra lớp còn thực hiện một vài phương thức khác. Nhưng chỉ có những phương thức khai báo trong Giao diện thừa kế từ Giao diện Remote hoặc các lớp con của nó mới được Client gọi từ JVM khác. Trong mục này ta nêu một số thuật ngữ thường xuyên được sử dụng trong phần này:

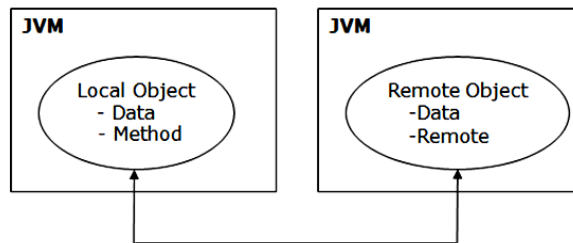
Giao diện Remote: Một Giao diện khai báo các phương thức cho phép gọi từ xa. Trong Java Giao diện Remote có các đặc điểm sau:

Thừa kế Giao diện có sẵn: `java.rmi.Remote`.

Mỗi phương thức trong Giao diện Remote phải được khai báo để đưa ra ngoại lệ `RemoteException` nhờ mệnh đề `throws java.rmi.RemoteException` và có thể có các ngoại lệ khác.

Đối tượng Remote: một đối tượng được tạo ra để cho phép những đối tượng khác trên một máy JVM khác gọi tới nó.

Phương thức Remote: Đối tượng Remote chứa một số các phương thức, những phương thức này có thể được gọi từ xa bởi các đối tượng trong JVM khác.



Hình 9.1: Mô hình triệu gọi đối tượng từ xa.

9.1.3 Các lớp trung gian Stub và Skeleton

Trong kỹ thuật lập trình phân tán RMI, để các đối tượng trên các máy Java ảo khác nhau có thể truyền tin với nhau thông qua các lớp trung gian: Stub và Skeleton.

Vai trò của lớp trung gian: Lớp trung gian tồn tại cả ở hai phía client (nơi gọi phương thức của các đối tượng ở xa) và server (nơi đối tượng thật sự được cài đặt để thực thi mã lệnh của phương thức). Trong Java trình biên dịch `rmic.exe` được sử dụng để tạo ra lớp trung gian này. Phía client lớp trung gian này gọi là Stub (lớp móc), phía server lớp trung gian này gọi là Skeleton (lớp nối) chúng giống như các lớp môi giới giúp các lớp ở xa truyền tin với nhau.

Cơ chế hoạt động của RMI

Các hệ thống RMI phục vụ cho việc truyền tin thường được chia thành hai loại: client và server. Một server cung cấp dịch vụ RMI, và client gọi các phương thức trên đối tượng của dịch vụ này.

Server RMI phải đăng ký với một dịch vụ tra tìm và đăng ký tên. Dịch vụ này cho phép các client truy tìm chúng, hoặc chúng có thể tham chiếu tới dịch vụ trong một mô hình khác. Một chương trình đóng vai trò như vậy có tên là `rmiregistry`, chương trình này chạy như một tiến trình độc lập và cho phép các ứng dụng đăng ký dịch vụ RMI hoặc nhận một tham chiếu tới dịch vụ được đặt tên. Mỗi khi server được đăng ký,

nó sẽ chờ các yêu cầu RMI từ các client. Gắn với mỗi đăng ký dịch vụ là một tên được biểu diễn bằng một chuỗi ký tự để cho phép các client lựa chọn dịch vụ thích hợp. Nếu một dịch vụ chuyển từ server này sang một server khác, client chỉ cần tra tìm trình đăng ký để tìm ra vị trí mới. Điều này làm cho hệ thống có khả năng dung thứ lỗi-nếu một dịch vụ không khả dụng do một máy bị sập, người quản trị hệ thống có thể tạo ra một thể hiện mới của dịch vụ trên một hệ thống khác và đăng ký nó với trình đăng ký RMI.

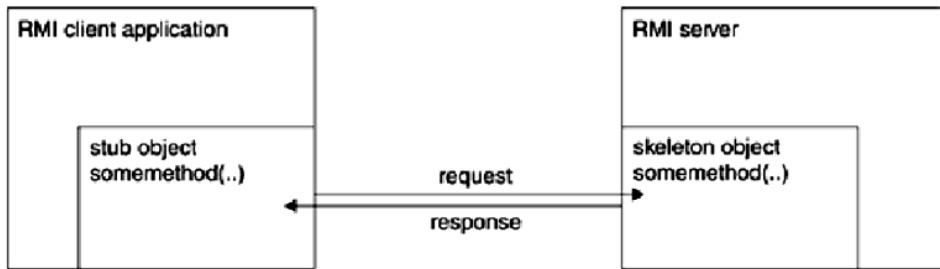
Các client RMI sẽ gửi các thông điệp RMI để gọi một phương thức trên một đối tượng từ xa. Trước khi thực hiện gọi phương thức từ xa, client phải nhận được một tham chiếu từ xa. Tham chiếu này thường có được bằng cách tra tìm một dịch vụ trong trình đăng ký RMI. Ứng dụng client yêu cầu một tên dịch vụ cụ thể, và nhận một URL trỏ tới tài nguyên từ xa. Khuôn dạng dưới đây được sử dụng để biểu diễn một tham chiếu đối tượng từ xa:

```
rmi://hostname:port/servicename
```

Trong đó hostname là tên của máy chủ hoặc một địa chỉ IP, port xác định dịch vụ, và servicename là một chuỗi ký tự mô tả dịch vụ.

Mỗi khi có được một tham chiếu, client có thể tương tác với dịch vụ từ xa. Các chi tiết liên quan đến mạng hoàn toàn được che dấu đối với những người phát triển ứng dụng-làm việc với các đối tượng từ xa đơn giản như làm việc với các đối tượng cục bộ. Điều này có thể có được thông qua sự phân chia hệ thống RMI thành hai thành phần, stub và skeleton.

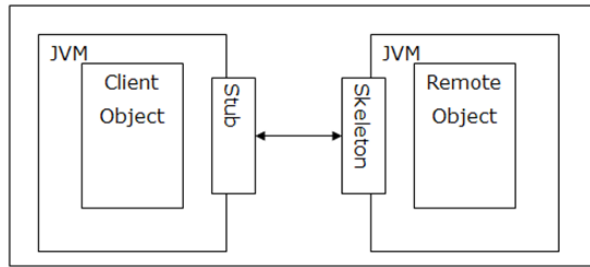
Đối tượng stub là một đối tượng ủy quyền, truyền tải yêu cầu đối tượng tới server RMI. Cần nhớ rằng mỗi dịch vụ RMI được định nghĩa như là một Giao diện, chứ không phải là một chương trình cài đặt, các ứng dụng client giống như các chương trình hướng đối tượng khác. Tuy nhiên ngoài việc thực hiện công việc của chính nó, stub còn truyền một thông điệp tới một dịch vụ RMI ở xa, chờ đáp ứng, và trả về đáp ứng cho phương thức gọi. Người phát triển ứng dụng không cần quan tâm đến tài nguyên RMI nằm ở đâu, nó đang chạy trên nền nào, nó đáp ứng đầy đủ yêu cầu như thế nào. Client RMI đơn giản gọi một phương thức trên đối tượng ủy quyền, đối tượng này quản lý tất cả các chi tiết cài đặt.



Hình 9.2: Minh họa vai trò của stub và skeleton.

Tại phía server, đối tượng skeleton có nhiệm vụ lắng nghe các yêu cầu RMI đến và truyền các yêu cầu này tới dịch vụ RMI. Đối tượng skeleton không cung cấp bản cài đặt của dịch vụ RMI. Nó chỉ đóng vai trò như là chương trình nhận các yêu cầu, và truyền các yêu cầu. Sau khi người phát triển tạo ra một Giao diện RMI, thì anh ta phải cung cấp một phiên bản cài đặt cụ thể của Giao diện. Đối tượng cài đặt này được gọi là đối tượng skeleton, đối tượng này gọi phương thức tương ứng và truyền các kết quả cho đối tượng stub trong client RMI. Mô hình này làm cho việc lập trình trở nên đơn giản, vì skeleton được tách biệt với cài đặt thực tế của dịch vụ. Tất cả những gì mà người phát triển dịch vụ cần quan tâm là mã lệnh khởi tạo (để đăng ký dịch vụ và chấp nhận dịch vụ), và cung cấp chương trình cài đặt của Giao diện dịch vụ RMI.

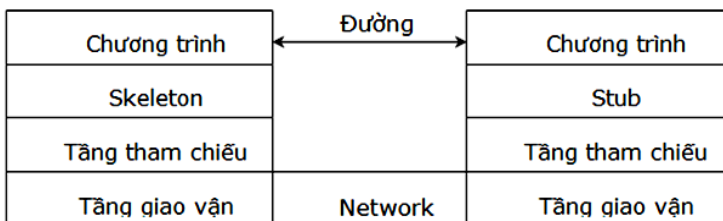
Với câu hỏi các thông điệp được truyền như thế nào, câu trả lời tương đối đơn giản. Việc truyền tin diễn ra giữa các đối tượng stub và skeleton bằng cách sử dụng các socket TCP. Mỗi khi được tạo ra, skeleton lắng nghe các yêu cầu đến được phát ra bởi các đối tượng stub. Các tham số trong hệ thống RMI không chỉ hạn chế đối với các kiểu dữ liệu nguyên tố-bất kỳ đối tượng nào có khả năng tuần tự hóa đều có thể được truyền như một tham số hoặc được trả về từ phương thức từ xa. Khi một stub truyền một yêu cầu tới một đối tượng skeleton, nó phải đóng gói các tham số (hoặc là các kiểu dữ liệu nguyên tố, các đối tượng hoặc cả hai) để truyền đi, quá trình này được gọi là marshalling. Tại phía skeleton các tham số được khôi phục lại để tạo nên các kiểu dữ liệu nguyên tố và các đối tượng, quá trình này còn được gọi là unmarshaling. Để thực hiện nhiệm vụ này, các lớp con của các lớp `ObjectOutputStream` và `ObjectInputStream` được sử dụng để đọc và ghi nội dung của các đối tượng.



Hình 9.3: Kiến trúc triệu gọi từ xa qua stub và skeleton

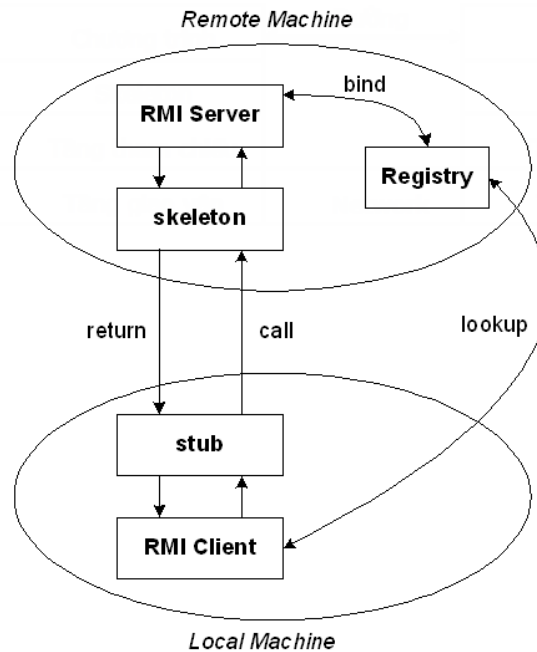
9.1.4 Kiến trúc RMI

Sự khác biệt căn bản giữa các đối tượng từ xa và các đối tượng cục bộ là các đối tượng từ xa nằm trên một máy khác. Thông thường, các tham số đối tượng được truyền cho các phương thức và các giá trị đối tượng được trả về từ các phương thức thông qua cách truyền theo tham chiếu. Tuy nhiên cách này không làm việc khi các phương thức gọi và các phương thức được gọi không cùng nằm trên một máy.



Hình 9.4: Minh họa giao tiếp giữa 2 ứng dụng của RMI.

Vì vậy, có ba cơ chế khác nhau được sử dụng để truyền các tham số cho các phương thức từ xa và nhận các kết quả trả về từ các phương thức ở xa. Các kiểu nguyên tố (int, boolean, double,...) được truyền theo tham trị. Các tham chiếu tới các đối tượng từ xa được truyền dưới dạng các tham chiếu cho phép tất cả phía nhận gọi các phương thức trên các đối tượng từ xa. Các đối tượng không thực thi Giao diện từ xa (nghĩa là các đối tượng không thực thi Giao diện Remote) được truyền theo tham trị; nghĩa là các bản sao đầy đủ được truyền đi bằng cách sử dụng cơ chế tuần tự hóa đối tượng. Các đối tượng không có khả năng tuần tự hóa thì không thể được truyền đi tới các phương thức ở xa. Các đối tượng ở xa chạy trên server nhưng có thể được gọi bởi các đối tượng đang chạy trên client. Các đối tượng không phải ở xa, các đối tượng khả tuần tự chạy trên các hệ thống client.



Hình 9.5: Minh họa kiến trúc xử lý của công nghệ RMI

Để quá trình truyền tin là trong suốt với người lập trình, truyền tin giữa client và server được cài đặt theo mô hình phân tầng như hình vẽ dưới đây

Đối với người lập trình, client dường như truyền tin trực tiếp với server. Thực tế, chương trình client chỉ truyền tin với đối tượng stub là đối tượng ủy quyền của đối tượng thực sự nằm trên hệ thống từ xa. Stub chuyển cuộc đàm thoại cho tầng tham chiếu, tầng này truyền tin trực tiếp với tầng giao vận. Tầng giao vận trên client truyền dữ liệu đi trên mạng máy tính tới tầng giao vận bên phía server. Tầng giao vận bên phía server truyền tin với tầng tham chiếu, tầng này truyền tin một phần của phần mềm server được gọi là skeleton. Skeleton truyền tin với chính server. Theo hướng khác từ server đến client thì luồng truyền tin được đi theo chiều ngược lại.

Cách tiếp cận có vẻ phức tạp nhưng ta không cần quan tâm đến vấn đề này. Tất cả đều được che dấu đi, người lập trình chỉ quan tâm đến việc lập các chương trình có khả năng gọi phương thức từ xa giống như đối với chương trình cục bộ.

Trước khi có thể gọi một phương thức trên một đối tượng ở xa, ta cần một tham chiếu tới đối tượng đó. Để nhận được tham chiếu này, ta yêu cầu một trình đăng ký tên rmiregistry cung cấp tên của tham chiếu. Trình đăng ký đóng vai trò như là một DNS nhỏ cho các đối tượng từ xa. Một client kết nối với trình đăng ký và cung cấp cho

nó một URL của đối tượng từ xa. Trình đăng ký cung cấp một tham chiếu tới đối tượng đó và client sử dụng tham chiếu này để gọi các phương thức trên server.

Trong thực tế, client chỉ gọi các phương thức cục bộ trên trong stub. Stub là một đối tượng cục bộ thực thi các Giao diện từ xa của các đối tượng từ xa.

Tầng tham chiếu từ xa thực thi giao thức tầng tham chiếu từ xa cụ thể. Tầng này độc lập với các đối tượng stub và skeleton cụ thể. Tầng tham chiếu từ xa có nhiệm vụ hiểu tầng tham chiếu từ xa có ý nghĩa như thế nào. Đôi khi tầng tham chiếu từ xa có thể tham chiếu tới nhiều máy ảo trên nhiều host.

Tầng giao vận gửi các lời gọi trên Internet. Phía server, tầng giao vận lắng nghe các liên kết đến. Trên cơ sở nhận lời gọi phương thức, tầng giao vận chuyển lời gọi cho tầng tham chiếu trên server. Tầng tham chiếu chuyển đổi các tham chiếu được gửi bởi client thành các tham chiếu cho các máy ảo cục bộ. Sau đó nó chuyển yêu cầu cho skeleton. Skeleton đọc tham số và truyền dữ liệu cho chương trình server, chương trình server sẽ thực hiện lời gọi phương thức thực sự. Nếu lời gọi phương thức trả về giá trị, giá trị được gửi xuống cho skeleton, tầng tham chiếu ở xa, và tầng giao vận trên phía server, thông qua Internet và sau đó chuyển lên cho tầng giao vận, tầng tham chiếu ở xa, stub trên phía client.

9.2 LẬP TRÌNH RMI

9.2.1 Gói java.rmi

Khi viết một applet hay một ứng dụng sử dụng các đối tượng ở xa, người lập trình cần nhận thức rằng các Giao diện và các lớp cần dùng cho phía client nằm ở trong gói java.rmi.

9.2.2 Giao Diện Remote

Giao diện này không khai báo bất kỳ phương thức nào. Các phương thức được khai báo trong phương thức này là các Giao diện có thể được gọi từ xa.

9.2.3 Lớp Naming

Lớp `java.rmi.Naming` truyền tin trực tiếp với một trình đăng ký đang chạy trên server để ánh xạ các URL `rmi://hostname/myObject` thành các đối tượng từ xa cụ thể trên host xác định. Ta có thể xem trình đăng ký như là một DNS cho các đối tượng ở xa. Mỗi điểm vào trong trình đăng ký bao gồm một tên và một tham chiếu đối tượng. Các client cung cấp tên và nhận về một tham chiếu tới URL.

URL `rmi` giống như URL `http` ngoại trừ phần giao thức được thay thế bằng `rmi`. Phần đường dẫn của URL là tên gắn với đối tượng từ xa trên server chứ không phải là tên một tệp tin.

Lớp `Naming` cung cấp các phương thức sau:

```
Public static String[] list(String url) throws RemoteException
```

Phương thức này trả về một mảng các chuỗi ký tự, mỗi chuỗi là một URL đã được gắn với một tham chiếu. Tham số `url` là URL của trình đăng ký `Naming`.

```
Public static Remote lookup(String url) throws RemoteException, NotBoundException,
AccessErrorException, MalformedURLException
```

Client sử dụng phương thức này để tìm kiếm một đối tượng từ xa gắn liền với tên đối tượng.

Phương thức này đưa ra ngoại lệ `NotBoundException` nếu server ở xa không nhận ra tên của nó. Nó đưa ra ngoại lệ `RemoteException` nếu trình không thể liên lạc được với trình đăng ký. Nó đưa ra ngoại lệ `AccessErrorException` nếu server từ chối tra tìm tên cho host cụ thể. Cuối cùng nếu URL không đúng cú pháp nó sẽ đưa ra ngoại lệ `MalformedURLException`.

```
Public static void bind(String url, Remote object) throws RemoteException,
AlreadyBoundException, MalformedURLException, AccessException
```

Server sử dụng phương thức `bind()` để liên kết một tên với một đối tượng ở xa. Nếu việc gán là thành công thì client có thể tìm kiếm đối tượng stub từ trình đăng ký.

Có rất nhiều tình huống có thể xảy ra trong quá trình gán tên. Phương thức này đưa ra ngoại lệ `MalformedURLException` nếu `url` không đúng cú pháp. Nó đưa ra ngoại lệ `RemoteException` nếu không thể liên lạc được với trình đăng ký. Nó đưa ra ngoại lệ

`AccessException` nếu client không được phép gán các đối tượng trong trình đăng ký. Nếu đối tượng URL đã gán với một đối tượng cục bộ nó sẽ đưa ra ngoại lệ `AlreadyBoundException`.

```
Public static void rebind(String url, Remote obj) throws RemoteException,  
AccessException, MalformedURLException
```

Phương thức này giống như phương thức `bind()` ngoại trừ việc là nó gán URL cho đối tượng ngay cả khi URL đã được gán.

9.2.4 Gói `java.rmi.registry`

Làm thế nào để nhận được một tham chiếu tới đối tượng? Client tìm ra các đối tượng ở xa hiện có bằng cách thực hiện truy vấn với trình đăng ký của server. Trình đăng ký cho ta biết những đối tượng nào đang khả dụng bằng cách truy vấn trình đăng ký của server. Ta đã biết lớp `java.rmi.Naming` cho phép chương trình Giao diện với trình đăng ký.

Giao diện `Registry` và lớp `LocateRegistry` cho phép các client tìm kiếm các đối tượng ở xa trên một server theo tên. `RegistryImpl` là lớp con của lớp `RemoteObject`, lớp này liên kết các tên với các đối tượng `RemoteObject`. Client sử dụng lớp `LocateRegistry` để tìm kiếm `RegistryImpl` cho một host và cổng cụ thể.

Giao diện `Registry`

Giao diện này cung cấp năm phương thức:

`Bind()` để gán một tên với một đối tượng từ xa cụ thể

`List()` liệt kê tất cả các tên đã được đăng ký với trình đăng ký

`Lookup()` tìm một đối tượng từ xa cụ thể với một URL cho trước gán với nó

`Rebind()` gán một tên với một đối tượng ở xa khác

`Unbind()` loại bỏ một tên đã được gán cho một đối tượng ở xa trong trình đăng ký

`Registry.REGISTRY_PORT` là cổng mặc định để lắng nghe các yêu cầu. Giá trị mặc định là 1099.

Lớp `LocateRegistry`

Lớp `java.rmi.registry.LocateRegistry` cho phép client tìm trong trình đăng ký trước tiên.

```
Public static Registry getRegistry() throws RemoteException
Public static Registry getRegistry(int port) throws RemoteException
Public static Registry getRegistry(String host) throws RemoteException
Public static Registry getRegistry(String host, int port) throws RemoteException
Public static Registry getRegistry(String host, int port, RMIClientSocketFactory
factory) throws RemoteException
```

Mỗi phương thức trên trả về một đối tượng `Registry` được sử dụng để nhận các đối tượng từ xa thông qua tên. Ví dụ để client có tìm thấy đối tượng ta có đăng ký đối tượng đó với trình đăng ký thông qua lớp `Registry`:

```
Registry r=LocateRegistry.getRegistry();
r.bind("MyName",this);
```

Client muốn gọi phương thức trên đối tượng từ xa có thể dùng các lệnh sau:

```
Registry r=LocateRegistry.getRegistry(www.somehose.com);
RemoteObjectInterface obj=(RemoteObjectInterface)r.lookup("MyName");
Obj.invokeRemoteMethod();
```

Ví dụ dưới đây minh họa cách tạo ra một trình đăng ký ngay trong server

```
import java.io.*;
import java.rmi.*;
import java.net.*;
import java.rmi.registry.*;
public class FileServer
{
    public static void main(String[] args) throws Exception
    {
        FileInterface fi=new FileImpl("FileServer");
        InetAddress dc=InetAddress.getLocalHost();
        LocateRegistry.createRegistry(1099);
        Naming.bind("rmi://" + dc.getHostAddress() + "/FileServer", fi);
        System.out.println("Server ready for client requests...");
    }
}
```

Như vậy khi thực thi chương trình ta không cần phải khởi động trình đăng ký vì việc tạo ra trình đăng ký và khởi động nó đã được tiến hành ở ngay trong chương trình phía server.

9.2.5 Gói `java.rmi.server`

Lớp `RemoteObject`: Về mặt kỹ thuật đối tượng từ xa không phải là một thể hiện của lớp `RemoteObject`. Thực tế, phần lớn các đối tượng từ xa là thể hiện của các lớp con của lớp `RemoteObject`.

Lớp `RemoteServer`: Lớp này là lớp con của lớp `RemoteObject`; nó là lớp cha của lớp `UnicastRemoteObject`. Lớp này có các constructor này:

```
Protected RemoteServer()  
Protected RemoteServer(RemoteRef r)
```

Lớp `RemoteServer` có một phương thức để xác định thông tin về client mà ta đang truyền tin với nó:

```
public static String getClientHost() throws ServerNotActiveException
```

Phương thức này trả về hostname của client mà gọi phương thức từ xa.

Lớp `UnicastRemoteObject`: Lớp `UnicastRemoteObject` là một lớp con cụ thể của lớp `RemoteServer`. Để tạo ra một đối tượng ở xa, ta phải thừa kế lớp `UnicastRemoteServer` và khai báo lớp này thực thi Giao diện `Remote`.

9.3 CÀI ĐẶT CHƯƠNG TRÌNH RMI

Để lập một hệ thống client/server bằng RMI ta sử dụng ba gói cơ bản sau: `java.rmi`, `java.rmi.server`, `java.rmi.registry`. Gói `java.rmi` bao gồm các Giao diện, các lớp và các ngoại lệ được sử dụng để lập trình cho phía client. Gói `java.rmi.server` cung cấp các Giao diện, các lớp và các ngoại lệ được sử dụng để lập trình cho phía server. Gói `java.rmi.registry` có các Giao diện, các lớp và các ngoại lệ được sử dụng để định vị và đặt tên các đối tượng.

Cài đặt chương trình phía Server: Để minh họa cho kỹ thuật lập trình RMI ở đây tác giả xin giới thiệu cách lập một chương trình `FileServer` đơn giản cho phép client tải về một tệp tin.

Bước 1: Đặc tả Giao diện `Remote`

```
import java.rmi.*;  
public interface FileInterface extends Remote
```

```
{  
    public byte[] downloadFile(String fileName)throws RemoteException;  
}
```

Bước 2: Viết lớp thực thi Giao diện

```
import java.rmi.*;  
import java.rmi.server.*;  
import java.io.*;  
public class FileImpl extends UnicastRemoteObject implements FileInterface  
{  
    private String name;  
    public FileImpl(String s)throws RemoteException{  
        super();  
        name=s;  
    }  
    public byte[] downloadFile(String fileName)throws RemoteException{  
        try{  
            File file=new File(fileName);  
            //Tạo một mảng b để lưu nội dung của tệp  
            byte b[]=new byte[(int)file.length()];  
            BufferedInputStream bis=new BufferedInputStream(new  
                FileInputStream(fileName));  
            bis.read(b,0,b.length);  
            bis.close();  
            return b;  
        }catch(Exception e){  
            System.err.println(e);  
            return null;  
        }  
    }  
}
```

Bước 3: Viết chương trình phía server

```
import java.io.*;  
import java.rmi.*;  
import java.net.*;  
public class FileServer {  
    public static void main(String[] args) throws Exception{  
        FileInterface fi=new FileImpl("FileServer");  
        InetAddress dc=InetAddress.getLocalHost();  
        Naming.rebind("rmi://" +dc.getHostAddress()+"/FileServer",fi);  
        System.out.println("Server ready for client requests...");  
    }  
}
```

Bước 4: Cài đặt client

```
import java.rmi.*;  
import java.io.*;
```

```

public class FileClient {
public static void main(String[] args) throws Exception{
    if(args.length!=2){
        System.out.println("Su dung:java FileClient fileName machineName ");
        System.exit(0);
    }
    String name="rmi://" +args[1]+"/FileServer";
    FileInterface fi=(FileInterface)Naming.lookup(name);
    byte[] filedata=fi.downloadFile(args[0]);
    File file =new File(args[0]);
    BufferedOutputStream bos=new BufferedOutputStream(new
    FileOutputStream(file.getName()));
    bos.write(filedata,0,filedata.length);
    bos.flush();
    bos.close();
}}

```

Triển khai ứng dụng: Để triển khai ứng dụng RMI ta cần thực hiện các bước sau:

Bước 1: Biên dịch các tệp chương trình

```

C:\MyJava>javac FileInterface.java
C:\MyJava>javac FileImpl.java
C:\MyJava>javac FileServer.java
C:\MyJava>javac FileClient.java

```

Ta sẽ thu được các lớp sau: FileInterface.class, FileImpl.class, FileServer.class, FileClient.class. Để tạo ra các lớp trung gian ta dùng lệnh sau:

```

C:\MyJava>rmic FileImpl

```

Sau khi biên dịch ta sẽ thu được hai lớp trung gian là FileImpl_Stub.class và FileImpl_Skel.class.

Bước 2: Tổ chức chương trình trên hai máy client và server như sau:

Phía Server	Phía Client
FileInterface.class	FileInterface.class
FileImpl.class	FileImpl_Stub.class
FileImpl_Skel.class	FileClient.class

FileServer.class	
------------------	--

Bước 3: Khởi động chương trình: Ở đây ta giả lập chương trình trên cùng một máy. Việc triển khai trên mạng không có gì khó khăn ngoài việc cung cấp hostname hoặc địa chỉ IP của server cung cấp dịch vụ

Khởi động trình đăng ký:

```
C:\MyJava>start rmiregistry
```

Khởi động server

```
C:\MyJava>start java FileServer
```

Khởi động client

```
C:\MyJava>java FileClient D:\RapidGet.exe localhost
```

CÂU HỎI ÔN TẬP

1. RMI là gì? Tại sao cần triệu gọi phương thức từ xa? Ưu/nhược điểm?
2. Trình bày mô hình kiến trúc hoạt động của việc triệu gọi phương thức từ xa?
3. Viết chương trình Server quản lý thông tin tài khoản người sử dụng ATM. Chương trình có đối tượng quản lý người sử dụng cho phép triệu gọi từ xa bởi các máy ATM. Thông tin người sử dụng gồm có: Tên, CMND, mật khẩu, số tiền, lãi xuất, thông tin các giao dịch trên tài khoản. Thông tin giao dịch gồm có: ngày giao dịch, số tiền, kiểu giao dịch (rút tiền hoặc gửi tiền). Chương trình quản lý đối tượng người sử dụng có các chức năng sau: Thêm một người sử dụng mới vào danh sách để quản lý. Tìm người sử dụng có tên và mật khẩu. Cho phép nhập/rút tiền trên tài khoản có tên và mật khẩu (hoặc CMND).

TÀI LIỆU THAM KHẢO

1. Elliotte Rusty Harold, Java Network Programming
2. Nguyễn Phương Lan- Hoàng Đức Hải, Java lập trình mạng, Nhà xuất bản Giáo dục
3. Darrel Ince & Adam Freemat, Programming the Internet with Java, Addison-Wesley
4. Mary Campione&Kathy Walrath&Alison Huml, Java™ Tutorial, Third Edition: A Short Course on the Basics, Addison Wesley The Complete Java.
5. Nguyễn Thúc Hải, Mạng máy tính và các hệ thống mở, Nhà xuất bản Giáo dục
6. Đoàn Văn Ban, Lập trình hướng đối tượng với Java, Nhà xuất bản Khoa học và Kỹ thuật
7. Douglas E.Comer, David L.Stevens, Client-Server Programming And Applications. In book: Internetworking with TCP/IPVolume III, Pearson Education, Singapore, 2004.
8. Herbert Schildt, Java™ 2: The Complete Reference Fifth Edition, Tata McGraw-Hill Publishing Company Limited, India, 2002.
9. Eliotte Rusty Harold, Java™ Network Programming, Third Edition, Oreilly, 2005.
10. Qusay H. Mahmoud, Advanced Socket Programming, <http://java.sun.com>, December 2001
11. Shengxi Zhou, Transport Java objects over the network with datagram packets, <http://www.javaworld.com>, 2006.