LẬP TRÌNH MẠNG

TÀI LIỆU

DÀNH CHO SINH VIÊN CÔNG NGHỆ THÔNG TIN

NGUYỄN MẠNH HÙNG - NGUYỄN TRỌNG KHÁNH

MỤC LỤC

MỤC LỤC ..... **i**
GIỚI THIỆU **1**
PHẦN I. **2**
CƠ SỞ CỦA LẬP TRÌNH MẠNG **2**
CHƯƠNG 1. GIỚI THIỆU VỀ LẬP TRÌNH MẠNG **3**
1.1. GIỚI THIỆU VỀ LẬP TRÌNH MẠNG 3
1.2. MỘT SỐ KIẾN THỨC MẠNG CƠ SỞ LẬP TRÌNH MẠNG 4
1.2.1. Mô hình OSI/ISO và họ giao thức TCP/IP 4
1.2.2. Giao thức truyền thông và phân loại (protocol) 4
1.2.3. Địa chỉ IP, mặt nạ (mask) 5
1.2.4. Địa chỉ cổng(port) 6
1.2.5. Giao diện socket, địa chỉ socket 8
1.3. CÁC MÔ HÌNH LẬP TRÌNH MẠNG 9
1.3.1. Mô hình client/server 9
1.3.2. Mô hình peer-to-peer 9
1.3.3. Mô hình đa tầng 9
1.4 . NGÔN NGỮ LẬP TRÌNH MẠNG 10
1.4.1. Giới thiệu chung 10
1.4.2. Lập trình mạng bằng ngôn ngữ Java 10
1.5. KỸ THUẬT LẬP TRÌNH MẠNG 10
1.6. KẾT LUẬN 11
CHƯƠNG 2. CƠ SỞ JAVA CHO LẬP TRÌNH MẠNG **12**
2.1. LẬP TRÌNH VÀO RA VỚI JAVA 12
2.1.1. Các lớp Java hỗ trợ nhập dữ liệu vào 12
2.1.2. Các kiểu nhập dữ liệu vào 14
2.1.3. Các lớp Java hỗ trợ xuất dữ liệu ra 15
2.1.4. Các kiểu xuất dữ liệu ra 18
2.2. LẬP TRÌNH THREAD VỚI JAVA 19
2.2.1 Giới thiệu về Thread trong Java 19
2.2.2. Lập trình với Thread trong Java 20
2.2.3. Ví dụ 20
2.3. LẬP TRÌNH THEO MÔ HÌNH MVC 22
2.3.1. Giới thiệu mô hình MVC 22
2.3.2 Ví dụ 23
2.4. KẾT LUẬN 28
2.5. BÀI TẬP 28
CHƯƠNG 3. LẬP TRÌNH VỚI CƠ SỞ DỮ LIỆU **30**
3.1. THIẾT KẾ CSDL CHO ỨNG DỤNG 30
3.1. Các bước để thiết kế CSDL cho ứng dụng 30

3.2. Ví dụ 30
3.2. SỬ DỤNG LỆNH SQL 33
3.2.1. Lệnh INSERT 33
3.2.2. Lệnh UPDATE 33
3.2.3. Lệnh DELETE 34
3.2.3. Lệnh SELECT 35
3.3. LẬP TRÌNH JAVA VỚI CSDL 38
3.3.1. Tạo kết nối vào CSDL bằng JDBC 38
3.3.2. Sử dụng lệnh SQL trong Java 39
3.3.3. Lấy dữ liệu ra từ SQL 40
3.3.4. Ví dụ 41
3.4. KẾT LUẬN 42
3.5. BÀI TẬP 42
PHẦN II. 45
LẬP TRÌNH VỚI SOCKET 45
CHƯƠNG 4. LẬP TRÌNH VỚI GIAO THỨC TCP/IP 46
4.1. GIỚI THIỆU GIAO THỨC TCP/IP 46
4.2. MỘT SỐ LỚP JAVA HỖ TRỢ GIAO THỨC TCP/IP 47
4.2.1. Lớp InetAddress 47
4.2.2. Lớp Socket 49
4.2.3. Lớp ServerSocket 51
4.3. LẬP TRÌNH ỨNG DỤNG MẠNG VỚI TCPSOCKET 52
4.3.1. Chương trình phía server 53
4.3.2. Chương trình client 53
4.3.3. Luồng nhập/xuất mạng và đọc/ghi dữ liệu qua luồng nhập/xuất 54
4.3.4. Một số ví dụ 55
4.4. CASE STUDY: LOGIN TỪ XA DÙNG GIAO THỨC TCP/IP 59
4.4.1 Bài toán 59
4.4.2 Kiến trúc hệ thống theo mô hình MVC 59
4.4.3 Cài đặt 62
4.4.5 Kết quả 67
4.5. KẾT LUẬN 67
4.6. BÀI TẬP 68
CHƯƠNG 5. LẬP TRÌNH VỚI GIAO THỨC UDP 69
5.1. GIỚI THIỆU GIAO THỨC UDP 69
5.2. MỘT SỐ LỚP JAVA HỖ TRỢ LẬP TRÌNH VỚI UDP 70
5.2.1. Lớp DatagramPacket 70
5.2.2. Lớp DatagramSocket 71
5.3. KỸ THUẬT LẬP TRÌNH TRUYỀN THÔNG VỚI GIAO THỨC UDP 72
5.3.1. Phía server 73
5.3.2. Phía client 73
5.3.3. Một số chương trình ví dụ 74

5.4. CASE STUDY: LOGIN TỪ XA DÙNG UDP 75
5.4.1 Bài toán 75
5.4.2 Kiến trúc hệ thống theo mô hình MVC 76
5.4.3 Cài đặt 78
5.4.5 Kết quả 85
5.5. KẾT LUẬN 85
5.6. BÀI TẬP 85
CHƯƠNG 6. LẬP TRÌNH CHUYÊN SÂU VỚI SOCKET 87
6.1. LẬP TRÌNH SOCKET VÀO RA KHÔNG CHẶN DÙNG 87
6.1.1. Gói Java NIO 87
6.1.2. Bộ đệm 89
6.1.2. Kênh 92
6.1.3. Một số ví dụ 94
6.1.5. Kết luận 101
6.2. LẬP TRÌNH KẾT NỐI NHIỀU GIAO TIẾP MẠNG 101
6.2.1. Giới thiệu Giao tiếp mạng 102
6.2.2. Lấy thông tin Giao tiếp mạng 105
6.2.3. Liệt kê Giao tiếp mạng 106
6.2.4. Các tham số Giao tiếp mạng 107
6.2.5. Kết hợp với Socket 108
6.3. LẬP TRÌNH MẠNG NGANG HÀNG P2P 108
6.3.1. Giới thiệu mạng ngang hàng P2P 109
6.3.2. Các loại ứng dụng P2P 111
6.3.3. Xây dựng ứng dụng trên mạng ngang hàng 111
6.3.4. Một số nền tảng hỗ trợ xây dựng ứng dụng P2P 114
6.3.5. Kết luận 122
6.4. LẬP TRÌNH TRUYỀN THÔNG MULTICAST 122
6.4.1. Giới thiệu truyền thông multicast và lớp MulticastSocket 122
6.4.2. Một số ví dụ gửi/nhận dữ liệu multicast 124
6.5. LẬP TRÌNH BẢO MẬT VỚI SSL 125
6.5.1. Giới thiệu SSL 125
6.5.2. Lập trình với SSL 130
6.6. LẬP TRÌNH ỨNG DỤNG MẠNG KHÔNG ĐỒNG NHẤT 133
6.6.1. Thứ tư byte trong Java 134
6.6.2. Tương tác với các nền tảng khác nhau 136
6.6.3. Kết luận 139
CHƯƠNG 7. LẬP TRÌNH VỚI CÁC GIAO THỨC TẦNG ỨNG DỤNG 140
7.1. LẬP TRÌNH GIAO THỨC DỊCH VỤ TELNET 140
7.1.1. Giới thiệu về Telnet 140
7.1.2. Cài đặt dịch vụ Telnet Client với Java 142
7.2. LẬP TRÌNH DỊCH VỤ TRUYỀN TỆP VỚI GIAO THỨC FTP 146
7.2.1. Dịch vụ truyền tệp FTP 146

7.2.2. Kỹ thuật cài đặt giao thức FTP với java 148
7.3. LẬP TRÌNH GỬI/NHẬN THƯ VỚI GIAO THỨC SMTP và POP3 150
7.3.1. Giao thức SMTP 150
7.3.2. Giao thức POP3 155
7.6. KẾT LUẬN 157
PHẦN III. 158
LẬP TRÌNH PHÂN TÁN 158
CHƯƠNG 8. LẬP TRÌNH PHÂN TÁN VỚI RMI 159
8.1. GIỚI THIỆU LẬP TRÌNH PHÂN TÁN VÀ RMI 159
8.1.1. Giới thiệu kỹ thuật lập trình phân tán 159
8.1.2. Giới thiệu kỹ thuật lập trình RMI 159
8.1.3. Các lớp hỗ trợ lập trình với RMI 163
8.2. XÂY DỰNG CHƯƠNG TRÌNH PHÂN TÁN RMI 163
8.2.1. Kỹ thuật lập trình RMI 163
8.2.2. Biên dịch chương trình 166
8.2.3. Thực thi chương trình ứng dụng 166
8.3. CASE STUDY 1: LOGIN TỪ XA DÙNG RMI 167
8.3.1. Bài toán 167
8.3.2. Thiết kế hệ thống 167
8.3.3. Cài đặt 170
8.3.4. Kết quả 175
8.4. KẾT LUẬN 176
8.5. BÀI TẬP 176
PHẦN IV. 178
LẬP TRÌNH ỨNG DỤNG WEB 178
CHƯƠNG 9. LẬP TRÌNH WEBSOCKET 179
9.1. GIỚI THIỆU WEBSOCKET 179
9.2. TẠO ỨNG DỤNG WEBSOCKET 180
9.2.1. Điểm cuối lập trình 181
9.2.2. Điểm cuối chú thích 181
9.2.3. Gửi và nhận thông điệp 182
9.2.4. Duy trì trạng thái máy khách 184
9.2.5. Sử dụng bộ mã hóa và bộ giải mã 185
9.2.6. Tham số đường dẫn 188
9.2.7. Xử lý lỗi 189
9.2.8. Đặc tả lớp cấu hình điểm cuối 189
9.3. ỨNG DỤNG WEBSOCKETBOT 190
9.3.1. Kiến trúc 190
CHƯƠNG 10. LẬP TRÌNH WEB VỚI JSP VÀ TOMCAT SERVER 197
10.1. GIỚI THIỆU JSP 197
10.1.1. Giới thiệu JSP 197
10.1.2. Kiến trúc một ứng dụng web với JSP-Tomcat 199

10.2. TRAO ĐỔI DỮ LIỆU GIỮA CÁC TRANG JSP 200
10.2.1. Dùng Parameter 200
10.2.2. Dùng bean 202
10.2.3. Dùng Response 204
10.2.4. Dùng Session 204
10.3. LẬP TRÌNH WEB VỚI JSP – TOMCAT 205
10.3.1. Cài đặt và deploy các lớp Java 206
10.3.2. Cấu hình ứng dụng web trên server Tomcat 208
10.3.3. Cài đặt các trang JSP 208
10.3.4. Test ứng dụng web 210
10.4. KẾT LUẬN 211
10.5. BÀI TẬP 211
CHƯƠNG 11. LẬP TRÌNH WEB VỚI SPRING FRAMEWORK 212
11.1. GIỚI THIỆU VỀ SPRING 212
11.2. LẬP TRÌNH ỨNG DỤNG WEB VỚI SPRING 214
11.3. CASE STUDY: ỨNG DỤNG WEB VỚI CHỨC NĂNG ĐĂNG KÍ VÀ ĐĂNG NHẬP 217
11.3.1. Tạo project 217
11.3.2. Chức năng đăng nhập 217
11.3.3. Chức năng đăng kí 224
11.4. KẾT LUẬN 226
11.5. BÀI TẬP 226
BÀI TẬP DỰ ÁN 228

GIỚI THIỆU

Mục tiêu của môn Lập trình mạng là cung cấp cho sinh viên những kiến thức và kỹ năng lập trình với bốn nền tảng: lập trình với socket, lập trình phân tán, lập trình hướng dịch vụ, và lập trình web. Nhằm cung cấp thêm cho sinh viên một kênh tham khảo cho môn học, tài liệu này cung cấp các kiến thức cơ bản và hướng dẫn các kỹ năng cần thiết cho 4 nền tảng lập trình này.

Tài liệu này có thể dùng cho sinh viên các ngành công nghệ thông tin và an toàn thông tin. Ngoài ra các bạn sinh viên đam mê lập trình cũng có thể sử dụng tham khảo. Để sử dụng được tốt nhất tài liệu này, yêu cầu người đọc phải có kiến thức cơ bản về các kỹ thuật lập trình, ngôn ngữ lập trình Java, và các khái niệm của lập trình hướng đối tượng.

Nội dung của tài liệu này cũng được chia làm 4 phần tương ứng với 4 nền tảng:

Phần 1: Nhắc lại kiến thức cơ bản về lập trình với Java.

Bao gồm lập trình vào ra dữ liệu với Java, lập trình với Thread và multi-thread, lập trình với mô hình MVC, lập trình với CSDL.

Phần 2: Lập trình socket

Bao gồm lập trình với các giao thức TCP/IP, UDP, FTP, SMTP, POP,...

Phần 3: Lập trình phân tán

Bao gồm lập trình phân tán với RMI, lập trình hướng dịch vụ với web service.

Phần 4: Lập trình web

Bao gồm các kỹ thuật lập trình ứng dụng web với các nền tảng: JSP-Tomcat, Struts 1, Struts 2, Spring framework.

Phần cuối là danh sách một số đề tài dự án cho sinh viên thao khảo coi như là bài tập lớn, có thể làm theo nhóm và triển khai trên hệ thống mạng.

TÀI LIỆU THAM KHẢO

[1] Java Network Programming, Elliotte Rusty Harold, Fourth Edition, by O'Reilly Media, 2014.
[2] Blog trực tuyến: http://coderandcode.blogspot.com/

Tài liệu tham khảo cho môn Lập trình mạng, phiên bản thứ nhất, tháng 9 năm 2016.

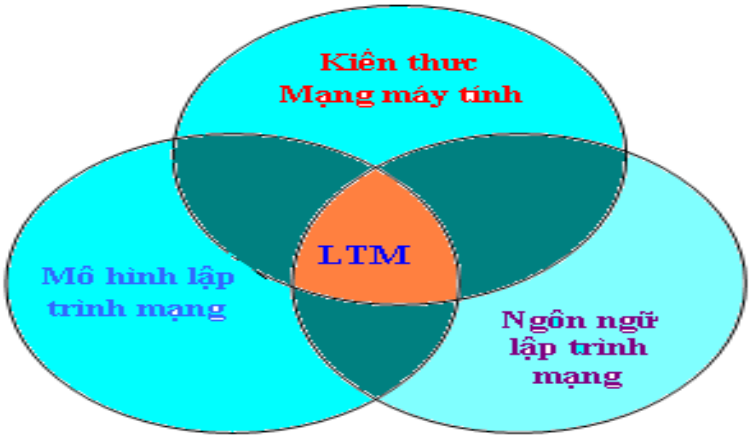
PHẦN I.
CƠ SỞ CỦA LẬP TRÌNH MẠNG

CHƯƠNG 1. GIỚI THIỆU VỀ LẬP TRÌNH MẠNG

1.1. GIỚI THIỆU VỀ LẬP TRÌNH MẠNG

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

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



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

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

LTM=KTM+MH+NN

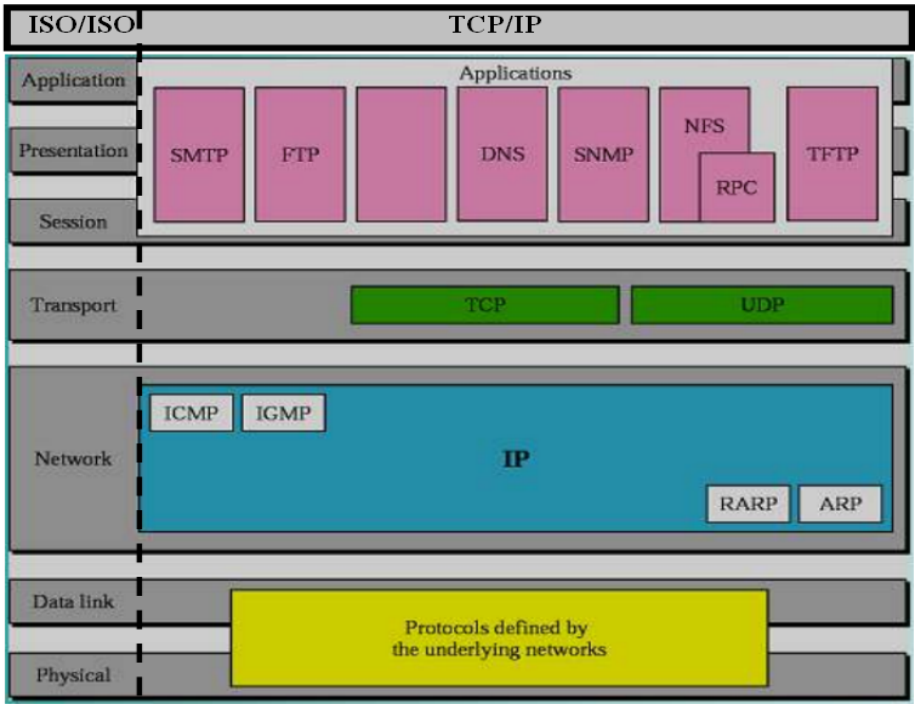
Trong đó:

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

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

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

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



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

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

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

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

a. Giao thức hoạt động theo hướng có kết nối

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

- Truyền thông theo kiểu điểm-điểm
- Dữ liệu truyền qua mạng là một dòng các byte liên tục truyền từ nơi gửi tới nơi nhận, mỗi byte có một chỉ số xác định.
- Quá trình truyền thông được thực hiện thông qua 3 giai đoạn:

- Thiết lập kết nối
- Truyền dữ liệu kèm theo cơ chế kiểm soát chặt chẽ
- Huỷ bỏ kết nối

- Giao thức tiêu biểu là giao thức TCP

b. Giao thức hoạt động hướng không kết nối

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

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

c. Một số giao thức truyền thông Internet phổ biến

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

1.2.3. Địa chỉ IP, mặt nạ (mask)

a. Địa chỉ IP

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

b. Mặt nạ(mask)

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

c. Một số địa chỉ IP đặc biệt

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

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

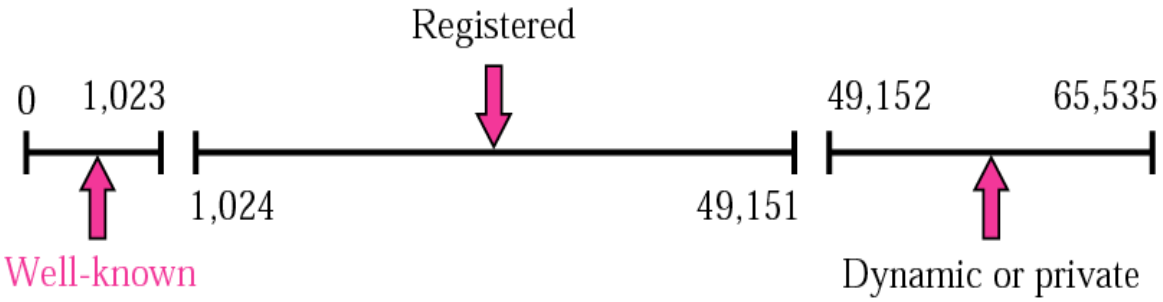
Địa chỉ *this host on this network*: có tất cả các bít netid và hostid đều bằng 0. Địa chỉ này là địa chỉ nguồn được máy trạm sử dụng tại thời điểm Bootstrap để truyền thông khi nó biết địa chỉ IP của nó.

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

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

1.2.4. Địa chỉ cổng(port)

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



Hình 1.3. Các dải địa chỉ cổng

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

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

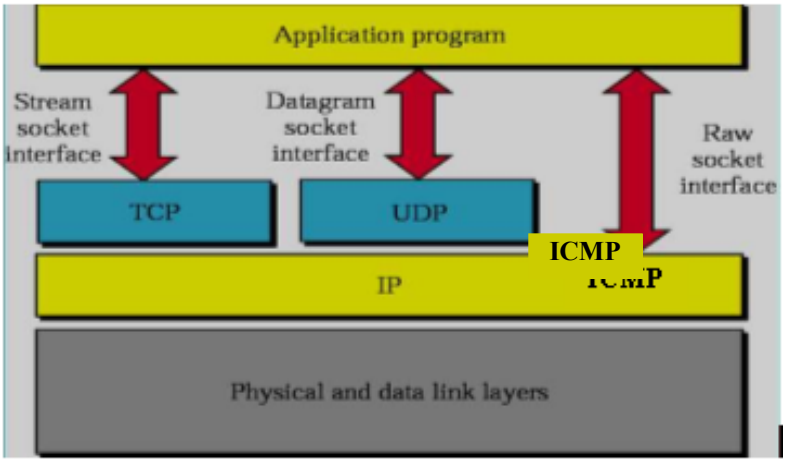
- Giải địa chỉ từ 1024 đến 49151: Giải địa chỉ cổng này người sử dụng được phép dùng, nhưng phải đăng ký để tránh trùng lặp.

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

1.2.5. Giao diện socket, địa chỉ socket

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

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



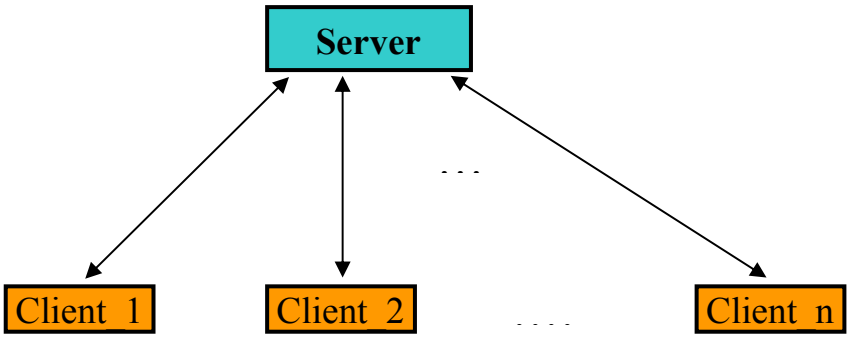
Hình 1.4. Các kiểu giao diện socket

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

1.3. CÁC MÔ HÌNH LẬP TRÌNH MẠNG

1.3.1. Mô hình client/server

Chương trình ứng dụng mạng tổ chức theo mô hình client/server được sử dụng phổ biến trong thực tế. Chương trình ứng dụng mạng theo mô hình này gồm có 2 phần mềm: Phần mềm server(phục vụ) và phần mềm client(máy khách) và nó thể hiện như hình 2. Một chương trình server có thể phục vụ nhiều chương trình client đồng thời hoặc tuần tự(kiểu lặp).



Hình 1.5. Mô hình client/server

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

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

1.3.2. Mô hình peer-to-peer

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

1.3.3. Mô hình đa tầng

Mô hình đa tầng gồm nhiều tầng mà tiêu biểu nhất là mô hình 3 tầng. Trong mô hình này, tầng thấp nhất là tầng thông tin, tầng trung gian và tầng đỉnh. Một ví dụ tiểu biểu của mô hình 3 tầng

đó là dịch vụ Web với tầng đỉnh là trình duyệt, tầng trung gian là webserver và tầng thông tin là cơ sở dữ liệu. Mô hình nhiều tầng sẽ được khảo sát kỹ trong phần lập trình ứng dụng mạng nâng cao với các kỹ thuật Servlet, EJB, Portlet..

1.4 . NGÔN NGỮ LẬP TRÌNH MẠNG

1.4.1. Giới thiệu chung

Nói chung tất cả các ngôn ngữ lập trình đều có thể sử dụng để lập trình mạng. Nhưng mỗi ngôn ngữ có những ưu, nhược điểm khác nhau và được hỗ trợ thư viện API ở các mức độ khác nhau. Tuỳ từng ứng dụng mạng cụ thể, hệ điều hành mạng cụ thể và thói quen lập trình mà người lập trình có thể chọn ngôn ngữ phù hợp để phát triển các ứng dụng mạng. Các ngôn ngữ lập trình phổ biến hiện nay gồm những ngôn ngữ sau: Hợp ngữ(Assembly Language), C/C$^{++,}$ VC^{++}, VB, Delphi, Java, .NET, ASP.

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

1.4.2. Lập trình mạng bằng ngôn ngữ Java

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

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

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

1.5. KỸ THUẬT LẬP TRÌNH MẠNG

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

- Kỹ thuật lập trình mạng với socket: Trong kỹ thuật này, chương trình ứng dụng mạng sẽ được xây dựng với các kiểu socket khác nhau. Kỹ thuật này cho phép mối quan hệ qua mạng giữa các chương trình chạy lỏng lẻo vì bản thân socket là giao diện mạng , không phải cơ chế truyền thông.
- Kỹ thuật lập trình phân tán: Trái với kỹ thuật lập trình socket, trong kỹ thuật này mối quan hệ giữa chương trình client và server là gắn kết chặt chẽ. Kỹ thuật lập trình này thực chất là kỹ thuật lập trình phân tán mã lệnh (đối tượng), cho phép phân tải tính toán lên các máy tính kết nối với nhau với quan hệ hữu cơ thay vì tập trung trên cùng một máy. Điều này cho phép tận dụng tài nguyên mạng để giải quyết các bài toán với khối lượng tính toán lớn, thời gian thực.
- Kỹ thuật lập trình ứng dụng web:

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

1.6. KẾT LUẬN

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

CHƯƠNG 2. CƠ SỞ JAVA CHO LẬP TRÌNH MẠNG

Nội dung chương này sẽ trình bày một số kiến thức cơ sở của Java được dùng nhiều trong lập trình mạng. Bao gồm:

- Lập trình vào ra với Java
- Lập trình với Thead trong Java
- Lập trình Java với mô hình MVC

2.1. LẬP TRÌNH VÀO RA VỚI JAVA

2.1.1. Các lớp Java hỗ trợ nhập dữ liệu vào

a. Scanner

Các hàm khởi tạo:

- Scanner(File): nhận tham số vào là một file (đọc dữ liệu từ file).
- Scanner(InputStream): nhận tham số là một dạng của InputStream để đọc dữ liệu vào từ luồng đó.
- Scanner(Path): nhận tham số là đường dẫn đến file, để đọc dữ liệu từ file.
- Scanner(Readable): nhận tham số từ một đối tượng dạng Readable để đọc dữ liệu từ đó.

Các phương thức hỗ trợ đọc dữ liệu vào:

- hasNext(): trả về true nếu vẫn còn dữ liệu đầu vào, false nếu ngược lại.
- next(): đọc dữ liệu vào dạng String.
- nextBoolean(): đọc dữ liệu vào dạng boolean.
- nextByte(): đọc dữ liệu vào dạng Byte.
- nextDouble(): đọc dữ liệu vào dạng double.
- nextFloat(): đọc dữ liệu vào dạng float
- nextInt(): đọc dữ liệu vào dạng int.
- nextLine(): đọc một dòng dữ liệu vào, trả về dạng String.
- nextLong(): đọc dữ liệu vào dạng long.
- nextShort(): đọc dữ liệu vào dạng short.

b. InputStream

InputStream là một lớp trừu tượng, ta không thể khởi tạo trực tiếp nó nhưng có thể sử dụng thông qua các lớp con kế thừa từ lớp này: các lớp trong tên có chữ "InputStream".

Các hàm khởi tạo:

- InputStream(): không tham số (trừu tượng)

Các phương thức hỗ trợ đọc dữ liệu vào:

- read(): trả về một byte đọc được (trừu tượng)
- read(byte[]): đọc một mảng các byte vào bộ đệm ở tham số vào. Trả về số lượng byte đọc được kiểu int.
- read(byte[] b, int off, int len): đọc tối đa *len* byte, ghi vào bộ đệm *b*, bắt đầu từ vị trí có chỉ số *off*. Trả về số lượng byte thực sự đọc được.

c. BufferedInputStream

Đây là một lớp con của lớp InputStream, cho nên nó kế thừa các phương thức của lớp InputStream. Ngoài ra còn có các phương thức riêng.

Các hàm khởi tạo:

- BufferedInputStream(InputStream): khởi tạo với luồng đọc vào.

Các phương thức hỗ trợ đọc dữ liệu vào:

- Tương tự các phương thức đọc của lớp InputStream.

d. DataInputStream

Đây là một lớp con của lớp InputStream, cho nên nó kế thừa các phương thức của lớp InputStream. Ngoài ra còn có các phương thức riêng.

Các hàm khởi tạo:

- DataInputStream(InputStream): khởi tạo với luồng đọc vào.

Các phương thức hỗ trợ đọc dữ liệu vào:

- readBoolean(): đọc dữ liệu vào dạng boolean.
- readByte(): đọc dữ liệu vào dạng Byte.
- readDouble(): đọc dữ liệu vào dạng double.
- readFloat(): đọc dữ liệu vào dạng float
- readInt(): đọc dữ liệu vào dạng int.
- readLine(): đọc một dòng dữ liệu vào, trả về dạng String.
- readLong(): đọc dữ liệu vào dạng long.
- readShort(): đọc dữ liệu vào dạng short.
- readUnsignedByte(): đọc dữ liệu vào dạng unsigned byte.
- readUnsignedShort(): đọc dữ liệu vào dạng unsigned short.
- readUTF(): đọc dữ liệu vào dạng String theo chuẩn unicode (UTF-8).

e. Reader

Reader là một lớp trừu tượng, ta không thể khởi tạo trực tiếp nó nhưng có thể sử dụng thông qua các lớp con kế thừa từ lớp này: các lớp trong tên có chữ "Reader".

Các hàm khởi tạo:

- Reader(): khởi tạo (trừu tượng).

Các phương thức hỗ trợ đọc dữ liệu vào:

- read(): đọc một kí tự, trả về mã character dạng int
- .read(char[] b): đọc dữ liệu vào mảng b. Trả về số lượng kí tự thực sự đọc được.
- read(char[] b, int off, int len): đọc tối đa *len* kí tự, ghi vào mảng *b*, bắt đầu từ vị trí *off*. Trả về số lượng kí tự thực sự đọc được.
- read(CharBuffer b): đọc các kí tự vào bộ đệm b. Trả về số lượng kí tự thực sự đọc được.

f. BufferedReader

Đây là một lớp con của lớp Reader, cho nên nó kế thừa các phương thức của lớp Reader. Ngoài ra còn có các phương thức riêng.

Các hàm khởi tạo:

- BuferedReader(Reader): Khởi tạo với một đối tượng đọc vào.

Các phương thức hỗ trợ đọc dữ liệu vào:

- readLine(): đọc một dòng văn bản, trả về dạng String.

g. InputStreamReader

Đây là một lớp con của lớp Reader, cho nên nó kế thừa các phương thức của lớp Reader. Ngoài ra còn có các phương thức riêng.

Các hàm khởi tạo:

- InputStreamReader(InputStream): Khởi tạo với một luồng đọc vào.

Các phương thức hỗ trợ đọc dữ liệu vào:

- Hoàn toàn tương tự lớp Reader.

2.1.2. Các kiểu nhập dữ liệu vào

a. Nhập dữ liệu vào từ bàn phím

Để nhập dữ liệu từ bàn phím, có thể dùng bất kì một trong các đối tượng đọc dữ liệu vào của Java. Chỉ cần truyền tham số khởi tạo là đối tượng System.in:

- Dùng Scanner: Scanner scn = new Scanner(System.in);
- Dùng BufferedInputStream: bis = new BufferedInputStream(System.in);
- Dùng DataInputStream: dis = new DataInputStream(System.in);

- Dùng BufferedReader: br = new BufferedReader(System.in);
- Dùng InputStreamReader: isr = new InputStreamReader(System.in);

b. Nhập dữ liệu vào từ file

Để nhập dữ liệu từ file, có thể dùng bất kì một trong các đối tượng đọc dữ liệu vào của Java. Chỉ cần truyền tham số khởi tạo là đối tượng mở file để đọc như là FileInputStream, FileReader:

- Dùng Scanner: Scanner scn = new Scanner(new FileInputStream("in.txt"));
- Dùng BufferedInputStream:

  bis = new BufferedInputStream(new FileInputStream("in.txt"));
- Dùng DataInputStream: dis = new DataInputStream(new FileInputStream("in.txt"));
- Dùng BufferedReader: br = new BufferedReader(new FileReader("in.txt"));
- Dùng InputStreamReader: isr = new InputStreamReader(new FileReader("in.txt"));

c. Nhập dữ liệu vào từ socket

Để nhập dữ liệu từ socket, có thể dùng bất kì một trong các đối tượng đọc dữ liệu vào của Java. Chỉ cần truyền tham số khởi tạo là đối tượng socket.getInputStream():

- Dùng Scanner: Scanner scn = new Scanner(socket.getInputStream());
- Dùng BufferedInputStream: bis = new BufferedInputStream(socket.getInputStream());
- Dùng DataInputStream: dis = new DataInputStream(socket.getInputStream());
- Dùng BufferedReader: br = new BufferedReader(socket.getInputStream());
- Dùng InputStreamReader: isr = new InputStreamReader(socket.getInputStream());

2.1.3. Các lớp Java hỗ trợ xuất dữ liệu ra

a. OutputStream

OutputStream là một lớp trừu tượng cho xuất dữ liệu ra theo luồng. Ta không thể khởi tạo trực tiếp đối tượng này nhưng có thể sử dụng các lớp con kế thừa từ lớp này.

Các hàm khởi tạo:

- OutputStream(): khởi tạo (trừu tượng)

Các phương thức hỗ trợ xuất dữ liệu ra:

- write(byte): ghi một byte ra luồng xuất (trừu tượng)
- write(byte[] b): ghi *b.length* byte từ mảng *b* ra luồng xuất.
- write(byte[] b, int off, int len): ghi *len* byte từ mảng *b*, bắt đầu từ vị trí *off* ra luồng xuất.

b. DataOutputStream

Đây là một lớp con của lớp OutputStream, cho nên nó được kế thừa các phương thức của lớp OutputStream. Ngoài ra nó còn có một số phương thức bổ sung.

Các hàm khởi tạo:

- DataOutputStream(OutputStream): khởi tạo để xuất dữ liệu theo luồng.

Các phương thức hỗ trợ xuất dữ liệu ra:

- writeBoolean(boolean): ghi dữ liệu ra dạng boolean.
- writeByte(byte): ghi dữ liệu ra dạng Byte.
- writeDouble(double): ghi dữ liệu ra dạng double.
- writeFloat(float): ghi dữ liệu ra dạng float
- writeInt(int): ghi dữ liệu ra dạng int.
- writeLine(String): ghi dữ liệu ra dạng String.
- writeLong(long): ghi dữ liệu ra dạng long.
- writeShort(short): ghi dữ liệu ra short.
- writeUnsignedByte(unsigned byte): ghi dữ liệu ra dạng unsigned byte.
- writeUnsignedShort(unsigned short): ghi dữ liệu ra dạng unsigned short.
- writeUTF(String): ghi dữ liệu ra dạng String theo chuẩn unicode (UTF-8).

c. BufferedOutputStream

Đây là một lớp con của lớp OutputStream, cho nên nó được kế thừa các phương thức của lớp OutputStream. Ngoài ra nó còn có một số phương thức bổ sung.

Các hàm khởi tạo:

- BufferedOutputStream(OutputStream): khởi tạo để xuất dữ liệu theo luồng.

Các phương thức hỗ trợ xuất dữ liệu ra:

- Hoàn toàn tương tự lớp OutputStream.

d. PrintStream

Đây là một lớp con của lớp OutputStream, cho nên nó được kế thừa các phương thức của lớp OutputStream. Ngoài ra nó còn có một số phương thức bổ sung.

Các hàm khởi tạo:

- PrintStream(File): nhận tham số vào là một file (ghi dữ liệu ra file).
- PrintStream(OutputStream): nhận tham số là một dạng của OutputStream để ghi dữ liệu ra luồng đó.

Các phương thức hỗ trợ xuất dữ liệu ra:

- print/println(boolean): ghi dữ liệu ra dạng boolean
- print/println(byte): ghi dữ liệu ra dạng byte.
- print/println(double): ghi dữ liệu ra dạng double.
- print/println(float): ghi dữ liệu ra dạng float.
- print/println(int): ghi dữ liệu ra dạng int.
- print/println(char[]): ghi dữ liệu ra dạng mảng kí tự.
- print/println(String): ghi dữ liệu ra dạng chuỗi.
- print/println(Object): ghi dữ liệu ra dạng đối tượng.
- println(): ghi dữ liệu ra dòng mới.

e. Writer

Writer cũng là một lớp trừu tượng để ghi dữ liệu ra. Dù không thể khởi tạo trực tiếp đối tượng này nhưng ta có thể sử dụng các lớp con kế thừa từ lớp này.

Các hàm khởi tạo:

- Writer(): khởi tạo (trừu tượng)

Các phương thức hỗ trợ xuất dữ liệu ra:

- append(char): xuất thêm một kí tự ra
- append(CharSequence csq): xuất thêm một chuỗi kí tự ra.
- append(CharSequence csq, int start, int end): xuất thêm một phần trong chuỗi kí tự ra, bắt đầu từ vị trí *start*, cho đến vị trí *end*.
- write(char): ghi một kí tự ra.
- write(char[]): ghi một mảng kí tự ra.
- write(String): ghi một chuỗi ra.
- write(char[] b, int off, int len): ghi *len* kí tự trong mảng *b*, bắt đầu từ vị trí *off* ra luồng.
- write(String b, int off, int len): ghi *len* kí tự trong chuỗi *b*, bắt đầu từ vị trí *off* ra luồng.

f. BufferedWriter

Đây là một lớp con kế thừa từ lớp Writer nên có thể sử dụng các phương thức kế thừa từ lớp Writer. Ngoài ra nó có bổ sung một số phương thức.

Các hàm khởi tạo:

- BufferedWriter(Writer): khởi tạo bằng một đối tượng ghi dữ liệu ra.

Các phương thức hỗ trợ xuất dữ liệu ra:

- Hoàn toàn tương tự lớp Writer.

g. OutputStreamWriter

Đây là một lớp con kế thừa từ lớp Writer nên có thể sử dụng các phương thức kế thừa từ lớp Writer. Ngoài ra nó có bổ sung một số phương thức.

Các hàm khởi tạo:

- OutputStreamWriter(OutputStream): khởi tạo bằng một luồng xuất dữ liệu ra.

Các phương thức hỗ trợ xuất dữ liệu ra:

- Hoàn toàn tương tự lớp Writer.

2.1.4. Các kiểu xuất dữ liệu ra

a. Xuất dữ liệu ra màn hình

Để xuất dữ liệu ra màn hình, có thể dùng bất kì một trong các đối tượng ghi dữ liệu ra của Java. Chỉ cần truyền tham số khởi tạo là đối tượng System.out:

- Dùng DataOutputStream: dos = new DataOutputStream(System.out);
- Dùng BufferedOutputStream: bos = new BufferedOutputStream(System.out);
- Dùng PrintStream: ps = new PrintStream(System.out);
- Dùng BufferedWriter: bw = new BufferedWriter(System.out);
- Dùng OutputStreamWriter: osw = new OutputStreamWriter(System.out);

b. Xuất dữ liệu ra file

Để xuất dữ liệu ra màn file, có thể dùng bất kì một trong các đối tượng ghi dữ liệu ra của Java. Chỉ cần truyền tham số khởi tạo là đối tượng FileOutputStream hoặc FileWriter:

- Dùng DataOutputStream:

  dos = new DataOutputStream(new FileOutputStream(“out.txt”));
- Dùng BufferedOutputStream:

  bos = new BufferedOutputStream(new FileOutputStream(“out.txt”));
- Dùng PrintStream: ps = new PrintStream(new FileOutputStream(“out.txt”));
- Dùng BufferedWriter: bw = new BufferedWriter(new FileWriter(“out.txt”));
- Dùng OutputStreamWriter: osw = new OutputStreamWriter(new FileWriter(“out.txt”));

c. Xuất dữ liệu ra socket

Để xuất dữ liệu ra màn socket, có thể dùng bất kì một trong các đối tượng ghi dữ liệu ra của Java. Chỉ cần truyền tham số khởi tạo là đối tượng socket.getOutputStream():

- Dùng DataOutputStream: dos = new DataOutputStream(socket.getOutputStream());

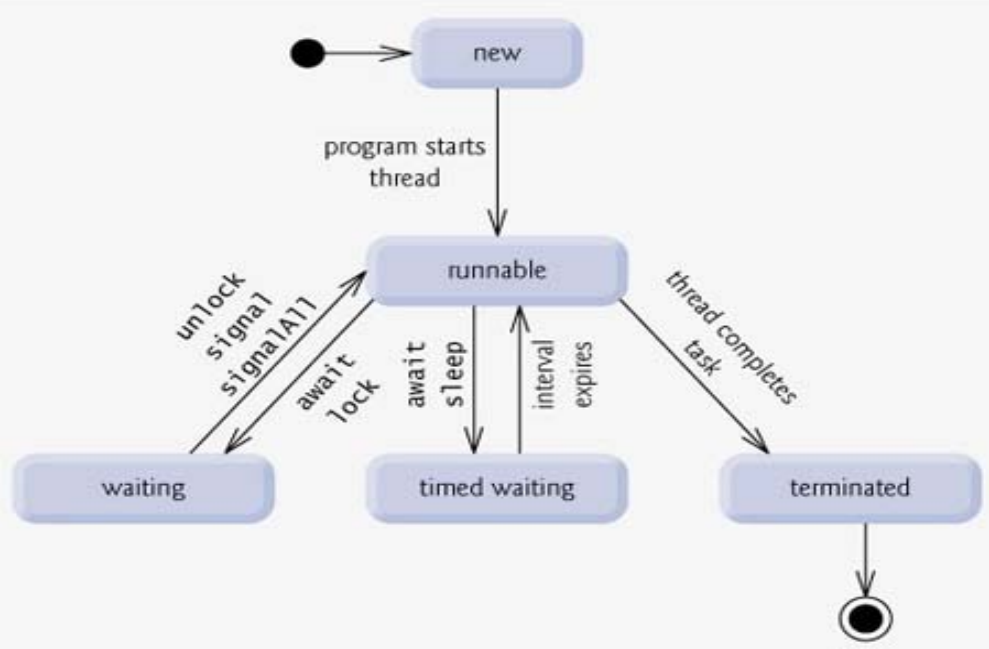
- Dùng BufferedOutputStream:

  bos = new BufferedOutputStream(socket.getOutputStream());
- Dùng PrintStream: ps = new PrintStream(socket.getOutputStream());
- Dùng BufferedWriter: bw = new BufferedWriter(socket.getOutputStream());
- Dùng OutputStreamWriter: osw = new OutputStreamWriter(socket.getOutputStream());

2.2. LẬP TRÌNH THREAD VỚI JAVA

2.2.1 Giới thiệu về Thread trong Java

Thread là một luồng, một tiến trình, một tiểu trình, hay đơn giản là một chương trình Java có thể chạy liên tục, độc lập, và đặc biệt là có thể chạy song song với các chương trình Java khác. Ngay cả khi chúng chạy trên các máy có hệ điều hành xử lí tuần tự, thì các thread của Java vẫn có thể chạy tựa như song song.



Hình 2.1: Mô hình hoạt động của Thread

Cơ chế hoạt động của một Thread trong Java được mô tả như trong hình 2.1:

- Ngay sau khi được khởi tạo, Thread có thể chạy liên tục không ngừng nghỉ, và chạy song song với các Thread khác (trạng thái runnable).
- Trong quá trình đang chạy, Thread chỉ có thể chuyển sang 2 trạng thái khác: chờ hoặc kết thúc.
- Trạng thái chờ: Thread ở dạng ngủ (sleep), không hoạt động gì cả, nhưng có thể phục hồi và tiếp tục các công việc đang dang dở (không phải khởi tạo lại từ đầu).

2.2.2. Lập trình với Thread trong Java

Để lập trình với Thread trong Java, cần thực hiện các bước như sau:

- Bước 1: Khai báo lớp đối tượng dưới dạng Thread. Có 2 cách khai báo.

  Cách 1: Kế thừa từ lớp Thread của Java

  ```
  public class LogWriter extends Thread{
  ```

  Cách 2: Cài đặt giao diện Runnable của Java

  ```
  public class LogWriter implements Runnable{
  ```

- Bước 2: Định nghĩa các hoạt động chính của Thread trong phương thức run()

  ```
  public void run(){
      định nghĩa ở đây
  }
  ```

- Bước 3: Gọi Thread thực hiện

  ```
  LogWriter lw1 = new LogWriter(...);
  lw1.start();
  ```

2.2.3. Ví dụ

Bài toán đặt ra như sau:

- Hệ thống có nhiều người dùng. Các người dùng có thể login vào dùng hệ thống bất cứ thời điểm nào. Có nghĩa là có thể có hai hay nhiều người dùng login vào đồng thời (song song).
- Mỗi người dùng, khi login vào hệ thống thì sẽ bị lưu vết lại bằng cách ghi thời điểm họ login vào một file log.txt ở bộ nhớ ngoài.
- Chương trình phải đảm bảo việc ghi ra file log đúng thời điểm và đúng thứ tự thời gian login vào hệ thống của từng người.

Để đảm bảo yêu cầu ghi ra file log đúng thứ thự thì phải sử dụng Thread cho mỗi lần một người login vào hệ thống. Nếu không việc ghi ra file log chỉ thực hiện được tuần tự chứ không thể thực hiện được song song cho nhiều người login vào hệ thống đồng thời.

Mã nguồn của chương trình như sau:

//LogWriter.java

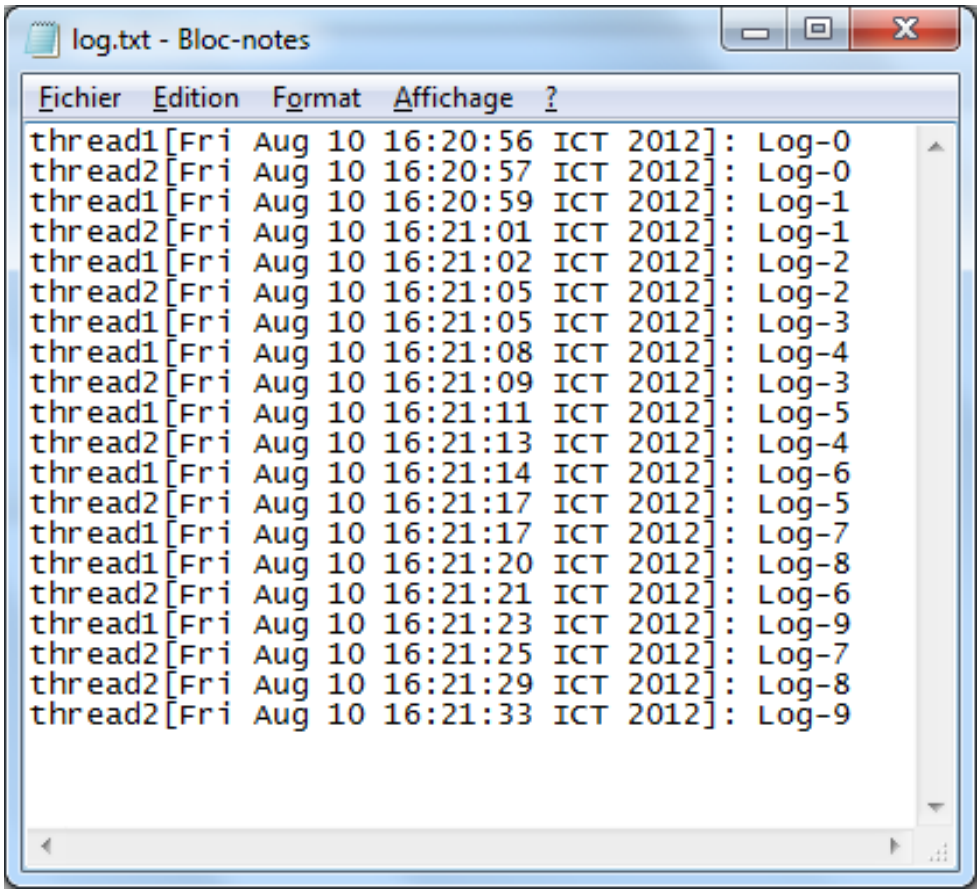
```
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.Writer;
import java.util.Calendar;

public class LogWriter extends Thread{
	private String filename;
	private long time;

	public LogWriter(String name, String filename, long time){
```

```
            super(name);
            this.filename = filename;
            this.time = time;
      }

      public void run(){
            for(int i=0; i<10; i++){
                  try{
                        Writer  wr = new BufferedWriter(new
                              FileWriter(filename,true));
                        this.sleep(time);
                        wr.append(getName() + "[" +
                              Calendar.getInstance().getTime() + "]: Log-" +
                              i + "\r\n");
                        wr.close();
                  }catch(Exception e){
                        System.out.println(e.getStackTrace());
                  }
            }
      }
}
```

//Hàm main

```
      public static void main(String[] args){
            LogWriter lw1 = new LogWriter("thread1", "log.txt", 3000);
            LogWriter lw2 = new LogWriter("thread2", "log.txt", 4000);
            lw1.start();
            lw2.start();
      }
```

Kết quả nội dung file log.txt như sau:

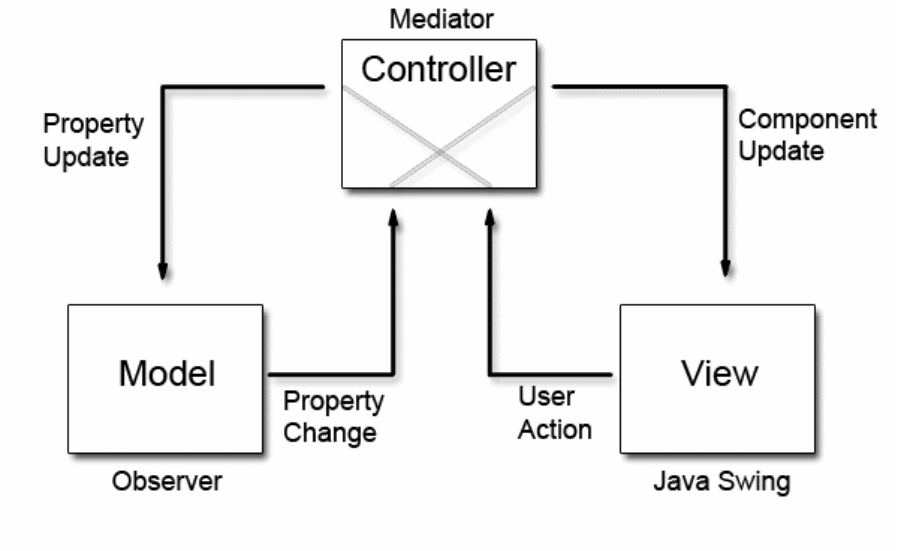


Trong đoạn ví dụ này, có 2 người dùng tên là thread1 và thread2. Thread1 cứ 3 giây lại ghi vào file log một lần. Trong khi đó, thread2 cứ 4 giây lại ghi vào file log một lần. Mỗi thread ghi vào file log 10 lần. Có thể nhận thấy nội dung các dòng log trong file log ghi đan xen giữa thread1 và thread2. Trong khi lời gọi hàm thức hiện là thread1 rồi mới đến thread2: nếu không có cơ chế thread thì thread1 sẽ ghi vào file log 10 lần của mình, sau đó mới đến lượt thread2 ghi. Tuy nhiên, vì cơ chế thread nên hai thread này chạy tự như song song nhau. Do đó, thời gian ghi cũng chạy song song nhau.

2.3. LẬP TRÌNH THEO MÔ HÌNH MVC

2.3.1. Giới thiệu mô hình MVC

Mô hình MVC (Model – View - Control) được sử dụng khá rộng rãi để thiết kế các phần mềm hiện nay. Theo đó, hệ thống được nhóm thành 3 thành phần chính (Hình 2.2):



Hình 2.2: Mô hình MVC tổng quan

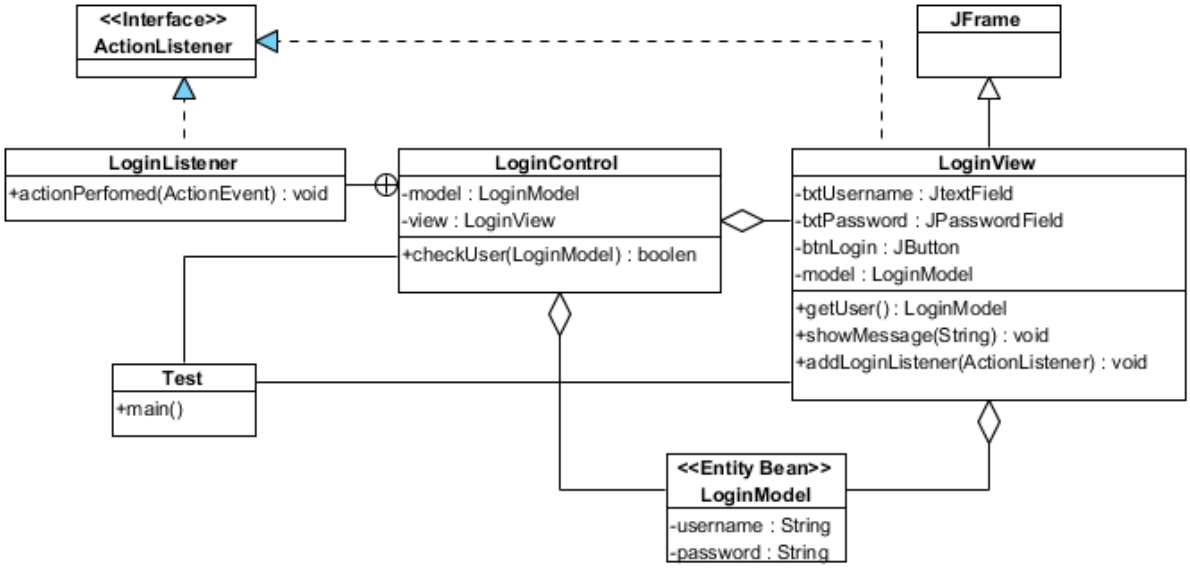
- Thành phần Model (M): mô hình, hay còn được gọi với nhiều tên khác như thực thể (entity, bean). Là các lớp chứa thông tin để xử lí của hệ thống. Các thông tin không nên để riêng lẻ mà nên hợp lại thành các lớp thực thể để trao đổi, truyền/nhận, và xử lí giữa các lớp thuộc các phần còn lại như Control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

2.3.2 Ví dụ

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

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

a. Thiết kế



Hình 2.3: Sơ đồ lớp của hệ thống

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

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

Tuần tự các bước xử lí như sau:

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

6. Lớp LoginView hiển thị kết quả đăng nhập lên cho người dùng

b. Cài đặt ứng dụng login theo mô hình MVC

Lớp LoginModel.java

```
package login_GUI_MVC;

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

	public LoginModel(){
	}

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

	public String getPassword() {
		return password;
	}

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

	public String getUserName() {
		return userName;
	}

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

Lớp LoginView.java

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

public class LoginView extends JFrame implements ActionListener{
	private JTextField txtUsername;
	private JPasswordField txtPassword;
```

```
	private JButton btnLogin;
	private LoginModel model;

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

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

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

		btnLogin.addActionListener(this);

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

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

	public void actionPerformed(ActionEvent e) {
	}

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

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

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

Lớp LoginControl.java

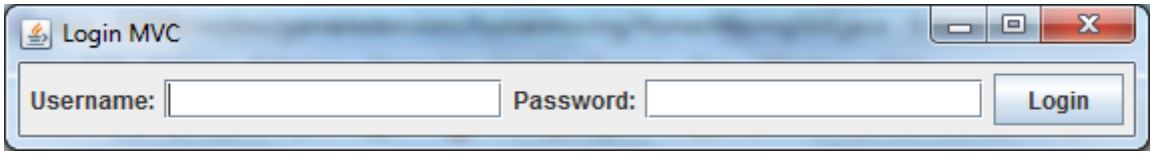
```
package login_GUI_MVC;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
```

```
import java.sql.Statement;


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

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

		view.addLoginListener(new LoginListener());
	}

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

	public boolean checkUser(LoginModel user) throws Exception {

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

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

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

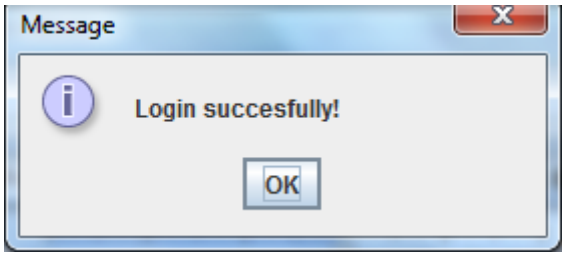
Lớp Test.java

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

Kết quả



Login thành công:



Login lỗi:

2.4. KẾT LUẬN

Nội dung chương này đã trình bày một số kiến thức cơ sở về Java hỗ trợ cho lập trình mạng, bao gồm: các kỹ thuật vào ra dữ liệu với Java, lập trình đa luồng với Thread trong Java, lập trình theo mô hình 3 tầng MVC.

Các kiến thức này là nền tảng cơ bản và sẽ được sử dụng nhiều trong các kỹ thuật lập trình mạng trong các phần tiếp theo.

2.5. BÀI TẬP

1. Viết chương trình copy file chỉ dùng cặp đối tượng: BufferedInputStream và BufferedOutputStream. Test chương trình trong 3 trường hợp: file text dung lượng nhỏ, file ảnh dung lượng vừa, file phim dung lượng lớn hơn 4G.
2. Viết chương trình copy file chỉ dùng cặp đối tượng: Scanner và PrintStream. Test chương trình trong 3 trường hợp: file text dung lượng nhỏ, file ảnh dung lượng vừa, file phim dung lượng lớn hơn 4G. So sánh chương trình với chương trình trong bài 1 xem chương trình nào copy file nhanh hơn?
3. Viết chương trình copy tất cả các file trong cùng một thư mục sang thư mục khác, trong đó mỗi file được copy trên một thread độc lập. So sánh thời gian copy với trường hợp copy không dùng thread?
4. Viết chương trình giải phương trình bậc 2 theo đúng mô hình MVC?
5. Viết chương trình giải hệ phương trình bậc nhất theo đúng mô hình MVC?
6. Viết chương trình tìm USCLN (BSCNN) của 2 số nguyên dương a và b theo đúng mô hình MVC?
7. Viết chương trình phân tích một số nguyên dương ra thừa số nguyên tố (ví dụ: 300 = 2*2*3*5*5) theo đúng mô hình MVC?

CHƯƠNG 3. LẬP TRÌNH VỚI CƠ SỞ DỮ LIỆU

Chương này sẽ trình bày các nội dung:

- Cách thiết kế CSDL cho ứng dụng
- Thao tác với dữ liệu bằng lệnh SQL
- Lập trình Java thao tác với CSDL.

3.1. THIẾT KẾ CSDL CHO ỨNG DỤNG

3.1. Các bước để thiết kế CSDL cho ứng dụng

Để tiến hành thiết kế CSDL cho một ứng dụng, ta cần thực hiện bốn bước như sau:

- Bước 1: Đọc bản mô tả hệ thống. Nếu bắt gặp một thực thể cần quản lí (có thông tin thuộc tính kèm theo), thì đề xuất nó thành một bảng, thuộc tính của đối tượng thành thuộc tính của bảng tương ứng.
- Bước 2: Xem xét quan hệ giữa các bảng (đối tượng) vừa trích được.
    - Nếu quan hệ là 1-1 thì gộp chung hai bảng thành một.
    - Nếu quan hệ là 1-n thì giữ nguyên (vì chuẩn hóa rồi).
    - Nếu quan hệ là n-n thì phải đề xuất thêm bảng trung gian (có thể nhiều hơn 1 bảng trung gian) sao cho quan hệ các bảng về dạng 1-n.
- Bước 3: Với mỗi quan hệ 1-n, bên bảng có nhãn n phải có khóa ngoài, tham chiếu đến khóa chính của bảng có nhãn 1.
- Bước 4: Bổ sung thêm các thuộc tính còn thiếu. Loại bỏ các thuộc tính gây dư thừa dữ liệu.

3.2. Ví dụ

Người ta yêu cầu chúng ta thiết kế CSDL cho bài toán quản lí đặt phòng khách sạn với mô tả như sau:

- Một khách sạn (id, tên, địa chỉ, số sao, mô tả) có nhiều phòng (id, tên phòng, hạng phòng, giá niêm yết, mô tả)
- Mỗi phòng có thể được đặt bởi nhiều khách hàng (id, tên, số id, kiểu thẻ id, địa chỉ, email, số điện thoại, mô tả) tại nhiều thời điểm khác nhau
- Mỗi khách hàng có thể đặt nhiều phòng tại nhiều thời điểm khác nhau nhưng chỉ ở 1 phòng tại 1 thời điểm nhất định, xác định 1 giá xác định
- Khách hàng chỉ có thể đặt phòng nếu phòng còn trống trong suốt thời gian khách hàng muốn đặt

- Khi nhận phòng, khách hàng chỉ cần xuất trình giấy tờ tùy thân, nhân viên khách sạn sẽ giao đúng chìa khóa phòng mà khách đã đặt.
- Khi trả phòng, nhân viên in phiếu thanh toán bao gồm tên khách sạn, tên khách hàng, số phòng, hạng phòng, ngày đến, ngày đi, và tổng số tiền thanh toán
- Khách hàng có thể thanh toán nhiều lần cho đến trước ngày trả phòng

Các bước tiến hành thiết kế CSDL cho hệ thống trên được tiến hành như sau:

Bước 1: Trích các đối tượng phải quản lí thông tin thành các bảng.

Đọc phần mô tả hệ thống, chúng ta dễ dàng xác định được ít nhất có ba đối tượng phải quản lí thông tin là: khách sạn, phòng, và khách hàng. Ngoài ra, khi khách hàng thanh toán, hệ thống phải xuất hóa đơn cho mỗi lần thanh toán, cho nên phải quản lí thêm đối tượng hóa đơn. Vậy ta có bốn đối tượng, tạo ra bốn bảng ban đầu:

- tblHotel: bảng lưu thông tin khách sạn, với các thuộc tính ban đầu: id (khóa chính), tên (name), số sao (starLevel), địa chỉ (address), mô tả (description).
- tblRoom: bảng lưu thông tin phòng của khách sạn, với các thuộc tính ban đầu: id (khóa chính), tên phòng (name), kiểu phòng (type), giá hiển thị (price), mô tả (description).
- tblUser: bảng lưu thông tin khách hàng. Lưu ý là thông tin khách hàng và thông tin người dùng hệ thống có thể gộp chung lại một bảng cho dễ quản lí, cho nên bảng này chuwacs các thuộc tính của khách hàng và người dùng hệ thống: id (khóa chính), username, password, tên đầy đủ (fullname), số id (idNumber), kiểu thẻ id (idType), địa chỉ (address), email, số điện thoại (tel), mô tả (description).
- tblBill: bảng lưu thông tin hóa đơn cho mỗi lần thanh toán của khách hàng. Trong đó có ít nhất các thuộc tính: id (khóa chính), ngày thanh toán (paymentDate), tổng số tiền thanh toán (paymentAmount), hình thức thanh toán (paymentType), và ghi chú (description).

Bước 2: Xét quan hệ giữa các bảng vừa trích được.

- 1 khách sạn có nhiều phòng, 1 phòng chỉ ở trong 1 khách sạn → quan hệ giữa bảng tblHotel và tblRoom là 1-n. Không phải chuẩn hóa nữa.
- 1 khách hàng đặt nhiều phòng, 1 phòng cũng có nhiều người đặt → quan hệ giữa tblRoom và tblUser là n-n, → phải chuẩn hóa → Tạo thêm một bảng đặt chỗ chi tiết tblBooking, xác định duy nhất: ngày bắt đầu (startDate), ngày kết thúc (endDate), giá phòng (price), ghi chú nếu có (description). Khi đó, quan hệ giữa tblRoom và tblBooking là 1-n (một phòng có nhiều lần đặt), và giữa tblUser và tblBooking cũng là 1-n (một người có nhiều lần đặt), là đạt chuẩn 3NF.
- Mỗi lần đặt phòng, khách hàng có thể thanh toán nhiều lần khác nhau → quan hệ giữa bảng tblBooking và tblBill là quan hệ 1-n. Đã đạt chuẩn 3NF.

Bước 3: Bổ sung khóa ngoại cho các quan hệ giữa các bảng.

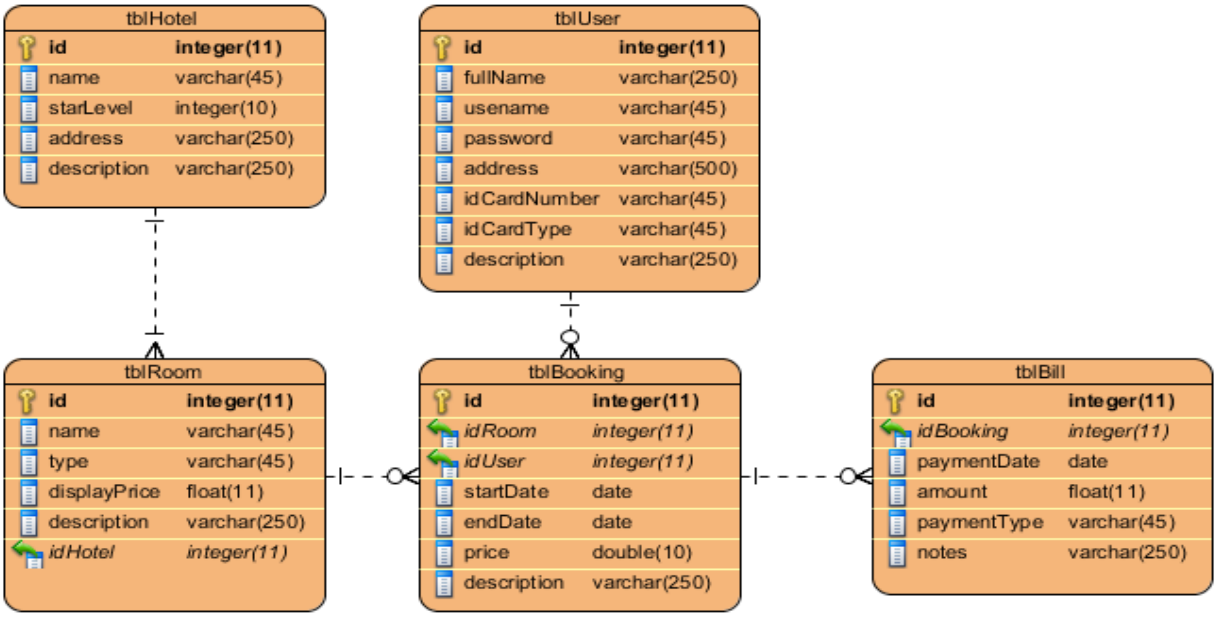
- Quan hệ giữa bảng tblHotel và bảng tblRoom là 1-n → bảng tblRoom phải bổ sung thêm thuộc tính khóa ngoài *idHotel*, tham chiếu đến khóa chính của bảng tblHotel.
- Quan hệ giữa bảng tblRoom và bảng tblBooking là 1-n → bảng tblBooking phải bổ sung thêm thuộc tính khóa ngoài *idRoom*, tham chiếu đến khóa chính của bảng tblRoom.
- Quan hệ giữa bảng tblUser và bảng tblBooking là 1-n → bảng tblBooking phải bổ sung thêm thuộc tính khóa ngoài *idUser*, tham chiếu đến khóa chính của bảng tblUser.
- Quan hệ giữa bảng tblBooking và bảng tblBill là 1-n → bảng tblBill phải bổ sung thêm thuộc tính khóa ngoài *idBooking*, tham chiếu đến khóa chính của bảng tblBooking.

Bước 4: Rà soát lại lần cuối các thuộc tính

- Sau khi rà soát lại các thuộc tính của các bảng, chỉ còn duy nhất thuộc tính giá phòng (price) là bị lặp lại ở hai bảng khác nhau: tblRoom và tblBooking. Tuy nhiên, điều này không gây dư thừa dữ liệu. Lí do là thuộc tính price ở bảng tblRoom chỉ dùng để hiển thị thông tin khi tìm kiếm phòng cho thuê, nó có thể dễ dàng thay đổi theo mùa vụ. Còn thuộc tính price trong bảng tblBooking xác định giá thuê cho một khách hàng, ở một phòng xác định vào một thời điểm xác định, thông tin này không được thay đổi và có giá trị lưu trữ cũng như thống kê doanh thu sau này. Do đó hai thuộc tính có vai trò khác nhau và không thay thế được cho nhau, cho nên vẫn phải lưu vào hai bảng như đã thiết kế.

Kết quả:

Kết thu được CSDL cho hệ thống đã mô tả được trình bày trong Hình 3.1.



Hình 3.1: Kết quả CSDL của hệ thống quản lí đặt phòng khách sạn đã mô tả

3.2. SỬ DỤNG LỆNH SQL

Các lệnh SQL thao tác với dữ liệu có thể chia làm 2 nhóm. Thứ nhất là nhóm lệnh làm thay đổi CSDL sau khi thực hiện, bao gồm 3 lệnh: insert, update, delete. Thứ hai là nhóm lệnh không làm thay đổi CSDL sau khi thực hiện, chỉ gồm lệnh: select.

3.2.1. Lệnh INSERT

Cú pháp:

INSERT INTO tên-bảng(các-thuộc-tính-của-bảng)

VALUES(các-giá-trị-tương-ứng-với-các-thuộc-tính-của-bảng)

Trong đó:

- tên-bảng: tên của bảng dữ liệu muốn thêm.
- các-thuộc-tính-của-bảng: liệt kê tên các thuộc tính của bảng, cách nhau bởi dấu phẩy. Nếu không liệt kê phần này thì các giá trị trong phần (các-giá-trị-tương-ứng-với-các-thuộc-tính-của-bảng) phải tương ứng với thứ tự cột trong bảng
- các-giá-trị-tương-ứng-với-các-thuộc-tính-của-bảng: liệt kê các giá trị tương ứng với các cột đã liệt kê phía trên, nếu các cột không được liệt kê ra thì phải theo thứ tự các cootj trong bảng thực tế. Các giá trị cách nhau bởi dấu phẩy. Giá trị kiểu chuỗi, date... phải đặt trong cặp dấu nháy.

Ví dụ câu lệnh:

```
INSERT INTO `tblhotel`(`address`,`id`,`name`,`starLevel`)
VALUES("Sai Gon",5,"Saigon Star",3);
```

sẽ thêm một dòng vào bảng tblHotel như trong Hình 3.2.

id	name	address	starLevel	description
1	Daiwoo	Kim Mã, Hà Nội	5	NULL
2	Sofitel	Tây Hồ, Hà Nội	5	NULL
3	Metropole	Hoàn Kiếm, Hà Nội	4	NULL
4	Bảo Sơn	Cầu Giấy, Hà Nội	3	NULL
5	Saigon Star	Sai Gon	3	NULL
NULL	NULL	NULL	NULL	NULL

Hình 3.2: Ví dụ câu lệnh Insert

3.2.2. Lệnh UPDATE

Cú pháp:

UPDATE tên-bảng

SET tên-cột-1 = giá-trị-1, tên-cột-2 = giá-trị-2

WHERE điều-kiện-lọc

Trong đó:

- tên-bảng: tên của bảng dữ liệu muốn cập nhật.
- Tên-cột = giá-trị: là các cặp tên cột và giá trị muốn cập nhật vào cột đấy. Nếu muốn cập nhật trên nhiều cột thì liệt kê theo cặp như vậy cho tất cả các cột, cách nhau bởi dấu phẩy.
- Điều-kiện-lọc: điều kiện lọc để lấy ra những dòng thõa mãn điều kiện này thì mới cập nhật giá trị vào các cột tương ứng.

Ví dụ câu lệnh:

```
UPDATE `tblhotel`
SET `address` = "Quận 1, TP. Hồ Chí Minh"
WHERE id = 5;
```

sẽ cập nhật cột address của dòng có id =5 trong bảng tblHotel như trong Hình 3.3.

id	name	address	starLevel	description
1	Daiwoo	Kim Mã, Hà Nội	5	NULL
2	Sofitel	Tây Hồ, Hà Nội	5	NULL
3	Metropole	Hoàn Kiếm, Hà Nội	4	NULL
4	Bảo Sơn	Cầu Giấy, Hà Nội	3	NULL
5	Saigon Star	Quận 1, TP. Hồ Chí Minh	3	NULL
NULL	NULL	NULL	NULL	NULL

Hình 3.3: Ví dụ kết quả câu lệnh Update

3.2.3. Lệnh DELETE

Cú pháp:

```
DELETE FROM tên-bảng
WHERE điều-kiện-lọc
```

Trong đó:

- tên-bảng: tên của bảng dữ liệu muốn xóa.
- điều-kiện-lọc: điều kiện lọc để lấy ra những dòng thõa mãn điều kiện này thì mới xóa.

Ví dụ câu lệnh:

```
DELETE FROM `tblhotel`
WHERE id = 5;
```

sẽ xóa dòng có id =5 trong bảng tblHotel như trong Hình 3.4.

id	name	address	starLevel	description
1	Daiwoo	Kim Mã, Hà Nội	5	NULL
2	Sofitel	Tây Hồ, Hà Nội	5	NULL
3	Metropole	Hoàn Kiếm, Hà Nội	4	NULL
4	Bảo Sơn	Cầu Giấy, Hà Nội	3	NULL
NULL	NULL	NULL	NULL	NULL

Hình 3.4: Ví dụ kết quả câu lệnh Delete

3.2.3. Lệnh SELECT

Cú pháp:

SELECT tên-các-cột
FROM tên-bảng
WHERE điều-kiện-lọc
GROUP BY tên-các-cột-bị-nhóm
ORDER BY tên-cột-cần-sắp-xếp
TOP n

Trong đó:

- tên-các-cột: tên các cột cần lấy dữ liệu ra. Nếu các cột từ nhiều bảng có trùng tên cột thì liệt kê theo cú pháp tên-bảng.tên-cột. Các cột cách nhau bởi dấu phẩy. Trong tên cột này có thể lồng một câu Select khác.
- tên-bảng: tên của bảng dữ liệu muốn truy vấn dữ liệu. Có thể liệt kê nhiều bảng, cách nhau bởi dấu phẩy, hoặc sử dụng lệnh JOIN.
- điều-kiện-lọc: điều kiện lọc để lấy ra những dòng thỏa mãn điều kiện này. Điều kiện lọc có thể liên quan chỉ 1 bảng, hoặc liên quan nhiều bảng đồng thời. Trong điều kiện lọc có thể lồng một câu Select khác.
- tên-các-cột-bị-nhóm: chỉ dùng trong trường hợp trong phần tên các cột sau chữ select có các toán tử: MIN, MAX, COUNT, SUM, AVERAGE... và chỉ liệt kê tên các cột không nằm trong các toán tử trên vào phần này.
- tên-cột-cần-sắp-xếp: liệ kê tên các cột là tiêu chí sắp xếp, mặc định là sắp xếp tăng dần theo giá trị các cột này. Nếu muốn sắp xếp giảm dần thì thêm từ khóa DESC vào sau tên từng cột.
- n: số lượng bản ghi trả về, giới hạn bởi n.

Ví dụ với CSDL của hệ thống quản lí đặt phòng khách sạn. Chúng ta có dữ liệu trong từng bảng như trong Hình 3.5.

Bảng tblHotel:

id	name	address	starLevel	description
1	Daiwoo	Kim Mã, Hà Nội	5	NULL
2	Sofitel	Tây Hồ, Hà Nội	5	NULL
3	Metropole	Hoàn Kiếm, Hà Nội	4	NULL
4	Bảo Sơn	Cầu Giấy, Hà Nội	3	NULL
5	Saigon Star	Quận 1, TP. Hồ Chí Minh	3	NULL

id	idHotel	name	type	displayPrice	description
1	4	102	single	700000	NULL
2	4	103	double	1000000	NULL
3	4	104	twink	1000000	NULL
4	4	202	single	650000	NULL
5	4	203	double	950000	NULL
6	4	204	twink	950000	NULL
7	4	302	single	900000	NULL
8	4	303	double	900000	NULL
9	4	304	twink	900000	NULL

id	username	password	fullName	idCardNumber	idCardType	address	description
1	xuanhinh	abc123	Xuân Hinh	123456789	CMTND	Nam Định	NULL
2	minhvuong	abc123	Minh Vượng	234567891	CMTND	Hà Nội	NULL
3	xuanbac	abc123	Xuân Bắc	345678912	CMTND	Hà Nội	NULL
4	tranthanh	abc123	Trấn Thành	456789123	CMTND	Xì Gòn	NULL
5	david	abc123	David James	B0809076	Passport	Anh	NULL
6	tom	abc123	Tom Cruise	B123456	Passport	Úc	NULL
7	nhanvien	abc123	Nhân Viên	147258369	CMTND	KS	NULL

id	idRoom	idUser	startDate	endDate	price
1	1	1	2013-08-25	2013-08-28	750000
2	1	2	2013-08-30	2013-09-02	800000
3	1	3	2013-09-07	2013-09-15	600000
4	4	4	2013-08-26	2013-08-30	750000
5	7	5	2013-08-28	2013-09-01	650000

Bảng tblRoom:

Bảng tblUser:

Bảng tblBooking:

Bảng tblBill:

id	idBooking	paymentDate	amount	paymentType	notes
1	2	2013-08-15	2400000	visa card	NULL
2	1	2013-08-28	2250000	cash	NULL
3	3	2013-09-07	2400000	cash	NULL
4	3	2013-09-15	2400000	master card	NULL

Hình 3.5: CSDL ban đầu

Khi đó, câu lệnh tìm kiếm tất cả người dùng mà tên có chứa chữ "a":

```
SELECT * FROM tblUser
WHERE fullName LIKE "%a%";
```

sẽ cho kết quả như Hình 3.6.

id	username	password	fullName	idCardNumber	idCardType	address	description
1	xuanhinh	abc123	Xuân Hinh	123456789	CMTND	Nam Định	NULL
3	xuanbac	abc123	Xuân Bắc	345678912	CMTND	Hà Nội	NULL
4	tranthanh	abc123	Trần Thành	456789123	CMTND	Xi Gòn	NULL
5	david	abc123	David James	B0809076	Passport	Anh	NULL
7	nhanvien	abc123	Nhân Viên	147258369	CMTND	KS	NULL

Hình 3.6: Danh sách người dùng mà tên chứa chữ "a"

Câu lệnh Tìm kiếm tất cả những người dùng có đặt phòng từ 15/08/2013 đến 30/08/2013:

```
SELECT a.fullName, a.idCardNumber, a.idCardType, a.address, b.name AS `room`,
        b.type, c.startDate, c.endDate, c.price
FROM tblUser a, tblRoom b, tblBooking c
WHERE c.startDate BETWEEN "2013-08-15" AND "2013-08-30"
AND c.endDate BETWEEN "2013-08-15" AND "2013-08-30"
AND b.id = c.idRoom
AND a.id = c.idUser;
```

sẽ cho kết quả như Hình 3.7

fullName	idCardNumber	idCardType	address	room	type	startDate	endDate	price
Xuân Hinh	123456789	CMTND	Nam Định	102	single	2013-08-25	2013-08-28	750000
Trần Thành	456789123	CMTND	Xi Gòn	202	single	2013-08-26	2013-08-30	750000

Hình 3.7: Danh sách khách hàng đặt phòng trong khoảng thời gian

Câu lệnh tìm kiếm và sắp xếp những khách hàng đã trả tiền nhiều nhất đến ít nhất:

```
SELECT a.id, a.fullName, a.idCardNumber, a.idCardType, a.address,
        SUM(c.amount) AS `amount`
FROM tblUser a INNER JOIN tblBooking b ON b.idUser = a.id
```

```
        INNER JOIN tblBill c ON c.idBooking = b.id
    GROUP BY a.id, a.fullName, a.idCardNumber, a.idCardType, a.address
    ORDER BY `amount` DESC;
```

sẽ cho kết quả như Hình 3.8

id	fullName	idCardNumber	idCardType	address	amount
3	Xuân Bắc	345678912	CMTND	Hà Nội	4800000
2	Minh Vượng	234567891	CMTND	Hà Nội	2400000
1	Xuân Hinh	123456789	CMTND	Nam Định	2250000

Hình 3.8: Kết quả thống kê khách hàng theo doanh thu

3.3. LẬP TRÌNH JAVA VỚI CSDL

3.3.1. Tạo kết nối vào CSDL bằng JDBC

Để tạo kết nối từ Java vào CSDL, cần thực hiện các bước như sau:

- Tải thư viện Jdbc driver dành cho hệ quản trị CSDL tương ứng (SQLServer hoặc MySQL), import vào library của project (trong Eclipse hoặc NetBean).
- Mở hệ quản trị CSDL ra và tạo một CSDL, đặt tên theo tên dự án của mình. Ví dụ hotelManagement. Sau đó tạo các bảng, các cột của bảng theo đúng như đã thiết kế.
- Mở dự án trong Eclipse hoặc NetBean, cài đặt phương thức kết nối vào CSDL của lớp DAO (data access object) như sau:

```
public Connection getConnection(String dbClass, String dbUrl, String
userName, String password) throws SQLException {
    Connection conn = null;
    try {
      Class.forName(dbClass);
      conn = DriverManager.getConnection (dbUrl, userName, password);
    }catch(ClassNotFoundException e) {
      e.printStackTrace();
    }catch(SQLException e) {
      throws e;
    }
    return conn;
  }
```

Trong đó:

- dbClass: là tên driver của JDBC. Tên này khác nhau nếu khác hệ quản trị CSDL:
  MySQL: `dbClass = "com.mysql.jdbc.Driver";`
  SQLServer: `dbClass = "com.microsoft.sqlserver.jdbc.SQLServerDriver";`
- dbUrl: địa chỉ truy cập đến server CSDL:
  MySQL: `dbUrl = "jdbc:mysql://your.database.domain:3306/yourDBname";`
  SQLServer: `dbUrl = "jdbc:sqlserver://your.database.domain:1433;instant=MSSQLSERVER;databasename=yourDBname";`
- your.database.domain: tên máy hoặc địa chỉ IP của máy mà server CSDL đang chạy.
- yourDBname: tên CSDL vừa tạo ở bước trên.

- username: tên đăng nhập khi cài hệ quản trị CSDL. Thường SQLServer thì là "admin", MySQL thì là "sa".
- password: mật khẩu khi cài đặt hệ quản trị CSDL.
- Có thể bỏ qua tham số username, password trong lệnh kết nối này.

Lưu ý:

- Hàm này thường được gọi trực tiếp trong hàm khởi tạo của lớp DAO, sau đó biến Connection sẽ được dùng chung cho tất cả các phương thức của lớp DAO.

3.3.2. Sử dụng lệnh SQL trong Java

Cách 1: dùng Statement

- Bước 1: khai báo câu lệnh SQL ở dạng String, ví dụ:

```
String query = "Select * FROM users";
String query = "INSERT INTO users VALUES(« aaa », « bbb »)";
String query = "UPDATE users SET password = « xxx » WHERE id = 111";
String query = "DELETE FROM users WHERE id = 111";
```

- Bước 2: Gọi đối tượng Statement để thực hiện

```
Statement stmt = conn.createStatement();
```

- Nếu câu SQL trong query là lệnh Select thì dùng phương thức:

```
ResultSet rs = stmt.executeQuery(query);
```

- Nếu câu SQL trong query là các lệnh Insert, Update, hay Delete thì dùng phương thức:

```
stmt.executeUpdate(query);
```

Lưu ý:

- Khi dùng cách này, muốn truyền tham biến vào câu SQL thì phải thực hiện cộng xâu vào String của câu SQL.

Cách 2: dùng PreparedStatement

- Bước 1: khai báo câu lệnh SQL ở dạng String tương tự dùng statement. Tuy nhiên điểm khác biệt là khi muốn truyền tham biến vào vị trí nào của câu SQL thì ta đặt dấu chấm hỏi (?) vào đúng vị trí đó. Ví dụ:

```
String query = "update products set SALES = ? where ID = ?";
```

- Bước 2: Gọi đối tượng Prepare Statement để thực hiện

```
PreparedStatement stmt = conn.preparedStatement(query);
```

- Bước 3: Truyền tham trị vào các vị trí có dấu hỏi, các dấu hỏi được tính thứ tự từ 1, theo thứ tự xuất hiện trong câu SQL:

```
query.setInt(1, value);
query.setInt(2, productId);
```

- Nếu câu SQL trong query là các lệnh Select thì dùng phương thức:

```
ResultSet rs = stmt.executeQuery();
```

- Nếu câu SQL trong query là các lệnh Insert, Update, hay Delete thì dùng phương thức:

```
stmt.executeUpdate();
```

Lưu ý:

- Khi dùng cách này, muốn truyền tham biến vào câu SQL thì phải thực hiện cộng xâu vào String của câu SQL.

3.3.3. Lấy dữ liệu ra từ SQL

Trong Java, dữ liệu trong các câu truy vấn được lấy ra từ đối tượng ResultSet. Nó có dạng như một bảng dữ liệu trong bộ nhớ, bao gồm các hàng và cột tương ứng với câu truy vấn trước đó. Đối tượng này có một số phương thức hay được dùng như sau:

- next(): trả về true nếu trong nó vẫn còn dữ liệu, ngược lại trả về false.
- last(): trỏ về dòng cuối cùng trong bảng kết quả
- beforeFirst(): trỏ về đầu bảng (trước dòng thứ nhất chứ không phửi trỏ vào dòng thứ nhất)
- getString(k)/getDate(k)/getInt(k)/getFloat(k)/getDouble(k)/getLong(k)/getBoolean(k): lấy về một giá trị dữ liệu dạng String/Date/Int/Float/Double/Long/Bolean từ cột thứ k. Có thể thay đổi số tự tự cột k bằng tên cột của bảng.

Ví dụ đoạn chương trình sau sẽ đọc thông tin tất cả người dùng có trong bảng users ra và lưu vào mảng listUser:

```
String query = "Select * FROM users";
User[] listUser = null;
try {
    Statement stmt = conn.createStatement();
    ResultSet rs = stmt.executeQuery(query);

      // get number of row in resultSet
      if (rs.last()) {
            listUser = new User[rs.getRow()];
            rs.beforeFirst(); // not rs.first()
      }

    int count = 0;
    while (rs.next()) {
      // do something with data...
      listUser[count] = new User(rs.getString("username"),
                  rs.getString("password"))
    }
}catch(SQLException e) {
    e.printStackTrace();
}
```

3.3.4. Ví dụ

Mục này sẽ trình bày một số ví dụ lập trình với CSDL dùng CSDL về hệ thống quản lí đặt phòng khách sạn như đã thiết kế trong mục 3.1. Mỗi chức năng ở mục này được trình bày như một phương thức của lớp DAO.

Chức năng thêm thông tin một phòng vào CSDL:

```
public void addRoom(int hotelID, Room r)throws SQLException{
    String sql = "INSERT INTO tblRoom(name, idHotel, type, displayPrice,
                  description) VALUES(?,?,?,?,?)";
    PreparedStatement prstm = null;
    try {
      prstm = conn.prepareStatement(sql);
      prstm.setString(1, r.getName());
      prstm.setInt(2, hotelID);
       prstm.setString(3, r.getType());
       prstm.setFloat(4, r.getDisplayPrice());
       prstm.setString(5, r.getDescription());
      prstm.executeUpdate();
    } catch (SQLException e ) {
        throw e;
    }
}
```

Tìm các phòng trống trong khoảng thời gian từ ngày sd đến ngày ed:

```
public Room[] searchFreeRoom(Date sd, Date ed)throws SQLException{
    Room[] result = null;
    String sql = "SELECT * FROM tblRoom WHERE id NOT IN (SELECT RoomId
from tblBooking WHERE ((StartDate BETWEEN ? AND ?) OR EndDate BETWEEN ?
AND ?) OR (? BETWEEN StartDate AND EndDate) OR (? BETWEEN StartDate AND
EndDate)));
    try {
        PreparedStatement prstm = conn.prepareStatement(sql);
        prstm.setDate(1, sd); prstm.setDate(3, sd); prstm.setDate(5, sd);
        prstm.setDate(2, ed); prstm.setDate(4, ed); prstm.setDate(6, ed);
        ResultSet rs = prstm.executeQuery();
        if (rs.last()) {
            result = new Room[rs.getRow()];
            rs.beforeFirst();
        }
        int count = 0;
        while (rs.next()) {
             result[count] = new Room(rs.getInt(1), rs.getString(2),
                      rs.getString(3), rs.getFloat(4), rs.getString(5));
            count++;
        }
    } catch (SQLException e ) { throw e;}
    return result;
}
```

Thêm thông tin đặt phòng vào hệ thống:

```
public void bookRoom(Booking b)throws SQLException{
```

```
    String sql = INSERT INTO tblBooking(idRoom, idUser, startDate, endDate,
price, description) VALUES(?, ?, ?, ?, ?, ?)";
    try {
        PreparedStatement prstm = conn.prepareStatement(sql);
        prstm.setInt(1, b.getRoom().getId());
        prstm.setInt(2, b.getUser().getId());
        prstm.setDate(3, b.getStartDate());
        prstm.setDate(4, b.getEndDate());
        prstm.setFloat(5, b.getPrice());
        prstm.setString(6, b.getDescription());
        prstm.executeUpdate();
    } catch (SQLException e ) {
        throw e;
    }
}
```

Thống kê doanh thu:

```
public double totalIncomeByPeriod(Date sd, Date ed)throws SQLException{
    double result = 0;
    String sql = SELECT SUM(b.amount)FROM tblBooking a INNER JOIN tblBill
b ON b.idBooking = a.id WHERE  ((a.startDate BETWEEN ? AND ?) AND
(a.endDate BETWEEN ? AND ?))" ;
    try {
        PreparedStatement prstm = conn.prepareStatement(sql);
        prstm.setDate(1, sd); prstm.setDate(3, sd);
        prstm.setDate(2, ed); prstm.setDate(4, ed);
        ResultSet rs = prstm.executeQuery();
        if(rs.next()){
           result = rs.getDouble(1);
        }
    } catch (SQLException e ) {
        throw e;
    }
    return result;
}
```

3.4. KẾT LUẬN

Chương này đã trình bày cách thiết kế CSDL cho ứng dụng, nhắc lại cách sử dụng các loại câu lệnh SQL cơ bản để thao tác với CSDL. Và cách lập trình Java thao tác với CSDL. Các kiến trúc trong chương này là nền tảng cơ bản hỗ trợ cho người học tiếp tục các modul về lập trình mạng trong các chương tiếp theo.

3.5. BÀI TẬP

1. Thiết kế CSDL cho phần mềm quản lí bán vé cho một chuỗi rạp chiếu phim với mô tả như sau:

- Hãng có một chuỗi rạp chiếu phim (Mã rạp, tên rạp, địa chỉ, giới thiệu).
- Mỗi rạp chiếu phim có nhiều phòng chiếu khác nhau (Mã phòng chiếu, số lượng ghế, đặc điểm phòng chiếu)

- Mỗi phim (Mã phim, tên phim, loại phim, năm sản xuất, mô tả) có thể được chiếu tại nhiều phòng chiếu khác nhau vào nhiều thời điểm khác nhau
- Mỗi phòng chiếu có thể chiếu nhiều phim khác nhau tại nhiều thời điểm khác nhau
- Mỗi một thời điểm nhất định, trong một phòng chiếu chỉ có duy nhất một phim được chiếu.

2. Thiết kế CSDL cho phần mềm quản lí bán vé cho một chuỗi rạp chiếu phim như bài 1, có bổ sung phần bán vé như sau:

- Cùng một phim, chiếu tại cùng 1 phòng chiếu nhưng nếu ở các khung giờ và ngày khác nhau có thể có giá vé khác nhau.
- Mỗi khách hàng có thể mua nhiều vé của cùng suất chiếu và thanh toán 1 lần.
- Nhân viên chỉ bán vé cho khách hàng khi phòng chiếu tại giờ chiếu mà khách hàng yêu cầu vẫn còn đủ số lượng ghế trống cho khách hàng.
- Khách hàng có thể trả lại một số vé sau khi đã mua, và có thể phải chịu tiền phạt: trả trước 48h thì miễn phí, trả trước 24h thì mất phí 20%, trả trước 12h thì mất phí 40%, trả trước 6h thì mất phí 60%, trả sau 6h thì mất phí 100%, tính từ giờ khởi chiếu

3. Thiết kế CSDL cho phần mềm quản lí bán vé cho một chuỗi rạp chiếu phim như bài 2, có bổ sung phần khách hàng thân thiết như sau:

- Mỗi khách hàng có một thẻ khách hàng thân thiết có thể tích điểm.
- Mỗi lần mua vé có xuất thẻ thì khách hàng được cộng điểm theo tỉ lệ: cứ 10000VND thì được cộng thêm 1 điểm. Ví dụ nếu hóa đơn thanh toán 116000VND thì được cộng 11 điểm.
- Khi số điểm đạt ngưỡng nào đấy thì có thể đổi vé xem phim miễn phí. Ví dụ, cứ 200 điểm được đổi 1 vé xem phim 2D, 400 điểm được đổi 1 vé xem phim 3D. Nếu khách hàng có 280 điểm và mua 2 vé xem phim với tổng hóa đơn là 200000VND, nếu khách hàng thanh toán hết thì sẽ được cộng 20 điểm vào thẻ. Nếu khách hàng muốn đổi vé miễn phí thì sẽ được đổi 1 vé, khách hàng chỉ phải trả 100000VND cho vé còn lại, trong thẻ còn 80 điểm, và sau giao dịch này khách hàng chỉ được cộng số điểm bằng tỉ lệ phần tiền thanh toán: 10 điểm ứng với 100000VND.

4. Với CSDL cho phần mềm quản lí bán vé cho một chuỗi rạp chiếu phim như trước, viết các lệnh phương thức lớp DAO cho các chức năng sau:

- Thêm một phim mới vào CSDL
- Lên lịch chiếu phim trong một ngày cho một phòng chiếu
- Bán một số vé của một phim trong một phòng chiếu ở một khung giờ nào đó

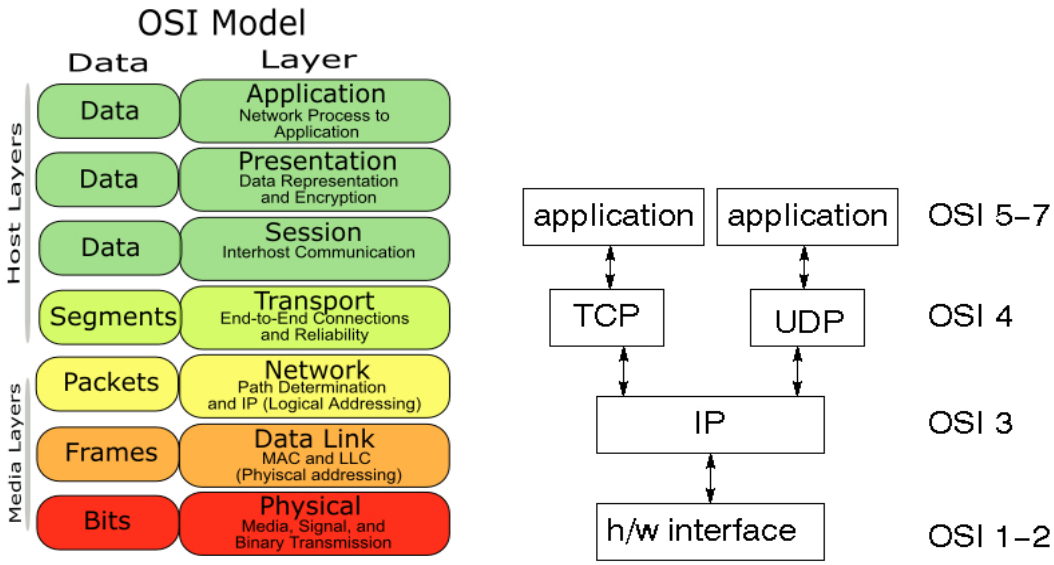
- Tìm kiếm các phim + phòng chiếu + giờ chiếu trong một khoảng thời gian nào đó mà đang còn ghế trống
- Thống kê các phim có danh thu cao nhất đến thấp nhất trong một khoảng thời gian nào đó
- Thống kê các khung giờ chiếu có đông khách hàng xem phim từ cao nhất đến thấp nhất
- Thống kê các khách hàng mua nhiều vé nhất đến ít vé nhất trong một khoảng thời gian nào đó

PHẦN II.
LẬP TRÌNH VỚI SOCKET

CHƯƠNG 4. LẬP TRÌNH VỚI GIAO THỨC TCP/IP

4.1. GIỚI THIỆU GIAO THỨC TCP/IP

Trong mô hình OSI 7 tầng (Physical, Data link, network, Transport, Session, Presentation, Application) thì giao thức UDP nằm ở tầng thứ 4 từ dưới lên, là tầng giao vận (Transport) (Hình 4.1).



Hình 4.1: Vị trí giao thức TCP/IP trong mô hình OSI 7 tầng

Đặc trưng của tầng giao vận là:

- Kết nối từ đầu cuối tới đầu cuối (End to end connections): các giao thức đảm bảo dữ liệu được truyền từ đầu cuối này tới đầu cuối kia chứ không chỉ quan tâm đến việc trung chuyển dữ liệu giữa các nút mạng như ở tầng thấp hơn.
- Tin cậy (Reliability): Các truyền là tin cậy vì có cơ chế kiểm tra lỗi và truyền lại.

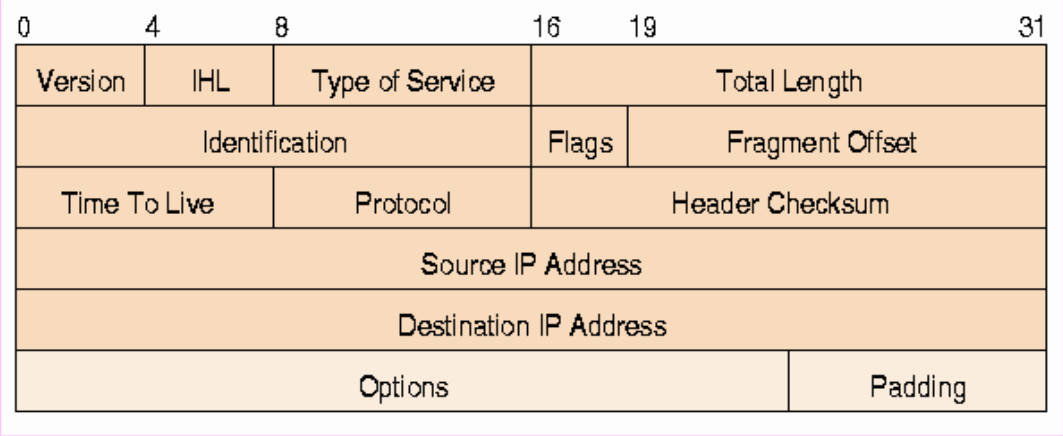
Tầng giao vận của mô hình OSI có 2 giao thức là TCP/IP và UDP. Đặc trưng khác biệt của TCP/IP là hướng kết nối (connection-oriented). Theo đó, việc truyền dữ liệu giữa hai đầu cuối được diễn ra theo theo hai bước:

- Bước 1: Tạo kênh kết nối. Bên gửi và bên nhận tạo ra một kênh riêng nối từ bên gửi đến tận bên nhận. Kênh này là kênh riêng vì không có ai được quyền nhảy vào truyền dữ liệu trên kênh này. Ngay cả khi hai bên không có dữ liệu trao đổi trên kênh thì kênh này vẫn tồn tại.
- Bước 2: Truyền dữ liệu. Dữ liệu được đóng gói vào các gói tin IP và truyền đi theo kênh riêng đã tạo được trên mạng.

Một gói tin IP có các trường thông tin như sau (Hình 4.2):

- Phiên bản của giao thức IP (version): phiên bản IPv4 hoặc Ipv6

- Chiều dài của header (IP header length - IHL): độ dài phần header.
- Kiểu dịch vụ (Type of service).
- Tổng chiều dài gói tin (total length): chiều dài bao gồm header và dữ liệu.
- Định dnah người gửi và nhận (identification): tên hay bí danh.
- Cờ (flags).
- Số thứ tự gói tin truyền (Fragment offset): mô tả thứ tự của gói tin được truyền ở bên gửi.
- Thời gian sống (Time to live): sau khoảng thời gian này, gói tịn bị hủy trên mạng.
- Giao thức (protocol).
- Mã kiểm tra header (header checksum): mã kiểm tra lỗi phần header.
- Địa chỉ IP của người gửi (source IP): địa chỉ IP (hoặc hostname) của người gửi.
- Địa chỉ IP của người nhận (destination IP): ịa chỉ IP (hoặc hostname) của người nhận.
- Các tùy chọn khác (options).
- Dự phòng (Padding).
- Dữ liệu (Data): là vùng chứa dữ liệu của gói tin gửi đi.



Hình 4.2: Cấu trúc header gói tin IP (IPv4)

Trong lập trình với giao thức TCP/IP, khi đóng gói gói tin IP, ta chỉ cần tập trung vào một số trường thông tin chủ yếu như: địa chỉ IP người gửi, địa chỉ IP người nhận, dữ liệu.

4.2. MỘT SỐ LỚP JAVA HỖ TRỢ GIAO THỨC TCP/IP

4.2.1. Lớp InetAddress

Java có các lớp quan trọng để thao tác với địa chỉ IP trong gói java.net. Lớp quan trọng nhất là lớp InetAddress. Lớp này cho phép lấy địa chỉ của một máy trạm bất kỳ trên mạng và cho phép dễ dàng hoán chuyển giữa địa chỉ IP và tên của một máy trạm(host). Mỗi đối tượng InetAddress chứa 2 thành phần chính của một máy trạm là hostname và địa chỉ IP của máy trạm đó. Ngoài ra

còn có 2 lớp khác kết thừa trực tiếp từ lớp InetAddress dành cho các phiên bản IPv4 và IPv6 là lớp Inet4Address, Inet6Address và 2 lớp khác là lớp SocketAddress , InetSocketAddress liên quan tới địa chỉ socket .

Hình 4.3. Lớp kế thừa từ lớp InetAddress và SocketAddress

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

public class InetAddress extends Object implements Serializable

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

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

	Trả về địa chỉ IP chứa trong đối tượng INetAddress là chuỗi dạng a.b.c.d
String	getHostName() Trả về tên máy trạm chưa trong đối tượng
static InetAddress	getLocalHost() Lấy đối tượng InetAddress của máy cục bộ
int	hashCode() Trả về hashcode của địa chỉ IP cục thể
boolean	isAnyLocalAddress() Kiểm tra địa chỉ InetAddress có phải địa chỉ wildcard không?
boolean	isLinkLocalAddress() Kiểm tra địa chỉ có phải là một địa chỉ link-local hay không.
boolean	isLoopbackAddress() Kiểm tra địa chỉ có phải là địa chỉ Loopback không.
boolean	isMCGlobal() Kiểm tra địa chỉ multicast có phạm vi toàn cục hay không?
boolean	isMCLinkLocal() Kiểm tra địa chỉ multicast có phải là địa chỉ có phạm vi liên kết hay không?
boolean	isMCNodeLocal() Kiểm tra địa chỉ multicast có phải là địa chỉ phạm vi nút mạng hay không?
boolean	isMulticastAddress() Kiểm tra địa chỉ InetAddress có phải là địa chỉ IP multicast hay không.
String	toString() Chuyển địa chỉ IP thành chuỗi.

4.2.2. Lớp Socket

Lớp Socket dùng để tạo đối tượng socket cho phép truyền thông với giao thức TCP.

a. Các hàm khởi tạo

public Socket(String host, int port) throws UnknownHostException, IOException

Hàm khởi tạo này cho phép tạo ra đối tượng Socket truyền thông với giao thức TCP và thực hiện kết nối với máy trạm từ xa có địa chỉ và số cổng được chỉ ra bởi tham số host và port tương ứng. Tham số host có thể là tên máy trạm, tên miền hoặc địa chỉ IP. Nếu không tìm thấy máy trạm từ xa hoặc đối tượng Socket không được mở thì nó ném trả về ngoại lệ *UnknownHostException* hoặc *IOException.* Ví dụ đoạn chương trình sau cho phép mở socket và kết nối tới máy trạm từ xa có tên miền www.yahoo.com và số cổng là 80.

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

public Socket(InetAddress host, int port) throws IOException

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

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

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

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

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

protected Socket( )

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

b. Một số phương thức quan trọng của lớp Socket

- *public InetAddress getInetAddress( ):* Phương thức cho phép trả về địa chỉ của máy trạm từ xa hiện đang kết nối với socket.
- *public int getPort( ):* Trả về số cổng trên máy trạm từ xa mà hiện đang kết nối với socket.

- *public int getLocalPort( )*: Trả về số cổng trên máy cục bộ
- *public InputStream getInputStream( ) throws IOException:* Trả về luồng nhập của socket là đối tượng InputStream.
- *public OutputStream getOutputStream( ) throws IOException*: Trả về luồng xuất của socket là đối tượng OutputStream.
- *public void close( ) throws IOException*: Đóng socket

c. Thiết lập các tuỳ chọn Socket

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

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

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

public void setTcpNoDelay(boolean on) throws SocketException

public boolean getTcpNoDelay( ) throws SocketException

4.2.3. Lớp ServerSocket

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

a. Các hàm khởi tạo

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

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

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

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

- *public ServerSocket( ) throws IOException*

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

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

b. Phương thức

- Phương thức accept()

Phương thức này có cú pháp sau:

public Socket accept( ) throws IOException

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

```
  ServerSocket server = new ServerSocket(5776);
  while (true) {
    Socket connection = server.accept( );
    OutputStreamWriter out
     = new OutputStreamWriter(connection.getOutputStream( ));
    out.write("You've connected to this server. Bye-bye now.\r\n");
  connection.close( );
}
```

- Phương thức close()

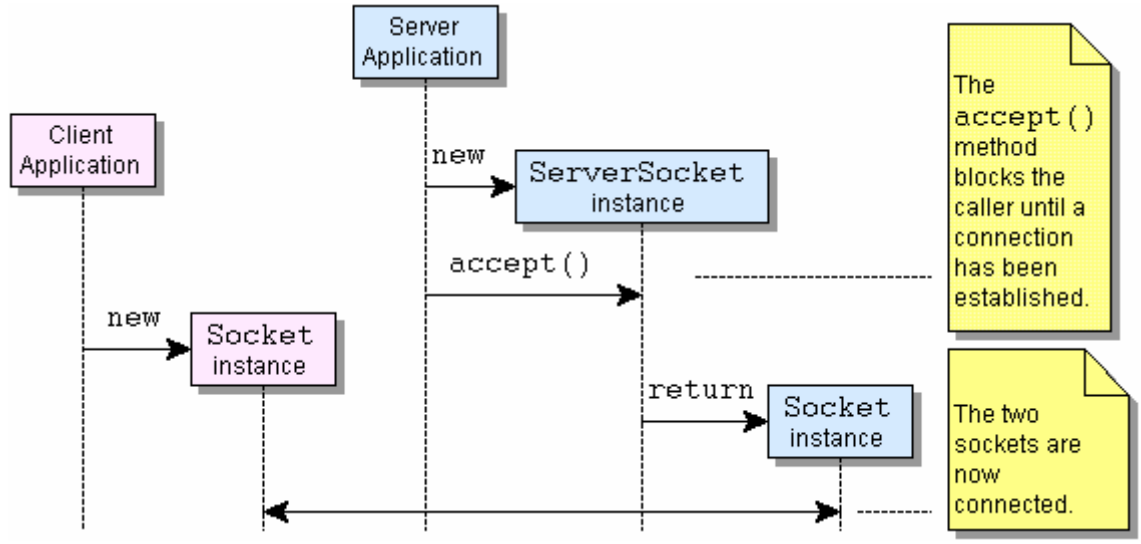
Phương thức close() có cú pháp sau:

public void close( ) throws IOException

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

4.3. LẬP TRÌNH ỨNG DỤNG MẠNG VỚI TCPSOCKET

Trong chương trình ứng dụng mạng xây dựng theo mô hình client/server, để chương trình client và chương trình server có thể truyền thông được với nhau thì mỗi phía phải thực hiện tối thiểu các thao tác cơ bản sau đây (Hình 4.4):



Hình 4.4. Quá trình khởi tạo truyền thông với TCPSocket

4.3.1. Chương trình phía server

- Tạo đối tượng ServerSocket với một số hiệu cổng xác định

```
ServerSocket myServer = new ServerSocket(số cổng);
```

- Đặt đối tượng ServerSocket ở trạng thái nghe tín hiệu đến kết nối bằng phương thức accept(). Nếu có tín hiệu đến kết nối phương thức accept() tạo ra đối tượng Socket mới để phục vụ kết nối đó.

```
Socket clientSocket = myServer.accept();
```

- Khai báo luồng nhập/xuất cho đối tượng Socket mới(tạo ra ở bước trên). Luồng nhập/xuất có thể là luồng kiểu byte hoặc kiểu char.

```
DataInputStream is = new
   DataInputStream(clientSocket.getInputStream());
PrintStream os = new
   PrintStream(clientSocket.getOutputStream());
```

- Thực hiện truyền dữ liệu với client thông qua luồng nhập/xuất

```
String input = is.read();
os.println(dữ liệu trả về);
```

- Server hoặc client hoặc cả 2 đóng kết nối

```
myServer.close();
```

- Server trở về bước 2 và đợi kết nối tiếp theo.

4.3.2. Chương trình client

- Tạo đối tượng Socket và thiết lập kết nối tới server bằng cách chỉ ra các tham số của server.

```
Socket mySocket = new Socket(tên máy chủ, số cổng);
```

- Khai báo luồng nhập/xuất cho Socket. Luồng nhập/xuất có thể là luồng kiểu byte hoặc kiểu char.

```
DataOutputStream os = new
   DataOutputStream(mySocket.getOutputStream());
DataInputStream is = new
   DataInputStream(mySocket.getInputStream());
```

- Thực hiện truyền dữ liệu qua mạng thông qua luồng nhập/xuất

```
os.writeBytes(dữ liệu gửi đi);
String responseStr = is.read();
```

- Đóng Socket, giải phóng các tài nguyên khác, kết thúc chương trình nếu cần.

```
mySocket.close();
```

Lưu ý:

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

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

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

- **Luồng kiểu byte**

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

- Với luồng nhập:

+ Tạo luồng nhập cho socket:

InputStream inp=cl.getInputStream();

+ Đọc dữ liệu: Có ba cách

-/ Đọc mỗi lần một byte: *inp.read()*

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

byte b=new byte[1024];

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

- Với luồng xuất:

+Tạo luồng xuất:

OutputStream outp=cl.getOutputStream();

+ Viết dữ liệu:

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

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

//byte[] b;

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

- **Luồng kiểu char:**

- Với luồng nhập:

+Tạo luồng nhập:

BufferedReader inp=new BuferedReader(

new InputStreamReader(cl.getInputStream()));

+ Đọc dữ liệu:

-/Đọc từng ký tự: *int ch=inp.read()*

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

- Với luồng xuất:

+ Tạo luồng xuất:

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

+ Viết dữ liệu:

outp.println(<data>);

4.3.4. Một số ví dụ

a. Chương trình quét cổng sử dụng Socket

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

```
            }
        }  // end main
}  // end PortScanner
```

b. Chương trình quét cổng cục bộ dùng lớp ServerSocket

```
import java.net.*;
import java.io.*;
public class LocalPortScanner {
        public static void main(String[] args) {
                for (int port = 1; port <= 65535; port++) {
                        try {
                                // the next line will fail and drop into the catch block if
                                // there is already a server running on the port
                                ServerSocket server = new ServerSocket(port);
                        }catch (IOException ex) {
                                System.out.println("There is a server on port "
                                        + port + ".");
                        } // end catch
                } // end for
        }
}
```

c. Chương trình finger client

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

```
import java.net.*;
import java.io.*;
public class FingerClient {
        public final static int DEFAULT_PORT = 79;
        public static void main(String[] args) {
                String hostname = "localhost";
                try {
                        hostname = args[0];
                }catch (ArrayIndexOutOfBoundsException ex) {
                        hostname = "localhost";
                }
                Socket connection = null;
                try {
                        connection = new Socket(hostname, DEFAULT_PORT);
                        Writer out = new OutputStreamWriter(
                                        connection.getOutputStream( ), "8859_1");
                        for (int i = 1; i < args.length; i++) out.write(args[i] + " ");
                        out.write("\r\n");
                        out.flush( );
                        InputStream raw = connection.getInputStream( );
                        BufferedInputStream buffer = new BufferedInputStream(raw);
                        InputStreamReader in = new InputStreamReader(buffer, "8859_1");
                        int c;
                        while ((c = in.read( )) != -1) {
                                //non-printable and non-ASCII as recommended by RFC 1288
                                if ((c >= 32 && c < 127) || c == '\t' || c == '\r'
                                        || c == '\n') {
                                        System.out.write(c);
                                }
```

```
                }
            }catch (IOException ex) {
                System.err.println(ex);
            }finally {
                try {
                    if (connection != null) connection.close( );
                }catch (IOException ex) {}
            }
        }
}
```

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

//TimeClient.java

```
import java.net.*;
import java.io.*;
import java.util.*;
public class TimeClient {
    public final static int    DEFAULT_PORT = 37;
    public final static String DEFAULT_HOST = "time.nist.gov";
    public static void main(String[] args) {
        String hostname = DEFAULT_HOST ;
        int port = DEFAULT_PORT;
        if (args.length > 0) {
            hostname = args[0];
        }
        if (args.length > 1) {
            try {
                port = Integer.parseInt(args[1]);
            }catch (NumberFormatException ex) {
                // Stay with the default port
            }
        }
        // The time protocol sets the epoch at 1900,
        // the Java Date class at 1970. This number
        // converts between them.
        long differenceBetweenEpochs = 2208988800L;
        // If you'd rather not use the magic number, uncomment
        // the following section which calculates it directly.

        InputStream raw = null;
        try {
            Socket theSocket = new Socket(hostname, port);
            raw = theSocket.getInputStream( );
            long secondsSince1900 = 0;
            for (int i = 0; i < 4; i++) {
                secondsSince1900 = (secondsSince1900 << 8) | raw.read( );
            }
            long secondsSince1970
            = secondsSince1900 - differenceBetweenEpochs;
            long msSince1970 = secondsSince1970 * 1000;
            Date time = new Date(msSince1970);
            System.out.println("It is " + time + " at " + hostname);
        } catch (UnknownHostException ex) {
            System.err.println(ex);
        }catch (IOException ex) {
            System.err.println(ex);
```

```
            }finally {
                  try {
                        if (raw != null) raw.close( );
                  }catch (IOException ex) {}
            }
      }  // end main
} // end TimeClient
```

//TimeServe.java

```
import java.net.*;
import java.io.*;
import java.util.Date;
public class TimeServer {
      public final static int DEFAULT_PORT = 37;
      public static void main(String[] args) {
            int port = DEFAULT_PORT;
            if (args.length > 0) {
                  try {
                        port = Integer.parseInt(args[0]);
                        if (port < 0 || port >= 65536) {
                              System.out.println("Port must between 0 and 65535");
                              return;
                        }
                  } catch (NumberFormatException ex) {}
            }
            // The time protocol sets the epoch at 1900,
            // the Date class at 1970. This number
            // converts between them.
            long differenceBetweenEpochs = 2208988800L;
            try {
                  ServerSocket server = new ServerSocket(port);
                  while (true) {
                        Socket connection = null;
                        try {
                              connection = server.accept( );
                              OutputStream out = connection.getOutputStream( );
                              Date now = new Date( );
                              long msSince1970 = now.getTime( );
                              long secondsSince1970 = msSince1970/1000;
                              long secondsSince1900 = secondsSince1970
                                          + differenceBetweenEpochs;
                              byte[] time = new byte[4];
                              time[0]= (byte) ((secondsSince1900 &
                                    0x00000000FF000000L) >> 24);
                              time[1] = (byte) ((secondsSince1900 &
                                    0x0000000000FF0000L) >> 16);
                              time[2] = (byte) ((secondsSince1900 &
                                    0x000000000000FF00L) >> 8);
                              time[3] = (byte) (secondsSince1900 &
                                    0x00000000000000FFL);
                              out.write(time);
                              out.flush( );
                        } catch (IOException ex) {
                        } finally {
                              if (connection != null) connection.close( );
```

```
                        }
                    }  // end while
                } catch (IOException ex) {
                    System.err.println(ex);
                } // end catch
        } // end main
} // end TimeServer
```

4.4. CASE STUDY: LOGIN TỪ XA DÙNG GIAO THỨC TCP/IP

4.4.1 Bài toán

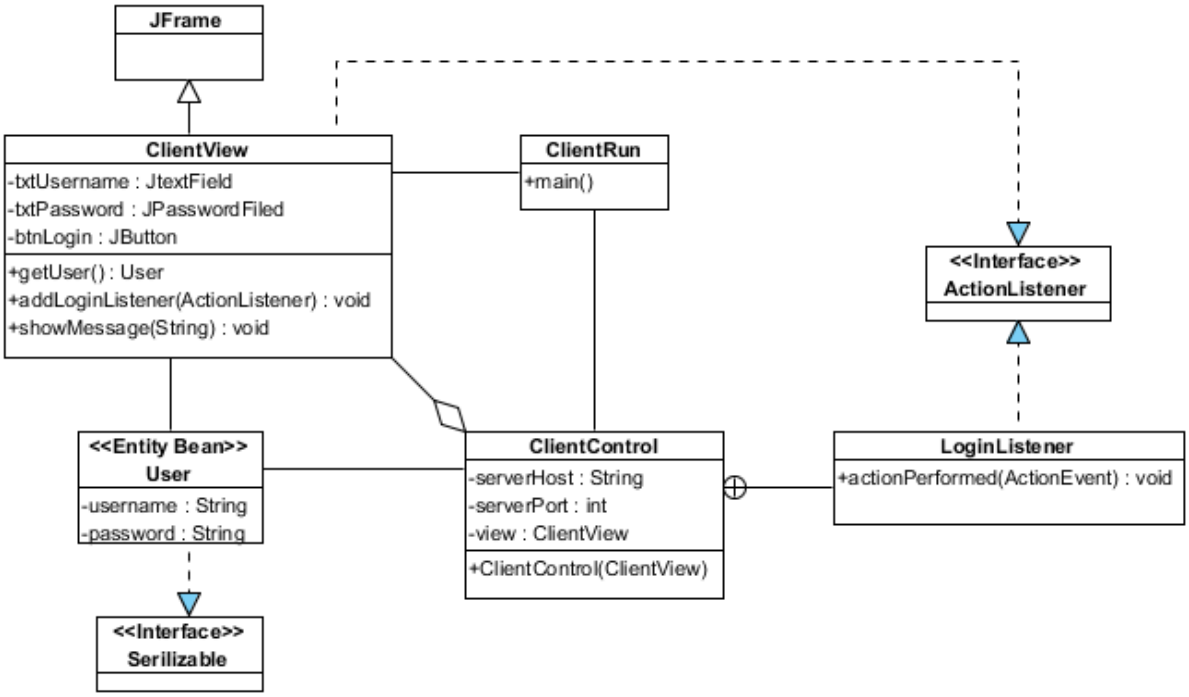
Bài toán login từ xa dùng giao thức TCP/IP đặt ra như sau:

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

4.4.2 Kiến trúc hệ thống theo mô hình MVC

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

a. Sơ đồ lớp phía client



Hình 4.5: Sơ đồ lớp phía client TCP/IP

Sơ đồ lớp của phía client được thiết kế theo mô hình MVC trong Hình 4.5, bao gồm 3 lớp chính tương ứng với sơ đồ M-V-C như sau:

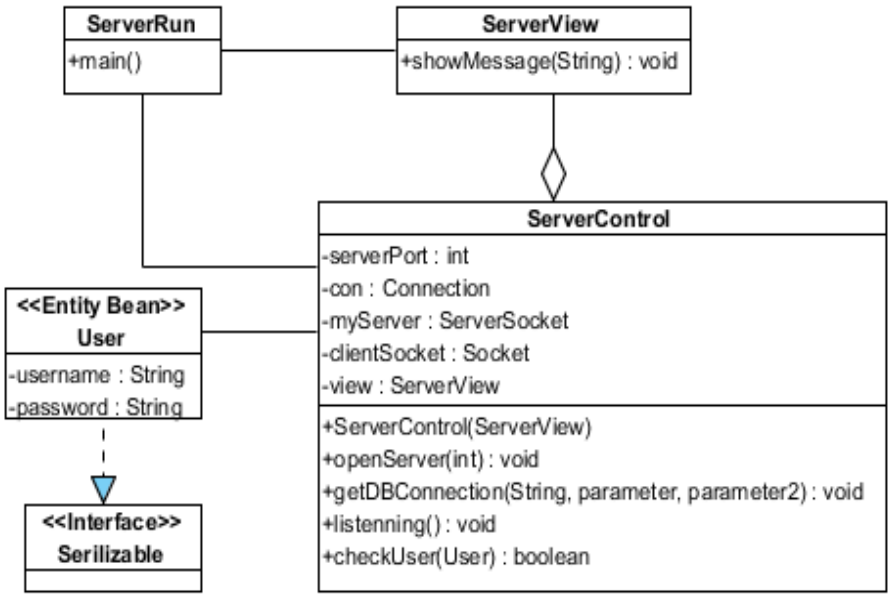
- Lớp User: là lớp tương ứng với thành phần model (M), bao gồm hai thuộc tính username và password, các hàm khởi tạo và các cặp getter/setter tương ứng với các thuộc tính.
- Lớp ClientView: là lớp tương ứng với thành phần view (V), là lớp form nên phải kế thừa từ lớp JFrame của Java, nó chứa các thuộc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

b. Sơ đồ lớp phía server

Sơ đồ lớp của phía server được thiết kế theo mô hình MVC trong Hình 4.6, bao gồm 3 lớp chính tương ứng với sơ đồ M-V-C như sau:

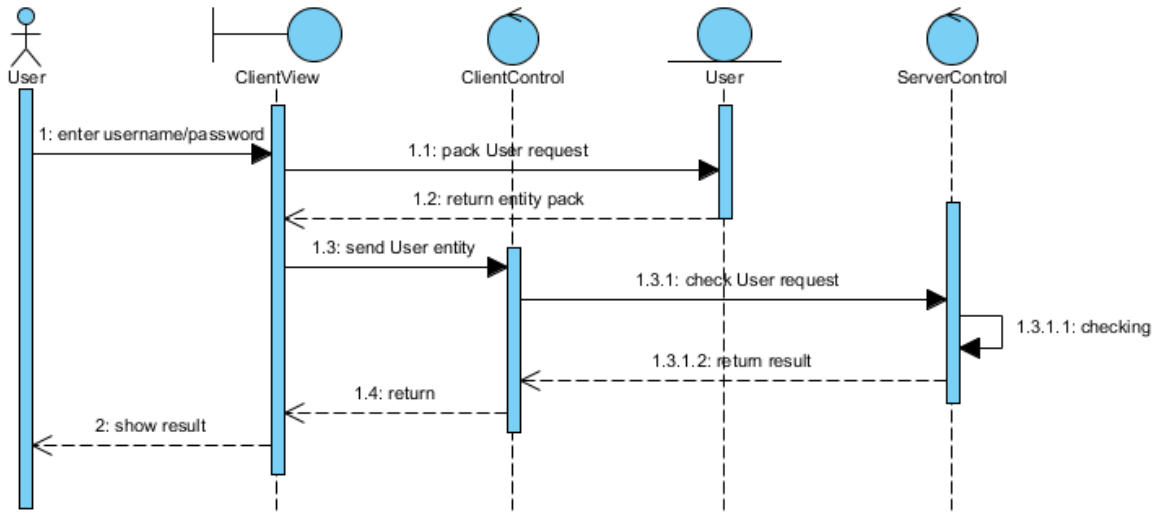
- Lớp User: là lớp thực thể, dùng chung thống nhất với lớp phía bên client.
- Lớp ServerView: là lớp tương ứng với thành phần view (V), là lớp dùng hiển thị các thông báo và trạng thái hoạt động bên server TCP.

- Lớp ServerControl: là lớp tương ứng với thành phần control (C), nó đảm nhiệm vai trò xử lí của server TCP, bao gồm: nhận thông tin đăng nhập từ phía các client, kiểm tra trong cơ sở dữ liệu xem các thng tin này đúng hay sai, sau đó gửi kết quả đăng nhập về cho client tương ứng.



Hình 4.6: Sơ đồ lớp phía server TCP/IP

c. Tuần tự các bước thực hiện



Hình 4.7: Tuần tự các bước thực hiện theo giao thức TCP/IP

Tuần tự các bước xử lí như sau (Hình 4.7):

1. Ở phía client, người dùng nhập username/password và click vào giao diện của lớp ClientView

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

4.4.3 Cài đặt

a. Các lớp phía client

User.java

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

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

	public User(){
	}

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

	public String getPassword() {
		return password;
	}

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

	public String getUserName() {
		return userName;
	}

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

ClientView.java

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

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

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

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

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

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

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

	public void actionPerformed(ActionEvent e) {
	}

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

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

```
	}

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

ClientControl.java

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

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

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

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

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

ClientRun.java

```
package tcp.client;
```

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

b. Các lớp phía server

ServerView.java

```
package tcp.server;

public class ServerView {
	public ServerView(){
	}

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

ServerControl.java

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

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

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

		while(true){
			listenning();
		}
	}

	private void getDBConnection(String dbName,
```

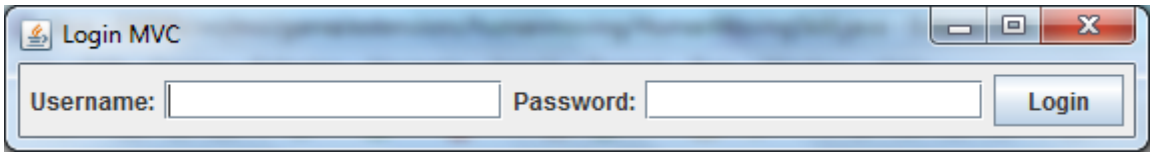
```
                                        String username, String password){
        String dbUrl = "jdbc:mysql://localhost:3306/" + dbName;
        String dbClass = "com.mysql.jdbc.Driver";

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

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

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

                  Object o = ois.readObject();
                  if(o instanceof User){
                         User user = (User)o;
                         if(checkUser(user)){
                                oos.writeObject("ok");
                         }
                         else
                                oos.writeObject("false");
                  }
           }catch (Exception e) {
                  view.showMessage(e.toString());
           }
    }


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

```
        }
}
```

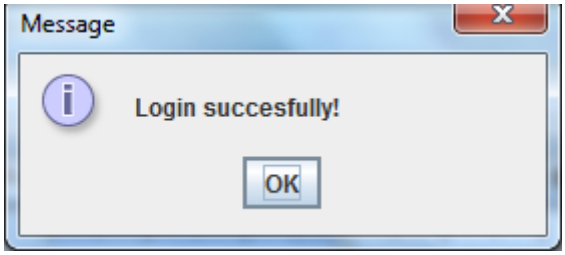
ServerRun.java

```
package tcp.server;

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

4.4.5 Kết quả



Login thành công:



Login lỗi:

4.5. KẾT LUẬN

Nội dung chương này đã nhắc lại một số kiến thức cơ bản về giao thức TCP/IP, Các lớp của Java cung cấp và hỗ trợ lập trình với giao thức TCP/IP, các bước lập trình với giao thức TCP/IP cho phía server và phía client. Chương này cũng trình bày một số ví dụ áp dụng với giao thức TCP/IP và case study lập trình giao thức TCP/IP theo mô hình MVC cho cả hai phía client và server.

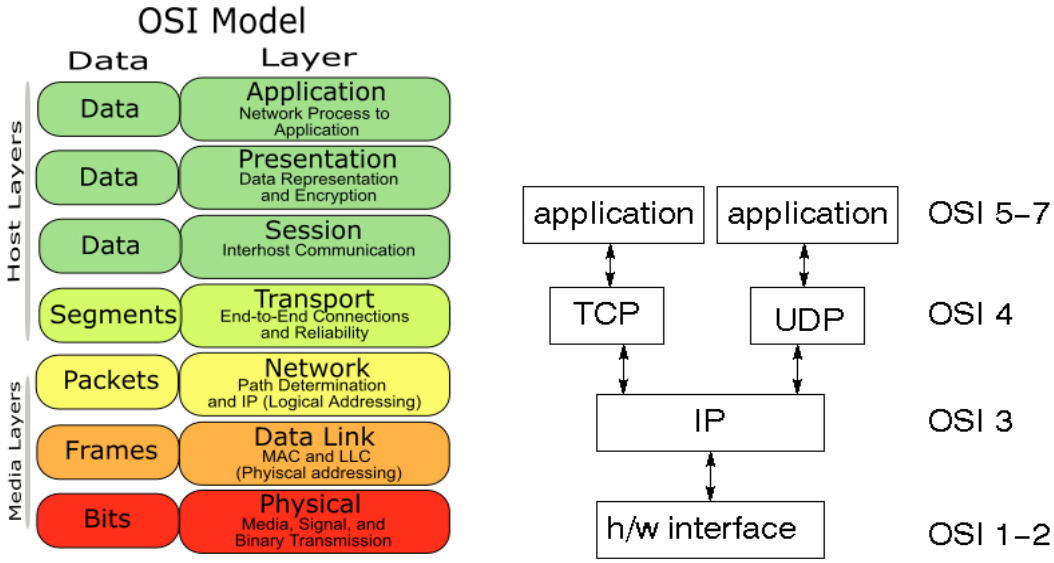
4.6. BÀI TẬP

1. Viết chương trình copy file từ máy chủ server về máy client dùng giao thức TCP/IP.
2. Viết chương trình giải phương trình bậc 2 theo đúng mô hình giao thức TCP/IP: client chỉ hiện giao diện nhập hệ số, sau đó chuyển lên server tính toán, kết quả lại được trả về cho client hiển thị.
3. Viết chương trình giải hệ phương trình bậc nhất theo đúng mô hình giao thức TCP/IP: client chỉ hiện giao diện nhập hệ số, sau đó chuyển lên server tính toán, kết quả lại được trả về cho client hiển thị.
4. Viết chương trình tìm USCLN (BSCNN) của 2 số nguyên dương a và b theo đúng mô hình giao thức TCP/IP: client chỉ hiện giao diện nhập số, sau đó chuyển lên server tính toán, kết quả lại được trả về cho client hiển thị.
5. Viết chương trình phân tích một số nguyên dương ra thừa số nguyên tố (ví dụ: 300 = 2*2*3*5*5) theo đúng mô hình giao thức TCP/IP: client chỉ hiện giao diện nhập số, sau đó chuyển lên server tính toán, kết quả lại được trả về cho client hiển thị.
6. Viết chương trình ứng dụng chát online dùng giao thức TCP/IP.

CHƯƠNG 5. LẬP TRÌNH VỚI GIAO THỨC UDP

5.1. GIỚI THIỆU GIAO THỨC UDP

Trong mô hình OSI 7 tầng (Physical, Data link, network, Transport, Session, Presentation, Application) thì giao thức UDP nằm ở tầng thứ 4 từ dưới lên, là tầng giao vận (Transport) (Hình 5.1).



Hình 5.1: Vị trí giao thức UDP trong mô hình OSI 7 tầng

Đặc trưng của tầng giao vận là:

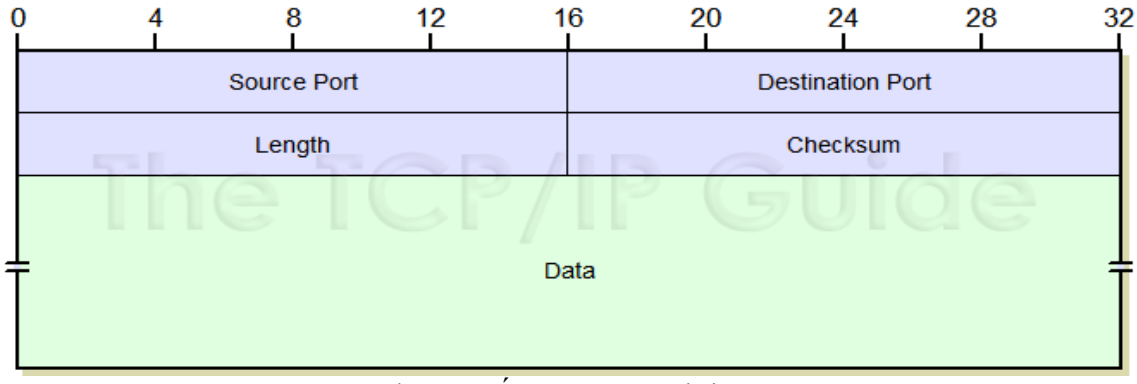
- Kết nối từ đầu cuối tới đầu cuối (End to end connections): các giao thức đảm bảo dữ liệu được truyền từ đầu cuối này tới đầu cuối kia chứ không chỉ quan tâm đến việc trung chuyển dữ liệu giữa các nút mạng như ở tầng thấp hơn.
- Tin cậy (Reliability): Các truyền là tin cậy vì có cơ chế kiểm tra lỗi (đối với TCP/IP) và cơ chế truyền lại (đối với UDP)

Tầng giao vận của mô hình OSI có 2 giao thức là TCP/IP và UDP. Đặc trưng khác biệt của UDP là không kết nối (truyền tin theo gói, không theo kênh như TCP/IP): Dữ liệu được đóng gói vào các gói tin UDP và truyền đi trên mạng. Tại các nút mạng, các bộ định tuyến đọc xem nếu gói tin gửi cho mình thì nhận, nếu không thì để gói tin di chuyển tiếp.

Một gói tin UDP có các trường thông tin như sau (Hình 5.2):

- Số cổng của người gửi (source port): một số nguyên dương là số hiệu cổng của người gửi.
- Số cổng của người nhận (destination port): cũng là một số nguyên dương.
- Chiều dài của gói tin (length): tính cả header lẫn dữ liệu
- Mã kiểm tra lỗi (Checksum): dùng kiểm tra phát hiện lỗi trong cả phần header lẫn dữ liệu.

- Dữ liệu (Data): là vùng chứa dữ liệu của gói tin gửi đi. Thường dữ liệu là một mảng các byte. Nếu dữ liệu là các dạng khác mảng các byte thì phải được chuyển về định dạng mảng các byte trước khi đóng gói vào gói tin UDP.



Hình 5.2: Cấu trúc một gói tin UDP.

5.2. MỘT SỐ LỚP JAVA HỖ TRỢ LẬP TRÌNH VỚI UDP

5.2.1. Lớp DatagramPacket

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

public final class DatagramPacket extends Object

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

a. Hàm khởi tạo

Lớp này có các hàm khởi tạo tạo gói tin gửi và gói tin nhận khác nhau:

* Hàm khởi tạo tạo gói tin nhận từ mạng:

public DatagramPacket(byte[] inBuffer, int length)

Tham số:

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

Ví dụ tạo gói tin nhận:

byte[] inBuff=new byte[512];//bộ đệm nhập

DatagramPacket inData=new DatagramPacket(inBuf, inBuff.length);

* Hàm khởi tạo tạo gói tinh gửi:

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

Tham số:

- outBuffer: Bộ đệm xuất chưa dữ liệu của gói tin gửi
- length: kích cỡ dữ liệu của gói tin gửi tính theo số byte và thường bằng outBuffer.length.
- destination: Địa chỉ nơi nhận gói tin
- port: Số hiệu cổng đích, nơi nhận gói tin.

Ví dụ:

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

b. Phương thức

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

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

5.2.2. Lớp DatagramSocket

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

public class DatagramSocket extends Object

a. Các phương thức khởi tạo

- *public DatagramSocket( ) throws SocketException:*

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

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

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

b. Các phương thức

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

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

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

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

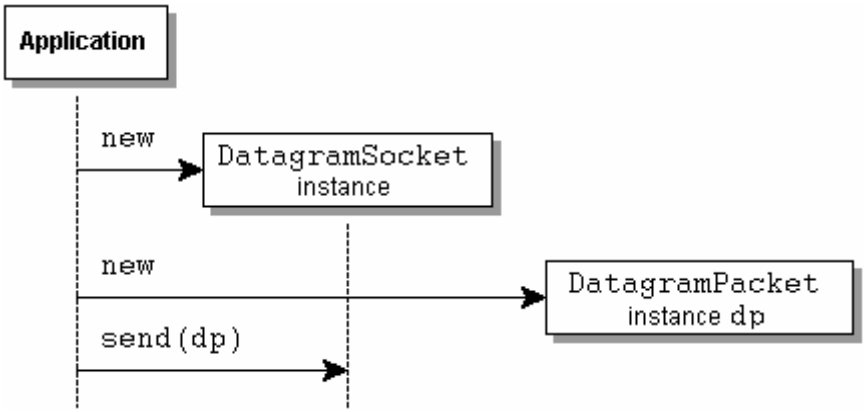
- *public void close( ):* Phương thức đóng socket.

Các phương thức khác thể hiện trong bảng sau:

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

5.3. KỸ THUẬT LẬP TRÌNH TRUYỀN THÔNG VỚI GIAO THỨC UDP

Trong mô hình client/server, để chương trình client và server có thể truyền thông được với nhau, mỗi phía phải thực hiện một số thao tác cơ bản sau đây (Hình 5.3)



Hình 5.3. Quá trình khởi tạo truyền thông UDPSocket

5.3.1. Phía server

- Tạo đối tượng DatagramSocket với số cổng xác định được chỉ ra

```
DatagramSocket myServer = new DatagramSocket(port);
```

- Khai báo bộ đệm nhập /xuất inBuffer/outBuffer dạng mảng kiểu byte

```
byte[] receiveData = new byte[1024];
```

- Khai báo gói tin nhận là đối tượng DatagramPacket.

```
DatagramPacket receivePacket = new
 DatagramPacket(receiveData, receiveData.length);
```

- Thực hiện nhận gói tin với phương thức receive()

```
myServer.receive(receivePacket);
```

- Bóc tách dữ liệu từ gói tin nhận được để xử lí

```
input = new String(receivePacket.getData());
```

- Sau khi xử lí xong, đóng gói một gói tin mới để trả lời (trả kết quả xử lí)

```
InetAddress IPAddress = receivePacket.getAddress();
int port = receivePacket.getPort();
byte[] sendData = (dữ liệu đã xử lí).getBytes();
DatagramPacket sendPacket =
new DatagramPacket(sendData, sendData.length, IPAddress, port);
```

- Gửi trả kết quả về máy client

```
myServer.send(sendPacket);
```

- Đóng socket, giải phóng các tài nguyên khác, kết thúc chương trình nếu cần, không quay về bước 3.

5.3.2. Phía client

- Tạo đối tượng DatagramSocket với số cổng nào đó

```
DatagramSocket myClient = new DatagramSocket(port);
```

- Khai báo gói tin gửi/nhận outData/inData là đối tượng DatagramPacket.

```
InetAddress IPAddress = InetAddress.getByName("localhost");
sendData = (dữ liệu gửi).getBytes();
```

```
DatagramPacket sendPacket = new DatagramPacket(sendData,
                sendData.length, IPAddress, số cổng);
```

- Thực hiện gửi gói tin với phương thức send()

```
myClient.send(sendPacket);
```

- Khai báo bộ đệm xuất/nhập outBuffer/inBuffer dạng mảng kiểu byte

```
byte[] receiveData = new byte[1024];
```

- Khai báo gói tin nhận là đối tượng DatagramPacket.

```
DatagramPacket receivePacket = new
DatagramPacket(receiveData, receiveData.length);
```

- Thực hiện nhận gói tin với phương thức receive()

```
myClient.receive(receivePacket);
```

- Bóc tách dữ liệu từ gói tin nhận được để xử lí

```
input = new String(receivePacket.getData());
```

- Đóng socket, giải phóng các tài nguyên khác, kết thúc chương trình nếu cần, không quay về bước 3.

Một số lưu ý:

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

5.3.3. Một số chương trình ví dụ

a. Chương trình minh hoạ

```
//UDPEchoClient.java
import java.net.*;
import java.io.*;
public class UDPEchoClient {
	public final static int DEFAULT_PORT = 7;
	public static void main(String[] args) {
		String hostname = "localhost";
		int port = DEFAULT_PORT;
		if (args.length > 0) {
			hostname = args[0];
		}
		try {
			InetAddress ia = InetAddress.getByName(hostname);
			Thread sender = new SenderThread(ia, DEFAULT_PORT);
			sender.start( );
			Thread receiver = new ReceiverThread(sender.getSocket( ));
			receiver.start( );
		}catch (UnknownHostException ex) {
			System.err.println(ex);
		}catch (SocketException ex) {
			System.err.println(ex);
		}
```

```
        }  // end main
}

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

5.4. CASE STUDY: LOGIN TỪ XA DÙNG UDP

5.4.1 Bài toán

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

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

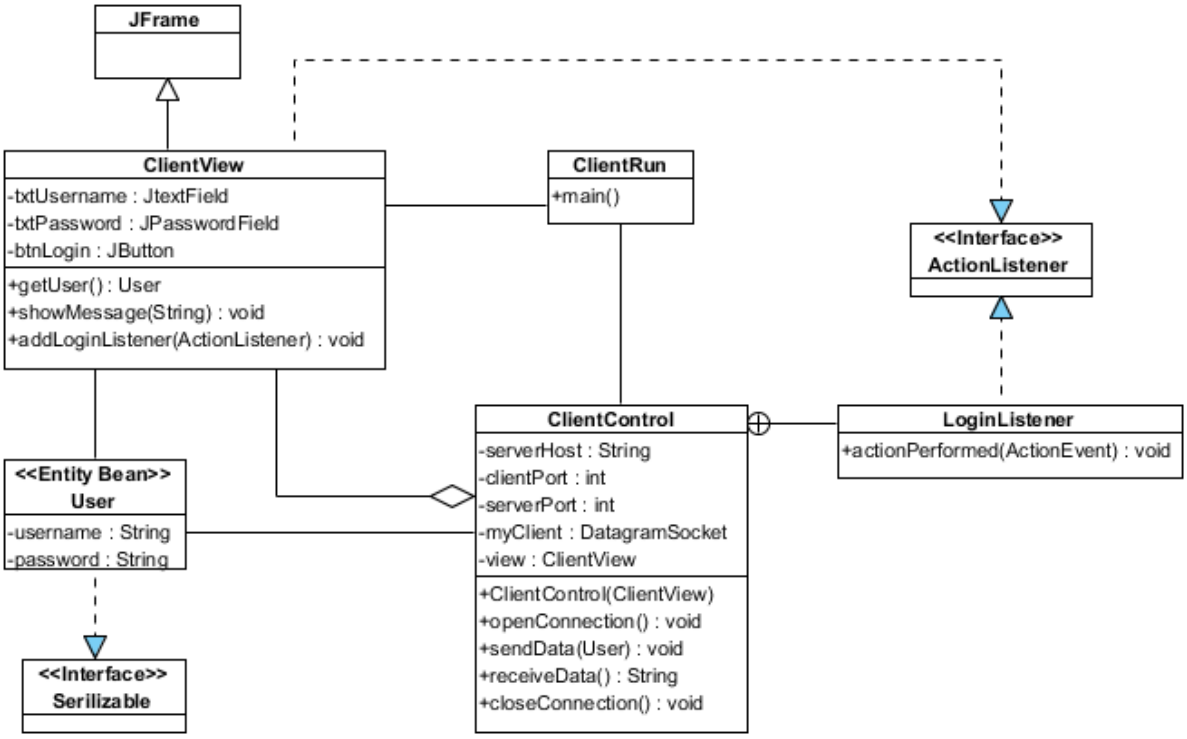
báo login thành công. Nếu đăng nhập sai thì thông báo là username/password không đúng.

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

5.4.2 Kiến trúc hệ thống theo mô hình MVC

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

a. Sơ đồ lớp phía client



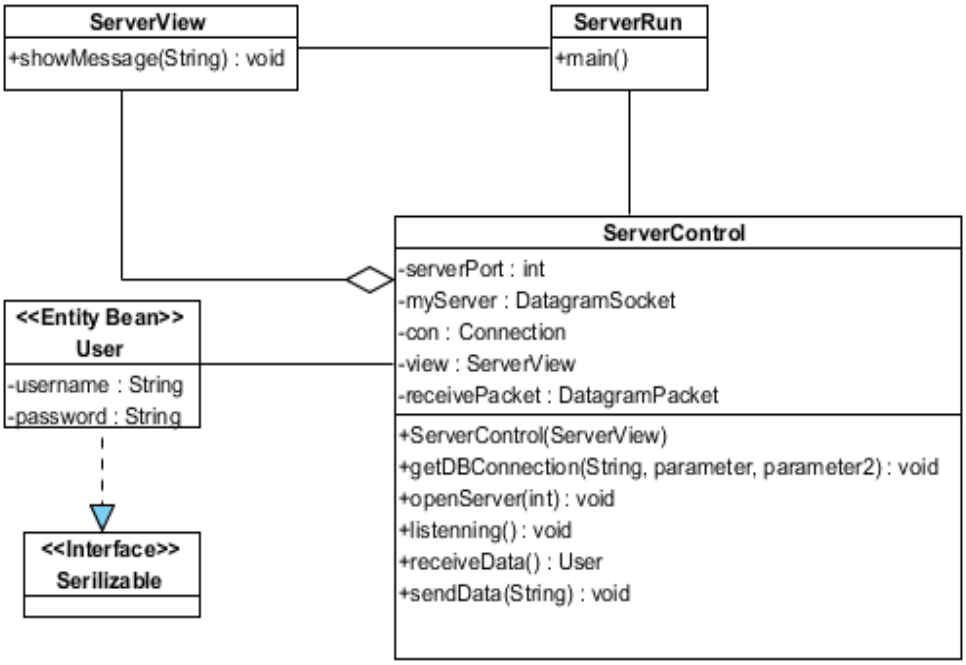
Hình 5.4: Sơ đồ lớp phía client UDP

Sơ đồ lớp của phía client được thiết kế theo mô hình MVC trong Hình 5.4, bao gồm 3 lớp chính tương ứng với sơ đồ M-V-C như sau:

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

sang server theo giao thức UDP, nhận kết quả đăng nhập từ server về và yêu cầu form giao diện hiển thị. Điều này đảm bảo nguyên tắc control điều khiển các phần còn lại trong hệ thống, đúng theo nguyên tắc của mô hình MVC.

b. Sơ đồ lớp phía server

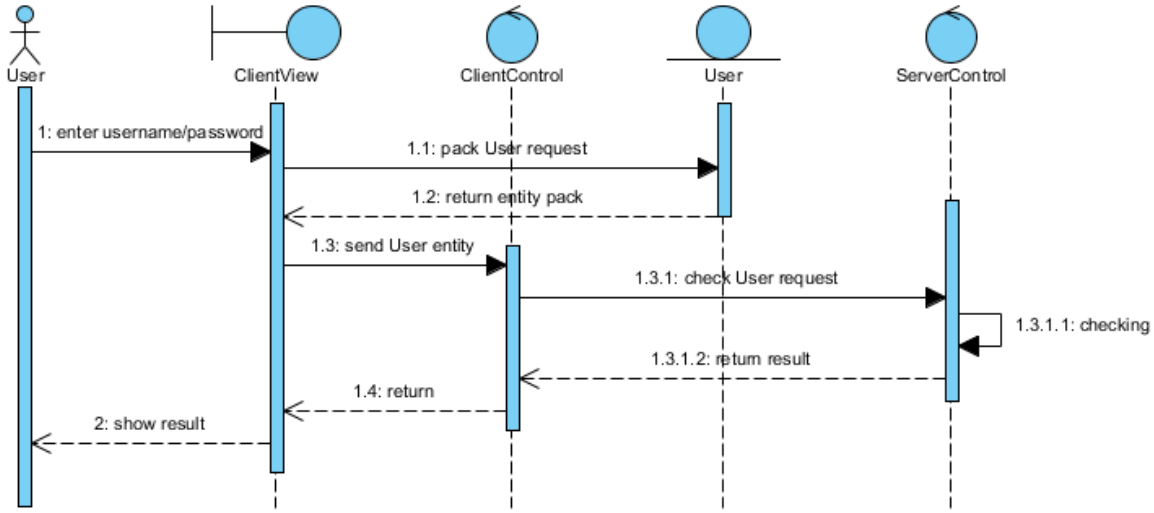


Hình 5.5: Sơ đồ lớp phía server UDP

Sơ đồ lớp của phía server được thiết kế theo mô hình MVC trong Hình 5.5, bao gồm 3 lớp chính tương ứng với sơ đồ M-V-C như sau:

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

c. Tuần tự các bước thực hiện



Hình 5.6: Tuần tự các bước thực hiện theo giao thức UDP

Tuần tự các bước xử lí như sau (Hình 5.6):

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

5.4.3 Cài đặt

a. Các lớp phía client

User.java

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

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

	public User(){
	}
```

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

	public String getPassword() {
		return password;
	}

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

	public String getUserName() {
		return userName;
	}

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

ClientView.java

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

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

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

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

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

```
		content.add(txtPassword);
		content.add(btnLogin);

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

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

	public void actionPerformed(ActionEvent e) {
	}

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

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

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

ClientControl.java

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

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

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

	class LoginListener implements ActionListener {
```

```
    public void actionPerformed(ActionEvent e) {
         openConnection();

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

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

         closeConnection();
    }
}

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

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

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

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

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

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

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

ClientRun.java

```
package udp.client;

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

b. Các lớp phía server

ServerView.java

```
package udp.server;

public class ServerView {
       public ServerView(){
       }

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

ServerControl.java

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

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

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

		while(true){
			listenning();
		}
	}

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

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

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

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

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

		sendData(result);
	}

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

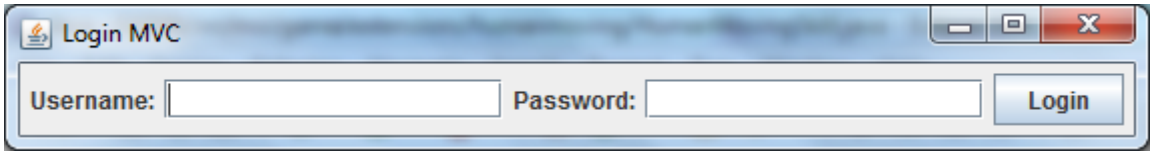
```
            oos.flush();

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

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

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

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

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


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

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

ServerRun.java

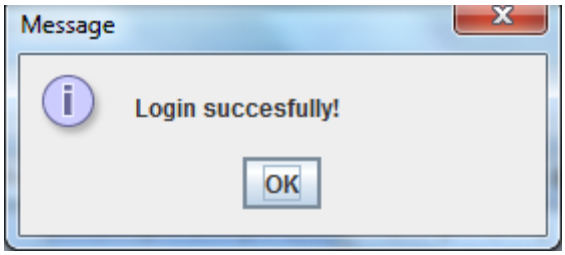
```
package udp.server;
```

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

5.4.5 Kết quả



Login thành công:



Login lỗi:

5.5. KẾT LUẬN

Nội dung chương này đã nhắc lại một số kiến thức cơ bản về giao thức UDP, Các lớp của Java cung cấp và hỗ trợ lập trình với giao thức UDP, các bước lập trình với giao thức UDP cho phía server và phía client. Chương này cũng trình bày một số ví dụ áp dụng với giao thức UDP và case study lập trình giao thức UDP theo mô hình MVC cho cả hai phía client và server.

5.6. BÀI TẬP

1. Viết chương trình copy file từ máy chủ server về máy client dùng giao thức UDP.

2. Viết chương trình giải phương trình bậc 2 theo đúng mô hình giao thức UDP: client chỉ hiện giao diện nhập hệ số, sau đó chuyển lên server tính toán, kết quả lại được trả về cho client hiển thị.
3. Viết chương trình giải hệ phương trình bậc nhất theo đúng mô hình giao thức UDP: client chỉ hiện giao diện nhập hệ số, sau đó chuyển lên server tính toán, kết quả lại được trả về cho client hiển thị.
4. Viết chương trình tìm USCLN (BSCNN) của 2 số nguyên dương a và b theo đúng mô hình giao thức UDP: client chỉ hiện giao diện nhập số, sau đó chuyển lên server tính toán, kết quả lại được trả về cho client hiển thị.
5. Viết chương trình phân tích một số nguyên dương ra thừa số nguyên tố (ví dụ: 300 = 2*2*3*5*5) theo đúng mô hình giao thức UDP: client chỉ hiện giao diện nhập số, sau đó chuyển lên server tính toán, kết quả lại được trả về cho client hiển thị.
6. Viết chương trình ứng dụng chát online dùng giao thức UDP.

CHƯƠNG 6. LẬP TRÌNH CHUYÊN SÂU VỚI SOCKET

6.1. LẬP TRÌNH SOCKET VÀO RA KHÔNG CHẶN DỪNG

Tốc độ truyền dữ liệu qua mạng chậm hơn nhiều so với tốc độ đọc ghi của CPU, bộ nhớ/ổ cứng. Điều này có thể dẫn tới sự không đồng bộ bên trong bản thân các ứng dụng mạng khi đọc ghi dữ liệu. Giải pháp Java truyền thống để cho phép CPU chạy trước mạng là sự kết hợp của bộ đệm và đa luồng. Đa luồng có thể tạo dữ liệu cho một số kết nối khác nhau cùng một lúc và lưu trữ dữ liệu đó trong bộ đệm cho đến khi mạng thực sự sẵn sàng để gửi đi; cách tiếp cận này hoạt động tốt cho các máy chủ và máy khách khá đơn giản không cần hiệu năng cao. Tuy nhiên, chi phí của việc tạo ra đa luồng và chuyển đổi giữa chúng có thể không thể bỏ qua. Ví dụ, cứ mỗi luồng lại yêu cầu thêm bộ nhớ trên RAM. Trên một máy chủ lớn có hàng hàng nghìn yêu cầu mỗi giây, sẽ rất tốn kém nếu chỉ định một luồng cho mỗi kết nối. Giải pháp là một luồng có thể chịu trách nhiệm cho nhiều kết nối: chọn một luồng sẵn sàng nhận dữ liệu, đưa vào nó càng nhiều dữ liệu để kết nối để quản lý nhanh nhất có thể, sau đó chuyển sang kết nối sẵn sàng tiếp theo.

Để hoạt động tốt, cách tiếp cận này cần được hỗ trợ bởi hệ điều hành bên dưới. Hầu hết mọi hệ điều hành hiện nay đang sử dụng để làm máy chủ lớn đều hỗ trợ cơ chế vào ra như trên, gọi là vào ra không chặn dừng. Tuy nhiên, giải pháp này có thể không được hỗ trợ tốt trên một số hệ thống, chẳng hạn như máy tính bảng, điện thoại di động, v.v. Giải pháp này thường được áp dụng trên máy chủ, nơi thường phải đối diện với hàng nghìn yêu cầu cùng lúc. Còn với máy khách và thậm chí cả các hệ thống ngang hàng hiếm khi cần xử lý nhiều kết nối đồng thời đến mức vào ra đa luồng.

Trong chương này, các lớp `Buffer` và `Channels` của gói **NIO** sẽ được giới thiệu. Đây là một giải pháp thay thế cho vào ra bởi IO API trước trong phần trước.

6.1.1. Gói Java NIO

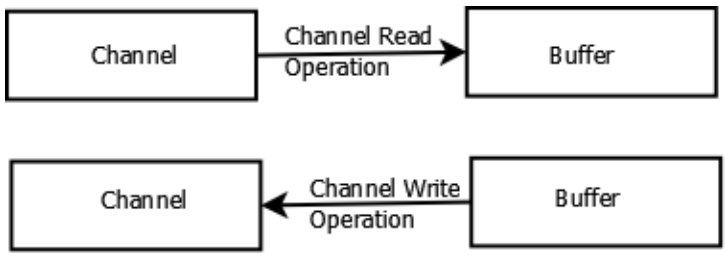
Java NIO có ba lớp quan trọng chính:

`Buffer`: Lớp bộ đệm dùng để chứa thông tin được đọc hoặc ghi đến kênh

`Channel`: Lớp kênh áp dụng kỹ thuật tương tự đọc ghi theo luồng để hỗ trợ hoạt động đọc/ghi không đồng bộ từ/vào nguồn dữ liệu/đích

`Selector`: Lớp này dùng để xử lý nhiều kênh trong một luồng duy nhất

Về mặt khái niệm, bộ đệm và kênh hoạt động cùng nhau để xử lý dữ liệu. Như thể hiện trong hình dưới, dữ liệu có thể được di chuyển theo một trong hai hướng giữa bộ đệm và kênh:



Kênh được kết nối với một số nguồn dữ liệu bên ngoài, trong khi bộ đệm được sử dụng bên trong để xử lý dữ liệu. Có một số loại kênh và bộ đệm. Một số này được liệt kê trong bảng sau.

Lớp kênh	**Mục đích**
`FileChannel`	Kết nối file
`DatagramChannel`	Hỗ trợ socket theo datagram
`SocketChannel`	Hỗ trợ socket theo luồn
`ServerSocketChannel`	Lắng nghe yêu cầu socket
`NetworkChannel`	Hỗ trợ các mạng socket
`AsynchronousSocketChannel`	Hỗ trợ các socket theo luồng không đồng bộ

Lớp bộ đệm	**Kiểu dữ liệu hỗ trợ**
`ByteBuffer`	byte
`CharBuffer`	char
`DoubleBuffer`	double
`FloatBuffer`	float
`IntBuffer`	int
`LongBuffer`	long
`ShortBuffer`	short

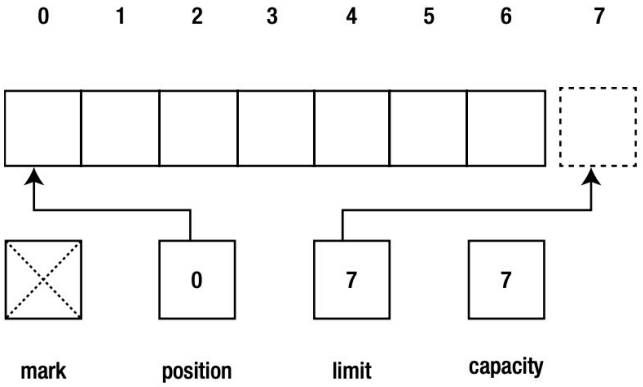
Lớp Selector được sử dụng trong trường hợp một ứng dụng sử dụng nhiều kết nối lưu lượng thấp có thể được xử lý bằng một luồng duy nhất. Điều này hiệu quả hơn việc tạo một luồng cho mỗi kết nối. Đây cũng là một kỹ thuật được sử dụng để làm cho một ứng dụng có thể mở rộng.

6.1.2. Bộ đệm

Bộ đệm tạm thời giữ dữ liệu đang được di chuyển đến/từ các kênh. Khi một bộ đệm được tạo, nó sẽ được tạo với kích thước cố định. Một phần hoặc toàn bộ bộ nhớ của bộ đệm có thể được sử dụng với một số trường của lớp Buffer có sẵn để quản lý dữ liệu trong một bộ đệm.

Lớp Buffer là lớp trừu tượng, nó xử lý 4 thuộc tính chính, như hình dưới:

- *capacity*: Tổng số mục dữ liệu có thể được lưu trữ trong bộ đệm. Dung lượng được chỉ định khi bộ đệm được tạo và không thể thay đổi sau đó.
- *limit*: Chỉ số của phần tử đầu tiên không được đọc hoặc ghi. Nói cách khác, nó xác định số lượng mục dữ liệu "sống" trong bộ đệm.
- *position*: Chỉ mục mục dữ liệu tiếp theo có thể đọc được hoặc vị trí mà mục dữ liệu có thể được ghi.
- *mark* : Chỉ mục vị trí của bộ đệm sẽ được đặt lại khi phương thức reset() của bộ đệm được gọi.



Các phương pháp *mark* và *reset* cũng điều khiển vị trí trong bộ đệm. Phương thức *mark* sẽ đặt dấu của bộ đệm vào vị trí của nó. Phương thức *reset* khôi phục vị trí đánh dấu về vị trí đã đánh dấu trước đó. Mối quan hệ giữa các thuật ngữ trên được mô tả như dưới:

0 <= mark <= position <= limit <= capacity

Bộ đệm có thể là trực tiếp hoặc không trực tiếp. Bộ đệm trực tiếp sẽ cố gắng sử dụng các phương thức IO gốc bất cứ khi nào có thể. Việc tạo ra một bộ đệm trực tiếp có xu hướng tốn kém hơn nhưng sẽ hoạt động hiệu quả hơn đối với các bộ đệm lớn, nằm trong bộ nhớ lâu hơn. Phương thức *allocateDirect* được sử dụng để tạo bộ đệm trực tiếp và nhận một số nguyên xác định kích thước của bộ đệm. Phương thức *allocate* cũng chấp nhận một đối số kích thước số nguyên nhưng tạo ra một bộ đệm không trực tiếp.

Bộ đệm không trực tiếp sẽ không hiệu quả bằng bộ đệm trực tiếp trong hầu hết các hoạt động. Tuy nhiên, bộ nhớ được sử dụng bởi bộ đệm không trực tiếp sẽ được bộ thu gom rác JVM thu hồi, trong khi bộ đệm trực tiếp có thể nằm ngoài sự kiểm soát của JVM. Điều này làm cho việc quản lý bộ nhớ với bộ đệm không trực tiếp dễ hơn.

Có một số phương pháp được sử dụng để truyền dữ liệu giữa kênh và bộ đệm. Chúng có thể được phân loại như sau:

- Tuyệt đối hoặc tương đối
- Chuyển hàng loạt
- Sử dụng các kiểu dữ liệu nguyên thủy
- Hỗ trợ chế độ *view*
- Nén, nhân bản và cắt một byte bộ đệm

Nhiều phương thức của lớp Buffer hỗ trợ chuỗi gọi lệnh. Phương thức kiểu *đặt - put* sẽ chuyển dữ liệu vào một bộ đệm, trong khi một phương thức kiểu *lấy - get* lấy thông tin từ một bộ đệm. Các phương thức này sẽ chuyển một byte duy nhất tại một thời điểm.

Các phương thức lấy và đặt này giựa trên vị trí hiện tại trong bộ đệm. Bên cạnh đó, cũng có một số phương thức tuyệt đối sử dụng chỉ mục trong bộ đệm để cô lập một phần tử đệm cụ thể.

Dữ liệu hàng loạt truyền các khối dữ liệu liền nhau. Các phương thức get và put sử dụng một mảng byte làm đối số để lưu giữ dữ liệu.

Khi tất cả dữ liệu trong lớp Buffer thuộc cùng một loại, *view* có thể được tạo cho phép truy cập thuận tiện vào dữ liệu bằng một kiểu dữ liệu cụ thể ví dụ như Float.

Thao tác nén sẽ thay đổi nội dung của bộ đệm để loại bỏ dữ liệu đã được xử lý. Nhân bản sẽ tạo một bản sao của bộ đệm, trong khi việc cắt lát sẽ tạo ra một bộ đệm mới dựa trên toàn bộ hoặc một phần của bộ đệm ban đầu. Các thay đổi đối với một trong hai bộ đệm sẽ được phản ánh trong bộ đệm còn lại. Tuy nhiên, các giá trị vị trí, giới hạn và dấu của mỗi vùng đệm là độc lập.

Buffer định nghĩa một số phương thức như sau:

Phương thức	Mô tả
Final int capacity()	Trả về số phần tử có trong vùng đệm
Final Buffer clear()	Xóa vùng đệm
Final Buffer flip()	Thiết lập giới hạn của vùng đệm về vị trí hiện hành và thiết lập lại vị trí hiện hành về 0
Final boolean hasRemaining()	Phương thức này trả về giá trị true nếu còn các phần tử trong vùng đệm. Trả về giá trị false nếu ngược lại
Abstract boolean isReadOnly()	Trả về giá trị true nếu vùng đệm là chỉ đọc. Trả về giá trị false nếu ngược lại
Final int limit()	Thiết lập giới hạn của vùng đệm là n
Final Buffer limit(int n)	Thiết lập giới hạn của vùng đệm là n và trả về tham chiếu tới vùng đệm được gọi
Final Buffer mark()	Thiết lập vị trí đánh dấu và trả về tham chiếu tới vùng đệm được gọi
Final int Position()	Trả về vị trí hiện hành của vùng đệm
Final Buffer position(int n)	Thiết lập vị trí của Buffer là n. Trả về một tham chiếu tới

	vùng đệm
Final Buffer reset()	Thiết lập lại vị trí hiện hành của vùng đệm và vị trí đánh dấu được thiết lập trước đó. Trả về một tham chiếu tới vùng đệm
Final Buffer rewind()	Thiết lập vị trí hiện hành của vùng đệm về 0

Các phương thức get và put cho phép ta nhận dữ liệu từ một vùng đệm và đặt dữ liệu vào một buffer, cụ thể như sau

Phương thức	Mô tả
Abstract byte get()	Trả về byte dữ liệu tại vị trí hiện hành
ByteBuffer get(byte[] vals)	Sao chép dữ liệu từ vùng đệm vào một mảng được tham chiếu bởi mảng vals. Trả về một tham chiếu tới buffer
ByteBuffer get(byte vals[], int start, int num)	Sao chép num số phần tử từ buffer vào mảng được tham chiếu bởi vals, bắt đầu tại chỉ mục được xác định bởi tham số start. Trả về tham chiếu tới vùng đệm. Nếu không còn phần tử nào trong vùng đệm, ngoại lệ BufferUnderflowException
Abstract byte get(int idx)	Trả về byte dữ liệu tại vị trí được xác định bởi chỉ mục idx trong vùng đệm
Abstract ByteBuffer put(byte b)	Sao chép byte dữ liệu b vào tại vị trí hiện hành
final ByteBuffer put(byte b[])	Sao chép tất cả các phần tử của mảng b vào vùng đệm, bắt đầu từ vị trí hiện hành. Trả về tham chiếu tới vùng đệm
ByteBuffer put(byte b[], int start, int num)	Sao chép num phần tử từ mảng b bắt đầu tại vị trí start vào vùng đệm. Trả về tham chiếu tới vùng đệm. Nếu vùng đệm không chưa được tất cả các phần tử của vùng đệm thì ngoại lệ BufferOverflowException sẽ được đưa ra
BufferByte put(ByteBuffer bb)	Sao chép tất cả các phần tử của vùng đệm BufferByte gọi, bắt đầu từ vị trí hiện hành . Nếu vùng đệm không chưa được tất cả các phần tử của vùng đệm thì ngoại lệ BufferOverflowException sẽ được đưa ra
Abstract ByteBuffer put(int idx, byte b)	Sao chép byte dữ liệu b tại vị trí idx vào vùng đệm. Trả về tham chiếu tới vùng đệm

6.1.2. Kênh

Một kênh biểu diễn một liên kết mở tới một nguồn hoặc đích vào ra. Các kênh cho phép truyền dữ liệu một cách hiệu quả giữa bộ đệm và các nguồn/đích dịch vụ vào ra dựa trên hệ điều hành.

Java hỗ trợ các kênh thông qua các gói java.nio.channels và java.nio.channels.spi. Tất cả các kênh là thực thể của lớp thực thi giao diện java.nio.channels.Channel. Channel khai báo các phương thức sau:

- *void close()*: Đóng kênh. Khi kênh đã bị đóng, việc gọi close() không có tác dụng. Khi một luồng khác đã gọi hàm close(), một lệnh gọi close() mới sẽ bị chặn cho đến khi lệnh gọi đầu tiên kết thúc, sau đó lệnh close() trả về mà không có hiệu lực. Phương thức này tạo ra java.io.IOException khi xảy ra lỗi I/O.
- *boolean isOpen()*: Trả lại trạng thái mở của kênh. Phương thức này trả về true khi kênh đang mở; nếu không, nó trả về false.

Các phương thức này chỉ ra rằng chỉ có hai thao tác chung cho tất cả các kênh: đóng kênh và xác định xem kênh đang mở hay đóng. Để hỗ trợ vào ra, Channel được mở rộng bởi các giao diện *java.nio.channels.WordsByteChannel* và *java.nio.channels.ReadableByteChannel*:

Lớp	Mô tả
WordsByteChannel	Khai báo phương thức trừu tượng *int write(ByteBuffer buffer)* để ghi một chuỗi các byte từ bộ đệm đến kênh hiện tại. Phương thức này trả về số byte được ghi. *java.nio.channels.NonWainedChannelException* sẽ được đưa ra khi kênh không được mở để ghi, *java.nio.channel.ClosedChannelException* khi kênh bị đóng, *java.nio.channels.AsynchronousCloseException* khi một luồng khác đóng kênh trong quá trình ghi, *java.nio.channels.ClosedByInterruptException* khi một luồng khác làm gián đoạn luồng hiện tại trong khi thao tác ghi đang diễn ra (do đó đóng kênh và thiết lập trạng thái ngắt của luồng hiện tại) và *IOException* khi một số lỗi I / O khác xảy ra.
ReadableByteChannel	Khai báo một phương thức int trừu *tượng int read(ByteBuffer buffer)* đọc các byte từ kênh hiện tại vào bộ đệm. Phương thức này trả về số byte

	được đọc (hoặc -1 khi không còn byte nào để đọc). *java.nio.channels.NonReadableChannelException* sẽ được tạo ra khi kênh không được mở để đọc; *ClosedChannelException* khi kênh bị đóng; *AsynchronousCloseException* khi một luồng khác đóng kênh trong quá trình đọc; *ClosedByInterruptException* khi một luồng khác ngắt luồng hiện tại trong khi thao tác ghi đang diễn ra, do đó đóng kênh và đặt trạng thái ngắt của luồng hiện tại; và *IOException* khi một số lỗi vào ra khác xảy ra.

Lớp Socket khai báo một phương thức *SocketChannel getChannel()* để trả về một thực thể kênh socket, mô tả một kết nối mở đến một socket. Không giống như socket, các kênh socket có thể lựa chọn và có thể hoạt động ở chế độ không chặn. Những khả năng này nâng cao khả năng mở rộng và tính linh hoạt của các ứng dụng lớn.

Các kênh socket được mô tả bởi các lớp trừu tượng *ServerSocketChannel*, *SocketChannel* và *DatagramChannel* của gói *java.nio.channels*. Lớp kế thừa *java.nio.channels.SelectableChannel* và thực thi *InterruptibleChannel*, làm cho các đối tượng *ServerSocketChannel*, *SocketChannel* và *DatagramChannel* có thể lựa chọn và ngắt được. Vì *SocketChannel* và *DatagramChannel* thực thi các giao diện *ByteChannel*, *GatheringByteChannel* và *ScatteringByteChannel*, nên ta có thể ghi/đọc vào/từ và thực hiện việc vào ra phân tán/tập hợp, trên các socket bên dưới.

Mỗi thực thể của ServerSocketChannel, SocketChannel và DatagramChannel tạo ra đối tượng socket ngang hàng từ lớp java.net.ServerSocket, Socket hoặc java.net.DatagramSocket. Mỗi lớp đã được trang bị để có thể hoạt động với các kênh. Ta có thể lấy đối tượng socket ngang hàng bằng cách gọi phương thức socket() của ServerSocketChannel, SocketChannel hoặc DatagramChannel.

Ta có thể nhận được một kênh bằng cách gọi phương thức getChannel() trên một đối tượng hỗ trợ kênh. Để nhận được một kênh, trước tiên ta phải nhận một đối tượng của các lớp này và sau đó gọi phương thức getChannel() trên đối tượng đó. Kiểu kênh cụ thể được trả về phụ thuộc vào kiểu đối tượng chịu tác động của phương thưc getChannel(). Ví dụ khi gọi phương thức getChannel() trên đối tượng FileInputStream, FileOutputStream hoặc RandomFileAccess thì kênh trả về là FileChannel. Khi gọi phương thức getChannel() trên đối tượng Socket thì kiểu kênh trả về là SocketChannel(). Các kênh FileChannel và SocketChannel hỗ trợ các phương thức read() và write() cho phép ta thực hiện các thao tác vào ra thông qua kênh.

Bản chất chặn của các socket được tạo từ các lớp Socket của Java hạn chế khả năng mở rộng của ứng dụng mạng Java. Ví dụ: phương thức *Socket accept()* của lớp *ServerSocket* chặn cho đến khi

có kết nối đến, tại thời điểm đó, nó tạo và trả về một thể hiện Socket cho phép máy chủ giao tiếp với máy khách. Nếu phương thức này không chặn, khả năng mở rộng sẽ cải thiện vì máy chủ có thể hoàn thành công việc hữu ích khác thay vì phải chờ đợi.

Lớp trừu tượng *SelectableChannel* là cha chung của các lớp *ServerSocketChannel*, *SocketChannel* và *DatagramChannel*. *SelectableChannel* cho phép các kênh Socket lựa chọn hoạt động ở chế độ chặn hoặc không chặn: luồng có thể kiểm tra đầu vào hoặc gửi đầu ra mà không bị chặn khi không có đầu vào hoặc khi bộ đệm đầu ra đầy.

SelectableChannel cung cấp các phương pháp sau để bật chặn hoặc không chặn, xác định xem kênh đang chặn hay không chặn và lấy khóa chặn.

Phương thức	Mô tả
SelectableChannel configureBlocking(boolean block)	Chỉ định trạng thái chặn của kênh có thể chọn đang gọi. Chuyển true để kênh bị chặn và false để kênh không bị chặn. Phương thức trả về kênh có thể chọn hoặc tạo ra ngoại lệ: ClosedChannelException khi kênh bị đóng, java.net.channels.IllegalBlockingModeException khi trạng thái chặn là true và kênh đã được đăng ký với một hoặc nhiều bộ chọn và IOException khi xảy ra lỗi vào ra.
boolean isBlocking()	Phương thức trả về true khi kênh có thể chọn đang gọi bị chặn; nếu không, trả về false. Các kênh mới được tạo mặc định bị chặn
Object blockingLock()	Trả về đối tượng trên đó configureBlocking() đồng bộ hóa. Đối tượng trả về hữu ích trong việc thực thi các bộ điều hợp yêu cầu giá trị chế độ chặn hiện tại không thay đổi trong một khoảng thời gian ngắn.

6.1.3. Một số ví dụ

Dưới đây là một số chương trình client/server được thạo bởi SocketChannerl và Buffer.

a) Sử dụng kênh với chương trình lấy thời gian

Chương trình lấy thời gian đã được giới thiệu trong Chương 4, sẽ được triển khai ở đây để chứng minh việc sử dụng bộ đệm và kênh. Ứng dụng khá đơn giản, nhưng chúng minh họa cách bộ đệm và kênh có thể được sử dụng cùng nhau.

Chương trình phía máy chủ được minh họa ở đoạn code dưới.

```
public class ServerSocketChannelTimeServer {
      public static void main(String[] args) {
            System.out.println("Time Server started");
            try {
                  ServerSocketChannel serverSocketChannel =
                        ServerSocketChannel.open();
                  serverSocketChannel.socket().bind(new
                        InetSocketAddress(5000));
                  while (true) {
                        System.out.println("Waiting for request ...");
                        SocketChannel socketChannel =
                              serverSocketChannel.accept();

                        if (socketChannel != null) {
                              String dateAndTimeMessage = "Date: " + new
                                    Date(System.currentTimeMillis());

                              ByteBuffer buf = ByteBuffer.allocate(64);
                              // If buffer is not large enough:
                                    BufferOverflowException
                              buf.put(dateAndTimeMessage.getBytes());
                              buf.flip();
                              while (buf.hasRemaining()) {
                                    socketChannel.write(buf);
                              }
                              System.out.println("Sent: " +
                                    dateAndTimeMessage);
                        }
                  }
            } catch (IOException ex) {
                  ex.printStackTrace();
            }
      }
}
```

Chương trình phía máy khách được minh họa ở đoạn code dưới.

```
public class SocketChannelTimeClient {
    public static void main(String[] args) {
        SocketAddress address = new InetSocketAddress("127.0.0.1", 5000);
        try (SocketChannel socketChannel = SocketChannel.open(address)) {
            ByteBuffer byteBuffer = ByteBuffer.allocate(64);
            int bytesRead = socketChannel.read(byteBuffer);
            while (bytesRead > 0) {
                byteBuffer.flip();
                while (byteBuffer.hasRemaining()) {
                    System.out.print((char) byteBuffer.get());
                }
```

```
                System.out.println();
                bytesRead = socketChannel.read(byteBuffer);
            }
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }

    private static void displayByteBuffer(ByteBuffer byteBuffer) {
        System.out.println("Capacity: " + byteBuffer.capacity()
                + " limit: " + byteBuffer.limit()
                + " position: " + byteBuffer.position());
    }
}
```

b) Ứng dụng trò chuyện

Chương trình phía máy chủ được minh họa ở đoạn code dưới

```
public class ChatServer {
    public ChatServer() {
        System.out.println("Chat Server started");
        try {
            ServerSocketChannel serverSocketChannel =
                  ServerSocketChannel.open();
            serverSocketChannel.socket().bind(new InetSocketAddress(5000));

            boolean running = true;
            while (running) {
                System.out.println("Waiting for request ...");
                SocketChannel socketChannel = serverSocketChannel.accept();

                System.out.println("Connected to Client");
                String message;
                Scanner scanner = new Scanner(System.in);
                while (true) {
                    System.out.print("> ");
                    message = scanner.nextLine();
                    if (message.equalsIgnoreCase("quit")) {
                        //HelperMethods.sendFixedLengthMessage(
                        //socketChannel, "Server terminating");
                            HelperMethods.sendMessage(
                                socketChannel, "Server terminating");
                        running = false;
                        break;
                    } else {
                        //HelperMethods.sendFixedLengthMessage(
                        //socketChannel, message);
                            HelperMethods.sendMessage(socketChannel, message);
                        // Receive message
```

```
                        System.out.println("Waiting for message from client
                        ...")
                        System.out.println("Message: " +
                        HelperMethods.receiveMessage(socketChannel));
                    }
                }
            }
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }

    public static void main(String[] args) {
        new ChatServer();
    }
}
```

Chương trình phía máy khách được minh họa ở đoạn code dưới

```
public class ChatClient {
      public ChatClient() {
            SocketAddress address = new InetSocketAddress("127.0.0.1", 5000);
            try (SocketChannel socketChannel = SocketChannel.open(address)) {
                  System.out.println("Connected to Chat Server");
                  String message;
                  Scanner scanner = new Scanner(System.in);
                  while (true) {
                        // Receive message
                        System.out.println("Waiting for message from the
                        server ...");
                        System.out.println("Message: " +
                        HelperMethods.receiveMessage(socketChannel));
                        // System.out.println("Message: "
                        // + HelperMethods.receiveFixedLengthMessage(
                        // socketChannel));
                        System.out.print("> ");
                        message = scanner.nextLine();
                        if (message.equalsIgnoreCase("quit")) {
                              HelperMethods.sendMessage(socketChannel, "Client
                              terminating");
                              // HelperMethods.sendFixedLengthMessage(
                              // socketChannel, "Client terminating");
                              break;
                        }
                        // Send message
                        HelperMethods.sendMessage(socketChannel, message);
                                    }
            } catch (IOException ex) {
                  ex.printStackTrace();
            }
```

```
    }

    public static void main(String[] args) {
        new ChatClient();
    }
}
```

c) Ứng dụng chấp nhận nhiều kết nối đồng thời

Chương trình phía máy chủ được minh họa ở đoạn code dưới

```
public class PartsServer {
    private static final HashMap<String,Float> parts = new HashMap<>();

    public PartsServer() {
        System.out.println("Part Server Started");
        initializeParts();
        try {
            ServerSocketChannel serverSocketChannel =
                  ServerSocketChannel.open();
            serverSocketChannel.socket().bind(new InetSocketAddress(5000));
            serverSocketChannel.setOption(SO_RCVBUF, 64);
            boolean running = true;
            while (running) {
                System.out.println("Waiting for client ...");
                SocketChannel socketChannel
                        = serverSocketChannel.accept();
                new Thread(new ClientHandler(socketChannel)).start();
            }
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }

    private void initializeParts() {
        parts.put("Hammer", 12.55f);
        parts.put("Nail", 1.35f);
        parts.put("Pliers", 4.65f);
        parts.put("Saw", 8.45f);
    }

    public static Float getPrice(String partName) {
        return parts.get(partName);
    }

    public static void main(String[] args) {
        new PartsServer();
    }
}
```

Chương trình phía máy khách được minh họa ở đoạn code dưới

```
public class PartsClient {

    public PartsClient() {
        System.out.println("PartsClient Started");
        SocketAddress address = new InetSocketAddress("127.0.0.1",
                  5000);
        try (SocketChannel socketChannel = SocketChannel.open(address)) {
            System.out.println("Connected to Parts Server");
            Scanner scanner = new Scanner(System.in);
            while (true) {
                // Get part name
                System.out.print("Enter part name: ");
                String partName = scanner.nextLine();
                if (partName.equalsIgnoreCase("quit")) {
                    HelperMethods.sendMessage(socketChannel, "quit");
                    break;
                } else {
                    HelperMethods.sendMessage(socketChannel, partName);
                    System.out.println("The price is " +
                        HelperMethods.receiveMessage(socketChannel));
                }
            }
            System.out.println("PartsClient Terminated");
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }

    public static void main(String[] args) {
        new PartsClient();
    }
}
```

d) Ứng dụng liên lạc không đồng bộ

Chương trình phía máy chủ được minh họa ở đoạn code dưới

```
public class AsynchronousServerSocketChannelServer {

    public AsynchronousServerSocketChannelServer() {
        System.out.println("Asynchronous Server Started");
        try (AsynchronousServerSocketChannel serverChannel
                = AsynchronousServerSocketChannel.open()) {
            InetSocketAddress hostAddress
                    = new InetSocketAddress("localhost", 5000);
            serverChannel.bind(hostAddress);
```

```
            System.out.println("Waiting for client to connect... ");
            Future acceptResult = serverChannel.accept();

            try (AsynchronousSocketChannel clientChannel
                    = (AsynchronousSocketChannel) acceptResult.get()) {
                System.out.println("Messages from client: ");
                while ((clientChannel != null) && (clientChannel.isOpen())) {
                    ByteBuffer buffer = ByteBuffer.allocate(32);
                    Future result = clientChannel.read(buffer);
                    // Wait until buffer is ready using
                    while (!result.isDone()) {
                        // do nothing
                    }
                    // Technique 2
                  // result.get();
                    // Technique 3
                  // result.get(10, TimeUnit.SECONDS);
                    buffer.flip();

                    String message = new String(buffer.array()).trim();
                    System.out.println(message);
                    if (message.equals("quit")) {
                        break;
                    }
                }
            }
        } catch (IOException | InterruptedException | ExecutionException ex) {
            ex.printStackTrace();
        }
    }

    public static void main(String[] args) {
        new AsynchronousServerSocketChannelServer();

    }


}
```

Chương trình phía máy khách được minh họa ở đoạn code dưới

```
public class AsynchronousSocketChannelClient {

    public static void main(String[] args) {
        System.out.println("Asynchronous Client Started");
        try (AsynchronousSocketChannel client =
                AsynchronousSocketChannel.open()) {
            InetSocketAddress hostAddress =
                    new InetSocketAddress("localhost", 5000);
            Future future = client.connect(hostAddress);
```

```
            future.get(); // Blocks until connection made

            System.out.println("Client is started: " + client.isOpen());
            System.out.println("Sending messages to server: ");

            Scanner scanner = new Scanner(System.in);
            String message;
            while (true) {
                System.out.print("> ");
                message = scanner.nextLine();
                ByteBuffer buffer = ByteBuffer.wrap(message.getBytes());
                Future result = client.write(buffer);
                while (!result.isDone()) {
                    // Wait
                }
                if (message.equalsIgnoreCase("quit")) {
                    break;
                }
            }
        } catch (IOException | InterruptedException | ExecutionException ex) {
            ex.printStackTrace();
        }
    }
}
```

6.1.5. Kết luận

Trong chương này, ta đã xem xét việc sử dụng các lớp Buffer và Channel của NIO. Kênh kết nối với nguồn bên ngoài và truyền dữ liệu đến và đi qua bộ đệm. Một số ví dụ liên quan đến việc sử dụng Bộ đệm và Kênh cũng được đưa ra minh họa cho các ứng dụng mạng.

Bộ đệm là kho lưu trữ tạm thời cho dữ liệu. Sử dụng bộ đệm cho phép dữ liệu được truy cập tuần tự hoặc ngẫu nhiên. Có nhiều thao tác đệm, cho phép nhiều lựa chọn tốt cho các ứng dụng.

Lớp ServerSocketChannel hỗ trợ máy chủ và sử dụng phương thức accept để chặn cho đến khi máy khách yêu cầu kết nối. Phương thức này sẽ trả về một thể hiện SocketChannel, thể hiện này sẽ được kết nối với SocketChannel của máy khách. Các lớp AsynchronousSocketChannel và AsynchronousSocketChannel hỗ trợ giao tiếp không đồng bộ cho phép giao tiếp không chặn giữa hai ứng dụng.

Cách thức hoạt động của lớp đệm và lớp kênh đã được giải thích và minh họa việc sử dụng chúng trong một số ứng dụng máy khách/máy chủ.

6.2. LẬP TRÌNH KẾT NỐI NHIỀU GIAO TIẾP MẠNG

Các hệ thống thường chạy với nhiều kết nối mạng đang hoạt động, chẳng hạn như kết nối có dây, kết nối không day 802.11 b/g, hoặc Bluetooth . Một số ứng dụng có thể cần truy cập thông tin này để thực hiện hoạt động mạng trên một kết nối cụ thể.

Phần này sẽ giới thiệu và hướng dẫn một số thao tác với ứng dụng mạng chạy trên hệ thống có nhiều giao tiếp mạng, đồng thời cũng giới thiệu lớp java.net.NetworkInterface, là lớp cung cấp quyền truy cập vào thông tin này.

6.2.1. Giới thiệu Giao tiếp mạng

Giao diện mạng là điểm kết nối giữa một máy tính và một mạng riêng hoặc công cộng. Giao diện mạng nói chung là một thẻ giao diện mạng (NIC hay còn gọi là card mạng), nhưng không nhất thiết phải dưới dạng vật lý. Thay vào đó, giao diện mạng có thể được thực thi qua phần mềm. Ví dụ, giao diện loopback (127.0.0.1 cho IPv4 và ::1 cho IPv6) không phải là thiết bị vật lý mà là một phần mềm mô phỏng giao diện mạng. Giao diện loopback thường được sử dụng trong các môi trường thử nghiệm.

Lớp java.net.NetworkInterface đại diện cho cả hai loại giao diện. Lớp đại diện cho một giao diện mạng dưới dạng tên (chẳng hạn như le0) và danh sách các địa chỉ IP được gán cho giao diện này. Mặc dù giao diện mạng thường được triển khai trên NIC vật lý, nó cũng có thể được thực hiện trong phần mềm; ví dụ: giao diện loopback (để kiểm tra một máy khách).

NetworkInterface hữu ích cho một hệ thống có nhiều NIC. Sử dụng NetworkInterface, ta có thể chỉ định NIC nào sẽ sử dụng cho một hoạt động mạng cụ thể.

Ví dụ: giả sử có một máy với hai NIC được cấu hình và ta muốn gửi dữ liệu đến một máy chủ. Ta tạo một socket như thế này:

```
Socket soc = new java.net.Socket();
soc.connect(new InetSocketAddress(address, port));
```

Để gửi dữ liệu, hệ thống xác định giao diện nào được sử dụng. Tuy nhiên, nếu muốn chỉ định NIC nào để sử dụng, ta có thể truy vấn hệ thống để biết các giao diện thích hợp và tìm một địa chỉ trên giao diện muốn sử dụng. Khi tạo socket và liên kết với địa chỉ đó, hệ thống sẽ sử dụng giao diện được liên kết. Ví dụ:

```
NetworkInterface nif = NetworkInterface.getByName("bge0");
Enumeration<InetAddress> nifAddresses = nif.getInetAddresses();
Socket soc = new java.net.Socket();
soc.bind(new InetSocketAddress(nifAddresses.nextElement(), 0));
soc.connect(new InetSocketAddress(address, port));
```

Ta cũng có thể sử dụng *NetworkInterface* để xác định giao diện cục bộ mà trên có hỗ trợ nhóm multicast. Ví dụ:

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

NetworkInterface có thể được sử dụng với các API Java theo nhiều cách khác ngoài hai cách sử dụng được mô tả ở đây. Sau đây tóm tắt các phương pháp quan trọng của NeworkInterface:

Phương thức	**Mô tả**
`static NetworkInterface getByInetAddress(InetAddress address)`	Trả về NetworkInterface tương ứng với địa chỉ đã cho hoặc null khi không có giao diện nào có địa chỉ này. Phương thức này tạo ra SocketException khi xảy ra lỗi vào ra và NullPointerException khi địa chỉ rỗng.
`static NetworkInterface getByName(String interfaceName)`	Trả về NetworkInterface với tên đã chỉ định hoặc trả về null khi không có giao diện mạng như vậy. Phương thức này tạo ra SocketException với lỗi vào ra và NullPointerException khi interfaceName là null.
`String getDisplayName()`	Trả về *tên hiển thị* của giao diện mạng.
`byte[] getHardwareAddress()`	Trả về một mảng byte chứa địa chỉ phần cứng của giao diện mạng, địa chỉ này thường được gọi là địa chỉ *điều khiển truy cập phương tiện (MAC)* . Khi giao diện không có địa chỉ MAC hoặc khi không thể truy cập địa chỉ, phương thức trả về null. Phương thức này tạo ra SocketException khi xảy ra lỗi vào ra.
`Enumeration<InetAddress> getInetAddresses()`	Trả về một *enumeration* với tất cả hoặc một tập hợp con của các địa chỉ liên kết với giao diện mạng.
`List<InterfaceAddress> getInterfaceAddresses()`	Trả về java.util.List chứa InterfaceAddresses của giao diện mạng.
`int getMTU()`	Trả về *đơn vị truyền tối đa (MTU)* của giao diện mạng. Phương thức này tạo ra SocketException khi xảy ra lỗi vào ra.
`String getName()`	Trả lại tên của giao diện mạng (chẳng hạn như eth0 hoặc lo).
`static Enumeration<NetworkInterface> getNetworkInterfaces()`	Trả về tất cả các giao diện mạng trên máy hoặc trả về null khi không tìm thấy giao diện mạng nào. Phương thức này tạo ra SocketException khi xảy ra lỗi vào ra.
`NetworkInterface getParent()`	Trả lại NetworkInterface cha của giao diện mạng khi giao diện mạng là giao diện con. Khi giao diện mạng không có cha hoặc khi đó là giao diện vật lý (không ảo), phương thức này trả về null. (Một giao diện mạng

	vật lý có thể được chia một cách hợp lý thành nhiều giao diện con *ảo* , thường được sử dụng trong định tuyến và chuyển mạch. Các giao diện con có thể được tổ chức thành một hệ thống phân cấp nơi giao diện mạng vật lý đóng vai trò gốc.)
`Enumeration<NetworkInterface> getSubInterfaces()`	Trả về một bảng liệt kê có chứa các giao diện con ảo được gắn vào giao diện mạng. Ví dụ, eth0: 1 là một giao diện con của eth0.
`boolean isLoopback()`	Trả về true khi giao diện mạng phản ánh dữ liệu đi trở lại chính nó dưới dạng dữ liệu đến. Phương thức này tạo ra SocketException khi xảy ra lỗi vào ra.
`boolean isPointToPoint()`	Trả về true khi giao diện mạng là điểm - điểm (chẳng hạn như kết nối PPP qua modem). Phương thức này tạo ra SocketException khi xảy ra lỗi vào ra.
`boolean isUp()`	Trả về true khi giao diện mạng được thiết *lập* (các mục định tuyến đã được thiết lập) và *đang chạy* (tài nguyên nền tảng đã được phân bổ). Phương thức này tạo ra SocketException khi xảy ra lỗi vào ra.
`boolean isVirtual()`	Trả về true khi giao diện mạng là giao diện con ảo. Trên một số nền tảng, giao diện con ảo là giao diện mạng được tạo ra dưới dạng giao diện mạng vật lý và được cung cấp các cài đặt khác nhau (chẳng hạn như địa chỉ hoặc MTU). Thông thường, tên của giao diện sẽ là tên của giao diện mẹ, theo sau là dấu hai chấm (:) và một số xác định con vì có thể có một số giao diện con ảo được gắn vào một giao diện mạng vật lý.
`boolean supportsMulticast()`	Trả về true khi giao diện mạng hỗ trợ *multicast*. Phương thức này tạo ra SocketException khi xảy ra lỗi vào ra.

Có thể sử dụng các phương thức này để thu thập thông tin hữu ích về các giao diện mạng, cụ thể:

- Lấy tên giao diện mạng và tên hiển thị
- Xác định xem giao diện mạng có phải là giao diện *loopback* hay không
- Xác định xem giao diện mạng có đang hoạt động hay không

- Nhận MTU
- Xác định xem giao diện mạng có hỗ trợ multicast hay không
- Liệt kê tất cả các giao diện con ảo của giao diện mạng

6.2.2. Lấy thông tin Giao tiếp mạng

Lớp NetworkInterface không có phương thức khởi tạo công khai. Do đó, ta không thể chỉ tạo mới đối tượng của lớp với toán tử new . Thay vào đó, các phương thức tĩnh sau cho phép truy xuất chi tiết giao diện từ hệ thống: getByInetAddress() , getByName() và getNetworkInterfaces(). Hai phương thức đầu được sử dụng khi ta đã biết địa chỉ IP hoặc tên của giao diện cụ thể. Phương thức thứ ba, getNetworkInterfaces() trả về danh sách đầy đủ các giao diện trên máy.

Các giao diện mạng có thể được tổ chức phân cấp. Lớp NetworkInterface có hai phương thức, getParent() và getSubInterfaces(), phù hợp với hệ thống phân cấp giao diện mạng. Phương thức getParent() trả về NetworkInterface cha của một giao diện. Nếu giao diện mạng là giao diện con, getParent() trả về một giá trị khác rỗng. Phương thức getSubInterfaces() trả về tất cả các giao diện con của giao diện mạng.

Ví dụ sau liệt kê tên của tất cả các giao diện mạng và giao diện con (nếu có) trên một máy.

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

public class ListNIFs
{
    public static void main(String args[]) throws SocketException {
        Enumeration<NetworkInterface> nets =
                    NetworkInterface.getNetworkInterfaces();
       for (NetworkInterface netIf : Collections.list(nets)) {
            out.printf("Display name: %s\n", netIf.getDisplayName());
            out.printf("Name: %s\n", netIf.getName());
            displaySubInterfaces(netIf);
            out.printf("\n");
        }
    }

    static void displaySubInterfaces(NetworkInterface netIf) throws
            SocketException {
        Enumeration<NetworkInterface> subIfs = netIf.getSubInterfaces();
        for (NetworkInterface subIf : Collections.list(subIfs)) {
            out.printf("\tSub Interface Display name: %s\n",
                    subIf.getDisplayName());
            out.printf("\tSub Interface Name: %s\n", subIf.getName());
        }
```

```
    }
}
```

6.2.3. Liệt kê Giao tiếp mạng

Một trong những thông tin hữu ích nhất mà ta có thể nhận được từ giao diện mạng là danh sách các địa chỉ IP được gán cho nó. Ta có thể lấy thông tin này qua NetworkInterface bằng cách sử dụng một trong hai cách. Đầu tiên thông qua phương thức getInetAddresses(), phương thức trả về Enumeration của InetAddress. Phương thức thứ 2 là getInterfaceAddresses(), trả về danh sách các thực thể của java.net.InterfaceAddress. Phương pháp này được sử dụng khi ta cần thêm thông tin về địa chỉ giao diện ngoài địa chỉ IP. Ví dụ: ta có thể cần thông tin bổ sung về mặt nạ mạng con và địa chỉ quảng bá khi địa chỉ là IPv4 và độ dài tiền tố mạng trong trường hợp là địa chỉ IPv6.

Chương trình ví dụ sau liệt kê tất cả các giao diện mạng và địa chỉ trên một máy:

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

public class ListNets {

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

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

6.2.4. Các tham số Giao tiếp mạng

Ta cũng có thể truy cập các thông số mạng về giao diện mạng ngoài tên và địa chỉ IP. Đồng thời, ta có thể khám phá xem giao diện mạng có đang chạy (up) hay không bằng phương thức isUP(). Các phương thức sau xác đinh loại giao diện mạng:

- isLoopback() cho biết giao diện mạng có phải là giao diện lặp lại hay không.
- isPointToPoint() cho biết giao diện có phải là giao diện điểm-điểm hay không.
- isVirtual() cho biết giao diện có phải là giao diện ảo hay không.

Phương thức supportsMulticast() cho biết giao diện mạng có hỗ trợ đa hướng hay không. Phương thức getHardwareAddress() trả về địa chỉ phần cứng vật lý của giao diện mạng, thường được gọi là địa chỉ MAC, khi nó khả dụng. Phương thức getMTU() trả về Đơn vị Truyền tối đa (MTU), kích thước gói lớn nhất có thể được truyền đi.

Ví dụ sau minh họa cách sử dụng các phương thức trên:

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

public class ListNetsEx {

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

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

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

        out.printf("Up? %s\n", netint.isUp());
        out.printf("Loopback? %s\n", netint.isLoopback());
        out.printf("PointToPoint? %s\n", netint.isPointToPoint());
```

```
            out.printf("Supports multicast? %s\n", netint.supportsMulticast());
            out.printf("Virtual? %s\n", netint.isVirtual());
            out.printf("Hardware address: %s\n",
                       Arrays.toString(netint.getHardwareAddress()));
            out.printf("MTU: %s\n", netint.getMTU());
            out.printf("\n");
        }
}
```

6.2.5. Kết hợp với Socket

Lớp Networkinterface được sử dụng cho hệ thống có nhiều NIC. Sử dụng NetworkInterface, ta có thể chỉ định NIC để sử dụng cho một hoạt động mạng cụ thể. Ví dụ: giả sử máy có hai NIC được cấu hình và ta muốn gửi dữ liệu đến một máy chủ. Ta tạo một socket như sau:

```
Socket socket = new Socket();
socket.connect(new InetSocketAddress(address, port));
```

Trước khi gửi dữ liệu, hệ điều hành (OS) xác định giao diện nào sẽ được sử dụng. Tuy nhiên, nếu ta muốn chỉ định NIC cụ thể, có thể truy vấn Hệ điều hành để biết các giao diện thích hợp và tìm một địa chỉ trên giao diện muốn sử dụng. Khi tạo socket và liên kết nó với địa chỉ này, hệ điều hành sẽ sử dụng giao diện được liên kết. Ví dụ:

```
NetworkInterface nif = NetworkInterface.getByName("bge0");
Enumeration<InetAddress> nifAddresses = nif.getInetAddresses();
Socket socket = new Socket();
socket.bind(new InetSocketAddress(nifAddresses.nextElement(), 0));
socket.connect(new InetSocketAddress(address, port));
```

Ta cũng có thể sử dụng NetworkInterface để xác định giao diện cục bộ hỗ trợ multicast. Ví dụ:

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

6.3. LẬP TRÌNH MẠNG NGANG HÀNG P2P

Mạng điểm-điểm (**P2P**) là kiến trúc có các nút thường xuyên đóng vai trò vừa là máy chủ và vừa là máy khách. Mục tiêu chính của hệ thống P2P là loại bỏ nhu cầu về các máy chủ riêng biệt để quản lý hệ thống. Cấu hình của mạng P2P sẽ thay đổi động với các nút tham gia và rời khỏi mạng theo cách không thể đoán trước. Các nút có thể khác nhau về các yếu tố, chẳng hạn như tốc độ xử lý, hỗ trợ băng thông và khả năng lưu trữ. Thuật ngữ ngang hàng ngụ ý mức độ bình đẳng giữa các nút.

Có nhiều định nghĩa và cách giải thích khác nhau về mạng P2P. Chúng có thể được mô tả như một kiến trúc phi tập trung, thay đổi liên tục và tự điều chỉnh. Máy chủ có xu hướng cung cấp dịch vụ, trong khi máy khách yêu cầu dịch vụ. Một nút P2P thường đóng cả 2 vai trò. Một mạng P2P thuần túy sẽ không có các nút được chỉ định làm máy khách hoặc máy chủ. Trong thực tế, những mạng này rất hiếm. Hầu hết các mạng P2P dựa vào cơ sở trung tâm, chẳng hạn như máy chủ DNS, để được hỗ trợ.

Một số mạng nhất định có thể là sự kết hợp giữa kiến trúc máy khách/máy chủ và kiến trúc P2P thuần túy hơn, trong đó không bao giờ có một nút cụ thể hoạt động như một máy chủ "chính". Ví dụ: một P2P chia sẻ tệp có thể sử dụng các nút của mạng để tải tệp xuống, trong khi máy chủ có thể cung cấp thêm thông tin hỗ trợ.

P2P có thể được phân loại theo nhiều cách. Có một số loại phân loại phổ biến được sử dụng trong việc hiểu bản chất của mạng P2P. Phân loại dựa trên cách **lập chỉ mục**, cách tìm kiếm một nút:

- **Tập trung** : Máy chủ trung tâm theo dõi vị trí của dữ liệu giữa các điểm
- **Cục bộ** : Mỗi điểm ngang hàng theo dõi dữ liệu của riêng mình
- **Phân tán** : Tham chiếu dữ liệu được duy trì bởi nhiều điểm

Mạng P2P kết hợp sử dụng lược đồ lập chỉ mục tập trung. Mạng P2P thuần túy sử dụng các chỉ mục cục bộ hoặc phân tán.

Các thuật toán được sử dụng để xác định vị trí của thông tin trong một hệ thống. Hệ thống được phân cấp không có máy chủ ghi đè thực thi thuật toán. Thuật toán hỗ trợ một hệ thống tự tổ chức tự động cấu hình lại chính nó khi các nút được thêm vào và loại bỏ. Ngoài ra, các hệ thống này sẽ cân bằng lý tưởng giữa tải và tài nguyên khi thành viên mạng thay đổi.

Trong phần này sẽ giới thiệu các nội dung:

- Khái niệm và thuật ngữ P2P
- Hỗ trợ Java cho mạng P2P
- Bản chất của bảng băm phân tán
- Nền tảng FreePastry hỗ trợ ứng dụng P2P

6.3.1. Giới thiệu mạng ngang hàng P2P

One way of understanding a P2P network is to examine its characteristics. These include the following:

- Các nút đóng góp tài nguyên cho hệ thống, bao gồm:
    - Lưu trữ dữ liệu
    - Tài nguyên tính toán
- Cung cấp hỗ trợ cho một loạt các dịch vụ

- Dễ mở rộng và có khả năng chịu lỗi
- Hỗ trợ cân bằng tải các tài nguyên
- Có thể hỗ trợ ẩn danh hạn chế

Bản chất của hệ thống P2P là người dùng có thể không truy cập được vào một nút cụ thể để sử dụng dịch vụ hoặc tài nguyên. Khi các nút tham gia và rời khỏi hệ thống một cách ngẫu nhiên, một nút cụ thể có thể không khả dụng. Thuật toán sẽ xác định cách hệ thống phản hồi các yêu cầu.

Các chức năng cơ bản của hệ thống P2P bao gồm:

- Đăng ký các điểm trong một mạng lưới
- Khám phá đồng đẳng — quá trình xác định điểm nào có thông tin quan tâm
- Gửi tin nhắn giữa các điểm

Không phải tất cả các điểm đều thực hiện tất cả các chức năng này.

Các tài nguyên của hệ thống P2P được xác định bằng cách sử dụng **Mã định danh duy nhất toàn cục** (**GUID**) thường được tạo bằng cách sử dụng hàm băm an toàn được kiểm tra trong các thành phần DHT. GUID là một giá trị được tạo ngẫu nhiên, để tránh xung đột.

Các nút của P2P được tổ chức bằng cách sử dụng các **lớp phủ định tuyến**. Đây là một loại **phần mềm trung gian** định tuyến các yêu cầu đến nút thích hợp. Lớp phủ đề cập đến một mạng nằm trên cùng của mạng vật lý được xác định bởi các tài nguyên sử dụng địa chỉ IP. Có thể hình dung một mạng được cấu tạo trên một loạt các nút dựa trên IP. Tuy nhiên, lớp phủ là một tập hợp con của các nút này thường tập trung vào một nhiệm vụ duy nhất.

Lớp phủ định tuyến sẽ xem xét các yếu tố, chẳng hạn như số lượng nút giữa người dùng và tài nguyên và băng thông của kết nối, để xác định nút nào sẽ đáp ứng yêu cầu. Thông thường, một tài nguyên có thể bị trùng lặp hoặc thậm chí bị chia nhỏ trên nhiều nút. Lớp phủ định tuyến sẽ cố gắng cung cấp đường dẫn tối ưu đến tài nguyên.

Khi các nút tham gia và rời khỏi hệ thống, lớp phủ định tuyến cần tính đến những thay đổi này. Khi một nút tham gia vào hệ thống, nó có thể được yêu cầu đảm nhận một số trách nhiệm. Khi một nút rời đi, các phần khác của hệ thống có thể cần phải nhận một số trách nhiệm của các nút rời đi.

Phần này sẽ tổng quan ngắn gọn về các ứng dụng P2P khác nhau, sau đó sẽ là phần thảo luận về hỗ trợ của Java cho kiến trúc này. Việc sử dụng các bảng băm phân tán được trình bày và trình bày một bản kiểm tra chuyên sâu về FreePastry, sẽ cung cấp thông tin chi tiết về cách thức hoạt động của nhiều khuôn khổ P2P.

6.3.2. Các loại ứng dụng P2P

Có rất nhiều ứng dụng dựa trên mạng P2P. Chúng có thể được sử dụng cho những việc sau:

- **Phân phối nội dung** : Chia sẻ tệp (tệp, nhạc, video, hình ảnh)
- **Tính toán phân tán** : Một vấn đề được chia thành các tác vụ nhỏ hơn và được thực hiện theo cách song song
- **Cộng tác** : Người dùng làm việc cùng nhau để giải quyết một vấn đề chung
- **Nền tảng** : Hệ thống trên đó các ứng dụng P2P được xây dựng, chẳng hạn như JXTA và Pastry

Tính toán phân tán tận dụng sức mạnh của số lượng lớn các máy tính nhỏ để thực hiện một tác vụ. Các vấn đề có thể giải quyết được với cách tiếp cận này đòi hỏi chúng phải được chia nhỏ thành các đơn vị nhỏ hơn và sau đó thực hiện đồng thời trên nhiều máy. Kết quả của các nhiệm vụ nhỏ hơn này sau đó cần được kết hợp để tạo ra một kết quả cuối cùng. Mạng P2P hỗ trợ một số ứng dụng mạng, ví dụ:

- **Skype** : Ứng dụng hội nghị truyền hình
- **Freecast** : Chương trình âm thanh phát trực tuyến ngang hàng
- **BitTorrent** : Hệ thống chia sẻ tệp ngang hàng phổ biến
- **Tor** : Chương trình bảo vệ danh tính của người dùng
- **Haihaisoft** : Ứng dụng được sử dụng để phân phối các chương trình TV được ghi sẵn
- **WoW** : Ứng dụng sử dụng P2P để cập nhật trò chơi
- **YaCy** : Công cụ tìm kiếm và trình thu thập thông tin web
- **Octoshape** : Ứng dụng hỗ trợ truyền hình trực tiếp

6.3.3. Xây dựng ứng dụng trên mạng ngang hàng

Java support beyond the low-level socket support that was detailed in earlier chapters consists of various frameworks. These range from well-known frameworks, such as JXTA, to small limited-capability protocols. These frameworks provide the basis for more specialized applications.

The following table lists several of these frameworks:

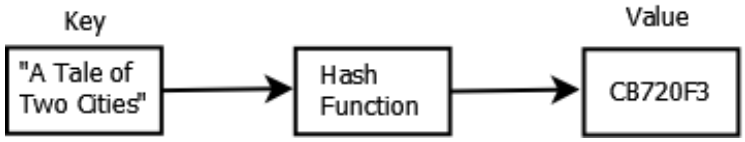
Nền tảng P2P	URL
TomP2P	`http://tomp2p.net/`
JXTA	`https://jxta.kenai.com/`
Hive2Hive	`https://github.com/Hive2Hive/Hive2Hive`
jnmp2p	`https://code.google.com/p/jnmp2p/`
FlexGP	`http://flexgp.github.io/flexgp/javalibrary.html`
JMaay	`http://sourceforge.net/projects/jmaay/`
P2P-MPI	`http://grid.u-strasbg.fr/p2pmpi/`

Pastry	`http://www.freepastry.org/`

Các nền tảng này sử dụng một số thuật toán để định tuyến thông điệp giữa các nút. Bảng băm thường là cơ sở của các nền tảng này.

Một **Bảng băm phân tán** (**DHT**) sử dụng một cặp khóa/giá trị để xác định vị trí các tài nguyên trong một mạng. Chức năng ánh xạ này được trải rộng trên các nút làm cho nó được phân phối. Kiến trúc này cho phép các mạng P2P dễ dàng mở rộng quy mô đến một số lượng lớn các nút và xử lý các nút tham gia và rời khỏi mạng một cách ngẫu nhiên. DHT là cơ sở để hỗ trợ các dịch vụ P2P cốt lõi. Nhiều ứng dụng sử dụng DHT, bao gồm BitTorrent, Freenet và YaCy.

Hình sau minh họa ánh xạ khóa với giá trị. Khóa thường là một chuỗi chứa danh tính của tài nguyên, chẳng hạn như tên của một cuốn sách; và giá trị là một số được tạo để đại diện cho tài nguyên. Số có thể được sử dụng để định vị tài nguyên trong mạng và có thể tương ứng với định danh của một nút.



Thực tế, mạng P2P đã được sử dụng từ lâu. Sự phát triển của các mạng này được phản ánh trong cách các tài nguyên được lập bản đồ như Napster, Gnutella và Freenet:

- Napster (https://en.wikipedia.org/wiki/Napster) là hệ thống cung cấp nội dung P2P quy mô lớn đầu tiên. Nó sử dụng một máy chủ để theo dõi các nút trong mạng. Các nút giữ dữ liệu thực tế. Khi một máy khách cần dữ liệu này, máy chủ sẽ tìm kiếm tập hợp các nút hiện tại đang lưu giữ dữ liệu và vị trí của nút này sẽ được gửi lại cho máy khách. Sau đó máy khách sẽ liên hệ với nút đang giữ dữ liệu.
- Gnutella (https://web.archive.org/web/20080525005017 , http: // www.Gnutella.com/) không sử dụng máy chủ trung tâm nhưng quảng bá đến mọi nút trong mạng. Điều này dẫn đến việc mạng tràn ngập tin nhắn, cách tiếp cận này đã được cập nhập trong các phiên bản sau.
- Freenet (https://freenetproject.org/) sử dụng sơ đồ định tuyến dựa trên khóa heuristic và tập trung vào các vấn đề kiểm duyệt và ẩn danh. Tuy nhiên, DHS sử dụng phương pháp định tuyến dựa trên khóa có cấu trúc hơn dẫn đến những điều sau:
    - Phân quyền
    - Khả năng chịu lỗi
    - Khả năng mở rộng
    - Hiệu quả

Tuy nhiên, DHT không hỗ trợ tìm kiếm đối sánh chính xác. Nếu loại tìm kiếm này là cần thiết, thì nó phải được thêm vào.

Một **Không gian khóa** là một tập hợp các chuỗi 160-bit (khóa) được sử dụng để xác định một phần tử. **Phân vùng không gian khóa** là quá trình chia tách không gian khóa giữa các nút của mạng. Một mạng lớp phủ kết nối các nút.

Thuật toán băm thường được sử dụng là **Thuật toán băm an toàn (SHA-1)** (https://en.wikipedia.org/wiki/SHA-1). SHA-1 được thiết kế bởi NSA tạo ra một giá trị băm 160 bit được gọi là tóm lược thông báo. Hầu hết các P2P không yêu cầu nhà phát triển phải thực hiện rõ ràng chức năng băm. Tuy nhiên, các mạng thường hướng dẫn cách thực hiện như thế nào. Sau đây là một ví dụ về việc sử dụng Java để tạo thông báo.

Phương thức *getInstance* của lớp *MessageDigest* chấp nhận một chuỗi xác định thuật toán để sử dụng và trả về một đối tượng MessageDigest . Phương thức *update* yêu cầu một mảng byte chứa khóa để băm. Trong ví dụ dưới, một chuỗi được sử dụng. Phương thức *digest* trả về một mảng byte giữ giá trị băm. Mảng byte sau đó được hiển thị dưới dạng số thập lục phân:

```
String message = "String to be hashed";
try {
    MessageDigest messageDigest = MessageDigest.getInstance("SHA-1");
    messageDigest.update(message.getBytes());
    byte[] digest = messageDigest.digest();
    StringBuffer buffer = new StringBuffer();
    for (byte element : digest) {
        buffer.append(Integer.toString((element & 0xff) + 0x100,
                16).substring(1));
    }
    System.out.println("Hex format : " + buffer.toString());
} catch (NoSuchAlgorithmException ex) {
   ex.printStackTrace();
}
```

Chạy chương trình trên ta nhận được kết quả sau:

Hex format : 434d902b6098ac050e4ed79b83ad93155b161d72

Để lưu trữ dữ liệu, ta có thể sử dụng tên tệp để tạo khóa. Hàm put có thể được sử dụng để lưu trữ dữ liệu:

```
put(key, data)
```

Để truy xuất dữ liệu tương ứng với một khóa, ta có thể sử dụng hàm get:

```
data = get(key)
```

Tất cả các nút trong lớp phủ hoặc chứa dữ liệu được đại diện bởi khóa hoặc là nút gần nút chứa dữ liệu. Thuật toán định tuyến xác định nút tiếp theo sẽ truy cập trên đường đến nút chứa dữ liệu.

6.3.4. Một số nền tảng hỗ trợ xây dựng ứng dụng P2P

a. JDHT

Để sử dụng JDHT, cần có các tệp JAR được liệt kê trong bảng sau. Tệp dks.jar là tệp jar chính được sử dụng, hai tệp JAR còn lại cũng có thể được sử dụng bởi JDHT:

JAR	***Site***
dks.jar	• http://dks.sics.se/jdht/ • https://www.ac.upc.edu/projects/cms/browser/cms/trunk/lib/dks.jar?rev=2
xercesImpl.jar	http://www.java2s.com/Code/Jar/x/ DownloadxercesImpljar.htm
Apache log4j.1.2.17	https://logging.apache.org/log4j/1.2/ download.html

Ví dụ sau tạo một đối tượng JDHT, sử dụng cổng 4440 làm mặc định, và phương thức put để thêm một cặp khóa/giá trị vào bảng:

```
public class JDHTExample {

    public static void main(String[] arg) {
        try {
            JDHT DHTExample = new JDHT();
            DHTExample.put("Java SE API",
                    "http://docs.oracle.com/javase/8/docs/api/");
            System.out.println(((JDHT) DHTExample).getReference());
            Scanner scanner = new Scanner(System.in);
            System.out.println("Press Enter to terminate application: ");
            scanner.next();
            DHTExample.close();
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }
}
```

Để máy khách kết nối với đối tượng này, ta cần tham chiếu đến nút qua lệnh:

```
System.out.println(((JDHT) DHTExample).getReference());
```

Đoạn mã sau sẽ tiếp tục chạy chương trình cho đến khi người dùng chấm dứt. Phương thức *close* sau đó được sử dụng để đóng bảng:

```
Scanner scanner = new Scanner(System.in);
System.out.println("Press Enter to terminate application: ");
```

```
scanner.next();
DHTExample.close();
```

Ứng dụng khách được mô tả như sau. Một thực thể JDHT mới được tạo qua cổng khác. Đối số thứ hai tham chiếu đến ứng dụng đầu tiên. Ta sẽ cần sao chép tham chiếu và dán nó vào máy khách. Một tham chiếu khác sẽ được tạo mỗi khi ứng dụng đầu tiên được thực thi:

```
public class JDHTClient {

    public static void main(String[] args) {
        try {
            JDHT myDHT = new JDHT(5550, "dksref://192.168.1.9:4440"
                    + "/0/2179157225/0/1952355557247862269");
            String value = (String) myDHT.get("Java SE API");
            System.out.println(value);
            myDHT.close();
        } catch (IOException | DKSTooManyRestartJoins |
                DKSIdentifierAlreadyTaken | DKSRefNoResponse ex) {
            ex.printStackTrace();
        }
    }
}
```

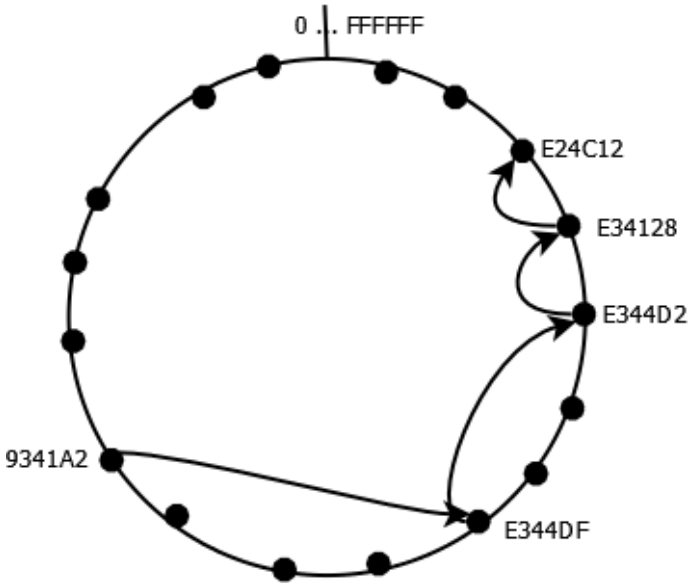
Ở đây ta sử dụng phương thức get để lấy giá trị được liên kết với khóa. Giá trị sau đó được hiển thị và ứng dụng được đóng.

a. FreePastry

Pastry (http://www.freepastry.org/) là một hệ thống lớp phủ định tuyến P2P. FreePastry (http://www.freepastry.org/FreePastry/) là một triển khai mã nguồn mở của Pastry và đủ đơn giản để ta có thể sử dụng để hiểu nhiều tính năng của hệ thống P2P. Pastry sẽ định tuyến thông điệp với một mạng lưới *n* nút trong *O(n)* bước. Nghĩa là, với một mạng các nút, nó yêu cầu tối đa là logarit cơ số 2 trong số *n* bước để đến được nút. Đây là một cách tiếp cận định tuyến hiệu quả. Tuy nhiên, trong trong một số trường hợp có thể chỉ cần đi qua ba nút để đến một tài nguyên, nó có thể yêu cầu một số lượng IP đáng kể để đến được nó.

Pastry sử dụng khái niệm **bộ lá** trong quá trình định tuyến. Mỗi nút có một bộ lá. Bộ lá là tập hợp các GUIDS và địa chỉ IP của các nút gần nút này nhất về mặt số học. Các nút được sắp xếp hợp lý trong một vòng tròn, như hình bên dưới.

Trong hình, mỗi dấu chấm đại diện cho một nút có định danh. Các địa chỉ được sử dụng ở đây nằm trong khoảng từ 0 đến FFFFFF. Các địa chỉ thực nằm trong khoảng từ 0 đến 2128. Nếu một thông báo đại diện cho một yêu cầu bắt nguồn từ địa chỉ 9341A2 và cần được gửi đến địa chỉ E24C12, thì dựa trên địa chỉ số, bộ định tuyến lớp phủ có thể định tuyến các thông báo qua các nút trung gian, như được mô tả bởi mũi tên:



Các ứng dụng này sử dụng API để hỗ trợ việc sử dụng chúng.

Để minh họa cách FreePastry hỗ trợ ứng dụng P2P, ta sẽ tạo hai nút và trình bày cách chúng có thể gửi và nhận tin nhắn. Ba lớp sau sẽ được sử dụng để xây dựng ứng dụng bootstrap:

- FreePastryExample: Lớp để khởi động mạng
- FreePastryApplication: Lớp thực thi chức năng của nút
- PastryMessage: Lớp thông báo được gửi giữa các nút

Có một số thành phần được sử dụng với các ứng dụng FreePastry. Bao gồm các:

- **Môi trường** : Lớp này đại diện cho môi trường của ứng dụng
- **Cổng ràng buộc** : Cổng này đại diện cho cổng cục bộ mà ứng dụng sẽ liên kết với
- **Cổng khởi động** : Đây là cổng khởi động được sử dụng cho InetAddress của nút
- **Địa chỉ khởi động** : Đây là địa chỉ IP của nút khởi động

Lớp FreePastryExample được định nghĩa như dưới. Nó chứa một phương thức main và một phương thức khởi tạo:

```
public class FreePastryExample {

    public FreePastryExample(int bindPort,
            InetSocketAddress bootAddress,
            Environment environment) throws Exception {
        NodeIdFactory nidFactory = new RandomNodeIdFactory(environment);
        PastryNodeFactory factory = new SocketPastryNodeFactory(nidFactory,
            bindPort, environment);
        PastryNode node = factory.newNode();
```

```
        FreePastryApplication application = new FreePastryApplication(node);
        node.boot(bootAddress);

        System.out.println("Node " + node.getId().toString() + " created");
        System.out.println("Node " + node.getId().toStringFull() + "
            created");
        environment.getTimeSource().sleep(10000);

        environment.getTimeSource().sleep(10000);
        LeafSet leafSet = node.getLeafSet();
        Collection<NodeHandle> collection = leafSet.getUniqueSet();
        for (NodeHandle nodeHandle : collection) {
            application.routeMessageDirect(nodeHandle);
            environment.getTimeSource().sleep(1000);
        }
    }

    public static void main(String[] args) throws Exception {
        Environment environment = new Environment();
        environment.getParameters().setString("nat_search_policy", "never");

        try {
            int bindPort = 9001;
            int bootPort = 9001;
            InetAddress bootInetAddress =
                  InetAddress.getByName("192.168.1.9");
            InetSocketAddress bootAddress
                    = new InetSocketAddress(bootInetAddress, bootPort);
            System.out.println("InetAddress: " + bootInetAddress);

            FreePastryExample freePastryExample
                    = new FreePastryExample(bindPort, bootAddress,
            environment);
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
}
```

Trong phương thức main, một thể hiện của lớp Environment được tạo trước. Lớp này giữ các cài đặt tham số cho nút. Tiếp theo, chính sách tìm kiếm NAT được thiết lập không bao giờ, cho phép ta sử dụng chương trình trong mạng LAN cục bộ mà không gặp khó khăn:

```
public static void main(String[] args) throws Exception {
      Environment environment = new Environment();
      environment.getParameters().setString("nat_search_policy", "never");
      ...
}
```

Các cổng và của thực thể InetSocketAddress được khởi tạo. Ta sẽ đặt cả hai cổng thành cùng một số tại thời điểm này. Ta đã sử dụng địa chỉ IP 192.168.1.14 để khởi tạo đối tượng InetAddress. Thay vào đó, ta sẽ cần sử dụng địa chỉ máy của mình. Đây là địa chỉ mạng LAN cục bộ. Không sử dụng 127.0.0.1 vì nó sẽ không hoạt động bình thường.

Đối tượng InetAddress cùng với giá trị bootPort được sử dụng để tạo cá thể InetSocketAddress. Tất cả điều này được đặt trong khối bắt ngoại lệ:

```
try {
    int bindPort = 9001;
    int bootPort = 9001;
    //    192.168.1.14
    InetAddress bootInetAddress = InetAddress.getByName("192.168.1.9");
    InetSocketAddress bootAddress = new InetSocketAddress(bootInetAddress,
                                        bootPort);

   ....
} catch (Exception ex) {
     ex.printStackTrace();

}
```

Việc cuối cùng là tạo một thể hiện của lớp FreePastryExample bằng cách gọi hàm tạo:

```
FreePastryExample freePastryExample = new FreePastryExample(bindPort,
                                        bootAddress, environment);
```

Hàm tạo sẽ tạo và khởi chạy ứng dụng của nút. Để thực hiện điều này, ta cần tạo một thực thể PastryNode và đính kèm ứng dụng vào đó. Để tạo nút, ta sẽ sử dụng *factory*.

Mỗi nút cần một ID duy nhất. Lớp RandomNodeIdFactory tạo ID dựa trên môi trường hiện tại. Sử dụng đối tượng này với cổng liên kết và môi trường, một đối tượng của SocketPastryNodeFactory sẽ được tạo. Tiếp đến, phương thức newNode được gọi để tạo đối tượng PastryNode của ta::

```
NodeIdFactory nidFactory = new RandomNodeIdFactory(environment);
PastryNodeFactory factory = new SocketPastryNodeFactory(nidFactory, bindPort,
                                    environment);
PastryNode node = factory.newNode();
...
}
```

Tiếp theo, thực thể của lớp FreePastryApplication được tạo và nút và sử dụng phương thức *boot* :

```
FreePastryApplication application = new FreePastryApplication(node);
node.boot(bootAddress);
...
```

Sau đó, ID của nút được hiển thị như trong mã tiếp theo. Vì sẽ có nhiều nút trong mạng, ta tạm dừng 10 giây để cho phép các nút khác bắt đầu. Ta đã sử dụng bộ đếm thời gian FreePastry để thực hiện độ trễ này. Một ID nút ngẫu nhiên được tạo và thông báo routeMessage của ứng dụng được gọi để gửi thông báo đến nút đó:

```
System.out.println("Node " + node.getId().toString() + " created");
environment.getTimeSource().sleep(10000);
Id randomId = nidFactory.generateNodeId();
application.routeMessage(randomId);
```

Trước khi thực thi chương trình, ta cần phát triển lớp ứng dụng.

Lớp FreePastryApplication thực thi giao diện Application và chức năng của nút, như đoạn mã dưới.

```
public class FreePastryApplication implements Application {

    protected Endpoint endpoint;
    private final String message;
    private final String instance = "Instance ID";

    public FreePastryApplication(Node node) {
        this.endpoint = node.buildEndpoint(this, instance);
        this.message = "Hello there! from Instance: "
                + instance + " Sent at: [" + getCurrentTime() + "]";
        this.endpoint.register();
    }

    private long getCurrentTime() {
        return this.endpoint
                .getEnvironment()
                .getTimeSource()
                .currentTimeMillis();
    }

    public void routeMessage(Id id) {
        System.out.println(
                "Message Sent\n\tCurrent Node: " + this.endpoint.getId()
                + "\n\tDestination: " + id
                + "\n\tTime: " + getCurrentTime());
        Message msg = new PastryMessage(endpoint.getId(), id, message);
```

```
        endpoint.route(id, msg, null);
    }

    public void routeMessageDirect(NodeHandle nh) {
        System.out.println("Message Sent Direct\n\tCurrent Node: "
                + this.endpoint.getId() + " Destination: " + nh
                + "\n\tTime: " + getCurrentTime());
        Message msg = new PastryMessage(endpoint.getId(), nh.getId(),
                "DIRECT-" + message);
        endpoint.route(null, msg, nh);
    }

    @Override
    public void deliver(Id id, Message message) {
        System.out.println("Message Received\n\tCurrent Node: " +
this.endpoint.getId()
                + "\n\tMessage: " + message + "\n\tTime: " +
getCurrentTime());
    }

    @Override
    public void update(NodeHandle handle, boolean joined) {
    }

    @Override
    public boolean forward(RouteMessage message) {
        return true;
    }

    @Override
    public String toString() {
        return "FreePastryApplication " + endpoint.getId();
    }


}
```

Lớp PastryMessage thực thi giao Message interface, như dưới:

```
    public class PastryMessage implements Message {
      private final Id from;
      private final Id to;
      private final String messageBody;

      public PastryMessage(Id from, Id to, String messageBody) {
        this.from = from;
```

```
    this.to = to;
    this.messageBody = messageBody;
  }

  @Override
  public String toString() {
    return "From: " + this.from
           + " To: " + this.to
           + " [" + this.messageBody + "]";
  }

  @Override
  public int getPriority() {
    return Message.LOW_PRIORITY;
  }
}
```

Trong ví dụ trên, ta không cung cấp một địa chỉ cụ thể cho các tin nhắn. Thay vào đó, địa chỉ được tạo ra một cách ngẫu nhiên. Ứng dụng này đã trình bày các yếu tố cơ bản của một ứng dụng FreePastry. Các lớp bổ sung được sử dụng để tạo điều kiện giao tiếp giữa các nút, chẳng hạn như mô hình xuất bản / cung cấp được Scribe hỗ trợ.

Ta có thể bắt đầu một nút thứ hai bằng cách sử dụng cùng một chương trình, nhưng sẽ cần sử dụng một cổng ràng buộc khác để tránh xung đột. Thông báo được gửi bởi một trong hai nút sẽ không nhất thiết phải được nhận bởi nút kia. Đây là kết quả của các tuyến được tạo bởi FreePastry.

Để gửi một tin nhắn trực tiếp đến một nút, cần mã định danh của nó. Để lấy mã định danh của một nút từ xa, ta cần sử dụng một bộ lá. Tập hợp này không hoàn toàn là một tập hợp vì đối với các mạng nhỏ, chẳng hạn như mạng ta đang sử dụng, cùng một nút có thể xuất hiện hai lần.

Lớp LeafSet đại diện cho tập hợp này và có phương thức get sẽ trả về một thể hiện NodeHandle cho mỗi nút. Ta có thể gửi tin nhắn đến các nút nếu có nút điều khiển này.

Để chứng minh cách tiếp cận này, ta có thể thêm phương thức sau vào lớp FreePastryApplication. Điều này tương tự như phương thức routeMessage, nhưng nó sử dụng một nút xử lý làm đối số của phương thức route:

```
public void routeMessageDirect(NodeHandle nh) {
        System.out.println("Message Sent Direct\n\tCurrent Node: "
                + this.endpoint.getId() + " Destination: " + nh
                + "\n\tTime: " + getCurrentTime());
        Message msg = new PastryMessage(endpoint.getId(), nh.getId(),
                "DIRECT-" + message);
        endpoint.route(null, msg, nh);
```

```
}
```

6.3.5. Kết luận

Phần này đã giới thiệu mạng P2P và một sô phương pháp, nền tảng để xây dựng các ứng dụng trên mạng P2P. Mạng P2P là mạng có kiến trúc trong đó tất cả các nút được xem là như nhau.

Bảng băm phân tán được sử dụng ddeewr hỗ trợ xác định và định vị các nút trong mạng. Các thuật toán định tuyến sử dụng bảng này để thực hiện các yêu cầu bằng cách gửi thông điệp giữa các nút.

Có sẵn một số nền tảng P2P mã nguồn mở dựa trên Java cho phép xây dựng ứng dụng trên đó. Nền tảng FreePastry đã được sử dụng để xây dựng ứng dụng minh họa.

6.4. LẬP TRÌNH TRUYỀN THÔNG MULTICAST

6.4.1. Giới thiệu truyền thông multicast và lớp MulticastSocket

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

- Videoconferencing
- Usenet news
- Computer configuration

Các địa chỉ multicast:

- 224.0.0.1: Tất cả các hệ thống ở trên mạng con cục bộ
- 224.0.0.2 : Tất cả các router tren mạng con cục bộ.
- 224.0.0.11: Các tác tử di động(agent) trên mạng con cục bộ
- 224.0.1.1 : Giao thức định thời mạng
- 224.0.1.20: Thử nghiệm mà không cho vượt ra khỏi mạng con cục bộ
- 224.2.X.X (Multicast Backbone on the Internet (MBONE)): Được sử dụng cho audio và video quảng bá trên mạng Internet .

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

*public class **MulticastSocket** extends* DatagramSocket

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

Cấu tử lớp MulticastSocket
`MulticastSocket()` Tạo socket muticast

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

Các phương thức của lớp MulticastSocket	
`InetAddress`	`getInterface()` Lấy địa chỉ giao tiếp mạng được sử dụng cho các gói tin multicast
`boolean`	`getLoopbackMode()` Lấy chuỗi thiết đặt đối local loopback của gói tin multicast
`NetworkInterface`	`getNetworkInterface()` Lấy tập giao tiếp mạng multicast
`int`	`getTimeToLive()` Lấy tham số time to live mặc định của các gói tin multicast gửi ra socket
`byte`	`getTTL()` Lấy tham số time- to -live
`void`	`joinGroup(InetAddress mcastaddr)` Ghép nhóm multicast
`void`	`joinGroup(SocketAddress mcastaddr, NetworkInterface netIf)` Ghép nhóm multicast cụ thể tại giao tiếp mạng cụ thể
`void`	`leaveGroup(InetAddress mcastaddr)` Loại bỏ một nhóm multicast
`void`	`leaveGroup(SocketAddress mcastaddr, NetworkInterface netIf)` Loại bỏ một nhóm multicast trên giao tiếp mạng cục bộ được chỉ ra.
`void`	`send(DatagramPacket p, byte ttl)` Gửi gói tin
`void`	`setInterface(InetAddress inf)` Đặt giao tiếp mạng multicast được sử dụng bởi phương thức mà hành vi của nó bị ảnh hưởng bởi giá trị của gao tiếp mạng.
`void`	`setLoopbackMode(boolean disiao tiếp mangable)` Cho phép hoặc làm mất hiệu lực vòng phản hồi cục bộ của lược đồ dữ liệu multicast
`void`	`setNetworkInterface(NetworkInterface netIf)` Chỉ ra giao tiếp mạng để gửi các lược đồ dữ liệu multicast qua
`void`	`setTimeToLive(int ttl)` Thiết đặt tham số TTL mặc định cho các gói tin multicast gửi trên MulticastSocket nhằm mục đích điều khiển phạm vi multicast.

void	setTTL(byte ttl) Thiết đặt tham số TTL

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

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

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

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

a. Ví dụ gửi dữ liệu multicast

```
import java.net.*;
// Which port should we send to
int port = 5000;
// Which address
String group = "225.4.5.6";
// Which ttl
int ttl = 1;
// Create the socket but we don't bind it as we are only going to send
data
MulticastSocket s = new MulticastSocket();
// Note that we don't have to join the multicast group if we are only
// sending data and not receiving
// Fill the buffer with some data
byte buf[] = byte[10];
for (int i=0; i<buf.length; i++) buf[i] = (byte)i;
// Create a DatagramPacket
DatagramPacket pack = new DatagramPacket(buf, buf.length,
                                    InetAddress.getByName(group), port);
// Do a send. Note that send takes a byte for the ttl and not an int.
s.send(pack,(byte)ttl);
```

```
// And when we have finished sending data close the socket
s.close();
```

b. Ví dụ nhận dữ liệu multicast

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

6.5. LẬP TRÌNH BẢO MẬT VỚI SSL

6.5.1. Giới thiệu SSL

a. SSL

SSL (Secure Socket Layer) là giao thức đa mục đích được thiết kế để tạo ra các giao tiếp giữa hai chương trình ứng dụng trên một cổng định trước (socket 443) nhằm mã hoá toàn bộ thông tin đi/đến, được sử dụng trong giao dịch điện tử như truyền số liệu thẻ tín dụng, mật khẩu, số bí mật cá nhân (PIN) trên Internet.

Trong các giao dịch điện tử trên mạng và trong các giao dịch thanh toán trực tuyến, thông tin/dữ liệu trên môi trường mạng Internet không an toàn thường được bảo đảm bởi cơ chế bảo mật thực hiện trên tầng vận tải có tên Lớp cổng bảo mật SSL (Secure Socket Layer) - một giải pháp kỹ thuật hiện nay được sử dụng khá phổ biến trong các hệ điều hành mạng máy tính trên Internet. Giao thức SSL được hình thành và phát triển đầu tiên năm 1994 bởi nhóm nghiên cứu Netscape dẫn dắt bởi Elgammal, và ngày nay đã trở thành chuẩn bảo mật thực hành trên mạng Internet. Phiên bản SSL hiện nay là 3.0 và vẫn đang tiếp tục được bổ sung và hoàn thiện. Tương tự như

SSL, một giao thức khác có tên là Công nghệ truyền thông riêng tư PCT (Private Communication Technology) được đề xướng bởi Microsoft, hiện nay cũng được sử dụng rộng rãi trong các mạng máy tính chạy trên hệ điều hành Windows NT. Ngoài ra, một chuẩn của Nhóm đặc trách kỹ thuật Internet IETF (Internet Engineering Task Force) có tên là Bảo mật lớp giao vận TLS (Transport Layer Security) dựa trên SSL cũng được hình thành và xuất bản dưới khuôn khổ nghiên cứu của IETF Internet Draft được tích hợp và hỗ trợ trong sản phẩm của Netscape.

b. Khóa – Key

Định nghĩa khóa

Khóa (key) là một thông tin quan trọng dùng để mã hóa thông tin hoặc giải mã thông tin đã bị mã hóa. Có thể hiểu nôm na khóa giống như là mật khẩu(password).

Độ dài khóa – Key Length

Độ dài khóa được tính theo bit: 128 bits, 1024 bits hay 2048 bits,... Khóa càng dài thì càng khó phá. Chẳng hạn như khóa RSA 1024 bits đồng nghĩa với việc chọn 1 trong 2^{1024} khả năng.

Password và PassParse

Password và passparse gần giống nhau. Password không bao giờ hết hạn(expire). Passparse chỉ có hiệu lực trong một khoảng thời gian nhất định có thể là 5 năm, 10 năm hay chỉ là vài ba ngày. Sau thời gian đó, phải thay đổi lại mật khẩu mới. Nói chung, mọi thứ trong SSL như passparse, khóa, giấy chứng nhận, chữ kí số (sẽ nói sau), ... đều chỉ có thời hạn sử dụng nhất định. Passparse được dùng để mở (mã hóa/giải mã) khóa riêng.

c. Thuật toán mã hóa

Mã hóa (encrypt) và giải mã (decrypt) thông tin dùng các hàm toán học đặt biệt. Được biết đến với cái tên là thuật toán mã hóa (cryptographic algorithm) và thường được gọi tắt là cipher. Các thuật toán mã hoá và xác thực của SSL được sử dụng bao gồm (phiên bản 3.0):

(1) DES - Chuẩn mã hoá dữ liệu (ra đời năm 1977), phát minh và sử dụng của chính phủ Mỹ.

(2) DSA - Thuật toán chữ ký điện tử, chuẩn xác thực điện tử, phát minh và sử dụng của chính phủ Mỹ.

(3) KEA - Thuật toán trao đổi khoá, phát minh và sử dụng của chính phủ Mỹ.

(4) MD5 - Thuật toán tạo giá trị "băm" (message digest), phát minh bởi Rivest.

(5) RC2, RC4 - Mã hoá Rivest, phát triển bởi công ty RSA Data Security.

(6) RSA - Thuật toán khoá công khai, cho mã hoá và xác thực, phát triển bởi Rivest, Shamir và Adleman.

(7) RSA key exchange - Thuật toán trao đổi khoá cho SSL dựa trên thuật toán RSA.

(8) SHA-1 - Thuật toán hàm băm an toàn, phát triển và sử dụng bởi chính phủ Mỹ.

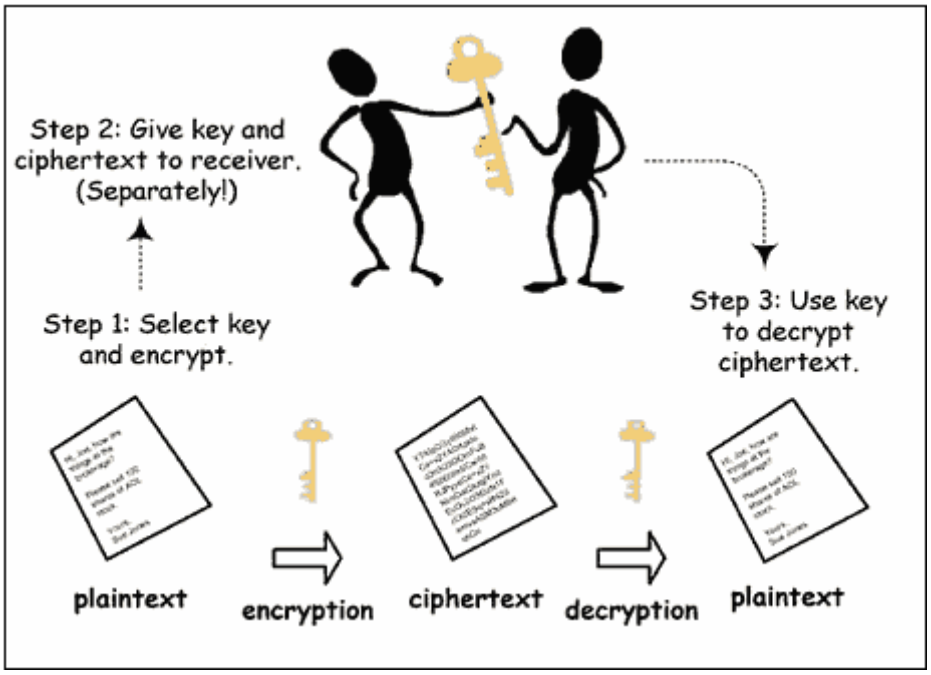
(9) SKIPJACK - Thuật toán khoá đối xứng phân loại được thực hiện trong phần cứng Fortezza, sử dụng bởi chính phủ Mỹ;

(10) Triple-DES - Mã hoá DES ba lần.

d. Các phương pháp mã hóa

Có hai phương pháp mã hóa được sử dụng phổ biến hiện nay là mã hóa bằng khóa đối xứng và mã hóa dùng cặp khóa chung - khóa riêng..

Mã hóa bằng khóa đối xứng (symmetric-key)



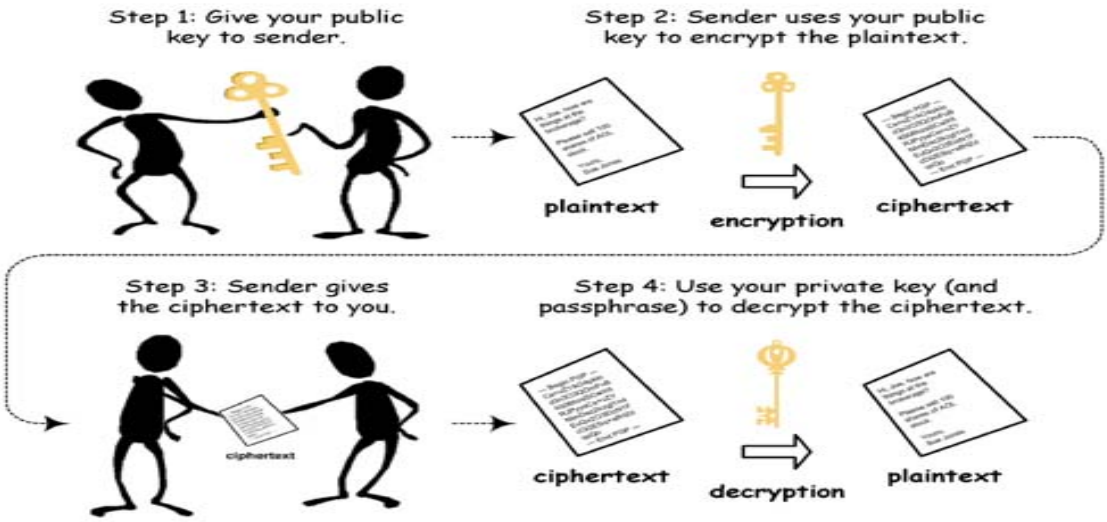
Hình 6.7. Mã hoá bằng khoá đối xứng

Khóa dùng để mã hóa cũng là khóa dùng để giải mã.

Một khe hở trong mã hóa đối xứng là bạn phải chuyển khóa cho người nhận để họ có thể giải mã. Việc chuyển khóa không được mã hóa qua mạng là một điều cực kì mạo hiểm. Nhỡ như khóa này rơi vào tay người khác thế là họ có thể giải mã được thông tin mà đã chuyển đi. Phương pháp mã hóa bằng khóa chung - khóa riêng ra đời nhằm giải quyết vấn đề này.

Thay vì chỉ có một khóa duy nhất dùng chung cho mã hóa và giải mã, sẽ có một cặp khóa gồm khóa chung chỉ dùng để mã hóa và khóa riêng chỉ dùng để giải mã. Khi người A muốn gửi thông điệp cho người B thì người B cần biết khóa chung của người A. (Khóa này được người A công bố công khai). Người B mã hóa các thông tin gởi đến người A bằng khóa chung của người A. Chỉ có người A mới có khóa riêng để giải mã các thông tin này. Nhỡ như thông tin này có rơi vào tay người khác thì họ cũng không thể giải mã được vì chỉ có người A mới có khóa riêng dành cho việc giải mã đúng thông điệp trên.

Mã hóa dùng cặp khóa chung – khóa riêng



Hình 6.8. Mã hoá công khai

e. Cơ chế làm việc của SSL – SSL Protocol

Điểm cơ bản của SSL là được thiết kế độc lập với tầng ứng dụng để đảm bảo tính bí mật, an toàn và chống giả mạo luồng thông tin qua Internet giữa hai ứng dụng bất kỳ, thí dụ như webserver và các trình duyệt (browser), do đó được sử dụng rộng rãi trong nhiều ứng dụng khác nhau trên môi trường Internet. Toàn bộ cơ chế hoạt động và hệ thống thuật toán mã hoá sử dụng trong SSL được phổ biến công khai, trừ khóa chia sẻ tạm thời được sinh ra tại thời điểm trao đổi giữa hai ứng dụng là tạo ngẫu nhiên và bí mật đối với người quan sát trên mạng máy tính. Ngoài ra, giao thức SSL còn đòi hỏi ứng dụng chủ phải được chứng thực bởi một đối tượng lớp thứ ba (CA) thông qua chứng chỉ điện tử (digital certificate) dựa trên mật mã công khai (thí dụ RSA).

Sau đây ta xem xét một cách khái quát cơ chế hoạt động của SSL để phân tích cấp độ an toàn của nó và các khả năng áp dụng trong các ứng dụng nhạy cảm, đặc biệt là các ứng dụng về thương mại và thanh toán điện tử.

Giao thức SSL dựa trên hai nhóm con giao thức là giao thức "bắt tay" (handshake protocol) và giao thức "bản ghi" (record protocol). Giao thức bắt tay xác định các tham số giao dịch giữa hai đối tượng có nhu cầu trao đổi thông tin hoặc dữ liệu, còn giao thức bản ghi xác định khuôn dạng cho tiến hành mã hoá và truyền tin hai chiều giữa hai đối tượng đó. Khi hai ứng dụng máy tính, thí dụ giữa một trình duyệt web và máy chủ web, làm việc với nhau, máy chủ và máy khách sẽ trao đổi "lời chào" (hello) dưới dạng các thông điệp cho nhau với xuất phát đầu tiên chủ động từ máy chủ, đồng thời xác định các chuẩn về thuật toán mã hoá và nén số liệu có thể được áp dụng giữa hai ứng dụng. Ngoài ra, các ứng dụng còn trao đổi "số nhận dạng/khoá theo phiên" (session ID, session key) duy nhất cho lần làm việc đó. Sau đó ứng dụng khách (trình duyệt) yêu cầu có chứng chỉ điện tử (digital certificate) xác thực của ứng dụng chủ (web server).

Chứng chỉ điện tử thường được xác nhận rộng rãi bởi một cơ quan trung gian (Thẩm quyền xác nhận CA - Certificate Authority) như RSA Data Sercurity hay VeriSign Inc., một dạng tổ chức

độc lập, trung lập và có uy tín. Các tổ chức này cung cấp dịch vụ "xác nhận" số nhận dạng của một công ty và phát hành chứng chỉ duy nhất cho công ty đó như là bằng chứng nhận dạng (identity) cho các giao dịch trên mạng, ở đây là các máy chủ webserver.

Sau khi kiểm tra chứng chỉ điện tử của máy chủ (sử dụng thuật toán mật mã công khai, như RSA tại trình máy trạm), ứng dụng máy trạm sử dụng các thông tin trong chứng chỉ điện tử để mã hoá thông điệp gửi lại máy chủ mà chỉ có máy chủ đó có thể giải mã. Trên cơ sở đó, hai ứng dụng trao đổi khoá chính (master key) - khoá bí mật hay khoá đối xứng - để làm cơ sở cho việc mã hoá luồng thông tin/dữ liệu qua lại giữa hai ứng dụng chủ khách. Toàn bộ cấp độ bảo mật và an toàn của thông tin/dữ liệu phụ thuộc vào một số tham số:

(i) Số nhận dạng theo phiên làm việc ngẫu nhiên.

(ii) Cấp độ bảo mật của các thuật toán bảo mật áp dụng cho SSL.

(iii) Độ dài của khoá chính (key length) sử dụng cho lược đồ mã hoá thông tin.

f. Bảo mật của giao thức SSL

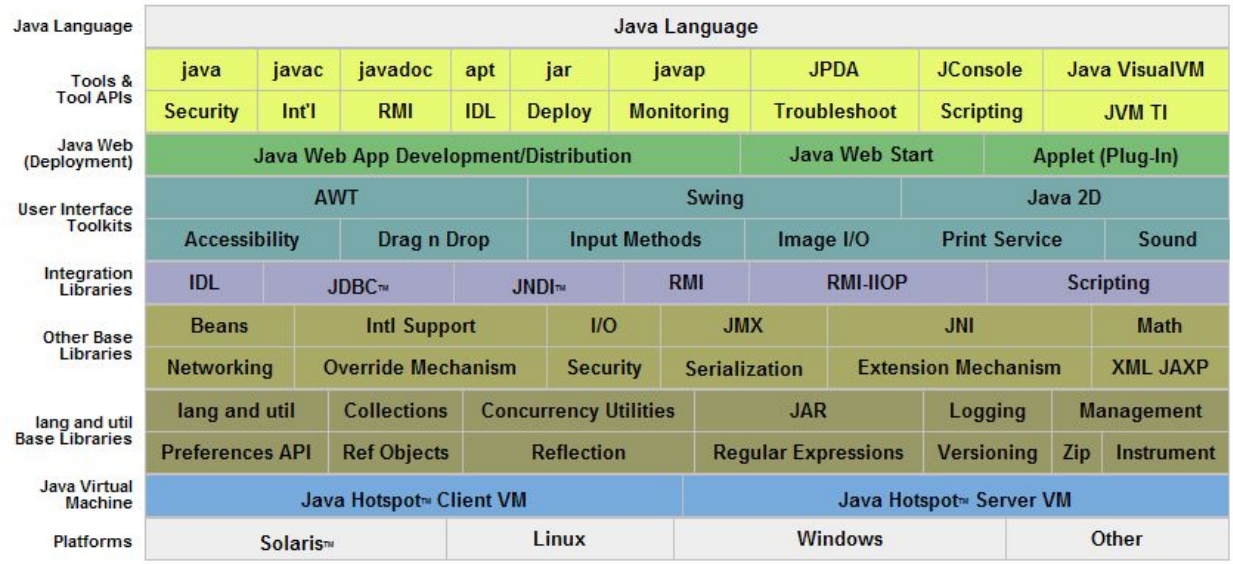
Mức độ bảo mật của SSL như trên mô tả phụ thuộc chính vào độ dài khoá hay phụ thuộc vào việc sử dụng phiên bản mã hoá 40 bits và 128bits. Phương pháp mã hoá 40 bits được sử dụng rộng rãi không hạn chế ngoài nước Mỹ và phiên bản mã hoá 128 bits chỉ được sử dụng trong nước Mỹ và Canada. Theo luật pháp Mỹ, các mật mã "mạnh" được phân loại vào nhóm "vũ khí" (weapon) và do đó khi sử dụng ngoài Mỹ (coi như là xuất khẩu vũ khí) phải được phép của chính phủ Mỹ hay phải được cấp giấy phép của Bộ Quốc phòng Mỹ (DoD). Đây là một lợi điểm cho quá trình thực hiện các dịch vụ thương mại và thanh toán điện tử trong Mỹ và các nước đồng minh phương Tây và là điểm bất lợi cho việc sử dụng các sản phẩm cần có cơ chế bảo mật và an toàn trong giao dịch điện tử nói chung và thương mại điện tử nói riêng trong các nước khác.

Các phương thức tấn công (hay bẻ khoá) của các thuật toán bảo mật thường dùng dựa trên phương pháp "tấn công vét cạn" (brute-force attack) bằng cách thử-sai miền không gian các giá trị có thể của khoá. Số phép thử-sai tăng lên khi độ dài khoá tăng và dẫn đến vượt quá khả năng và công suất tính toán, kể cả các siêu máy tính hiện đại nhất. Thí dụ, với độ dài khoá là 40 bits, thì số phép thử sẽ là 2^{40}=1,099,511,627,776 tổ hợp. Tuy nhiên độ dài khoá lớn kéo theo tốc độ tính toán giảm (theo luỹ thừa nghịch đảo) và dẫn đến khó có khả năng áp dụng trong thực tiễn. Một khi khoá bị phá, toàn bộ thông tin giao dịch trên mạng sẽ bị kiểm soát toàn bộ. Tuy nhiên do độ dài khoá lớn (thí dụ 128 bits, 256 bits), số phép thử-sai trở nên "không thể thực hiện" vì phải mất hàng năm hoặc thậm chí hàng nghìn năm với công suất và năng lực tính toán của máy tính mạnh nhất hiện nay.

Ngay từ năm 1995, bản mã hoá 40 bits đã bị phá bởi sử dụng thuật toán vét cạn. Ngoài ra, một số thuật toán bảo mật (như DES 56 bits, RC4, MD4,...) hiện nay cũng bị coi là không an toàn khi áp dụng một số phương pháp và thuật toán tấn công đặc biệt. Đã có một số đề nghị thay đổi trong luật pháp Mỹ nhằm cho phép sử dụng rộng rãi các phần mềm mã hoá sử dụng mã hoá 56 bits song hiện nay vẫn chưa được chấp thuận.

6.5.2. Lập trình với SSL

a. Thư viện java hỗ trợ lập trình SSL



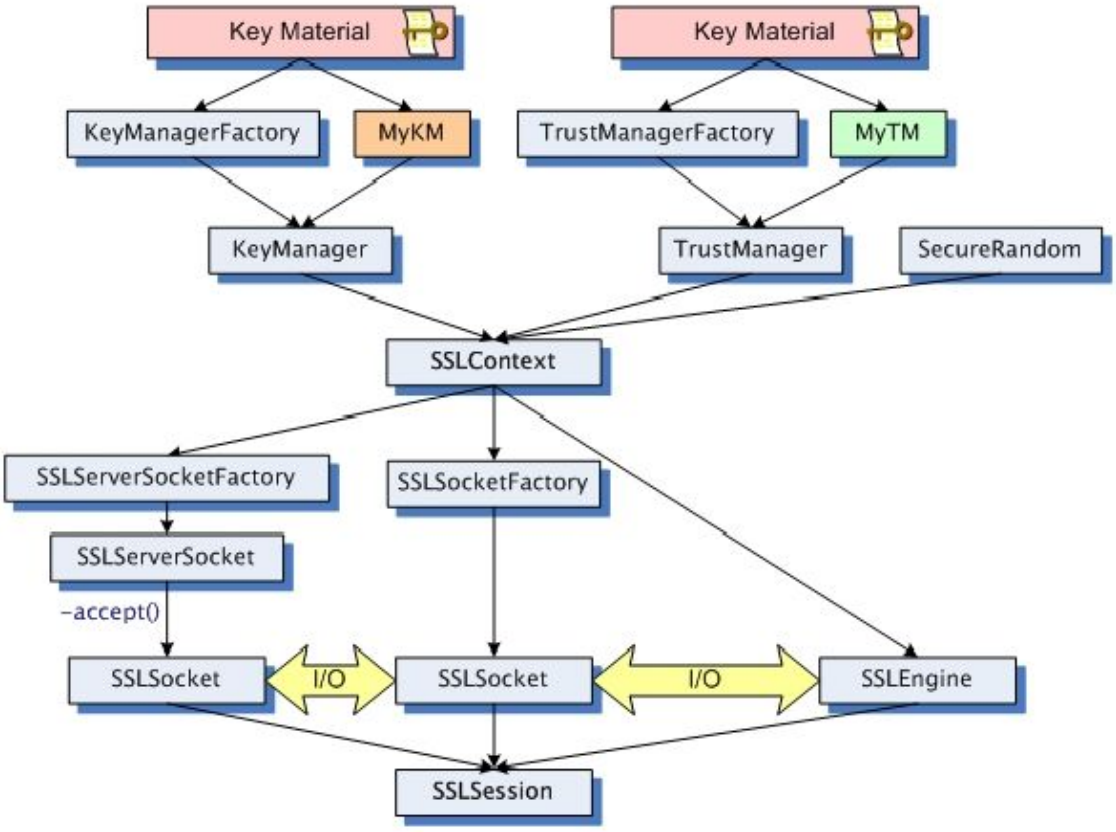
Hình 6.9. Kiến trúc JDK

Java™ security bao gồm tập hợp rất nhiều APIs, công cụ, và cài đặt của các thuật toán bảo mật thông dụng (commonly-used security algorithms), các cơ chế (mechanisms) và các giao thức (protocols). Java security APIs được sử dụng rộng rãi. Bao gồm mã hóa (cryptography), hạ tầng khóa chung (public key infrastructure), trao đổi bảo mật (secure communication), xác thực (authentication), và điều khiển truy cập (access control). Bao gồm rất nhiều lớp thư viện như Java Authentication and Authorization Service (JAAS), Java Cryptography Extension (JCE), Java Secure Socket Extension (JSSE)… Tuy nhiên trong báo cáo này chỉ tập trung vào JSSE. Và cụ thể hơn là JSSE hỗ trợ SSL. Các gói thư viện hỗ trợ lập trình với SSL:

- Gói javax.net.ssl (JSSE)
- Gói javax.rmi.ssl (SSL/TLS-based RMI Socket Factories)

Lớp SSL

Để truyền thông an toàn, cả 2 phía của kết nối đều phải sử dụng SSL. Trong java, các lớp điểm cuối của kết nối là `SSLSocket` và `SSLEngine`. Hình 7.4. cho thấy các lớp chính được sử dụng để tạo ra `SSLSocket/SSLEngines`.



Hình 6.10. Các lớp java SSL

Ví dụ về sử dụng các lớp SSL

Chương trình ví dụ có mã lệnh cho phép server và client có thể xác thực nhau. Muốn vậy thì client phải có chứng chỉ của server (thực tế là một tập (chain) chứng chỉ). Trường hợp ví dụ chứng chỉ của server là chứng chỉ tự ký (self-certificate). Sau đó khi chạy chương trình thì trỏ tới nó.

Sau khi đánh lệnh trên thì sẽ hiện ra các thông tin để điền vào như mật khẩu, tên cá nhân, tên tổ chức, thành phố,…

Server source code (EchoServer.java)

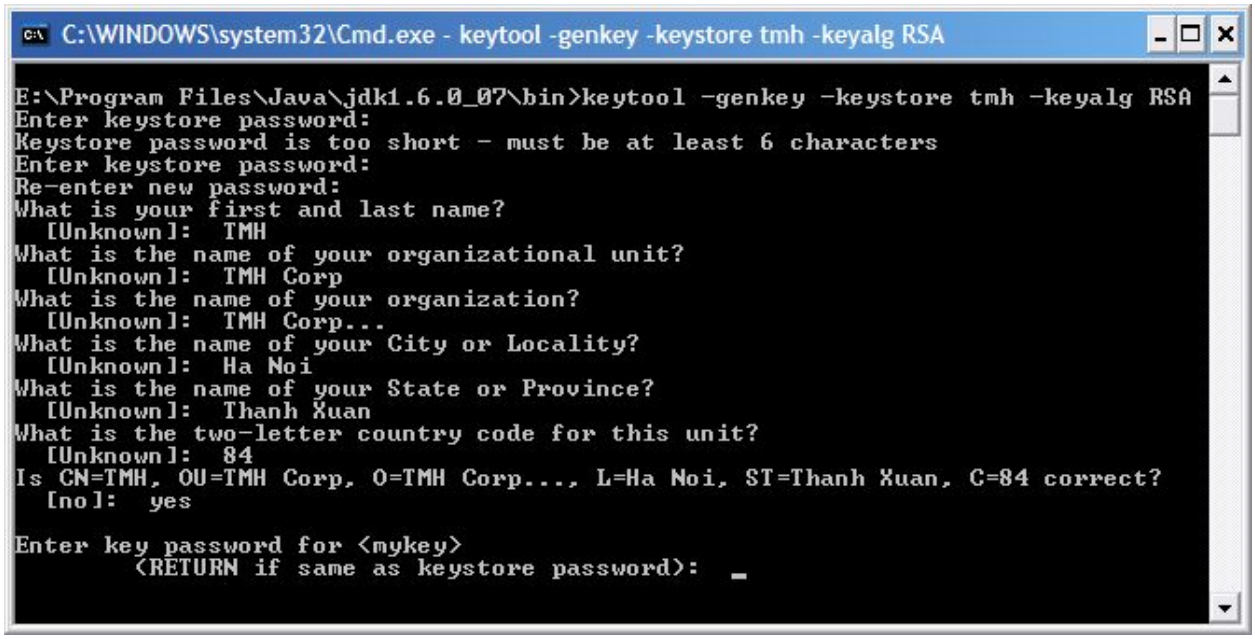
```
import javax.net.ssl.SSLServerSocket;
import javax.net.ssl.SSLServerSocketFactory;
import javax.net.ssl.SSLSocket;
import java.io.BufferedReader;
import java.io.InputStream;
import java.io.InputStreamReader;
public class EchoServer {
      public static void main(String[] arstring) {
            try {
                  SSLServerSocketFactory sslserversocketfactory =
                  (SSLServerSocketFactory) SSLServerSocketFactory.getDefault();
                  SSLServerSocket sslserversocket =
                  (SSLServerSocket)sslserversocketfactory.createServerSocket(9999);
                  SSLSocket sslsocket = (SSLSocket) sslserversocket.accept();
```

```
                    InputStream inputstream = sslsocket.getInputStream();
                    InputStreamReader inputstreamreader = new
                          InputStreamReader(inputstream);
                    BufferedReader bufferedreader = new
                          BufferedReader(inputstreamreader);
                    String string = null;
                    while ((string = bufferedreader.readLine()) != null) {
                          System.out.println(string);
                          System.out.flush();
                    }
             } catch (Exception exception) {
                    exception.printStackTrace();
             }
      }
}
```

Client source code (EchoClient.java)

```
import javax.net.ssl.SSLSocket;
import javax.net.ssl.SSLSocketFactory;
import java.io.*;
public class EchoClient {
      public static void main(String[] arstring) {
             try {
                    SSLSocketFactory sslsocketfactory =
                          (SSLSocketFactory) SSLSocketFactory.getDefault();
                    SSLSocket sslsocket = (SSLSocket)
                          sslsocketfactory.createSocket("localhost", 9999);
                    InputStream inputstream = System.in;
                    InputStreamReader inputstreamreader = new
                          InputStreamReader(inputstream);
                    BufferedReader bufferedreader = new
                          BufferedReader(inputstreamreader);
                    OutputStream outputstream = sslsocket.getOutputStream();
                    OutputStreamWriter outputstreamwriter = new
                          OutputStreamWriter(outputstream);
                    BufferedWriter bufferedwriter = new
                          BufferedWriter(outputstreamwriter);
                    String string = null;
                    while ((string = bufferedreader.readLine()) != null) {
                          bufferedwriter.write(string + '\n');
                          bufferedwriter.flush();
                    }
             } catch (Exception exception) {
                    exception.printStackTrace();
             }
      }
}
```

Sau khi dịch chạy chương trình , được kết quả sau:



Giải thích:

-genkey: Lệnh tạo key

-keystore mySrvKeystore: Tên key là mySrvKeystore

-keyalg RSA: Thuật toán dùng để mã hóa là RSA

Về phần ứng dụng qua giao diện dòng lệnh thì sử dụng chương trình mẫu giống như ở trên. Chạy như sau:

Tạo chứng nhận:

keytool -genkey -keystore mySrvKeystore -keyalg RSA

Mật khẩu sẽ điền là 123456

Sau khi tạo xong chứng chỉ thì copy file key vào trong thư mục chứa file

Phía server thì chứng chỉ được lưu trong keyStore

Chạy chương trình:

java -Djavax.net.ssl.keyStore=mySrvKeystore -Djavax.net.ssl.keyStorePassword=123456 EchoServer

Phía Client thì chứng chỉ được lưu trong trustStore

Chạy chương trình:

java -Djavax.net.ssl.trustStore=mySrvKeystore -Djavax.net.ssl.trustStorePassword=123456 EchoClient

6.6. LẬP TRÌNH ỨNG DỤNG MẠNG KHÔNG ĐỒNG NHẤT

Khả năng tương tác của mạng đề cập đến khả năng trao đổi thông tin một cách đáng tin cậy và chính xác, của các hệ thống khác nhau về công nghệ triển khai. Nghĩa là các yếu tố, ví dụ như phần cứng, hệ điều hành và ngôn ngữ triển khai, có thể khác nhau giữa các nền tảng, nhưng chúng sẽ không ảnh hưởng đến khả năng giao tiếp của các hệ thống này.

Có một số yếu tố có thể ảnh hưởng đến khả năng tương tác, bao gồm từ các vấn đề ở mức thấp, ví dụ như thứ tự byte được sử dụng bởi các kiểu dữ liệu gốc, đến các công nghệ cấp cao, ví dụ như các dịch vụ web mà các chi tiết triển khai được ẩn đi. Phần này sẽ đề cập đến vấn đề này.

Đầu tiên, thứ tự byte, được sử dụng để hỗ trợ các kiểu dữ liệu gốc, sẽ được giới thiệu. Đây là vấn đề cơ bản để chuyển dữ liệu. Các thứ tự byte khác nhau sẽ dẫn đến sự khác biệt về cách thông tin được diễn giải.

Tiếp theo, ta sẽ thảo luận về cách các ứng dụng Java có thể tương tác với các ứng dụng được viết bằng các ngôn ngữ khác nhau. Có thể là các ngôn ngữ dựa trên JVM hoặc các ngôn ngữ hoàn toàn khác với Java.

Cấu trúc giao tiếp mạng cơ bản giựa trên socket. Thực thể này thường hoạt động trong môi trường TCP/IP. Cách các socket Java có thể tương tác với các socket được viết bằng các ngôn ngữ khác nhau, cụ thể là C #, cũng được đề cập trong phần này.

Hỗ trợ đáng kể nhất cho khả năng tương tác được cấu trúc dưới dạng các tiêu chuẩn truyền thông, thường được các dịch vụ web định hình. Các ứng dụng này hỗ trợ giao tiếp giữa các hệ thống khác nhau bằng cách sử dụng phần mềm trung gian được chuẩn hóa. Phần lớn các chi tiết của giao tiếp bị che giấu bởi các triển khai phần mềm trung gian.

Phần này sẽ đề cập:

1. Cách Java xử lý thứ tự byte
2. Giao tiếp với các ngôn ngữ khác
3. Giao tiếp với socket
4. Sử dụng phần mềm trung gian để thực hiện khả năng tương tác.

6.6.1. Thứ tư byte trong Java

Có hai loại thứ tự byte: **big endian**, and **little endian**. Các thuật ngữ này đề cập đến thứ tự mà một số lượng nhiều byte được lưu trữ trong bộ nhớ. Để minh họa điều này, hãy xem xét cách một số nguyên được lưu trữ trong bộ nhớ. Một số nguyên bao gồm 4 byte, các byte này được gán cho vùng bộ nhớ 4 byte. Tuy nhiên, các byte này có thể được lưu trữ theo nhiều cách khác nhau. **Big endian** đặt byte quan trọng nhất đầu tiên, trong khi **little endian** đặt byte ít quan trọng nhất trước.

Xem xét khai báo và khởi tạo một số nguyên sau:

```
int number = 0x01234567;
```

Trong ví dụ sau, bốn byte bộ nhớ được hiển thị bằng cách sử dụng **big endian**, giả sử rằng số nguyên đã được cấp cho địa chỉ 1000 :

Address	**Byte**

1000	01
1001	23
1002	45
1003	67

Bảng sau đây cho thấy cách số nguyên sẽ được lưu trữ bằng cách sử dụng **little endian**:

Address	**Byte**
1000	67
1001	45
1002	23
1003	01

Các hệ thống khác nhau có thể sử dụng thứ tự byte khác nhau, ví dụ:

- Bộ xử lý Intel sử dụng little endian
- Bộ xử lý ARM có thể sử dụng cả 2
- Bộ vi xử lý Motorola 68K sử dụng big endian
- Motorola PowerPC sử dụng big endian
- Bộ xử lý Sun SPARK sử dụng big endian

Việc gửi dữ liệu, ví dụ như chuỗi ASCII, không phải là một vấn đề vì các byte này được lưu trữ theo thứ tự liên tiếp. Đối với các loại dữ liệu khác, ví dụ như float và longs, nó có thể là một vấn đề.

Nếu ta cần biết máy hiện tại hỗ trợ biểu diễn nào, lớp ByteOder trong gói java.nio có thể xác định thứ tự byte hiện tại. Câu lệnh sau sẽ hiển thị giá trị này:

```
System.out.println(ByteOrder.nativeOrder());
```

Các phương thức của lớp DataOutputStream tự động sử dụng big endian. Lớp ByteBuffer cũng sử dụng big endian theo mặc định. Tuy nhiên, như đoạn mã dưới đây, thứ tự có thể được chỉ định:

```
ByteBuffer buffer = ByteBuffer.allocate(4096);
System.out.println(buffer.order());
buffer.order(ByteOrder.LITTLE_ENDIAN);
System.out.println(buffer.order());
```

Sau khi được thiết lập, các phương thức khác, ví dụ như phương thức slice, không thay đổi thứ tự byte được sử dụng.

Thứ tự byte thường được xử lý tự động trên máy. Tuy nhiên, khi ta chuyển dữ liệu giữa các máy sử dụng các thiết bị khác nhau, ta có thể gặp sự cố. Có thể các byte được chuyển sẽ không đúng thứ tự tại máy đích.

Các mạng thường sử dụng big endian, còn được gọi là **thứ tự byte mạng**. Bất kỳ dữ liệu nào được gửi qua một socket đều cần sử dụng big endian. Khi gửi thông tin giữa các ứng dụng Java, thông thường không phải là vấn đề. Tuy nhiên, có thể phát sinh vấn đề khi tương tác với các công nghệ khác không phải Java.

6.6.2. Tương tác với các nền tảng khác nhau

Đôi khi, ta cần truy cập các thư viện được viết bằng một ngôn ngữ khác để xây dựng một ứng dụng. Tuy không chỉ là vấn đề mạng, nhưng Java cung cấp hỗ trợ vấn đề này theo một số cách. Giao diện trực tiếp với các ngôn ngữ khác không diễn ra trên mạng mà xảy ra trên cùng một máy. Ta sẽ xem xét ngắn gọn một số vấn đề về giao diện này.

Với thư viện Java, thông thường chỉ cần tải các lớp JAR tương ứng. Với các ngôn ngữ không phải Java, ta có thể sử dụng các API của **Java Native Interface** (**JNI**) hoặc một số thư viện khác. Tuy nhiên, nếu ngôn ngữ này là ngôn ngữ dựa trên JVM, thì quá trình này sẽ dễ dàng hơn.

a) Giao tiếp với các ngôn ngữ dựa trên JVM

Máy ảo JAVA (**JVM**) thực thi mã byte Java. Tuy nhiên, đây không phải là ngôn ngữ duy nhất sử dụng JVM. Các ngôn ngữ khác cũng sử dụng bao gồm những ngôn ngữ sau:

Nashorn : Sử dụng JavaScript

Clojure : Phương ngữ Lisp

Groovy : Đây là một ngôn ngữ kịch bản

Scala : Kết hợp các phương pháp hướng đối tượng và chức năng

JRuby : Triển khai Java của Ruby

Jthon : Triển khai Java của Python

Jacl : Triển khai Java của Tcl

TuProlog : Triển khai dựa trên Java của Prolog

Sử dụng cùng một cơ sở JVM sẽ tạo điều kiện thuận lợi cho việc chia sẻ mã và thư viện. Thông thường, có thể không chỉ sử dụng các thư viện được phát triển bằng một ngôn ngữ dựa trên JVM khác, mà còn có thể lấy từ các lớp được phát triển bằng các ngôn ngữ khác nhau.

Nhiều ngôn ngữ đã được chuyển sang JVM vì việc sử dụng JVM dễ dàng hơn so với việc tạo nhiều trình biên dịch hoặc trình thông dịch cho các nền tảng khác nhau. Ví dụ: Ruby và Python có triển khai JVM vì lý do này. Những ngôn ngữ này có thể tận dụng tính di động của JVM và quy trình biên dịch **Just-In-Time** (**JIT**). Ngoài ra, JVM có một thư viện lớn các mã đã được thử nghiệm tốt để xây dựng.

Nashorn là một công cụ JavaScript được xây dựng trên JVM và đã được thêm vào trong Java 8. Điều này cho phép mã JavaScript được tích hợp dễ dàng vào ứng dụng Java. Chuỗi mã sau đây minh họa quá trình này. Một thực thể của công cụ JavaScript được lấy và sau đó mã JavaScript được thực thi:

```
try {
     ScriptEngine engine = new
                 ScriptEngineManager().getEngineByName("nashorn");
     engine.eval("print('Executing JavaScript code');");
} catch (ScriptException ex) {
     // Handle exceptions
}
```

Có thể tìm hiểu thêm thông tin chi tiết về công nghệ này tại https://docs.oracle.com/javase/8/docs/technotes/Guide/scripting/nashorn /.

b) Giao tiếp với các ngôn ngữ không dựa trên JVM

Một kỹ thuật phổ biến để truy cập mã của ngôn ngữ khác là thông qua các API JNI . API này cung cấp một phương tiện truy cập mã C/C ++. Cách tiếp cận này ta có thể hiểu thêm tại http://www.ibm.com/developerworks/java/tutorials/ j-jni /j-jni.html.

c) Giao tiếp thông qua các socket đơn giản

Có thể chuyển thông tin giữa các ứng dụng được viết bằng các ngôn ngữ khác nhau bằng socket. Khái niệm socket không phải là duy nhất đối với Java và đã được triển khai trong nhiều ngôn ngữ. Do socket hoạt động ở cấp độ TCP/IP nên chúng có thể giao tiếp mà không cần nỗ lực nhiều.

Việc xem xét khả năng tương tác quân tâm cính đến dữ liệu được truyền. Sự không tương thích có thể xảy ra khi cách biểu diễn bên trong của dữ liệu khác nhau đáng kể giữa hai ngôn ngữ khác nhau. Điều này có thể là do việc sử dụng big endian so với little endian trong cách một kiểu dữ liệu được biểu diễn bên trong và liệu một kiểu dữ liệu cụ thể có tồn tại ở ngôn ngữ khác hay không. Ví dụ, trong C không có kiểu dữ liệu Boolean. Nó được biểu diễn bằng số nguyên.

Trong phần này, ta sẽ phát triển một máy chủ bằng Java và một máy khách trong C #. Để chứng minh cách dùng socket để tương tác giữa 2 nền tảng khác nhau, một chuỗi string sẽ được chuyển giữa hai ứng dụng. Ta sẽ thấy rằng việc chuyển ngay cả một kiểu dữ liệu đơn giản, ví dụ như chuỗi, có thể gặp một số khó khăn.

- Chương trình máy chủ Java

Máy chủ được khai báo trong lớp JavaSocket , như dưới, tương tự như phiên bản máy chủ echo đã được giới thiệu. Socket máy chủ được tạo và sau đó chặn chờ cho đến khi phương thức accept trả lại socket kết nối với máy khách:

```
public class JavaSocket {

    public static void main(String[] args) {
        System.out.println("Server Started");
        try (ServerSocket serverSocket = new ServerSocket(5000)) {
            Socket socket = serverSocket.accept();
            System.out.println("Client connection completed");

            Scanner scanner = new Scanner(socket.getInputStream());
            PrintWriter pw = new PrintWriter(socket.getOutputStream(), true);
            String message = scanner.nextLine();
            System.out.println("Server received: " + message);
            pw.println(message);
            System.out.println("Server sent: " + message);

            socket.close();
        } catch (IOException ex) {
            ex.printStackTrace();
        }
        System.out.println("Server Terminated");
    }
}
```

d) Chương trình máy khách bằng C#

Lớp CSharpClient, như dưới, triển khai ứng dụng khách. Câu lệnh using tương ứng với câu lệnh import trong Java. Tương tự như Java, phương thức đầu tiên để thực thi là phương thức Main .

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

namespace CSharpSocket
{
    class CSharpClient
    {
        public static void Main(string[] args)
        {
            Console.WriteLine("Client Started");
            IPEndPoint serverAddress = new
                  IPEndPoint(IPAddress.Parse("127.0.0.1"), 5000);
            Socket clientSocket = new Socket(AddressFamily.InterNetwork,
                  SocketType.Stream, ProtocolType.Tcp);
            clientSocket.Connect(serverAddress);

            Console.Write("Enter message: ");
            String message = Console.ReadLine();

            byte[] messageBuffer;
```

```
            messageBuffer = System.Text.Encoding.UTF8.GetBytes(message +
                  "\n");
            clientSocket.Send(messageBuffer);

            byte[] receiveBuffer = new byte[32];
            clientSocket.Receive(receiveBuffer);
            String recievedMessage =
System.Text.Encoding.ASCII.GetString(receiveBuffer);
            Console.WriteLine("Client received: [" + recievedMessage + "]");

            clientSocket.Close();
            Console.WriteLine("Client Terminated");
        }
    }
}
```

Trong đoạn mã trên, biến IPEndPoint biểu diễn địa chỉ Internet và lớp Socket , biểu diễn cho socket. Phươn thức Connect để kết nối đến máy chủ. Phương thức write của lớp Console hiển thị thông tin trong một cửa sổ lệnh. Tại đây, người dùng được yêu cầu gửi một thông báo đến máy chủ. Phương thức ReadLine đọc đầu vào từ người dùng. Phương pháp Send sẽ truyền dữ liệu đến máy chủ. Tuy nhiên, nó yêu cầu dữ liệu phải được đặt vào bộ đệm byte. Thông điệp và ký tự xuống dòng được mã hóa và chèn vào bộ đệm. Ký tự được nối thêm là cần thiết để máy chủ có thể đọc chuỗi chính xác và biết khi nào chuỗi bị kết thúc. Phương thức Receive đọc trả lời từ máy chủ. Tương tự như phương thức Send, nó yêu cầu một bộ đệm byte. Bộ đệm này được tạo với kích thước 32 byte. Điều này giới hạn kích thước của thông báo. Cuối cùng, bộ đệm nhận được chuyển đổi thành chuỗi và hiển thị kết quả

6.6.3. Kết luận

Phần này giới thiệu một số yếu tố ảnh hưởng đến khả năng tương tác của mạng. Ở mức thấp, thứ tự byte trở nên quan trọng. Các hệ thống sử dụng một trong hai thứ tự byte: big endian hoặc little endian. Thứ tự có thể được xác định và kiểm soát bởi các ứng dụng Java. Giao tiếp mạng thường sử dụng big endian lớn truyền dữ liệu.

Nếu ta cần giao tiếp với các ngôn ngữ khác, các ngôn ngữ dựa trên JVM dễ thực hiện hơn vì chúng chia sẻ cùng một cơ sở mã byte. Nếu cần làm việc với các ngôn ngữ khác, thì JNI thường được sử dụng.

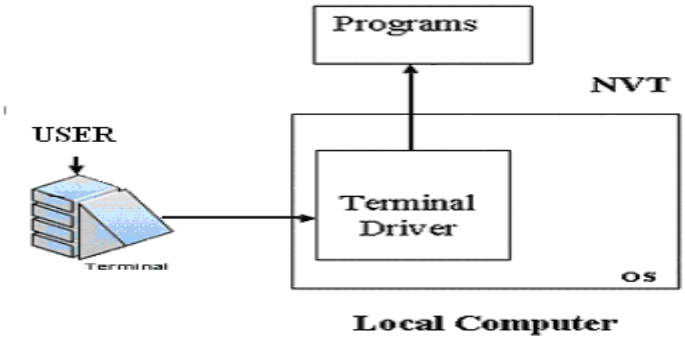
Socket không phải là một khái niệm duy nhất của Java. Nó thường được sử dụng trong môi trường TCP/IP, có nghĩa là một socket được viết bằng một ngôn ngữ có thể dễ dàng giao tiếp với một socket được viết bằng một ngôn ngữ khác. Một ví dụ minh sử dụng máy chủ Java và máy khách C # đã được xây dựng để thể hiện quan điểm trên.

CHƯƠNG 7. LẬP TRÌNH VỚI CÁC GIAO THỨC TẦNG ỨNG DỤNG

7.1. LẬP TRÌNH GIAO THỨC DỊCH VỤ TELNET

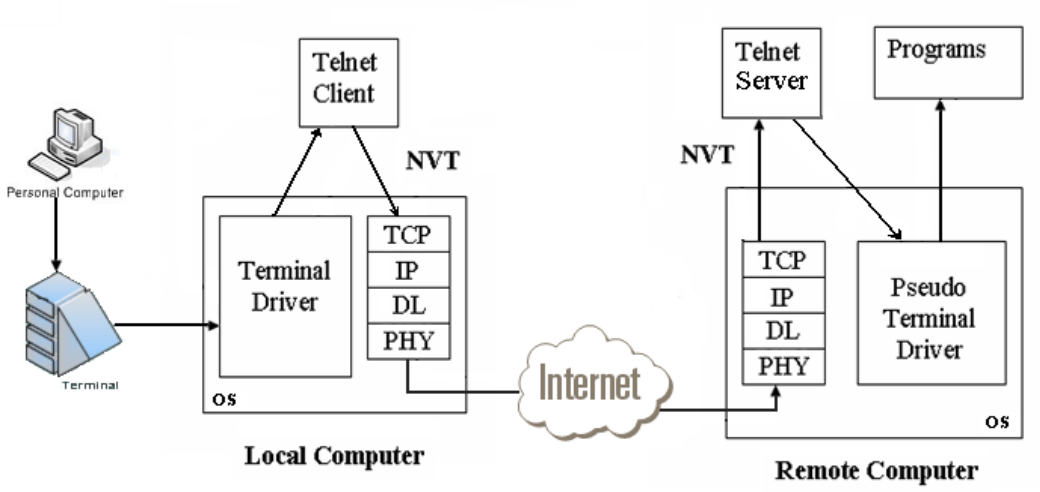
7.1.1. Giới thiệu về Telnet

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



Hình 6.1. Đăng nhập cục bộ

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



Hình 6.2. Đăng nhập từ xa

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

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

cùng người sử dụng phải nắm được các chế độ hoạt động của Telnet trước khi cài đặt chương trình Telnet.

7.1.2. Cài đặt dịch vụ Telnet Client với Java

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

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

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

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

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

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

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

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

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

```
//TelnetClient.java
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.net.*;
//Terminal hiển thị chữ trên cửa số
class Terminal extends Canvas{
	// Kích cỡ font chữ
	private int charWidth, charHeight;
	// text[0] là dòng thao tác hiện tại
	private String[] text;
	// Khoảng cách với viền cửa sổ chính chương trình
	private final int margin=4;
	// Số dòng lệnh tối đa được lưu lại
	private final int lines=50;

	// Constructor, khởi tạo các giá trị ban đầu
	Terminal()   {
		charHeight=12;
		setFont(new Font("Monospaced", Font.PLAIN, charHeight));
		charWidth=getFontMetrics(getFont()).stringWidth(" ");
		text=new String[lines];
		for (int i=0; i<lines; ++i)
			text[i]="";
		setSize(80*charWidth+margin*2, 25*charHeight+margin*2);
		requestFocus();
```

```
            // Lắng nghe sự kiện con trỏ chuột
            addMouseListener(new MouseAdapter() {
                  public void mousePressed(MouseEvent e){
                        requestFocus();
                  }
            });
      }
      // In và lưu lại các kí tự người dùng nhập từ bàn phím
      public void put(char c){
            Graphics g=getGraphics();
            if (c=='\r'){  // Return
                  for (int i=lines-1; i>0; --i)
                        text[i]=text[i-1];
                  text[0]="";
                  update(g);  // Clear screen and paint
            }
            // Các kí tự điều khiển: backspace, delete, telnet EC
            else if (c==8 || c==127 || c==247){
                  int len=text[0].length();
                  if (len>0){
                        --len;
                        text[0]=text[0].substring(0, len);
                        g.setColor(getBackground());
                        g.fillRect(len*charWidth+margin,
                              getSize().height-margin-charHeight,
                              (len+1)*charWidth+margin, getSize().height-margin);
                  }
            }
            else if (c=='\t'){        // Tab với khoảng cách 8 space
                  text[0]+="        ";
                  text[0].substring(0, text[0].length()&-8);
            }
            else if (c>=32 && c<127){  // Kí tự có thể in
                  g.drawString(""+c, margin+text[0].length()*charWidth,
                              getSize().height-margin);
                  text[0]+=c;
            }
            g.dispose();
      }
      // Hiển thị những gì đã gõ từ bàn phím
      public void paint(Graphics g)    {
            int height=getSize().height;
            for (int i=0; i<lines; ++i)
                  g.drawString(text[i], margin, height-margin-i*charHeight);
      }
}
//luồng nhận sẽ chờ các kí tự đến từ một luồng vào (Input
//stream) và gửi đến Termial. Đàm phán các lựa chọn đầu cuối
class Receiver extends Thread{
      private InputStream in;
      private OutputStream out;
      private Terminal terminal;
      public Receiver(InputStream in, OutputStream out, Terminal terminal){
            this.in=in;
            this.out=out;
            this.terminal=terminal;
            start();
```

```
	}
	//Đọc các kí tự và gửi đến đầu cuối
	public void run()   {
		while (true){
			try {
				int c=in.read();
				if (c<0){    // EOF
					System.out.println("Connection closed ");
					return;
				}
				else if (c==255){   // Đàm phán các lựa chọn đầu cuối
					int c1=in.read();  // 253=do, 251=will
					int c2=in.read();  // option
					if (c1==253)  // do option, send "won't do option"
				out.write(new byte[] {(byte)255, (byte)252, (byte)c2});
					else if (c1==251) // send "don't do option"
				out.write(new byte[] {(byte)255, (byte)254, (byte)c2});
				}
				else
					terminal.put((char)c);
			}
			catch (IOException x) {
				System.out.println("Receiver: "+x);
			}
		}
	}
}
//TelnetWindow. Gửi dữ liệu bàn phím từ terminal đến một socket từ
//xa và bắt đầu nhận các kí tự từ socket và hiển thị các kí tự đó  trên terminal
class TelnetWindow extends Frame{
	Terminal terminal;
	InputStream in;
	OutputStream out;
	// Constructor
	TelnetWindow(String hostname, int port){
		super("telnet "+hostname+" "+port);  // Set title\
		// Thiết lập cửa sổ
		add(terminal=new Terminal());
		// Xử lý việc đóng cửa sổ
		addWindowListener(new WindowAdapter() {
			public void windowClosing(WindowEvent e){
				dispose();
				try {
					out.close();
				}
				catch (IOException x) {
					System.out.println("Closing connection: "+x);
				}
			}
			public void windowClosed(WindowEvent e) {
				System.exit(0);
			}
		});
		// Xử lý các thao tác với bàn phím
		terminal.addKeyListener(new KeyAdapter() {
			public void keyTyped(KeyEvent e) {
				char k=e.getKeyChar();
```

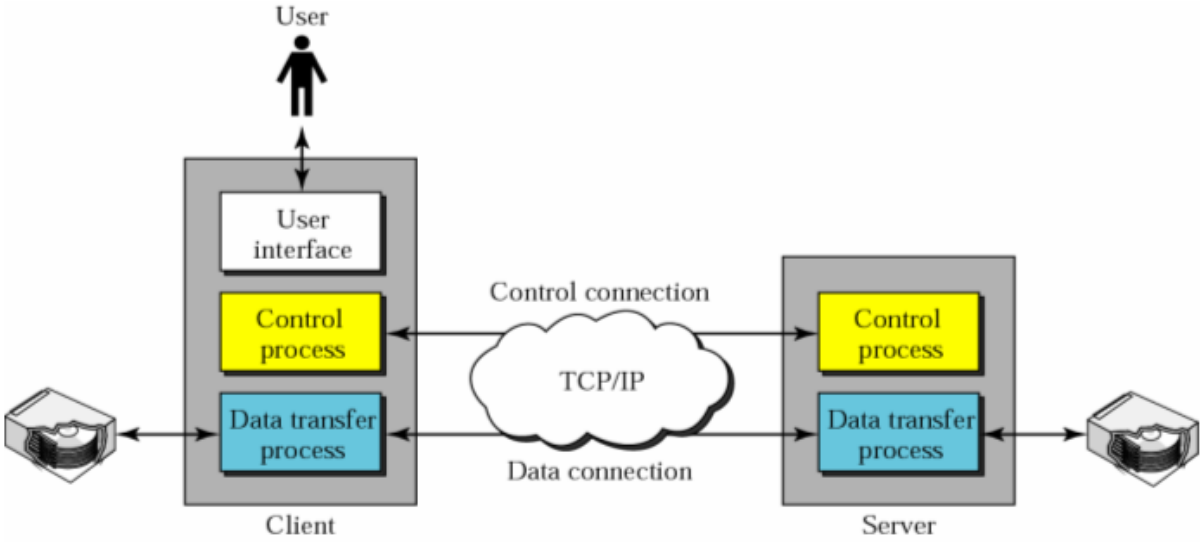
```
                        try {
                                terminal.put(k);
                                out.write((int)k);
                                if (k=='\r'){
                                        out.write('\n');  // Convert CR to CR-LF
                                        out.flush();
                                }
                        }
                        catch (IOException x) {
                                System.out.println("Send: "+x);
                        }}});
                try {
                        // Mở một connection
                        System.out.println("Opening connection to "+
                                hostname+" on port "+port);
                        Socket socket=new Socket(hostname, port);
                        InetAddress addr=socket.getInetAddress();
                        System.out.println("Connected to "+addr.getHostAddress());
                        in=socket.getInputStream();
                        out=socket.getOutputStream();
                        // Hiển thị cửa sổ
                        pack();
                        setVisible(true);
                        // Bắt đầu nhận dữ liệu từ server
                        new Receiver(in, out, terminal);
                        System.out.println("Ready");
                }
                catch (UnknownHostException x) {
                        System.out.println("Unknown host: "+hostname+" "+x);
                        System.exit(1);
                }
                catch (IOException x) {
                        System.out.println(x);
                        System.exit(1);
                }
        }
}
//Chương trình chính
public class TelnetClient{
        public static void main(String[] argv){
                // Phân tách các đối số: telnet hostname port
                String hostname="";
                int port=23;
                try {
                        hostname=argv[0];
                        if (argv.length>1)
                                port=Integer.parseInt(argv[1]);
                }   catch (ArrayIndexOutOfBoundsException x) {
                        System.out.println("Usage: java telnet hostname [port]");
                        System.exit(1);
                }
                catch (NumberFormatException x) {}
                TelnetWindow t1=new TelnetWindow(hostname, port);
        }
}
```

Chạy thử chương trình

- Bước 1: Dịch chương trình TelnetClient.java
- Bước 2: Kiểm tra xem trên máy từ xa, trình Telnet server đã được khởi tạo chạy chưa, nếu chưa thì chạy nó và dùng trình quản trị Telnet Server, thiết lập các tham số phù hợp.
- Bước 3: Chạy chương trình Telnet Client từ máy cục bộ.

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

7.2.1. Dịch vụ truyền tệp FTP

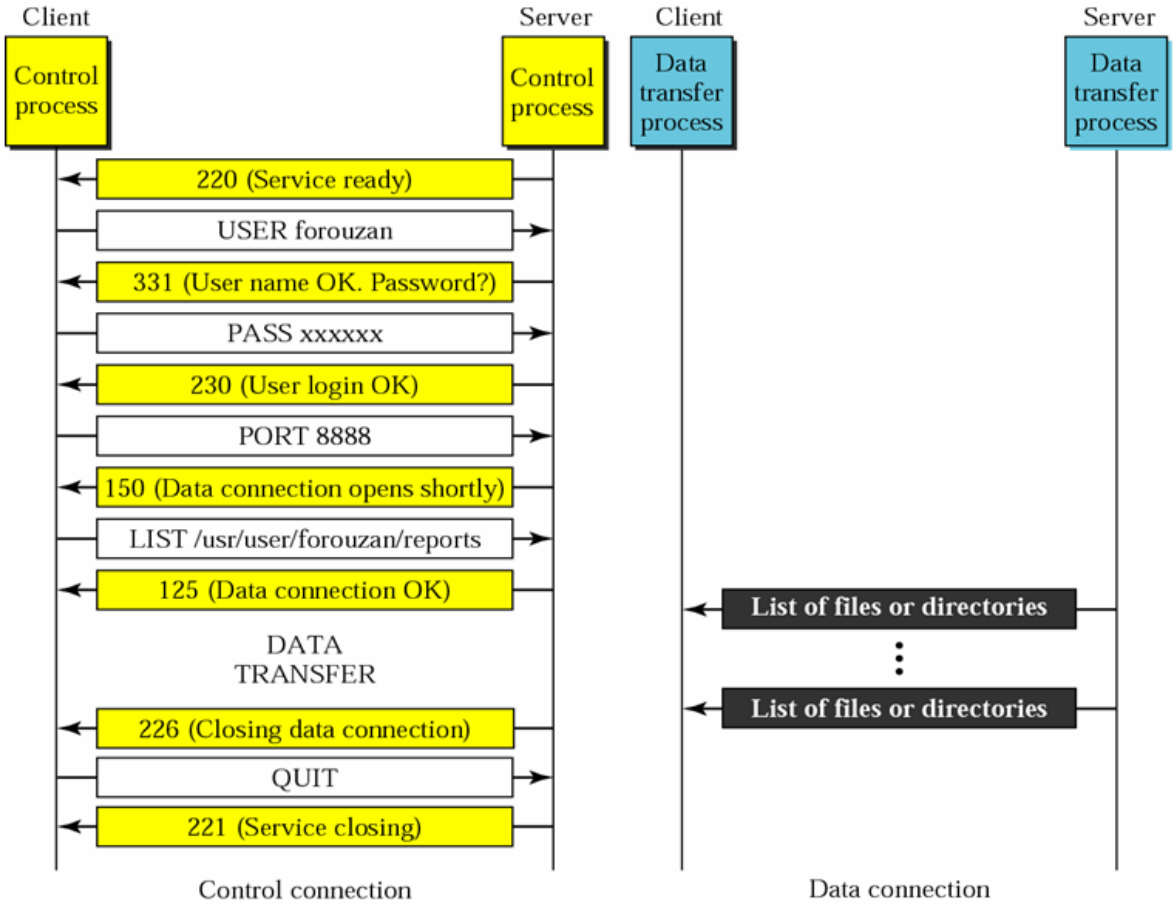


Hình 6.3. Mô hình FTP

Đặc điểm

- FTP là giao thức chuẩn của TCP/IP
- FTP sử dụng kết nối TCP, là kết nối truyền thông tin cậy
- FTP gồm 2 phần mềm: Phần mềm FTPClient cài trên máy cục bộ và FTPServer cài trên máy từ xa(File Server).
- FTP sử dựng 2 kết nối truyền thông đồng thời để tăng hiệu quả của việc truyền tệp qua mạng:

- Kết nối điều khiển: Sử dụng phương thức truyền thông đơn giản và dữ liệu truyền dưới dạng text(NVT-ASCII 7bít). Kết nối này cho phép truyền lệnh từ client tới server và truyền đáp ứng từ server về client. Kết nối này sử dụng số cổng mặc định là 21 phía server.



Hình 6.4. Ví dụ quá trình truyền tệp FTP

- Kết nối dữ liệu: Kết nối này sử dụng các phương thức truyền thông phức tạp vì phải truyền nhiều kiểu dữ liệu khác nhau. Kết nối này được thiết lập mỗi khi truyền một tệp và huỷ sau khi truyền xong tệp đó. Kết nối này bao giờ cũng được khởi tạo sau kết nối điều khiển và kết thúc trước khi huỷ bỏ kết nối điều khiển(kết nối điều khiển duy trì trong suốt phiên làm việc). Kết nối dữ liệu sử dụng số cổng mặc định phía server là 20. Có 2 cách thiết lập kết nối dữ liệu: dùng lệnh PORT và lệnh PASV.
  - FTP có 3 chế độ truyền tệp:
    - Cất tệp trên máy cục bộ lên máy tính từ xa dưới sự giám sát của lệnh STOR.
    - Lấy một tệp trên máy tính từ xa về máy tính cục bộ dưới sự giám sát của lệnh RETR.
    - Lấy danh sách các mục trong một thư mục trên máy từ xa về máy cục bộ dưới sự giám sát của lệnh LIST.
  - Mô hình hoạt động của FTP thể hiện như hình 6.3.

Ví dụ quá trình truyền tệp giữa FTPclient và FTPserver như hình 6.4.

7.2.2. Kỹ thuật cài đặt giao thức FTP với java

a. Các bước cài đặt:

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

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

b.Chương trình truyền tệp FTP

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

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

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

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

  Hoặc viết phương thức kết nối như ví dụ sau:

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

InputStreamReader(connectionSocket.getInputStream()));

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

        return true;
    }
```

Hoặc hàm giải phóng kết nối:

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

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

- Khai báo các phương thức để thực hiện gửi các lệnh của FTP tới FTPServer
- Phương thức thực hiện đăng nhập với lệnh USER và PASS

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

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

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

- **Phương thức đọc/ghi dữ liệu:**

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

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

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

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

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

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

c. Chương trình ví dụ

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

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

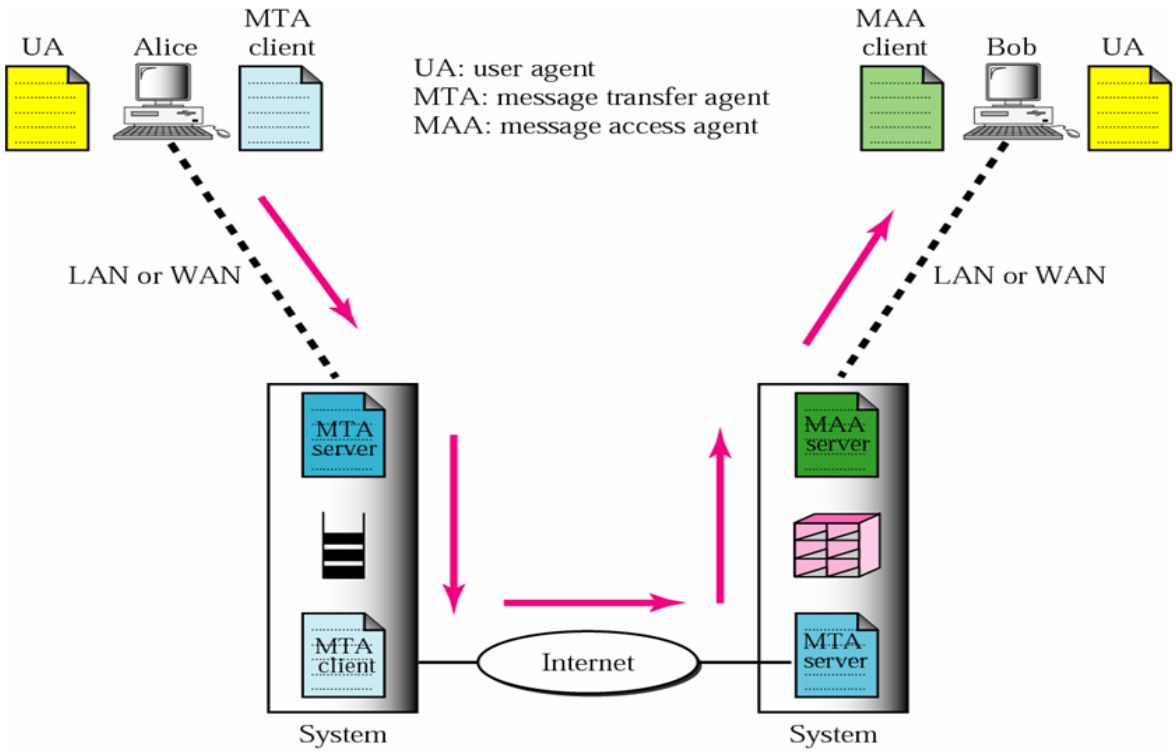
7.3. LẬP TRÌNH GỬI/NHẬN THƯ VỚI GIAO THỨC SMTP và POP3

7.3.1. Giao thức SMTP

a. Giới thiệu

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

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



Hình 6.5. Mô hình người gửi/nhận kết nối mail server qua LAN/WAN

b. Cài đặt chương trình gửi thư với SMTP

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

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

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

Ví dụ về một giao dịch gửi thư của SMTP:

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

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

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.PrintStream;
import java.net.Socket;

public class SendMail    {
       Object mailLock        = null;  //In case we want a multi-threaded mailer
       public String mailServerHost = "";
       public String from           = "";
       public String to             = "";
       public String replyTo        = "";
       public String subject        = "Java is Fun";
       public String mailData       ="HyperSendMail";
       public String errorMsg = "";
       public Socket mailSendSock = null;
       public  BufferedReader inputStream = null;
       public PrintStream outputStream    =  null;
       public String serverReply          = "";
       SendMail() {
              // Doesn't do anything but we need this for extension purposes.
       }

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

```
}

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

		if(replyTo == null)
			replyTo = from;
		outputStream.println("RCPT TO: <"+to+">");
		serverReply = inputStream.readLine();
		if(serverReply.startsWith("5")){
			close("Reply error:"+serverReply);
			return;
		}
		outputStream.println("DATA");
		serverReply = inputStream.readLine();
		if(serverReply.startsWith("5")){
			close("DATA Server error : "+serverReply);
			return;
		}
		outputStream.println("From: "+from);
		outputStream.println("To: "+to);
		if(subject != null)
			outputStream.println("Subject: "+subject);
		if(replyTo != null)
			outputStream.println("Reply-to: "+replyTo);
		outputStream.println("");
		outputStream.println(mailData);
		outputStream.print("\r\n.\r\n");
		outputStream.flush();
		serverReply = inputStream.readLine();
		if(serverReply.startsWith("5")){
			close("DATA finish server error: "+serverReply);
			return;
		}
		outputStream.println("quit");
		serverReply = inputStream.readLine();
		if(serverReply.startsWith("5")) {
			close("Server error on QUIT: "+serverReply);
			return;
		}
		inputStream.close();
		outputStream.close();
```

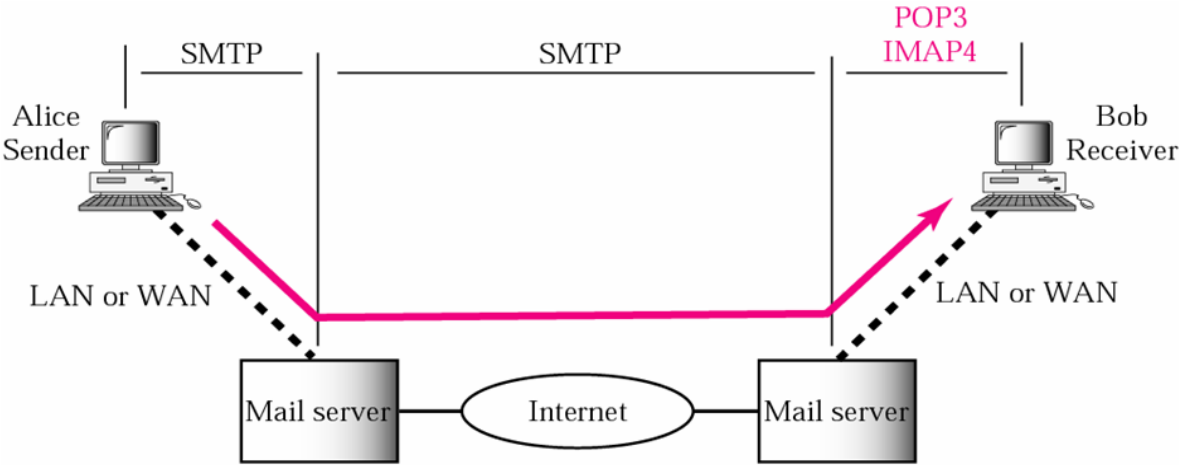
```
                mailSendSock.close();
        }
        catch(Exception any){
                any.printStackTrace();
                close("send() Exception");
        }
        close("Mail sent");
}

public boolean open(){
        synchronized(mailLock) {
                try {
                        mailSendSock = new Socket(mailServerHost, 25);
                        outputStream = new PrintStream(
                                mailSendSock.getOutputStream());
                        inputStream = new BufferedReader(new InputStreamReader(
                                mailSendSock.getInputStream()));
                        serverReply = inputStream.readLine();
                        if(serverReply.startsWith("4"))         {
                                errorMsg = "Server refused the connect message : "
                                        +serverReply;
                                return false;
                        }
                }
                catch(Exception openError)        {
                        openError.printStackTrace();
                        close("Mail Socket Error");
                        return false;
                }
                System.out.println("Connected to "+mailServerHost);
                return true;
        }

}

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

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

```
                    "test test test!");              // Message Data
            sm.send();                          // Send it!
        }
}
```

7.3.2. Giao thức POP3

a. Giới thiệu

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



Hình 6.6: Giao thức POP3

b. Chương trình truy cập hộp thư với giao thức POP3

Các thao tác cơ bản:

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

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

```
//CheckMail.java
import java.net.*;
import java.io.*;
public class CheckMail {
```

```
	public static void main(String s[]) {
		//  CheckMail [mailServer] [user] [password]
		try {
			CheckMail t = new CheckMail();
			int i = t.checkMyMail(s[0], s[1], s[2]);
			if (i==0) {
				System.out.println("No mail waiting.");
			}
			else {
				System.out.println("There " + (i==1?"is " :"are ") + i +
						" message" +(i==1?"":"s")+ " waiting.");
			}
		}
		catch (Exception e) {
			e.printStackTrace();
		}
	}
	private void send(BufferedWriter out, String s) throws IOException {
		out.write(s+"\n");
		out.flush();
	}
	private String receive(BufferedReader in) throws IOException {
		return in.readLine();
	}
	private int checkMyMail
	(String server, String user, String pass) throws IOException {
		Socket s = new Socket(server, 110);
		BufferedReader in  = new BufferedReader(
				new InputStreamReader(s.getInputStream()));
		BufferedWriter out = new BufferedWriter(
				new OutputStreamWriter(s.getOutputStream()));
		receive(in);
		send(out, "USER " + user);
		receive(in);
		send(out, "PASS " + pass);
		receive(in);
		return getNumberOfMessages(in, out);
	}
	public int getNumberOfMessages
	(BufferedReader in, BufferedWriter out) throws IOException {
		int i = 0;
		String s;
		send(out, "LIST");
		receive(in);
		while((s = receive(in)) != null) {
			if (!(s.equals("."))) {
				i++;
			}
			else
				return i;
		}
		return 0;
	}
}
```

7.6. KẾT LUẬN

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

PHẦN III.
LẬP TRÌNH PHÂN TÁN

CHƯƠNG 8. LẬP TRÌNH PHÂN TÁN VỚI RMI

8.1. GIỚI THIỆU LẬP TRÌNH PHÂN TÁN VÀ RMI

8.1.1. Giới thiệu kỹ thuật lập trình phân tán

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

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

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

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

8.1.2. Giới thiệu kỹ thuật lập trình RMI

a. Đặc trưng của kỹ thuật RMI

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

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

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

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

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

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

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

Trong đó:

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

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

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

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

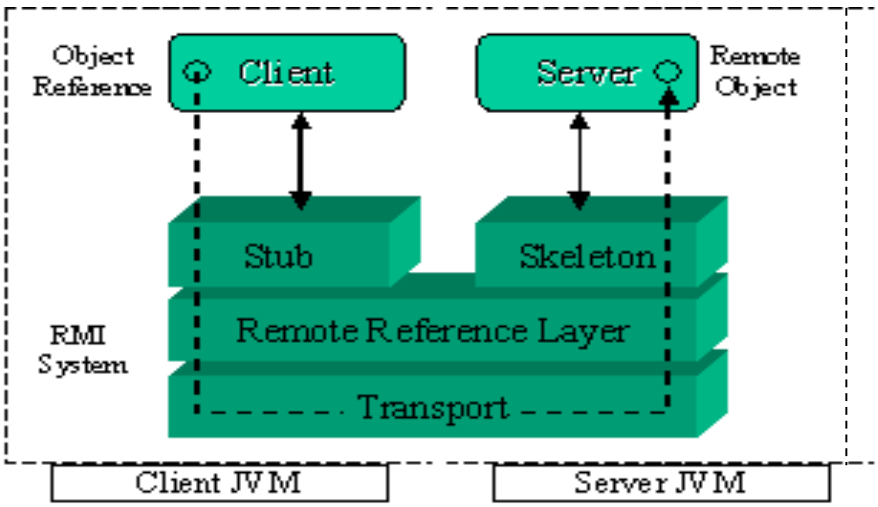
b. Kiến trúc của chương trình Client – Server theo cơ chế RMI

Hình 7.1. là kiến trúc của một chương trình phân tán đối tượng RMI theo mô hình Client /Server:

Trong đó:

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

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



Hình 7.1. Kiến trúc Client/Server của chương trình RMI

c. Các cơ chế hoạt động RMI

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

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

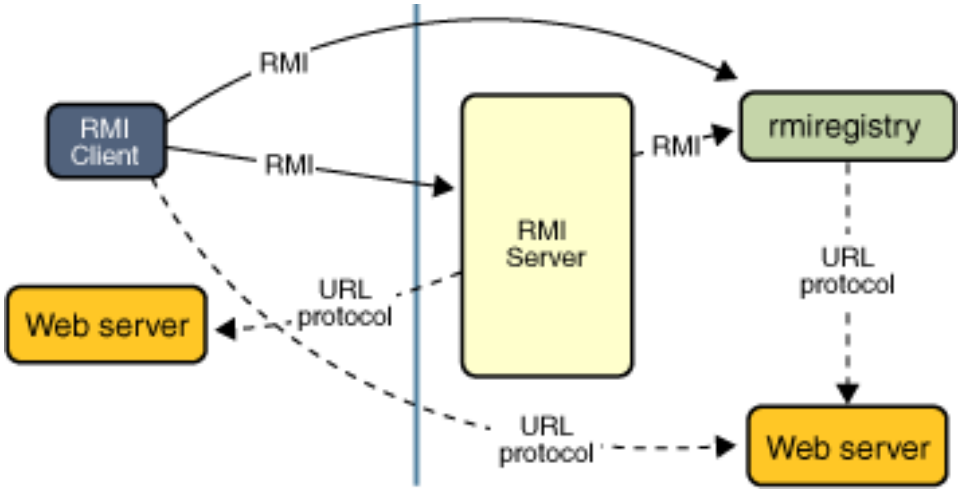
Cơ chế định vị đối tượng ở xa (Locate remote objects): Cơ chế này xác định cách thức mà chương trình Client có thể lấy được **tham chiếu** (Stub) đến các đối tượng ở xa. Thông thường người ta sử dụng một dịch vụ danh bạ (Naming Service) lưu giữ các tham chiếu đến các đối tượng cho phép gọi từ xa mà client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

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

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

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

Hình 7.2. mô tả một ứng dụng phân tán RMI sử dụng dịch vụ danh bạ để lấy các tham chiếu tới các đối tượng ở xa.



Hình 7.2. Vai trò của dịch vụ tên

Trong đó:

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

Hình 7.2. minh họa cũng cho thấy cách thức mà hệ thống RMI sử dụng một WebServer sẵn có để truyền tải mã bytecodes của các lớp qua lại giữa Client và Server

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

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

- Cài đặt đối tượng Stub và trả về tham khảo đối tượng xa cho Client (6).
- Client thực thi một lời gọi phương thức từ xa thông qua đối tượng Stub (7).

8.1.3. Các lớp hỗ trợ lập trình với RMI

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

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

8.2. XÂY DỰNG CHƯƠNG TRÌNH PHÂN TÁN RMI

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

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

8.2.1. Kỹ thuật lập trình RMI

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

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

- *Cài đặt các chương trình Client*: Các chương trình Client có sử dụng các Remote Object có thể được cài đặt ở bất kỳ thời điểm nào sau khi các Remote Interface đã được định nghĩa.

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

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

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

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

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

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

```
//TT.java

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

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

```
//TTImpl.java

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

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

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

TT c=new TTImpl();

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

UnicastRemoteObject.exportObject(Obj);

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

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

Chương trình server được viết như sau:

```
//TTServer.java

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

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

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

Chương trình client:

```
//TTClient.java

import java.rmi.*;
```

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

8.2.2. Biên dịch chương trình

Giai đoạn này gồm 2 bước:

Bước 1: Biên dịch các tệp chương trình thành dạng bytecode dùng trình javac.exe. Trong chương trình trên có 4 tệp:

javac.exe

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

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

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

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

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

rmic TTImpl [Enter]

Sau khi dịch, 2 tệp mới được tạo ra:

TTImpl_Stub.class *(5)*

TTImpl_Skel.class *(6)*

8.2.3. Thực thi chương trình ứng dụng

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

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

Bước 2: Chạy chương trình

Thực hiện mở 3 của sổ lênh:

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

rmiregistry [porrt] [Enter]

- Của sổ thứ 2: Chạy chương trình server:

  java TTServer [Enter]

- Của sổ thứ 3: Chạy chương trình client:

  java TTClient [Enter]

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

8.3. CASE STUDY 1: LOGIN TỪ XA DÙNG RMI

8.3.1. Bài toán

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

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

8.3.2. Thiết kế hệ thống

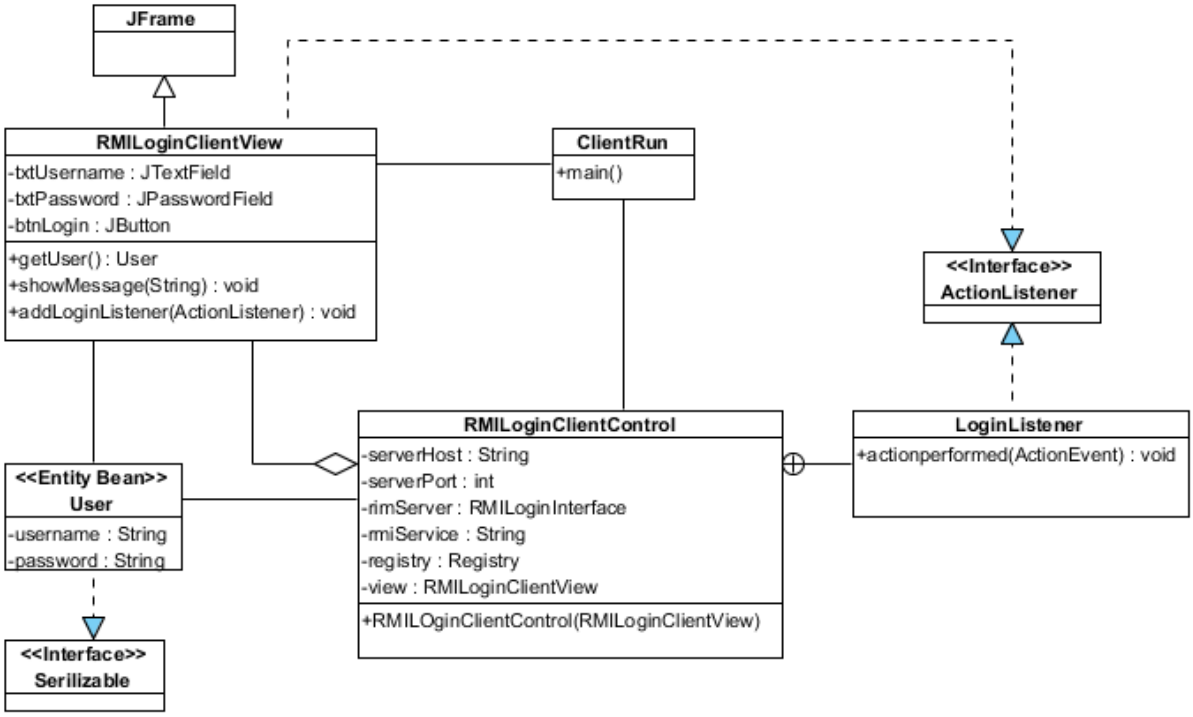
Vì hệ thống được thiết kế theo mô hình client/server RMI nên mỗi phía client, server sẽ có một sơ đồ lớp riêng, các sơ đồ này được thiết kế theo mô hình MVC.

a. Sơ đồ lớp phía client

Sơ đồ lớp của phía client được thiết kế theo mô hình MVC trong Hình 7.3, bao gồm 3 lớp chính tương ứng với sơ đồ M-V-C như sau:

- Lớp User: là lớp tương ứng với thành phần model (M), bao gồm hai thuộc tính username và password, các hàm khởi tạo và các cặp getter/setter tương ứng với các thuộc tính.

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



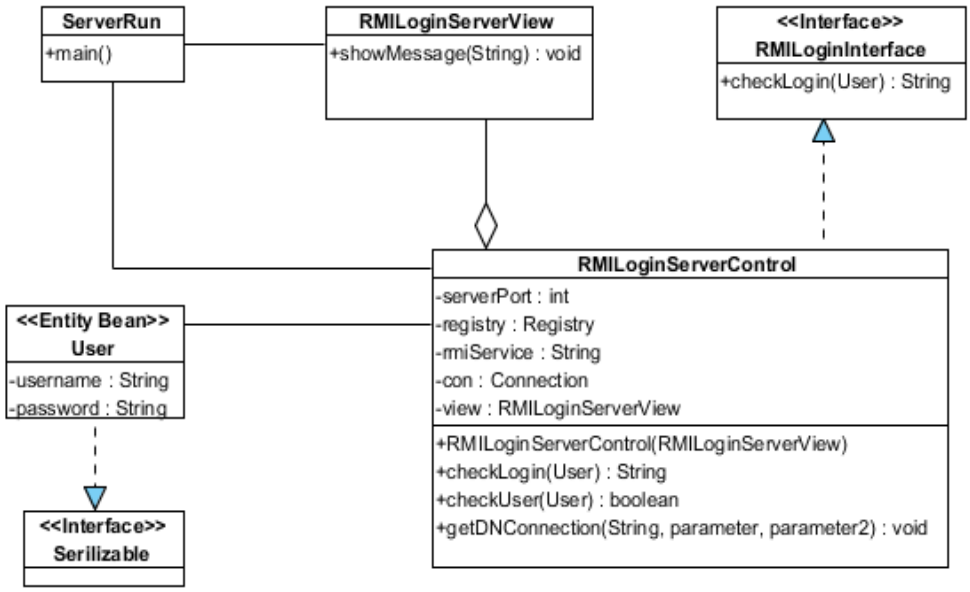
Hình 7.3: Sơ đồ lớp phía client RMI

b. Sơ đồ lớp phía server

Sơ đồ lớp của phía server được thiết kế theo mô hình MVC trong Hình 7.4, bao gồm 3 lớp chính tương ứng với sơ đồ M-V-C như sau:

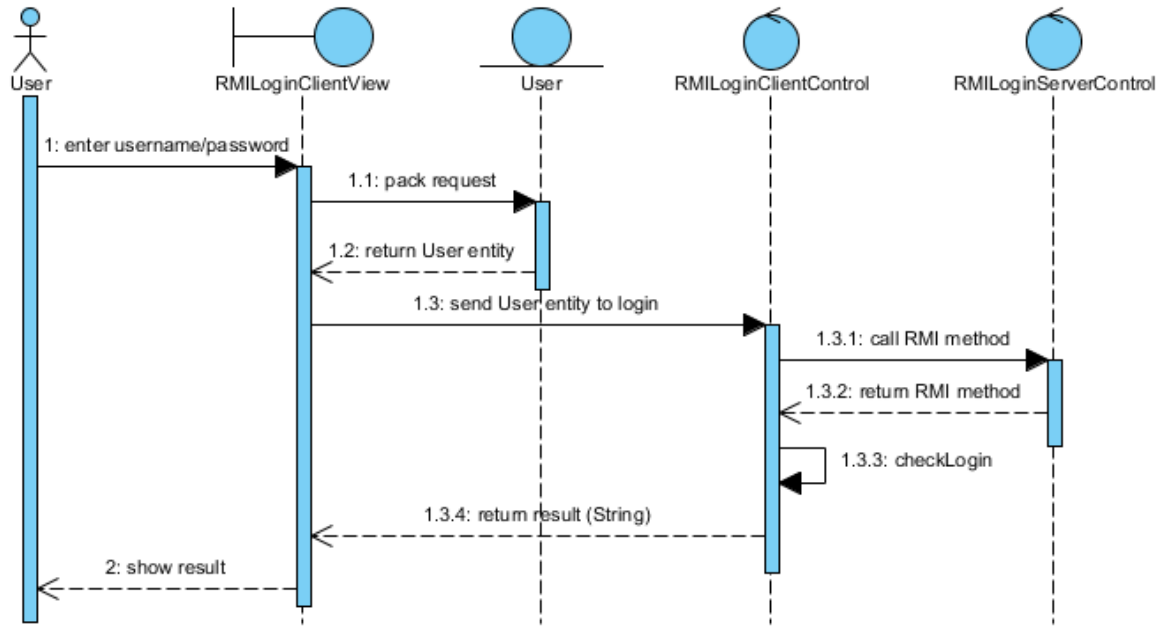
- Lớp User: là lớp thực thể, dùng chung thống nhất với lớp phía bên client.
- Lớp RMILoginServerView: là lớp tương ứng với thành phần view (V), là lớp dùng hiển thị các thông báo và trạng thái hoạt động bên server RMI.
- Giao diện RMILoginInterface: là giao diện (interface) khai báo đối tượng từ xa,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

báo trong RMILoginInterface, sau đó đăng kí bản thân nó vào server RMI để phục vụ các lời triệu gọi từ phía các client.



Hình 7.4: Sơ đồ lớp phía server RMI

c. Tuần tự các bước thực hiện



Hình 7.5: Tuần tự các bước thực hiện khi login từ xa với RMI

Tuần tự các bước xử lí như sau (Hình 7.5):

1. Ở phía client, người dùng nhập username/password và click vào giao diện của lớp RMILoginClientView

2. Lớp RMILoginClientView sẽ đóng gói thông tin username/password trên form vào một đối tượng model User bằng phương thức getUser() và chuyển xuống cho lớp RMILoginClientControl xử lí
3. Lớp RMILoginClientControl sẽ triệu gọi làm checkLogin() từ phía server RMI
4. Server trả về cho bên client một skeleton của phương thức checkLogin().
5. Bên phía client, khi nhận được skeleton, nó gọi phương thức checkLogin() để kểm tra thông tin đăng nhập.
6. Kết quả kiểm tra sẽ được lớp RMILoginClientControl sẽ chuyển cho lớp RMILoginClientView hiển thị bằng phương thức showMessage()
7. Lớp RMILoginClientView hiển thị kết quả đăng nhập lên cho người dùng

8.3.3. Cài đặt

a. Các lớp phía client RMI

Lớp User.java

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

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

	public User(){
	}

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

	public String getPassword() {
		return password;
	}

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

	public String getUserName() {
		return userName;
	}

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

Lớp RMILoginClientView.java

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

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

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

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

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

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

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

	public void actionPerformed(ActionEvent e) {
	}

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

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

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

Lớp RMILoginClientControl.java

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

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

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

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

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

```
    }
}
```

Lớp ClientRun.java

```
package rmi.client;

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

}
```

b. Các lớp phía server RMI

Lớp RMILoginServerView.java

```
package rmi.server;

public class RMILoginServerView {
	public RMILoginServerView(){
	}

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

}
```

Interface RMILoginInterface.java

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

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

Lớp RMILoginServerControl.java

```
package rmi.server;
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.rmi.server.UnicastRemoteObject;
import java.sql.Connection;
```

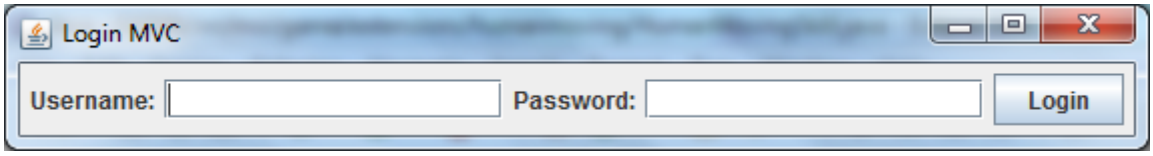
```
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
import rmi.client.User;

public class RMILoginServerControl extends UnicastRemoteObject implements
RMILoginInterface{
	private int serverPort = 3232;
	private Registry registry;
	private Connection con;
	private RMILoginServerView view;
	private String rmiService = "rmiLoginServer";

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

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

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

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

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

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

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

	      if (rs.next()) {
```

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

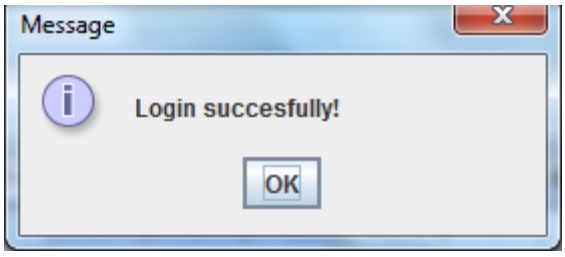
Lớp ServerRun.java

```
package rmi.server;

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

8.3.4. Kết quả



Login thành công:



Login lỗi:

8.4. KẾT LUẬN

Qua các mục của chương này, chúng ta đã làm sáng tỏ kỹ thuật lập trình, cơ chế truyền thông và bản chất của lập trình phân tán đối tượng của RMI. Thông qua đó, sinh viên có thể hiểu đượoc các kỹ thuật lập trình khác như RPC, DCOM, CORBA, EJB, WebService... với các kỹ thuật lập trình OOP, SOP và kiến trúc nhiều tầng. Ngoài các vấn đề nêu trong chương, còn một số kỹ thuật khác của RMI không kém phần quan trọng mà sẽ được đề cập đến trong bài giảng và thông qua bài tập của sinh viên như: Vấn đề định nghĩa bộ đăng ký, vấn đề tuần tự hoá đối tượng, Kỹ thuật gọi đối tượng từ xa bằng phương thức động, kỹ thuật kích hoạt đối tượng từ xa tự động, chính sách bảo mật từ phía client.v.v..

8.5. BÀI TẬP

1. Viết chương trình giải phương trình bậc 2 dùng kiến trúc RMI: client chỉ hiện giao diện nhập dữ liệu vào, sau đó gọi hàm từ server RMI về tính toán, kết quả lại được client hiển thị.
2. Viết chương trình giải hệ phương trình bậc nhất dùng kiến trúc RMI: client chỉ hiện giao diện nhập dữ liệu vào, sau đó gọi hàm từ server RMI về tính toán, kết quả lại được client hiển thị.
3. Viết chương trình tìm USCLN (BSCNN) của 2 số nguyên dương a và b dùng kiến trúc RMI: client chỉ hiện giao diện nhập dữ liệu vào, sau đó gọi hàm từ server RMI về tính toán, kết quả lại được client hiển thị.
4. Viết chương trình phân tích một số nguyên dương ra thừa số nguyên tố (ví dụ: 300 = 2*2*3*5*5) dùng kiến trúc RMI: client chỉ hiện giao diện nhập dữ liệu vào, sau đó gọi hàm từ server RMI về tính toán, kết quả lại được client hiển thị.
5. Viết chương trình ứng dụng máy tính Calculator với các phép toán cơ bản: nhân, chia, cộng, trừ, căn, lũy thừa, logarit, sin, cos, tg, cotg dùng kiến trúc RMI: client chỉ hiện giao diện nhập dữ liệu vào, sau đó gọi hàm từ server RMI về tính toán, kết quả lại được client hiển thị.

PHẦN IV.
LẬP TRÌNH ỨNG DỤNG WEB

CHƯƠNG 9. LẬP TRÌNH WEBSOCKET

Nội dung chương này sẽ trình bày về:

- Giới thiệu về WebSocket
- Thiết kế, cài đặt và triển khai ứng dụng với WebSocket

9.1. GIỚI THIỆU WEBSOCKET

Trong mô hình yêu cầu- phản hồi truyền thống được sử dụng trong HTTP, máy khách yêu cầu tài nguyên và máy chủ cung cấp phản hồi. Việc trao đổi luôn do máy khách khởi xướng; máy chủ không thể gửi bất kỳ dữ liệu nào nếu không có máy khách yêu cầu trước. Mô hình này hoạt động tốt cho World Wide Web khi máy khách thỉnh thoảng đưa ra các yêu cầu đối với các tài liệu không thường xuyên thay đổi, nhưng hạn chế của cách tiếp cận này ngày thể hiện rõ khi nội dung thay đổi nhanh chóng và người dùng mong đợi trải nghiệm tương tác hơn trên Web. Giao thức WebSocket giải quyết những hạn chế này bằng cách cung cấp một kênh giao tiếp song công giữa máy khách và máy chủ. Kết hợp với các công nghệ ứng dụng khách khác, ví dụ như JavaScript và HTML5, WebSocket cho phép các ứng dụng web mang lại trải nghiệm người dùng phong phú hơn.

Trong ứng dụng WebSocket, máy chủ công bố một **điểm cuối (endpoint)** WebSocket và máy khách sử dụng URI của điểm cuối để kết nối với máy chủ. Giao thức WebSocket là đối xứng sau khi kết nối được thiết lập; máy khách và máy chủ có thể gửi tin nhắn cho nhau bất kỳ lúc nào khi kết nối đang mở và cũng có thể đóng kết nối bất kỳ lúc nào. Máy khách thường chỉ kết nối với một máy chủ và máy chủ chấp nhận kết nối từ nhiều máy khách.

Giao thức WebSocket có hai phần: bắt tay và truyền dữ liệu. Máy khách bắt đầu bắt tay bằng cách gửi yêu cầu tới điểm cuối WebSocket qua URI. Việc bắt tay tương thích với cơ sở hạ tầng trên nền HTTP hiện có: các máy chủ web xem đó như một yêu cầu nâng cấp kết nối HTTP. Ví dụ sau minh họa việc bắt tay từ một máy khách:

```
GET /path/to/websocket/endpoint HTTP/1.1 Host: localhost
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: xqBt3ImNzJbYqRINxEFlkg==
Origin: http://localhost
Sec-WebSocket-Version: 13
```

Phản hồi của máy chủ cho máy khách về việc như sau:

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: K7DJLdLooIwIG/MOpvWFB3y3FE8=
```

Máy chủ áp dụng một thao tác lên giá trị của Sec-WebSocket-Key để tạo ra giá trị của tiêu đề Sec-WebSocket-Accept . Máy khách áp dụng thao tác tương tự cho giá trị của Sec-WebSocket-Key và kết nối được thiết lập thành công nếu kết quả khớp với giá trị nhận được từ máy chủ. Máy khách và máy chủ có thể gửi tin nhắn cho nhau sau khi bắt tay thành công.

WebSocket hỗ trợ tin nhắn văn bản (mã hóa UTF-8) và tin nhắn nhị phân. Các khung điều khiển trong WebSocket là *close* , *ping* và *pong* (phản hồi đối với *ping*). Khung ping và pong cũng có thể chứa dữ liệu ứng dụng.

Các điểm cuối WebSocket được đại diện bởi các URI có dạng như sau:

```
ws://host:port/path?query wss://host:port/path?query
```

Lược đồ ***ws*** đại diện cho một kết nối WebSocket không được mã hóa và lược đồ ***wss*** đại diện cho một kết nối được mã hóa. Thành phần cổng ***port*** là tùy chọn; số cổng mặc định là 80 cho các kết nối không được mã hóa và 443 cho các kết nối được mã hóa. Thành phần đường dẫn ***path*** cho biết vị trí của điểm cuối trong máy chủ. Thành phần truy vấn ***query*** là tùy chọn.

Các trình duyệt web hiện đại triển khai giao thức WebSocket và cung cấp API JavaScript để kết nối với điểm cuối, gửi tin nhắn và chỉ định phương thức gọi lại cho các sự kiện WebSocket (ví dụ như kết nối đã mở, tin nhắn đã nhận và kết nối đóng).

9.2. TẠO ỨNG DỤNG WEBSOCKET

Nền tảng Java EE hỗ trợ API Java cho WebSocket (JSR 356), cho phép tạo, cấu hình và triển khai các điểm cuối WebSocket trong các ứng dụng web. API ứng dụng khách WebSocket được chỉ định trong JSR 356 cũng cho phép truy cập các điểm cuối WebSocket từ xa từ bất kỳ ứng dụng Java nào.

API Java cho WebSocket bao gồm các gói sau.

- Gói javax.websocket.server chứa chú thích, lớp, và giao diện để tạo ra và điểm cuối máy chủ cấu hình.
- Gói javax.websocket chứa chú thích, lớp, giao diện, và các ngoại lệ chung cho máy khách và máy chủ.

Điểm cuối WebSocket là các thực thể của lớp javax.websocket.Endpoint . API Java cho WebSocket cho phép ta tạo hai loại điểm cuối: điểm cuối lập trình và điểm cuối chú thích. Để tạo một **điểm cuối lập trình** , ta thừa kế lớp Endpoint và ghi đè các phương thức. Để tạo một **điểm cuối chú thích**, thì chỉ cần viết lớp Java sau đó khai báo các chú thích với những phương thức cần. Sau khi tạo một điểm cuối, ta triển khai qua một URI cụ thể trong ứng dụng để các máy khách từ xa có thể kết nối tới.

Quy trình tạo và triển khai điểm cuối WebSocket như sau.

1. Tạo một lớp điểm cuối.
2. Thực thi các phương thức của điểm cuối.

3. Thêm logic nghiệp vụ vào điểm cuối.
4. Triển khai điểm cuối bên trong ứng dụng web.

Quy trình này khác đối với điểm cuối lập trình và điểm cuối chú thích, sẽ được trình bày chi tiết trong các phần sau.

9.2.1. Điểm cuối lập trình

Ví dụ sau đây cho thấy cách tạo một điểm cuối bằng cách thừa kế lớp `Endpoint` :

```
public class EchoEndpoint extends Endpoint {
      @Override
      public void onOpen(final Session session, EndpointConfig config) {
            session.addMessageHandler(new MessageHandler.Whole<String>() {
               @Override
               public void onMessage(String msg) {
                  try {
                     session.getBasicRemote().sendText(msg);
                  } catch (IOException e) { ... }
               }
            });
         }

      }
```

Điểm cuối này lặp lại mọi thông điệp nhận được. Lớp Endpoint định nghĩa ba phương thức liên quan đến một số bước trong vòng đời: onOpen , onClose và onError . Lớp EchoEndpoint thực hiện phương thức onOpen , đây là phương thức trừu tượng duy nhất trong lớp Endpoint .

Tham số Session biểu diễn cho cuộc nói chuyện giữa điểm cuối này và các điểm cuối từ xa. Phương thức addMessageHandler đăng ký trình xử lý thông điệp, và phương thức getBasicRemote trả về một đối biểu diễn cho điểm cuối từ xa.

Trình xử lý thông báo được thực thi như một lớp nội ẩn. Phương thức onMessage của trình xử lý thông điệp được gọi khi điểm cuối nhận được một tin nhắn văn bản.

Mã sau triển khai điểm cuối lập trình này trong ứng dụng Java EE:

```
ServerEndpointConfig.Builder.create      (EchoEndpoint. Class , "/      echo" )
.build();
```

Khi triển khai, điểm cuối sẽ sẵn có tại `ws://<host>:<port>/<application>/echo`; ví dụ, `ws://localhost:8080/echoapp/echo`.

9.2.2. Điểm cuối chú thích

Ví dụ sau cho thấy cách tạo cùng một điểm cuối như ví dụ trên bằng cách sử dụng cơ chế chú thích (annotation):

```
@ServerEndpoint("/echo")
    public class EchoEndpoint {
       @OnMessage
       public void onMessage(Session session, String msg) {
          try {
             session.getBasicRemote().sendText(msg);
          } catch (IOException e) { ... }
       }
    }
```

Cùng một chức năng, điểm cuối chú thích đơn giản hơn điểm cuối lập trình, nó được triển khai tự động với ứng dụng đến đường dẫn tương đối được xác định trong chú thích ServerEndpoint. Thay vì phải tạo thêm lớp cho trình xử lý thông báo, ví dụ này sử dụng chú thích OnMessage để chỉ định phương thức được gọi để xử lý thông báo.

Bảng dưới liệt kê các chú thích có sẵn trong gói javax.websocket để chỉ rõ phương thức xử lý các sự kiện xảy ra trong vòng đời của WebSocket. Các ví dụ trong bảng hiển thị các tham số phổ biến cho các phương thức này.

Bảng Chú thích liên quan đến các sự kiện trong vòng đời của WebSocket

Chú giải	Sự kiện	Ví dụ
`OnOpen`	Kết nối được mở	`@OnOpen public void open(Session session, EndpointConfig conf) { }`
`OnMessage`	Nhận được tín nhắn	`@OnMessage public void message(Session session, String msg) { }`
`OnError`	Lỗi kết nối	`@OnError public void error(Session session,Throwable error) { }`
`OnClose`	Đóng kết nối	`@OnClose public void close(Session session,CloseReason reason) { }`

9.2.3. Gửi và nhận thông điệp

Điểm cuối WebSocket có thể gửi và nhận thông điệp văn bản và thông điệp nhị phân. Ngoài ra, nó cũng có thể gửi khung ping và nhận khung pong. Phần này mô tả cách sử dụng các giao diện Session và RemoteEndpoint để gửi thông báo đến máy ngang hàng được kết nối và cách sử dụng chú thích OnMessage để nhận thông báo.

a) Gửi thông điệp

Thực hiện các bước sau tại điểm cuối để gửi thông điệp:

1. Lấy đối tượng Session từ kết nối.

   Đối tượng `Session` là tham số trong các phương thức chú thích của điểm cuối, như minh họa trong bảng trên. Khi thông điệp là phản hồi cho một thông điệp từ máy khác, đối tượng Session đã được khởi tạo sẵn bên trong phương thức nhận thông điệp (phương thức được chú thích bằng @OnMessage). Nếu không phải là phản hồi, đối tượng Session là biến thực thể của lớp điểm cuối trong phương thức được chú thích với @OnOpen để ta có thể truy cập nó từ các phương thức khác.

2. Sử dụng đối tượng Session để lấy đối tượng RemoteEndpoint.

   Phương thức Session.getBasicRemote và Session.getAsyncRemote trả về các đối tượng RemoteEndpoint.Basic và RemoteEndpoint.Async. Giao tiếp RemoteEndpoint.Basic cung cấp phương thức chặn để gửi thông điệp; giao diện RemoteEndpoint.Async cung cấp phương pháp không chặn dừng.

3. Sử dụng đối tượng RemoteEndpoint để gửi tới kết nối ngang hàng.

   Dưới đây là một số phương thức có thể sử dụng để gửi thông điệp.

   - `void RemoteEndpoint.Basic.sendText(String text)`

     Gửi thông điệp, phương thức này chặn luồng cho đến khi toàn bộ thông điệp được truyền đi.

   - `void RemoteEndpoint.Basic.sendBinary(ByteBuffer data)`

     Gửi thông điệp nhị phân, phương thức này chặn luồng cho đến khi toàn bộ thông điệp được truyền đi.

   - `void RemoteEndpoint.sendPing(ByteBuffer appData)`

     Gửi một khung ping.

   - `void` RemoteEndpoint`.sendPong(ByteBuffer appData)`

     Gửi một khung pong.

Ví dụ trong điểm cuối chú thích trình bày cách sử dụng những phương thức trên để trả lời mọi thông điệp văn bản đến.

b) Gửi thông điệp tới tất cả kết nối và điểm cuối

Mỗi thực thể của lớp điểm cuối được liên kết với một và chỉ một kết nối và kết nối ngang hàng; tuy nhiên, có những trường hợp trong đó thực thể điểm cuối cần gửi thông báo đến tất cả các kết nối ngang hàng được kết nối. Ví dụ bao gồm các ứng dụng trò chuyện và đấu giá trực tuyến. Giao diện `Session` cung cấp phương thức `getOpenSessions` cho phép thực hiện

chức năng này. Ví dụ sau minh họa cách sử dụng phương thức để chuyển tiếp thông điệp văn bản đến cho tất cả các thực thể ngang hàng được kết nối:

```
public class EchoAllEndpoint {
       @OnMessage
       public void onMessage(Session session, String msg) {
          try {
             for (Session sess : session.getOpenSessions()) {
                if (sess.isOpen())
                   sess.getBasicRemote().sendText(msg);
             }
          } catch (IOException e) { ... }
       }

    }
```

c) Nhận thông điệp

Chú thích OnMessage chỉ định phương thức xử lý thông điệp đến. Có thể có nhiều nhất ba phương thức được chú thích với @OnMessage trong một điểm cuối, một phương thức cho mỗi loại thông báo: văn bản, nhị phân và pong. Ví dụ sau minh họa cách chỉ định các phương thức để nhận cả ba loại thông điệp:

```
public class ReceiveEndpoint {
       @OnMessage
       public void textMessage(Session session, String msg) {
          System.out.println("Text message: " + msg);
       }
       @OnMessage
       public void binaryMessage(Session session, ByteBuffer msg) {
          System.out.println("Binary message: " + msg.toString());
       }
       @OnMessage
       public void pongMessage(Session session, PongMessage msg) {
          System.out.println("Pong message: " +
                             msg.getApplicationData().toString());
       }
    }
```

9.2.4. Duy trì trạng thái máy khách

Vì bộ chứa tạo một thể hiện của lớp điểm cuối cho tất cả kết nối, ta có thể xác định và sử dụng các biến thể hiện để lưu trữ thông tin trạng thái máy khách. Ngoài ra, phương thức Session.getUserProperties cung cấp một bản đồ có thể sửa đổi để lưu trữ các thuộc tính của người dùng. Ví dụ, điểm cuối sau trả lời thông điệp văn bản đến có nội dung của thông điệp trước đó từ mỗi máy khách:

```
public class DelayedEchoEndpoint {
        @OnOpen
        public void open(Session session) {
           session.getUserProperties().put("previousMsg", " ");
        }
        @OnMessage
        public void message(Session session, String msg) {
           String prev = (String) session.getUserProperties()
                                         .get("previousMsg");
           session.getUserProperties().put("previousMsg", msg);
           try {
              session.getBasicRemote().sendText(prev);
           } catch (IOException e) { ... }
        }
 }
```

Để lưu trữ thông tin chung cho tất cả các máy khách được kết nối, ta có thể sử dụng các biến lớp (tĩnh); tuy nhiên, cần chú ý đảm bảo quyền truy cập an toàn theo luồng.

9.2.5. Sử dụng bộ mã hóa và bộ giải mã

Các API cho WebSocket cung cấp hỗ trợ chuyển đổi giữa các thông điệp WebSocket và các kiểu Java tùy chỉnh bằng cách sử dụng bộ mã hóa và bộ giải mã. Một bộ mã hóa nhận một đối tượng Java và tạo ra một biểu diễn có thể được truyền đi dưới dạng thông điệp WebSocket; ví dụ, các bộ mã hóa thường tạo ra các biểu diễn JSON, XML hoặc nhị phân. Một bộ giải mã thực hiện chức năng ngược lại; nó đọc một thông điệp WebSocket và tạo một đối tượng Java.

Cơ chế này đơn giản hóa các ứng dụng WebSocket, vì nó tách rời logic nghiệp vụ khỏi tuần tự hóa và giải mã hóa các đối tượng.

a) Chuyển đổi đối tượng Java thành các thông báo WebSocket

Quy trình để thực thi và sử dụng bộ mã hóa trong các điểm cuối như sau.

1. Thực thi một trong các giao tiếp sau:
   - `Encoder.Text<T>` cho thông điệp văn bản
   - `Encoder.Binary<T>` cho thông điệp nhị phân

   Các giao diện này xác định phương thức `encode`. Thực thi một lớp bộ mã hóa cho từng kiểu Java tùy chỉnh mong muốn gửi dưới dạng thông báo WebSocket.

1. Thêm tên của thực thi bộ mã hóa vào tham số tùy chọn của `encoders` của chú thích ServerEndpoint.
2. Sử dụng phương thức `sendObject(Object data)` của giao diện RemoteEndpoint.Basic hoặc RemoteEndpoint.Async để gửi đối tượng như thông

điệp. Bộ chứa tìm kiếm một bộ mã hóa phù hợp với kiểu đã dùng và sử dụng nó để chuyển đổi đối tượng thành thông điệp WebSocket.

Ví dụ, nếu ta có hai kiểu Java (MessageA và MessageB) muốn gửi dưới dạng thông điệp văn bản, chỉ cần thực thi giao diện Encoder.Text <MessageA> và Encoder.Text <MessageB> như sau:

```
public class MessageATextEncoder implements Encoder.Text<MessageA> {
        @Override
        public void init(EndpointConfig ec) { }
        @Override
        public void destroy() { }
        @Override
        public String encode(MessageA msgA) throws EncodeException {
           // Access msgA's properties and convert to JSON text...
           return msgAJsonString;
        }
}
```

Tương tự với `Encoder.Text<MessageB>`, sau đó thêm tham số `encoders` vào chú `ServerEndpoint`:

```
@ServerEndpoint(value = "/myendpoint",encoders = { MessageATextEncoder.class,
                  MessageBTextEncoder.class })
public class EncEndpoint { ... }
```

Bây giờ, ta có thể gửi các đối tượng MessageA và MessageB dưới dạng thông điệp WebSocket bằng phương thức sendObject như sau:

```
MessageA msgA = new MessageA(...);
MessageB msgB = new MessageB(...);
session.getBasicRemote.sendObject(msgA);
session.getBasicRemote.sendObject(msgB);
```

Như trong ví dụ này, ta có thể có nhiều bộ mã hóa cho thông điệp văn bản và nhiều bộ mã hóa cho thông điệp nhị phân. Giống như điểm cuối, các bản sao bộ mã hóa được liên kết với một và chỉ một kết nối WebSocket và thực thể ngang hàng, do đó, chỉ có một luồng thực thi mã của một thể hiện bộ mã hóa tại bất kỳ thời điểm nào.

b) Chuyển đổi thông báo WebSocket thành các đối tượng Java

Quy trình thực thi và sử dụng bộ giải mã trong các điểm cuối như sau.

1. Thực thi một trong các giao diện sau:

- Decoder.Text<T> cho thông điệp văn bản
- Decoder.Binary<T> cho thông điệp nhị phân

Các giao diện này chỉ định các phương thức *willDecode* và *decode*.

Không giống như với bộ mã hóa, ta có thể chỉ định nhiều nhất *một* bộ giải mã cho thông điệp nhị phân và *một* bộ giải mã cho thông điệp văn bản.

2. Thêm tên của các thực thi bộ giải mã vào tham số tùy chọn của bộ giải mã trong chú thích ServerEndpoint.

3. Sử dụng chú thích OnMessage trong điểm cuối để chỉ định một phương thức lấy kiểu Java tùy chỉnh của ta làm tham số. Khi điểm cuối nhận được thông điệp có thể được giải mã bởi một trong những bộ giải mã ta đã chỉ định, bộ chứa sẽ gọi phương thức được chú thích với @OnMessage lấy kiểu Java tùy chỉnh của ta làm tham số nếu phương thức này tồn tại.

Ví dụ, nếu có hai kiểu Java (MessageA và MessageB) cần gửi và nhận dưới dạng thông điệp văn bản, hai kiểu này sẽ thừa kế lớp `Message`. Vì chỉ có thể xác định một bộ giải mã cho thông điệp văn bản, bộ giải mã cho lớp `Message` được thực thi như sau:

```
public class MessageTextDecoder implements Decoder.Text<Message> {
        @Override
        public void init(EndpointConfig ec) { }
        @Override
        public void destroy() { }
        @Override
        public Message decode(String string) throws DecodeException {
           // Read message...
           if ( /* message is an A message */ )
              return new MessageA(...);
           else if ( /* message is a B message */ )
              return new MessageB(...);
        }
        @Override
        public boolean willDecode(String string) {
           // Determine if the message can be converted into either a
           // MessageA object or a MessageB object...
           return canDecode;

        }
}
```

Sau đó, thêm tham số `decoder` vào chú thích ServerEndpoint :

```
@ServerEndpoint(
              value = "/myendpoint",
              encoders = { MessageATextEncoder.class,
                 MessageBTextEncoder.class },
              decoders = { MessageTextDecoder.class }
```

```
        )
public class EncDecEndpoint { ... }
```

Tiếp đến, xác định một phương thức trong lớp điểm cuối nhận các đối tượng MessageA và MessageB như sau:

```
@OnMessage
public void message(Session session, Message msg) {
   if (msg instanceof MessageA) {
      // We received a MessageA object...
   } else if (msg instanceof MessageB) {
      // We received a MessageB object...
   }
}
```

Giống như các điểm cuối, các thể hiện của bộ giải mã được liên kết với một và chỉ một kết nối WebSocket và kết nối ngang hàng, vì vậy chỉ có một luồng thực thi mã của một thể hiện của bộ giải mã tại bất kỳ thời điểm nào.

9.2.6. Tham số đường dẫn

Chú thích ServerEndpoint cho phép ta sử dụng mẫu URI để xác định các bộ phận URI của điểm cuối, như tham số ứng dụng. Ví dụ:

```
@ServerEndpoint("/chatrooms/{room-name}")
public class ChatEndpoint {
   ...
}
```

Nếu điểm cuối được triển khai bên trong ứng dụng web có tên chatapp tại máy chủ Java EE cục bộ ở cổng 8080, máy khách có thể kết nối với điểm cuối bằng bất kỳ URI nào sau đây
http://localhost:8080/chatapp/chatrooms/currentnews

http://localhost:8080/chatapp/chatrooms/music

http://localhost:8080/chatapp/chatrooms/cars

http://localhost:8080/chatapp/chatrooms/technology

Các điểm cuối chú thích có thể nhận các tham số đường dẫn dưới dạng đối số trong các phương thức được chú thích bằng @OnOpen , @OnMessage và @OnClose . Trong ví dụ này, điểm cuối sử dụng tham số trong phương thức @OnOpen để xác định phòng trò chuyện mà máy khách muốn tham gia:

```
@ServerEndpoint("/chatrooms/{room-name}")
   public class ChatEndpoint {
      @OnOpen
      public void open(Session session,
```

```
                        EndpointConfig c,
                        @PathParam("room-name") String roomName) {
            // Add the client to the chat room of their choice ...
        }

    }
```

Các tham số đường dẫn được sử dụng làm đối số trong các phương thức này có thể là chuỗi, kiểu gốc hoặc kiểu bao bọc tương ứng.

9.2.7. Xử lý lỗi

Để chỉ định một phương pháp xử lý lỗi trong điểm cuối WebSocket chú thích, sử dụng chú thích @OnError, như sau:

```
@ServerEndpoint("/testendpoint")
public class TestEndpoint {
    ...
    @OnError
    public void error(Session session, Throwable t) {
        t.printStackTrace();
        ...
    }

}
```

Phương thức này được gọi khi có sự cố kết nối, lỗi thời gian chạy từ trình xử lý thông báo hoặc lỗi chuyển đổi khi giải mã thông điệp.

9.2.8. Đặc tả lớp cấu hình điểm cuối

Các API cho WebSocket cho phép ta định cấu hình cách bộ chứa tạo các điểm cuối máy chủ. Ta có thể cung cấp logic cấu hình điểm cuối tùy chỉnh để:

- Truy cập chi tiết của yêu cầu HTTP ban đầu cho kết nối WebSocket
- Thực hiện kiểm tra tùy chỉnh trên tiêu đề `Origin` của HTTP
- Sửa đổi phản hồi bắt tay WebSocket
- Chọn một giao thức con của WebSocket từ những giao thức được máy khách yêu cầu
- Kiểm soát việc khởi tạo và khởi tạo các thực thể điểm cuối

Để cung cấp logic cấu hình điểm cuối tùy chỉnh, chỉ cần thừa kế lớp ServerEndpointConfig.Configurator và ghi đè một số phương thức. Tiếp đến, trong lớp điểm cuối, chỉ định lớp trình cấu hình bằng cách sử dụng tham số bộ `configurator` của chú thích ServerEndpoint .

Ví dụ, lớp trình cấu hình sau tạo đối tượng yêu cầu bắt tay cho các điểm cuối:

```
public class CustomConfigurator extends ServerEndpointConfig.Configurator {

        @Override
        public void modifyHandshake(ServerEndpointConfig conf,
                                    HandshakeRequest req,
                                    HandshakeResponse resp) {

            conf.getUserProperties().put("handshakereq", req);
        }


    }
```

Lớp điểm cuối dưới đây định cấu hình các điểm cuối với trình cấu hình tùy chỉnh, cho phép cập đối tượng yêu cầu bắt tay:

```
@ServerEndpoint(value = "/myendpoint",configurator = CustomConfigurator.class)
      public class MyEndpoint {
          @OnOpen
          public void open(Session s, EndpointConfig conf) {
              HandshakeRequest req = (HandshakeRequest)
                  conf.getUserProperties().get("handshakereq");
              Map<String,List<String>> headers = req.getHeaders();
              ...
          }
}
```

Lớp điểm cuối có thể sử dụng đối tượng yêu cầu bắt tay để truy cập các chi tiết của yêu cầu HTTP ban đầu, ví dụ như các tiêu đề , hoặc đối tượng HttpSession.

9.3. ỨNG DỤNG WEBSOCKETBOT

Phần này sẽ trình bày ví dụ sử dụng WebSocket để xây dựng ứng dụng trò chuyện. Phòng trò chuyện cho phép nhiều người cùng tham gia và thảo luận. Người dùng có thể hỏi những câu hỏi đơn giản tới bot, luôn sẵn sàng trong phòng chat.

9.3.1. Kiến trúc

a) Kiến trúc của ví dụ

Ứng dụng websocketbot bao gồm các phần tử sau:

- Một bean CDI (BotBean) chứa logic để bot trả lời thông điệp
- Điểm cuối WebSocket (BotEndpoint) thực thi phòng trò chuyện
- Một tập hợp các lớp (Message, ChatMessage, InfoMessage , JoinMessage và UsersMessage) đại diện cho các thông điệp của ứng dụng

- Một tập hợp các lớp (ChatMessageEncoder, InfoMessageEncoder, JoinMessageEncoder và UsersMessageEncoder) mã hóa thông điệp ứng dụng thành thông điệp văn bản WebSocket dưới dạng dữ liệu JSON
- Lớp phân giải (MessageDecoder) phân tích thông điệp văn bản WebSocket dưới dạng dữ liệu JSON và giải mã chúng thành đối tượng JoinMessage hoặc ChatMessage
- Trang HTML (index.html) sử dụng JavaScript để triển khai ứng dụng khách cho phòng trò chuyện

b) Bean CDI

Bean CDI (BotBean) là lớp Java chứa phương thức `respond` . Phương thức này so sánh tin nhắn trò chuyện đến với một tập hợp các câu hỏi được xác định trước và trả về phản hồi phù hợp.

```
@Named
    public class BotBean {
        public String respond(String msg) { ... }
}
```

c) Điểm cuối WebSocket

Điểm cuối WebSocket (BotEndpoint) là một điểm cuối chú thích thực hiện các chức năng sau:

- Nhận thông điệp từ máy khách
- Chuyển tiếp thông điệp cho máy khách
- Duy trì danh sách các máy khách được kết nối
- Gọi chức năng bot

Điểm cuối chỉ định URI triển khai và các bộ mã hóa/giải mã thông báo bằng chú thích ServerEndpoint . Điểm cuối nhận được một thực thể của lớp BotBean và một tài nguyên dịch vụ thực thi được quản lý thông qua tiêm phụ thuộc:

```
@ServerEndpoint(value = "/websocketbot",decoders = { MessageDecoder.class },
                    encoders = { JoinMessageEncoder.class,
                    ChatMessageEncoder.class,
                    InfoMessageEncoder.class, UsersMessageEncoder.class })
            /* There is a BotEndpoint instance per connection */
      public class BotEndpoint {
             private static final Logger logger =
                  Logger.getLogger("BotEndpoint");
               /* Bot functionality bean */
               @Inject private BotBean botbean;
               /* Executor service for asynchronous processing */
               @Resource(name="comp/DefaultManagedExecutorService")
```

```
    private ManagedExecutorService mes;

    @OnOpen
    public void openConnection(Session session) {
        logger.log(Level.INFO, "Connection opened.");
    }
    ...
}
```

Phương thức `message` xử lý thông điệp đến từ máy khách. Bộ giải mã chuyển thông điệp văn bản đến thành đối tượng JoinMessage hoặc ChatMessage, kế thừa từ lớp `Message`. Phương thức `Message` có tham số là đối tượng `Message`:

```
@OnMessage
public void message(Session session, Message msg) {
   logger.log(Level.INFO, "Received: {0}", msg.toString());

   if (msg instanceof JoinMessage) {
      /* Add the new user and notify everybody */
      JoinMessage jmsg = (JoinMessage) msg;
      session.getUserProperties().put("name", jmsg.getName());
      session.getUserProperties().put("active", true);
      logger.log(Level.INFO, "Received: {0}", jmsg.toString());
      sendAll(session, new InfoMessage(jmsg.getName() +
              " has joined the chat"));
      sendAll(session, new ChatMessage("Duke", jmsg.getName(),
              "Hi there!!"));
      sendAll(session, new UsersMessage(this.getUserList(session)));

   } else if (msg instanceof ChatMessage) {
      /* Forward the message to everybody */
      ChatMessage cmsg = (ChatMessage) msg;
      logger.log(Level.INFO, "Received: {0}", cmsg.toString());
      sendAll(session, cmsg);
      if (cmsg.getTarget().compareTo("Duke") == 0) {
         /* The bot replies to the message */
         mes.submit(new Runnable() {
            @Override
            public void run() {
               String resp = botbean.respond(cmsg.getMessage());
               sendAll(session, new ChatMessage("Duke",
                       cmsg.getName(), resp));
            }
         });
```

```
        }
    }
}
```

Nếu thông điệp là thông điệp tham gia, điểm cuối sẽ thêm người dùng mới vào danh sách và thông báo cho tất cả các máy khách được kết nối. Nếu thông điệp là trò chuyện, điểm cuối sẽ chuyển tiếp nó đến tất cả các máy khách được kết nối.

Nếu một thông điệp trò chuyện gửi cho bot, điểm cuối sẽ nhận được phản hồi bằng cách sử dụng BotBean và gửi đến tất cả các máy khách được kết nối. Phương thức sendAll tương tự như ví dụ trong gửi tin nhắn đến tất cả thực thể ngang hàng kết nối với một điểm cuối ở ví dụ trên.

Lựa chọn giữa xử không đồng bộ và đồng bộ

Điểm cuối WebSocket gọi phương thức BotBean.respond để nhận phản hồi từ bot. Trong ví dụ này, đây là một thao tác chặn; người dùng gửi tin nhắn sẽ không thể gửi hoặc nhận các tin nhắn trò chuyện khác cho đến khi thao tác hoàn tất. Để tránh sự cố này, điểm cuối lấy một dịch vụ thực thi từ bộ chứa và thực hiện thao tác chặn trong một luồng khác bằng cách sử dụng phương thức ManagedExecutorService.submit từ các tiện ích đồng thời (Concurrency Utilities) cho Java EE.

Đặc tả API cho WebSocket yêu cầu thực thi Java EE khởi tạo các lớp điểm cuối một lần cho mỗi kết nối. Điều này tạo điều kiện thuận lợi cho phát triển của các điểm cuối WebSocket, vì ta được đảm bảo rằng chỉ một luồng đang thực thi mã trong lớp điểm cuối WebSocket tại bất kỳ thời điểm nào. Khi ta đưa vào một luồng mới trong một điểm cuối, như trong ví dụ này, ta phải đảm bảo rằng các biến và phương thức được nhiều hơn một luồng truy cập là luồng an toàn. Trong ví dụ này, mã trong BotBean là luồng an toàn và phương thức BotEndpoint.sendAll đã được `synchronized`.

d) Thông điệp Ứng dụng

Các lớp đại diện cho thông điệp ứng dụng (`Message`, `ChatMessage`, `InfoMessage`, `JoinMessage`, và `UsersMessage`) chỉ chứa các thuộc tính và các phương thức getter và setter. Ví dụ:

```
public class ChatMessage extends Message {
    private String name;
    private String target;
    private String message;
    /* ... Constructor, getters, and setters ... */
}
```

e) Các lớp mã hóa

Các lớp bộ mã hóa chuyển đổi các đối tượng thông báo ứng dụng thành văn bản JSON bằng cách sử dụng API cho Xử lý JSON. Ví dụ, lớp ChatMessageEncoder được triển khai như sau:

```
public class ChatMessageEncoder implements Encoder.Text<ChatMessage> {
   @Override
```

```
        public void init(EndpointConfig ec) { }
        @Override
        public void destroy() { }
        @Override
        public String encode(ChatMessage chatMessage) throws EncodeException {
            // Access properties in chatMessage and write JSON text...
        }
    }
```

f) Bộ giải mã thông điệp

Lớp bộ giải mã thông điệp (MessageDecoder) chuyển đổi thông điệp văn bản WebSocket thành tin nhắn ứng dụng bằng cách phân tích cú pháp văn bản JSON. Ví dụ như sau:

```
/* Encode a ChatMessage as JSON.
     * For example, (new ChatMessage("Peter","Duke","How are you?"))
     * is encoded as follows:
     * {"type":"chat","target":"Duke","message":"How are you?"}
     */
    /* Decode a JSON message into a JoinMessage or a ChatMessage.
     * For example, the incoming message
     * {"type":"chat","name":"Peter","target":"Duke","message":"How are you?"}
     * is decoded as (new ChatMessage("Peter", "Duke", "How are you?"))
     */
    public class MessageDecoder implements Decoder.Text<Message> {
        /* Stores the name-value pairs from a JSON message as a Map */
        private Map<String,String> messageMap;

        @Override
        public void init(EndpointConfig ec) { }
        @Override
        public void destroy() { }

        /* Create a new Message object if the message can be decoded */
        @Override
        public Message decode(String string) throws DecodeException {
            Message msg = null;
            if (willDecode(string)) {
                switch (messageMap.get("type")) {
                    case "join":
                        msg = new JoinMessage(messageMap.get("name"));
                        break;
                    case "chat":
                        msg = new ChatMessage(messageMap.get("name"),
                                              messageMap.get("target"),
```

```
                                        messageMap.get("message"));
                }
            } else {
                throw new DecodeException(string, "[Message] Can't decode.");
            }
            return msg;
        }

        /* Decode a JSON message into a Map and check if it contains
         * all the required fields according to its type. */
        @Override
        public boolean willDecode(String string) {
            // Convert JSON data from the message into a name-value map...
            // Check if the message has all the fields for its message type...
        }
    }
```

g) Trang HTML

Trang HTML (index.html) chứa một trường cho tên người dùng. Sau khi người dùng nhập tên và nhấp vào **Join** , ba vùng văn bản có sẵn: một vùng để nhập và gửi tin nhắn, một vùng dành cho phòng trò chuyện và một vùng chứa danh sách người dùng. Trang này cũng chứa bảng điều khiển WebSocket hiển thị các tin nhắn được gửi và nhận dưới dạng văn bản JSON.

Mã JavaScript trên trang sử dụng API WebSocket để kết nối với điểm cuối, gửi tin nhắn và chỉ định phương thức gọi lại. API WebSocket được hỗ trợ bởi hầu hết các trình duyệt hiện nay và được sử dụng rộng rãi để phát triển ứng dụng khách với HTML5.

CHƯƠNG 10. LẬP TRÌNH WEB VỚI JSP VÀ TOMCAT SERVER

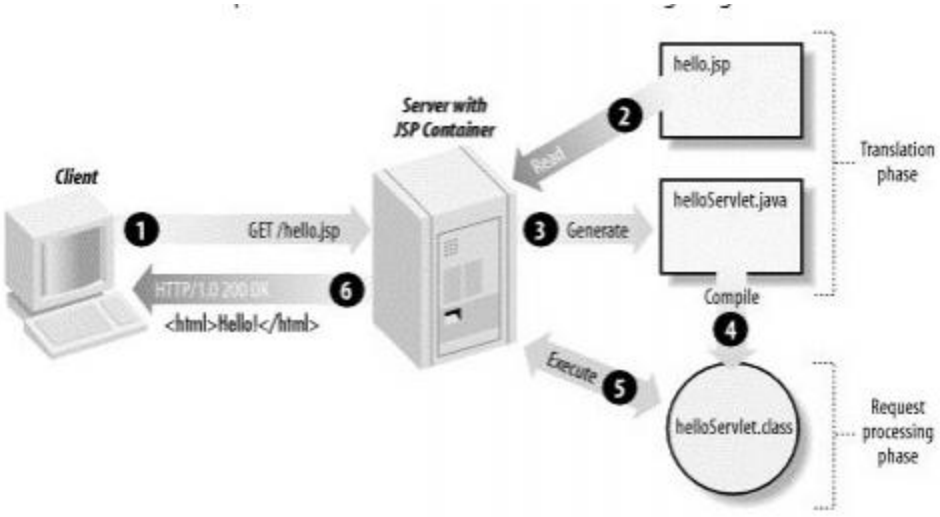
Nội dung chương này sẽ trình bày về:

- Giới thiệu về JSP
- Thiết kế, cài đặt và triển khai ứng dụng web với JSP-Tomcat

10.1. GIỚI THIỆU JSP

10.1.1. Giới thiệu JSP

JSP (viết tắt của tiếng Anh JavaServer Pages) còn được biết đến với một cái tên khác là Java Scripting Preprocessor - tạm dịch là "Bộ tiền xử lý văn lệnh Java" - là một công nghệ Java cho phép các nhà phát triển tạo nội dung HTML, XML hay một số định dạng khác của trang web một cách năng động, trong khi hồi âm yêu cầu của trình khách. Công nghệ này cho phép người ta nhúng mã Java và một số hành động xử lý đã được định trước (pre-defined actions) vào trong nội dung tĩnh của trang.



Hình 9.1: Chu trình hoạt động của một trang JSP

Trang JSP có chu trình sống xác định tính từ khi hệ thống đọc biên dịch trang JSP, gọi thực thi và loại bỏ trang ra khỏi bộ nhớ. Chu trình sống của trang JSP gồm có 5 giai đoạn sau:

- ***Biên dịch trang*:** Khi trình duyệt yêu cầu trang JSP, Web server sẽ kiểm tra xem trang JSP đã được biên dịch hay chưa. Nếu chưa biên dịch hoặc đã biên dịch nhưng trang JSP mới vừa thay đổi mã nguồn thì Web Server sẽ thực hiện biên dịch trang JSP. Quá trình biên dịch JSP thực tế là chuyển trang JSP thành servlet. File biên dịch .class của trang chỉ diễn ra một lần. Nếu trang đã biên dịch và sau đó không bị thay đổi trong mã nguồn thì

quá trình biên dịch sẽ không xảy ra nữa, do đó mà tốc độ thực thi sẽ nhanh hơn. Sau khi biên dịch, mã trang sẽ được nạp vào bộ nhớ để thực thi. Quá trình biên dich trang JSP sẽ được diễn ra như sau:

- Bước 1: Kiểm tra xem trang đã được dịch thành mã nguồn tương đương servlet hay chưa.
- Bước 2: Nếu chưa được biên dịch thì trang JSP sẽ được biên dịch thành file nguồn .java theo cấu trúc của servlet. Gọi trình biên dịch javac biên dịch file nguồn .java thành file thực thi của servlet .class.
- Bước 3: Nạp servlet đã biên dịch ở bước 2, thực thi trả kết quả về cho trình khách.
- Bước 4: Nếu file JSP đã được biên dịch trước đó : thực hiện kiểm tra xem nội dung file .jsp có thay đổi không, Nếu có thì quay lại bước 2 biên dịch lại trang, nếu không thì quay lại bước 3.

- ***Nạp trang*** : Kể từ giai đoạn này, quá trình nạp trang tương tự như servlet (trang JSP sau khi biên dịch có thể coi như một servlet). Chỉ có một điểm khác là servlet chỉ được nạp một lần trong khi mã trang JSP mặc dù đã biên dịch nhưng phải nạp lại nhiều lần mỗi khi web server nhận được yêu cầu trang từ trình duyệt.

- ***Khởi tạo*** : Khi nạp mã trang thành công, Web server sẽ gọi đến phương thức khỏi tạo trang. Và mặc dù JSP được biên dịch ra servlet nhưng phương thức khởi tạo cho trang JSP lại mang tên là jspInit() chứ không phải là init() như servlet.

- ***Thực thi***: Sau quá trình khởi tạo, Web server sẽ gọi đến phương thức _jspService (khác với servlet gọi đến doPost(), doGet() hoặc service()). Phương thức _jspService sẽ chuyển đến hai lớp đối tượng HttpServletRequest và HttpServletResponse để đọc và ghi kết xuất trả về trình khách.

- ***Dọn dẹp*** : Khi trang JSP đã thực thi xong, trình chủ Web Server sẽ gọi phương thức jspDestroy() để giải phóng mã trang khỏi bộ nhớ. Tương tự như trong Servlet, có thể cài đặt phương thức jspDestroy() thực hiện giải phóng vùng nhớ hoặc đóng kết nối trả về tài nguyên cho hệ thống.

Về cơ bản, một trang JSP có chức năng giống một lớp Servlet. Tuy nhiên, điểm khác biệt lớn nhất giữa chúng là về cú pháp. Trong khi Servlet là một lớp thuần Java, được viết và biên dịch với Java thì JSP là một trang lẫn lộn giữa HTML và Java. Thường người ta dùng các cú pháp lệnh điều khiển của Java để điều khiển sự hiển thị của HTML trong cùng một trang JSP.

Ví dụ, trang chủ của người dùng sau khi đăng nhập thành công sẽ hiện lên câu "welcome + username của người dùng". Nếu viết dưới dạng servlet thì nó là một lớp Java có mã nguồn như sau:

```
User user= (User)session.getAttribute("user");
if(user == null){
	response.sendRedirect("Home.jsp?ok=0");
}

out.println("<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">");
out.println("<HTML><HEAD><TITLE>JSP demo test</TITLE>");
out.println("<META http-equiv=Content-Type content="text/html; charset=iso-
8859-1">");
out.println("</HEAD><BODY leftMargin=0 topMargin=0>");
out.println(user.getUsername() + "!");
out.println("</BODY></HTML>");
```

Trong khi nếu viết với JSP thì là một trang JSP với mã nguồn như sau:

```
<%@page language="java" import = " java.util.*, java.awt.*, entity.*, dao.*"
%>
<%
	User user= (User)session.getAttribute("user");
	if(user == null){
		response.sendRedirect("Home.jsp?ok=0");
	}
 %>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML><HEAD><TITLE>JSP demo test</TITLE>
<META http-equiv=Content-Type content="text/html; charset=iso-8859-1">
</HEAD>
<BODY leftMargin=0 topMargin=0>
	Welcome <%= user.getUsername() %>!
</BODY>
</HTML>
```

Dễ dàng thấy sự khác biệt là trong Servlet, mã HTML phải được in ra theo lệnh out.println(), trong khi đó, trong trang JSP, mã HTML được viết trực tiếp như một file HTML thông thường. Chỉ có những phần code Java thì phải bỏ trong các đoạn bắt đầu bằng "<%" và kết thúc bằng "%>".

10.1.2. Kiến trúc một ứng dụng web với JSP-Tomcat

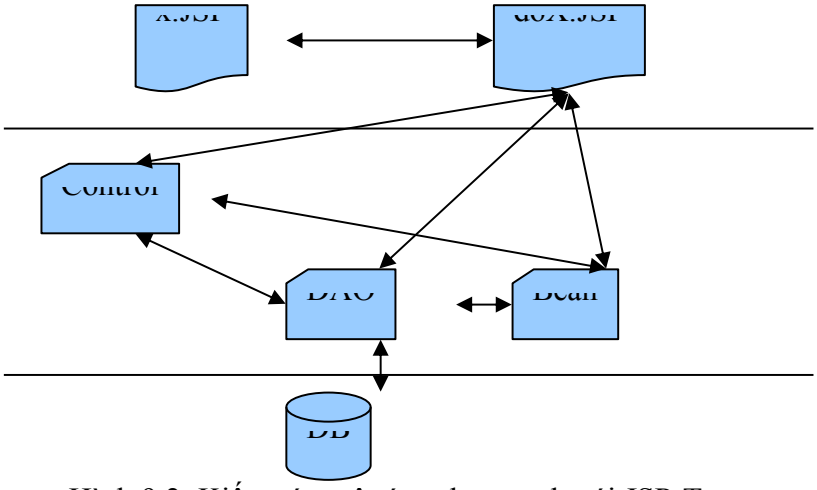
Mô hình kiến trúc tổng quan theo J2EE của một ứng dụng web sử dụng JSP-Tomcat được mô tả như Hình 9.2. Trong đó:

- Tầng lưu trữ: là tầng dưới cùng, bao gồm cá hệ thống lưu trữ như các hệ quản trị CSDL, các hệ thống file trên bộ nhớ ngoài.

- Tầng nghiệp vụ: là tầng giữa, bao gồm các lớp Java: các lớp Java để tính toán nghiệp vụ (Business/control), các lớp để truy cập cơ sở dữ liệu (DAO), các lớp thực thể (Entity/bean/model).

- Tầng giao diện: là tầng trên cùng, bao gồm các trang jsp: Các trang jsp hiện form để nhập dữ liệu, các trang jsp hiện kết quả tính toán, các trang jsp để chuyển hướng lẫn nhau giữa các trang jsp trong cùng tầng trên này.

Các ví dụ ứng dụng của chương này đều dựa trên kiến trúc này. Các bài tập về ứng dụng web với JSP đều khuyến khích thiết kế và cài đặt ứng dụng theo kiến trúc này.



Hình 9.2: Kiến trúc một ứng dụng web với JSP-Tomcat

10.2. TRAO ĐỔI DỮ LIỆU GIỮA CÁC TRANG JSP

10.2.1. Dùng Parameter

Giả sử trang có form nhận dữ liệu tên là x.jsp, sau khi nhận dữ liệu trên form (được submit), thì trang x.jsp sẽ chuyển dữ liệu sang trang doX.jsp để xử lí. Khi đó, để truyền dữ liệu theo kiểu dùng đối tượng Parameter, mỗi bên phải tiến hành như sau.

Bên gửi dữ liệu – trang x.jsp

- Khai báo form nhập dữ liệu với các thuộc tính:

```
method = "POST"
action = "doX.jsp"
```

- Khai báo các ô nhập với tên biến duy nhất, gọi là các *biến form*, và phải nhớ tên biến này để sang trang doX.jsp gọi lại.

- Khai báo một nút nhấn có nhiệm vụ submit form trên trang x.jsp gửi dữ liệu đến trang doX.jsp

Ví dụ, trang login.jsp hiện giao diện nhập username và password để người dùng đăng nhập, nó khai báo trang xử lí là doLogin.jsp, khi người dùng click submit form thì nó gửi usernane và password của người đăng nhập để gửi về server (trang doLogin.jsp) kiểm tra đăng nhập.

```
<form method="POST" name = "Dangnhap" action="doLogin.jsp" >
       <p align="left">Username:
       <input type="text" name="username" size="12"></p>
       <p align="left">Password:
       <input type="password" name="password" size="12"></p>
       <p align="left">
      <input type="submit" value="Submit" name="B1">
      <input type="reset" value="Reset" name="B2"></p>
</form>
```

Bên nhận dữ liệu – trang doX.jsp

- Gọi phương thức getParameter của đối tượng request theo cú pháp:

  tên_biến = request.getParameter(tên_biến_đã_khai_báo_trong_form_x.jsp)

Lưu ý:

- Phương thức này luôn trả về một đối tượng dạng String. Cho nên nếu muốn lấy dữ liệu kiểu số, hoặc kiểu ngày tháng thì phải ép kiểu.

- Bên form nhập của trang x.jsp có bao nhiêu biến form thì bên trang doX.jsp này phải gọi bấy nhiêu lần phương thức getParameter. Mỗi lần tương ứng với một biến form từ trang x.jsp.

- Phần code này là code Java cho nên phải đặt trong cặp kí hiệu "<%" và "%>" để phân biệt với code HTML thông thường trong trang jsp.

Ví dụ, trang doLogin.jsp phải khai báo 2 biến dạng String để nhận 2 giá trị tương ứng là username và password gửi từ trang login.jsp ở trên như sau:

```
<%
  String username = (String)request.getParameter("username");
  String password = (String)request.getParameter("password");
%>
```

10.2.2. Dùng bean

Bean trong jsp cũng là một dạng lớp Java bean mô tả một thực thể, một đối tượng mà ứng dngj phải xử lí. Trong đó có chứa các thuộc tính của thực thể, các phương thức get/set để truy nhập các thuộc tính của thực thể. Do vậy, để dùng được bean, trước hết phải khai báo lớp bean trong Java.

Khai báo lớp Bean trong Java

- Định nghĩa một lớp thực thể thuần trong Java với cá thuộc tính của thực thể, các hàm khởi tạo, các phương thức get/set tương ứng với từng thuộc tính của thực thể.

Ví dụ, để minh họa cho chức năng đăng nhập, ta cần khai báo lớp User dưới dạng bean như sau:

```
package entity;
public class User {
	private String username;
	private String password;

	public User(){}

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

	public String getUsername() {
		return username;
	}

	public void setUsername(String username) {
		this.username = username;
	}

	public String getPassword() {
		return password;
	}

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

Bên gửi dữ liệu – trang x.jsp

- Khai báo form nhập dữ liệu với các thuộc tính:

  method = “POST”

action = “doX.jsp”

- Khai báo các ô nhập với tên biến *phải trùng với tên các thuộc tính của lớp bean đã khai báo*, gọi là các *biến form*, và phải nhớ tên biến này để sang trang doX.jsp gọi lại.

- Khai báo một nút nhấn có nhiệm vụ submit form trên trang x.jsp gửi dữ liệu đến trang doX.jsp

Lưu ý:

- Form nhập dữ liệu trong cách dùng bean, về cơ bản không khác gì form nhập dữ liệu trong cách dùng parameter. Tuy nhiên, có một điểm khác biệt là tên các biến form phải trùng với tên các thuộc tính của lớp bean đã khai báo, có phân biệt chữ hoa chữ thường.

Ví dụ, trang login.jsp hiện giao diện nhập username và password để người dùng đăng nhập, nó khai báo trang xử lí là doLogin.jsp, khi người dùng click submit form thì nó gửi usernane và password của người đăng nhập để gửi về server (trang doLogin.jsp) kiểm tra đăng nhập.

```
<form method="POST" name = "Dangnhap" action="doLogin.jsp" >
       <p align="left">Username:
       <input type="text" name="username" size="12"></p>
       <p align="left">Password:
       <input type="password" name="password" size="12"></p>
       <p align="left">
      <input type="submit" value="Submit" name="B1">
      <input type="reset" value="Reset" name="B2"></p>
</form>
```

Bên nhận dữ liệu – trang doX.jsp

- Khai báo biến dùng thẻ jsp:bean theo cú pháp:

```
<jsp:useBean id="tên_biến" class="tên_lớp_bean" scope="request"/>
```

- Gọi lệnh để jsp tự ánh xạ dữ liệu trên form nhập vào các thuộc tính tương ứng của đối tượng:

```
<jsp:setProperty name="tên_biến" property="*"/>
```

Trong đó:

- tên_biến: là tên biến khai báo để dùng trong trang doX.jsp. Biến này sau khi khai báo thì sau đó có thể sử dụng trong thẻ jsp lẫn trong code Java.

- tên_lớp_bean: là tên lớp đã khai báo lớp bean. Nếu có package thì phải nêu cả tên package.

- Phần property: nếu muốn jsp ánh xạ toàn bộ giá trị trên các biến form vào tất cả các thuộc tính của lớp bean thì để dấu "*", nếu chỉ muốn ánh xạ một số thuộc tính thì liệt kê tên các thuộc tính trong giá trị của property, mỗi thuộc tính cách nhau bởi dấu phẩy.

Ví dụ, trang doLogin.jsp phải khai báo biến dạng User để nhận 2 giá trị tương ứng là username và password gửi từ trang login.jsp ở trên như sau:

```
<jsp:useBean id="user" class="entity.User" scope="request"/>
<jsp:setProperty name="user" property="*"/>
```

10.2.3. Dùng Response

Response là đối tượng dùng để chuyển trang từ trang jsp này sang trang jsp khác. Cú pháp cơ bản khi gọi đối tượng này để chuyển trang là gọi phương thức sendRedirect của nó như sau:

```
response.sendRedirect(tên_trang_jsp_muốn_chuyển_tới);
```

Trong đó, tham số truyền vào là một String, là tên trang jsp mà chương trình muốn chuyển tới. Ví dụ:

```
response.sendRedirect("login.jsp");
```

Tuy nhiên, chúng ta cũng có thể điều khiển việc truyền dữ liệu từ trang này sang trang khác trong khi gọi phương thức sendRedirect của đối tượng response bằng cách thêm các cặp tên biến – giá trị vào ngay sau tên trang jsp cần chuyển tới, theo cú pháp:

```
response.sendRedirect(tên_trang_jsp_muốn_chuyển_tới?tên_biến_1=giá_trị_1&tên_biến_2=giá_trị_2);
```

Ví dụ:

```
response.sendRedirect("login.jsp?ok=0");
```

Lệnh này sẽ chuyển đến trang login.jsp, đồng thời gửi theo một biến có tên là "ok", và giá trị của biến này là 0. Do đó, ở trang login.jsp, muốn lấy được giá trị của biến "ok" này, chỉ cần gọi phương thức getParameter của request như khi dùng Parameter.

10.2.4. Dùng Session

Session là đối tượng gắn liền với mỗi phiên truy nhập trực tuyến của người dùng. Nó cho phép lưu trữ tạm thời một số dữ liệu liên quan đến phiên truy nhập đó. Chúng ta có thể tận dụng khả năng này để chuyển dữ liệu giữa các trang jsp bằng cách truy nhập vào một biến chung được lưu trữ trong session. Cơ chế thực hiện như sau:

Bên trang muốn gửi dữ liệu đi

Trang jsp muốn gửi dữ liệu đi thông qua session thì phải ghi dữ liệu vào biến chung trong session, dùng phương thức setAttribute của đối tượng session:

```
session.setAttribute(tên_biến, giá_trị);
```

Ví dụ, trang doLogin.jsp sau khi nhận được username và password từ trang login.jsp, nó đóng gó thành đối tượng User và lưu vào biến session tên là user như sau:

```
User user = new User(username, password);
session.setAttribute("user", user);
```

Bên trang muốn nhận dữ liệu

Trang jsp muốn nhận dữ liệu về thông qua session thì pahir đọc dữ liệu từ biến chung trong session thông qua phương thức getAttribute của đối tượng session:

```
Kiểu_trả_về tên_biến = (ép kiểu)session.getAttribute(tên_biến_session);
```

Lưu ý là phương thức getAttribute của session luôn trả về dữ liệu dạng Object, cho nên kiểu dữ liệu ban đầu của chúng ta thế nào, chúng ta phải ép kiểu về kiểu đối tượng đấy. Ví dụ, để lấy giá trị biến user trong session, ta phải ép kiểu về đối tượng User như sau:

```
User user= (User)session.getAttribute("user");
if(user == null){
	response.sendRedirect("Home.jsp?ok=0");
}
```

Lưu ý:

- Trong trường hợp này, lớp User phải implements giao diện Serializable.
- Trong trường hợp hết thời gian sống của phiên làm việc (session timeout), lệnh đọc biến session thế này sẽ trả về null. Do đó, trước khi xử lí tiếp theo, ta nên kiểm tra xem giá trị biến có nul hay không.

10.3. LẬP TRÌNH WEB VỚI JSP – TOMCAT

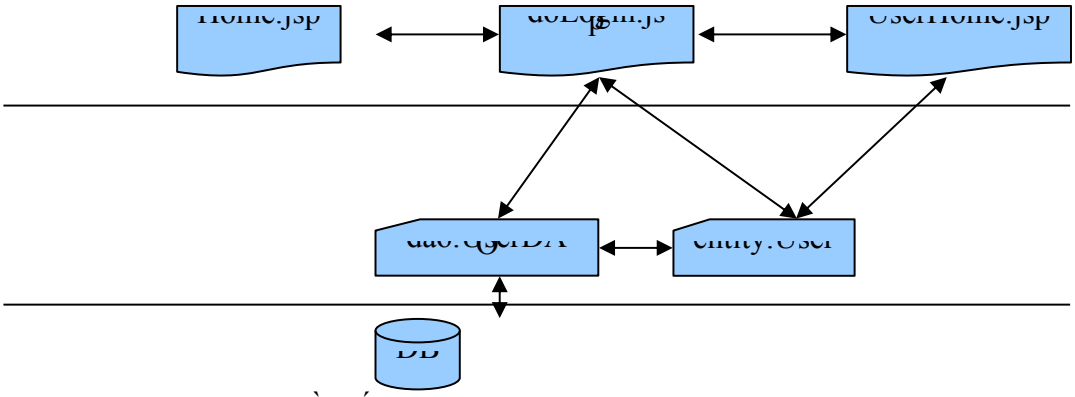
Để minh họa cho mục này, chúng ta sẽ thực hiện với việc xây dựng ứng dụng web với chức năng đăng nhập. Theo đó người dùng login vào trang web, nếu thành công sẽ chuyển sang trang chủ của người dùng, nếu không thành công thì báo lỗi và yêu cầu đăng nhập lại. Sơ đồ các lớp trong hệ thống được mô tả như trong hình 9.3.

Trong đó:

- Tầng lưu trữ có CSDL, trong đó có bảng users với hai cột username và password.

- Tầng nghiệp vụ có 2 lớp: lớp thực thể User có 2 thuộc tính username và password. Lớp truy nhập CSDL UserDAO có phương thức checkLogin để kiểm tra xem thông tin người dùng đăng nhập có đúng hay không.

- Tầng view có 3 trang jsp: trang home.jsp là giao diện đăng nhập. Trang doLogin.jsp là trang xử lí thông tin đăng nhập. Nếu đăng nhập thành công thì chuyển đến trang userHome.jsp. Nếu đăng nhập không thành công thì quay lại trang home.jsp thông báo lỗi và yêu cầu đăng nhập lại.



Hình 9.3: Sơ đồ kiến trúc ứng dụng web với chức năng login

10.3.1. Cài đặt và deploy các lớp Java

Các lớp Java được cài đặt bằng cách tạo một project mới trong Eclipse/NetBean, sau đó tạo 2 package là entity và dao. Trong package entity tạo lớp User. Trong package dao tạo lớp UserDAO với code như sau:

entity.User

```
package entity;
public class User implements Serializable{
	private String username;
	private String password;

	public User(){}

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

	public String getUsername() {
		return username;
	}
```

```
	public void setUsername(String username) {
		this.username = username;
	}

	public String getPassword() {
		return password;
	}

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

dao.UserDAO

```
package dao;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

public class UserDAO {
	Connection conn = null;

	public UserDAO() {
		String dbUsername = "sa";
		String dbPassword = "sa";
	    String dbUrl = "jdbc:mysql://your.database.domain/yourDBname";
	    String dbClass = "com.mysql.jdbc.Driver";

	    try {

		Class.forName(dbClass);
		conn = DriverManager.getConnection (dbUrl, dbUsername, dbPassword);

	    }catch(ClassNotFoundException e) {
		e.printStackTrace();
	    }

	    catch(SQLException e) {
		e.printStackTrace();
	    }
	  }

	public boolean checkLogin(entity.User user){
		String query = "Select * FROM users WHERE username = ?
			AND password = ?";

	    try {
		PreparedStatement ps = conn.prepareStatement(query);
		ps.setString(1, user.getUsername());
		ps.setString(2, user.getPassword());
```

```
            ResultSet rs = ps.executeQuery();
            if (rs.next()) {
                  return true;
            }
          }catch(SQLException e) {
            e.printStackTrace();
          }
            return false;
      }
}
```

10.3.2. Cấu hình ứng dụng web trên server Tomcat

Các bước cấu hình thư mục ứng dụng web trên server Tomcat như sau:

- ***Bước 1***: Tạo một thư mục tên "test" trong: thư mục gốc của Tomcat/webapps
- ***Bước 2***: Tạo thư mục và copy các thư mục entity và dao trong thư mục đã build của project java ở trên (các lớp có đuôi .class) vào: thư mục gốc của Tomcat/webapps/test/WEB-INF/classes
- ***Bước 3***: Copy thư viện JDBC driver của CSDL tương ứng với máy bạn vào: thư mục gốc của Tomcat/lib (hoặc common/lib). Nếu đã có sẵn rồi thì thôi.
- ***Bước 4***: Vào thư mục gốc của Tomcat/webapps/test tạo ra 3 file jsp có tên: Home.jsp, doLogin.jsp, và UserHome.jsp.
- ***Bước 5***: Vào thư mục gốc Tomcat/webapps/test/WEB-INF, tạo một file tên web.xml có nội dung như sau:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
    PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
    "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
    <session-config>
      <session-timeout>30</session-timeout>
    </session-config>
</web-app>
```

10.3.3. Cài đặt các trang JSP

Tiếp theo là cài đặt các trang jsp đã tạo ra ở bước 4 của mục trước: Home.jsp, doLogin.jsp, và UserHome.jsp.

Home.jsp

```
<%@page language="java" import = " java.util.*, java.awt.*, entity.*, dao.*" %>
<%  String msg = request.getParameter("ok"); %>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML><HEAD><TITLE>JSP demo test</TITLE>
<META http-equiv=Content-Type content="text/html; charset=iso-8859-1">
</HEAD>
<BODY leftMargin=0 topMargin=0>
      <table border="0" cellpadding="0" cellspacing="0" >
        <tr>
          <td width="100%" height="133"> 
          <form method="POST" name = "Dangnhap" action="doLogin.jsp" >
            <p align="left">Username:
            <input type="text" name="username" size="12"></p>
            <p align="left">Password:
           <input type="password" name="password" size="12"></p>
            <p align="left">     <input type="submit" value="Submit" name="B1">
                                 <input type="reset" value="Reset" name="B2"></p>
             </form>
          <p></td> </tr>  </table>
      <%    if((msg!=null)&&(msg.equals("0"))){   %>
            <SCRIPT language=JavaScript>
                    alert ("Password ban nhap khong dung. Nhap lai!");
            </SCRIPT>
           <p align="center"></p>
</BODY>
       <%} %>
</HTML>
```

doLogin.jsp dùng Parameter

```
<%@page language="java" import = "java.util.*, java.awt.*, entity.*, dao.*"%>
 <%
  String username = (String)request.getParameter("username");
  String password = (String)request.getParameter("password");
  User user = new User(username, password);
  session.setAttribute("user",user);
  UserDAO userDAO = new UserDAO("sa","sa");
  if(userDAO.checkLogin(user)){
      response.sendRedirect("UserHome.jsp");
  }
  else{
      response.sendRedirect("Home.jsp?ok=0");
  }
%>
```

doLogin.jsp dùng bean

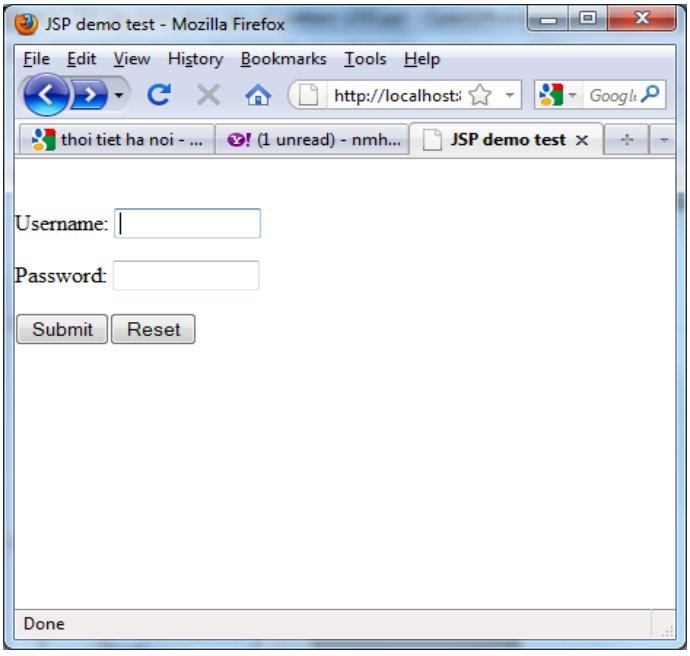
```
<%@page language="java" import = "java.util.*, java.awt.*, entity.*, dao.*"%>
 <jsp:useBean id="user" class="entity.User" scope="request"/>
   <jsp:setProperty name="user" property="*"/>
<%
  session.setAttribute("user",user);
```

```
  UserDAO userDAO = new UserDAO("sa","sa");
  if(userDAO.checkLogin(user)){
      response.sendRedirect("UserHome.jsp");
  }
  else{
      response.sendRedirect("Home.jsp?ok=0");
  }
%>
```

UserHome.jsp

```
<%@page language="java" import = " java.util.*, java.awt.*, entity.*, dao.*" %>
<%
      User user= (User)session.getAttribute("user");
      if(user == null){
            response.sendRedirect("Home.jsp?ok=0");
      }
 %>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML><HEAD><TITLE>JSP demo test</TITLE>
<META http-equiv=Content-Type content="text/html; charset=iso-8859-1">
</HEAD>
<BODY leftMargin=0 topMargin=0>
      Welcome <%= user.getUsername() %>!
</BODY>
</HTML>
```

10.3.4. Test ứng dụng web



Hình 9.4: Kết quả trang Home.jsp

Các bước để kiểm tra một ứng dụng web chạy đúng hay không như sau:

- Bước 1: Khởi động hệ quản trị CSDL có CSDL về ứng dụng đăng nhập đã xây dựng
- Bước 2: Vào thư mục gốc của Tomcat/bin, chạy file startup.bat để khởi động Tomcat server.
- Bước 3: Mở một trình duyệt web bất kì, gõ địa chỉ sau vào URL:

http://localhost:8080/test/Home.jsp

Trang Home.jsp sẽ hiện ra như Hình 9.4. Thử đăng nhập đúng và đăng nhập sai xem hết quả thu được là gì?

10.4. KẾT LUẬN

Nội dung chương này đã giới thiệu về ngôn ngữ JSP để lập trình ứng dụng web trên nền tảng Java. Chương này cũng đã giới thiệu các bước để thiết kế, cài đặt, cấu hình và triển khai một ứng dụng web dựa trên JSP-Tomcat.

Về cơ bản, kiến trúc ứng dụng web trong chương này đã dựa trên mô hình ba tầng MVC cơ bản. Tuy nhiên các ứng dụng này vẫn chưa theo các framework có sẵn. Các chương tiếp theo sẽ hướng dẫn cách thiết kế, cài đặt và triển khai ứng dụng web trên các framework có sẵn.

10.5. BÀI TẬP

1. Xây dựng một ứng dụng web với JSP-Tomcat để giới thiệu các địa điểm vui chơi giải trí theo các thành phố, điểm du lịch của Việt Nam. Với các chức năng:
   - Bên phía người quản trị: có thể thêm, sửa, xóa một địa chỉ vui chơi giải trí, ăn uống
   - Bên người sử dụng phổng thông, có thể vào tìm kiếm các địa chỉ vui chơi, giải trí, ăn uống theo địa điểm.
2. Xây dựng một ứng dụng web với JSP-Tomcat để giới thiệu các món ăn của một nhà hàng. Với các chức năng:
   - Bên phía người quản trị: có thể thêm, sửa, xóa một món ăn trong menu
   - Bên người sử dụng phổng thông, có thể vào tìm kiếm và xem các món ăn.
3. Xây dựng một ứng dụng web với JSP-Tomcat để làm một diễn đàn trao đổi kinh nghiệm học tập, vui chơi, giải trí, làm thêm của sinh viên. Với các chức năng:
   - Người dùng có thể vào đăng bài mới
   - Người dùng vào xem một bài đăng và bình luận vào bài đăng đó

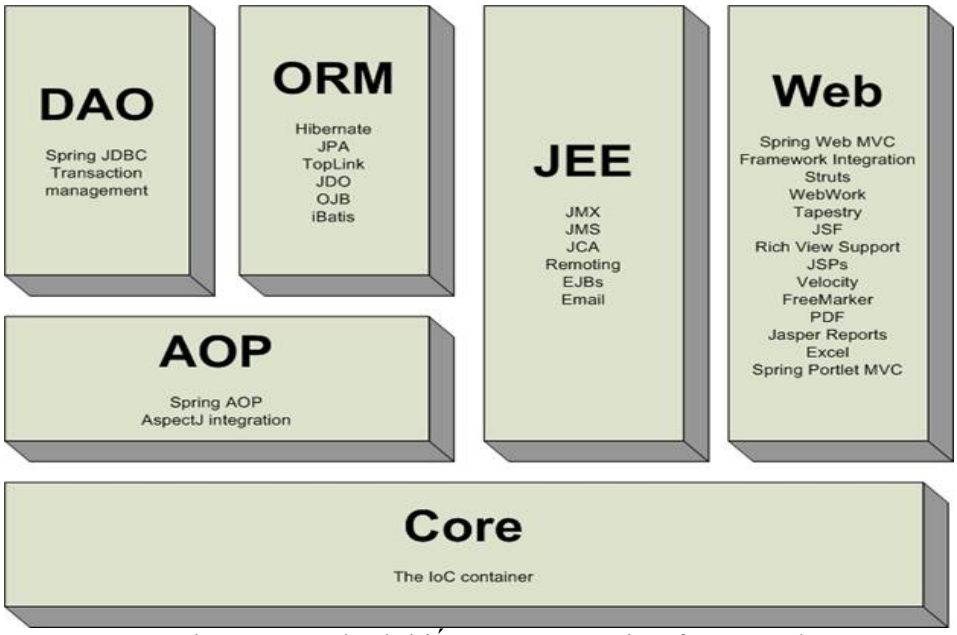
CHƯƠNG 11. LẬP TRÌNH WEB VỚI SPRING FRAMEWORK

Chương này sẽ trình bày:

- Giới thiệu về Spring framework
- Cách thiết kế, cài đặt, triển khai ứng dụng web với Spring

11.1. GIỚI THIỆU VỀ SPRING

Spring Framework là một cấu trúc dùng để xây dựng chương trình ứng dụng web mã nguồn mở dành cho ngôn ngữ lập trình Java, theo mô hình MVC và nền tảng J2EE. Phiên bản đầu tiên của nó do Rod Johnson viết và công bố năm 2002.



Hình 11.1: Mô hình kiến trúc của Spring framework

Kiến trúc của Spring framework được mô tả như Hình 11.1. Trong đó:

- **Core: The IoC Container:** Đây là phần quan trọng nhất và cũng là phần cơ bản, nền tảng của Spring. Nó giữ vai trò về cấu hình và quản lý lifecycle của các java object. Bài hôm nay chúng ta sẽ tìm hiểu về phần này.

- **AOP** (Spring AOP AspectJ integration): Spring AOP module tích hợp chức năng lập trình hướng khía cạnh vào Spring framework thông qua cấu hình của nó. Spring AOP module cung cấp các dịch vụ quản lý giao dịch cho các đối tượng trong bất kỳ ứng dụng nào sử dụng Spring. Với Spring AOP chúng ta có thể tích hợp declarative transaction management vào trong ứng dụng mà không cần dựa vào EJB component. Spring AOP module cũng đưa lập trình metadata vào trong Spring. Sử dụng cái này chúng ta có thể

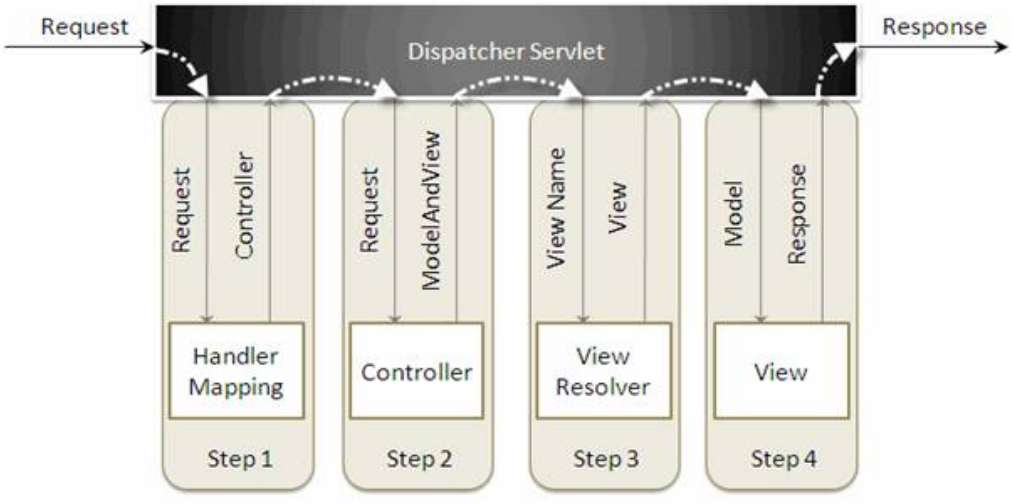
thêm annotation vào source code để hướng dẫn Spring nơi và làm thế nào để liên hệ với aspect.

- **DAO** (Spring JDBC transaction management): Tầng JDBC và DAO đưa ra một cây phân cấp exception để quản lý kết nối đến database, điều khiển exception và thông báo lỗi được ném bởi vendor của database. Tầng exception đơn giản điều khiển lỗi và giảm khối lượng code mà chúng ta cần viết như mở và đóng kết nối. Module này cũng cung cấp các dịch vụ quản lý giao dịch cho các đối tượng trong ứng dụng Spring.

- **ORM** (Hibernate, JPA OJB TopLink, JDO OJB iBatis): Spring có thể tích hợp với một vài ORM framework để cung cấp Object Relation tool bao gồm: JDO, Hibernate, OJB và iBatis SQL Maps.

- **Spring Web module:** Nằm trên application context module, cung cấp context cho các ứng dụng web. Spring cũng hỗ trợ tích hợp với Struts, JSF và Webwork. Web module cũng làm giảm bớt các công việc điều khiển nhiều request và gắn các tham số của request vào các đối tượng domain.

- **Spring MVC Framework:** MVC Framework thì cài đặt đầy đủ đặc tính của MVC pattern để xây dựng các ứng dụng Web. MVC framework thì cấu hình thông qua giao diện và chứa được một số kỹ thuật view bao gồm: JSP, Velocity, Tiles và generation of PDF và Excel file.



Hình 11.2: Tuần tự các bước xử lí request trong Spring

Tuần tự xử lí một request trong Spring được mô tả như trong Hình 11.2:

- DispatcherServlet nhận request và dựa vào HandlerMapping để tìm và invoke Controller tương ứng.
- Controller xử lý Request và return một ModelAndView object. ModelAndView object này chứa model data và ViewName.
- ViewResolver nhận ViewName từ DispatcherServlet để tìm View tương ứng và return View về cho DispatcherServlet.
- DispatcherServlet sẽ pass Model vào View và response về user.

Thư viện Spring có thể tải về từ địa chỉ:

https://github.com/spring-projects/spring-framework/releases

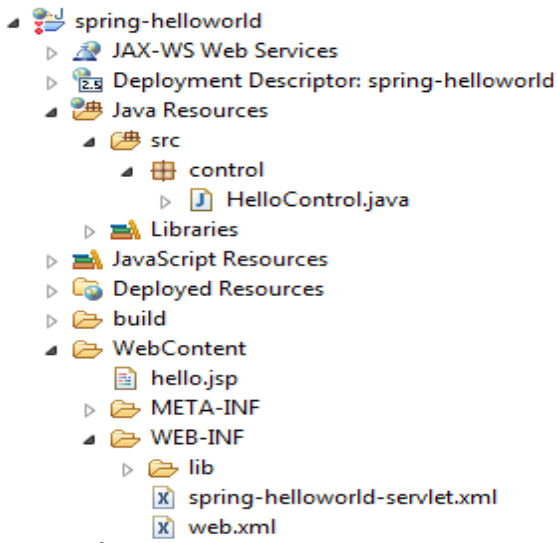
11.2. LẬP TRÌNH ỨNG DỤNG WEB VỚI SPRING

Để minh họa các bước tổng quan để cài đặt và triển khai một ứng dụng web trên Spring, chúng ta cùng xem các bước tiến hành với ví dụ helloworld.

Bước 1: Tạo một project ứng dụng web trong Eclipse phiên bản J2EE, đặt tên là spring-helloworld. Khi đó , ta nên tập trung xây dựng ở 2 thư mục: java resources và web content.

- Trong thư mục java resources, tạo một package tên là control, trong package này, tạo một lớp java tên HelloControl.java
- Trong thư mục web content, tạo một trang jsp tên hello.jsp
- Trong thư mục web content/WEB-INF, tạo 2 file xml: web.xml, và spring-helloworld-servlet.xml
- Copy toàn bộ thư viện Spring (file đuôi .jar) vào thư mục web content/WEB-INF/lib

Cấu trúc thư mục dự án như hình 11.3.



Hình 11.3: Cấu trúc thư mục dự án helloworld

Bước 2: Cài đặt các lớp Java

HelloControl.java

```
package control;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.ui.ModelMap;

@Controller
public class HelloControl{

   @RequestMapping(value = "/hello", method = RequestMethod.GET)
   public String printHello(ModelMap model) {
      model.addAttribute("message", "Hello Spring MVC Web Framework!");
      return "hello";
   }
}
```

Bước 3: Cài đặt các trang jsp

hello.jsp

```
<%@ page contentType="text/html; charset=UTF-8" %>
<html>
<head>
<title>Hello World</title>
</head>
<body>
   <h2>${message}</h2>
</body>
</html>
```

Bước 4: Cấu hình các file config

web.xml

```
<web-app id="WebApp_ID" version="2.4"
   xmlns="http://java.sun.com/xml/ns/j2ee"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
   http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">

   <display-name>Spring MVC Web Application</display-name>

   <servlet>
      <servlet-name>spring-helloworld</servlet-name>
      <servlet-class>org.springframework.web.servlet.DispatcherServlet
      </servlet-class>
      <init-param>
            <param-name>contextConfigLocation</param-name>
            <param-value>/WEB-INF/spring-helloworld-servlet.xml</param-value>
        </init-param>
      <load-on-startup>1</load-on-startup>
   </servlet>

   <servlet-mapping>
      <servlet-name>spring-helloworld</servlet-name>
      <url-pattern>/</url-pattern>
   </servlet-mapping>
</web-app>
```

spring-helloworld-servlet.xml

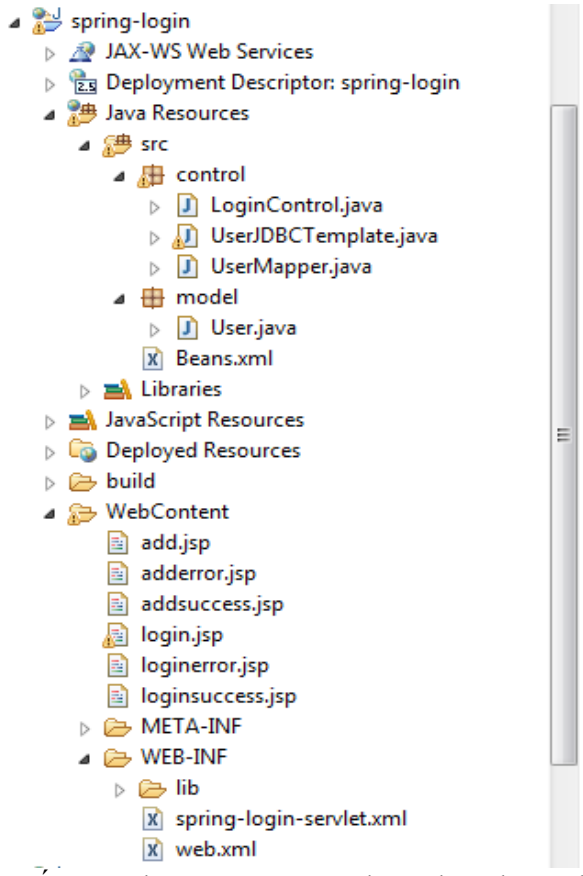
```
<beans xmlns="http://www.springframework.org/schema/beans"
   xmlns:context="http://www.springframework.org/schema/context"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="
   http://www.springframework.org/schema/beans
   http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
   http://www.springframework.org/schema/context
   http://www.springframework.org/schema/context/spring-context-3.0.xsd">

   <context:component-scan base-package="control" />

   <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
      <property name="prefix" value="/" />
      <property name="suffix" value=".jsp" />
   </bean>
</beans>
```

11.3. CASE STUDY: ỨNG DỤNG WEB VỚI CHỨC NĂNG ĐĂNG KÍ VÀ ĐĂNG NHẬP

11.3.1. Tạo project

Bài toán đặt ra như sau: Xây dựng một ứng dụng web trên nền tảng Spring cho phép người dùng web có thể đăng kí làm thành viên với thông tin cá nhân của mình. Sau khi đăng kí thành viên thành công, người dùng có thể đăng nhập vào hệ thống theo username và password của mình.

Để bắt đầu xây dựng ứng dụng, vào Eclipse tạo mới một project tên là spring-login. Cấu trúc thư mục với hai thư mục cần chú ý là java resources và web content như Hình 11.4.



Hình 11.4: Cấu trúc thư mục của ứng dụng đăng kí và đăng nhập

Các mục tiếp theo sẽ trình bày chi tiết cách cài đặt và cấu hình cho từng chức năng đăng nhập và đăng kí.

11.3.2. Chức năng đăng nhập

Để thực hiện chức năng này, ta cần tạo và cấu hình các file trong thư mục dự án như sau:

- Trong thư mục java resources, tạo 1 packeage: Package model có chứa lớp User. Package control có các lớp: UserMapper, UserJDBCTemplate, LoginControl.

- Trong thư mục web content tạo ra 3 trang jsp: login.jsp, loginerror.jsp, loginsuccess.jsp
- File Beans.xml đặt trong thư mục java resources.
- File web.xml và spring-login-servlet.xml đặt trong thư mục web content/WEB-INF

Cài đặt các lớp java

model.User.java

```
package model;

public class User {
	private Integer id;
	private String username;
	private String password;
	private String fullName;
	private String idCardNumber;
	private String idCardType;
	private String address;
	private String description;

	public Integer getId() {
		return id;
	}
	public void setId(Integer id) {
		this.id = id;
	}
	public String getFullName() {
		return fullName;
	}
	public void setFullName(String fullName) {
		this.fullName = fullName;
	}
	public String getIdCardNumber() {
		return idCardNumber;
	}
	public void setIdCardNumber(String idCardNumber) {
		this.idCardNumber = idCardNumber;
	}
	public String getIdCardType() {
		return idCardType;
	}
	public void setIdCardType(String idCardType) {
		this.idCardType = idCardType;
	}
	public String getAddress() {
		return address;
	}
	public void setAddress(String address) {
		this.address = address;
	}
	public String getDescription() {
		return description;
```

```
	}
	public void setDescription(String description) {
		this.description = description;
	}
	public String getUsername() {
		return username;
	}
	public void setUsername(String username) {
		this.username = username;
	}
	public String getPassword() {
		return password;
	}
	public void setPassword(String password) {
		this.password = password;
	}
}
```

control.UserMapper.java

```
package control;

import java.sql.ResultSet;
import java.sql.SQLException;
import org.springframework.jdbc.core.RowMapper;
import model.User;

public class UserMapper implements RowMapper<User> {
	public User mapRow(ResultSet rs, int rowNum) throws SQLException {
		User user = new User();
		user.setId(rs.getInt("id"));
		user.setUsername(rs.getString("username"));
		user.setPassword(rs.getString("password"));
		user.setFullName(rs.getString("fullName"));
		user.setIdCardNumber(rs.getString("idCardNumber"));
		user.setIdCardType(rs.getString("idCardType"));
		user.setAddress(rs.getString("address"));
		user.setDescription(rs.getString("description"));
		return user;
	}
}
```

control.UserJDBCTemplate.java

```
package control;

import java.util.List;
import javax.sql.DataSource;
import model.User;
import org.springframework.jdbc.core.JdbcTemplate;

public class UserJDBCTemplate {
	private DataSource dataSource;
	private JdbcTemplate jdbcTemplateObject;

	public void setDataSource(DataSource dataSource) {
```

```
            this.dataSource = dataSource;
            this.jdbcTemplateObject = new JdbcTemplate(dataSource);
         }

         public boolean checkLogin(User user) {
                  String SQL = "select * from tblUser
                         where username = ? and password = ?";
                  List<User> result = jdbcTemplateObject.query(SQL,
                         new Object[]{user.getUsername(),user.getPassword()},
                         new UserMapper());
                  if(result.size()>0)
                     return true;
                  return false;
         }
}
```

control.LoginControl.java

```
package control;

import model.User;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.ui.ModelMap;

@Controller
public class LoginControl{
       private ApplicationContext context = null;
    private UserJDBCTemplate userJDBCTemplate = null;

    public LoginControl(){
       context = new ClassPathXmlApplicationContext("Beans.xml");
           userJDBCTemplate = (UserJDBCTemplate)context.getBean("userJDBCTemplate");
       }

       @RequestMapping(value = "/login", method = RequestMethod.GET)
       public ModelAndView userLogin() {
              return new ModelAndView("login", "command", new User());
       }

       @RequestMapping(value = "/loginCheck", method = RequestMethod.POST)
       public String checkUser(@ModelAttribute("SpringWeb")User user,
          ModelMap model) {
             model.addAttribute("username", user.getUsername());

             if(userJDBCTemplate.checkLogin(user)){
                return "loginsuccess";
             }

             return "loginerror";
       }
```

```
}
```

Cài đặt các trang jsp

login.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">

<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Spring Login Example</title>
</head>

<body>
<h3>Login Form</h3>

<form:form method="POST" action="/spring-login/loginCheck">
<table>
	<tr><td>User Name:</td></tr>
	<tr><td><form:input path="username" /></td></tr>
	<tr><td>Password:</td></tr>
	<tr><td><form:password path="password" /></td></tr>
	<tr><td><input type="submit" value="Submit" /></td></tr>
</table>
</form:form>
</body>
</html>
```

loginsuccess.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<%@ taglib prefix="core" uri="http://java.sun.com/jsp/jstl/core"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Spring Login Example</title>
</head>

<body>
<h3>Welcome <core:out value="${username}" /></h3>
<table>
	<tr>
		<td><a href="login">Back</a></td>
	</tr>
</table>
```

```
</body>
</html>
```

loginerror.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Spring Login Example</title>
</head>

<body>
<h3>Login Error !!! Click below to login again</h3>
<table>
	<tr>
		<td><a href="login">Retry</a></td>
	</tr>
</table>
</body>
</html>
```

Cấu hình các file xml

Beans.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd ">

   <!-- Initialization for data source -->
   <bean id="dataSource"
class="org.springframework.jdbc.datasource.DriverManagerDataSource">
      <property name="driverClassName" value="com.mysql.jdbc.Driver"/>
      <property name="url" value="jdbc:mysql://localhost:3306/hotelmanagement"/>
      <property name="username" value="root"/>
      <property name="password" value="12345678"/>
   </bean>

   <!-- Definition for userJDBCTemplate bean -->
   <bean id="userJDBCTemplate" class="control.UserJDBCTemplate">
      <property name="dataSource"  ref="dataSource" />
   </bean>

</beans>
```

web.xml

```
<web-app id="WebApp_ID" version="2.4"
```

```
   xmlns="http://java.sun.com/xml/ns/j2ee"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
   http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">

   <display-name>Spring MVC Web Application</display-name>

   <servlet>
      <servlet-name>spring-login</servlet-name>
      <servlet-class>org.springframework.web.servlet.
             DispatcherServlet</servlet-class>
      <init-param>
            <param-name>contextConfigLocation</param-name>
            <param-value>/WEB-INF/spring-login-servlet.xml</param-value>
        </init-param>
      <load-on-startup>1</load-on-startup>
   </servlet>

   <servlet-mapping>
      <servlet-name>spring-login</servlet-name>
      <url-pattern>/</url-pattern>
   </servlet-mapping>

   <welcome-file-list>
        <welcome-file>login</welcome-file>
       </welcome-file-list>
</web-app>
```

spring-login-servlet.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
   xmlns:context="http://www.springframework.org/schema/context"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:mvc="http://www.springframework.org/schema/mvc"

   xsi:schemaLocation="
      http://www.springframework.org/schema/beans
      http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
      http://www.springframework.org/schema/context
      http://www.springframework.org/schema/context/spring-context-3.0.xsd
      http://www.springframework.org/schema/mvc
      http://www.springframework.org/schema/mvc/spring-mvc-3.0.xsd">

   <mvc:annotation-driven />

   <context:component-scan base-package="control" />

   <bean id="viewResolver" class="org.springframework.web.servlet.view.
             InternalResourceViewResolver">
      <property name="prefix" value="/" />
      <property name="suffix" value=".jsp" />
   </bean>

       <bean id="messageSource" class="org.springframework.context.support.
             ReloadableResourceBundleMessageSource">
             <property name="basename" value="/WEB-INF/messages" />
```

```
	</bean>
</beans>
```

11.3.3. Chức năng đăng kí

Khi bổ sung chức năng đăng kí người dùng mới, ta không cần phải thêm lớp java nào nhưng phải bổ sung một số phương thức và các lớp UserJDBCTemplate và LoginControl. Đối với các trang jsp thì cần thêm 3 trang: add.jsp, addsuccess.jsp, và adderror.jsp. Đối với 3 file xml chúng ta không cần phải cập nhật thêm gì.

Bổ sung phương thức cho các lớp java

Thêm phương thức cho lớp UserJDBCTemplate.java

```
public boolean create(User user) {
	String SQL = "select * from tblUser where username = ?";
	List<User> result = jdbcTemplateObject.query(SQL,
		new Object[]{user.getUsername()}, new UserMapper());
	if(result.size()>0)
		return false;

	SQL = "insert into tblUser (username, password, fullName,
	   idCardNumber, idCardType, address, description)
	   values (?,?,?,?,?,?,?)";
	 jdbcTemplateObject.update( SQL, user.getUsername(),
	   user.getPassword(), user.getFullName(), user.getIdCardNumber(),
	   user.getIdCardType(), user.getAddress(), user.getDescription());
	return true;
}
```

Thêm phương thức cho lớp LoginControl.java

```
@RequestMapping(value = "/add", method = RequestMethod.GET)
public ModelAndView userAdd() {
	return new ModelAndView("add", "command", new User());
}

@RequestMapping(value = "/addUser", method = RequestMethod.POST)
public String addUser(@ModelAttribute("SpringWeb")User user,
   ModelMap model) {
      model.addAttribute("username", user.getUsername());

      if(userJDBCTemplate.create(user)){
         return "addsuccess";
      }

      return "adderror";
}
```

Thêm các trang jsp

add.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">

<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Spring Registration Example</title>
</head>

<body>
<h3>Registration Form</h3>
<form:form method="POST" action="/spring-login/addUser">
<table>
	<tr><td>User Name:</td><td><form:input path="username" /></td></tr>
	<tr><td>Password:</td><td><form:password path="password" /></td></tr>
	<tr><td>Full name:</td><td><form:input path="fullName" /></td></tr>
	<tr><td>ID card number:</td><td><form:input path="idCardNumber" /></td></tr>
	<tr><td>ID card type:</td><td><form:input path="idCardType" /></td></tr>
	<tr><td>Address:</td><td><form:input path="address" /></td></tr>
	<tr><td>Description:</td><td><form:input path="description" /></td></tr>
	<tr><td><input type="submit" value="Submit" /></td></tr>
</table>
</form:form>
</body>
</html>
```

addsuccess.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<%@ taglib prefix="core" uri="http://java.sun.com/jsp/jstl/core"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Spring Login Example</title>
</head>

<body>
<h3>Your registration is success! Click the link below to login!</h3>
<table>
	<tr>
		<td><a href="login">Login</a></td>
	</tr>
</table>
</body>
</html>
```

adderror.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<%@ taglib prefix="core" uri="http://java.sun.com/jsp/jstl/core"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Spring Login Example</title>
</head>

<body>
<h3>The username: <core:out value="${username}" /> is already existed !!! Click below
to login again</h3>
<table>
	<tr>
		<td><a href="add">Retry</a></td>
	</tr>
</table>
</body>
</html>
```

11.4. KẾT LUẬN

Nội dung chương này đã giới thiệu về Spring framework, một framework dựa trên mô hình MVC và nền tảng J2EE cho các ứng dụng web. Chương này cũng đã trình bày về cách thức thiết kế, cài đặt và triển khai ứng dụng web trên nền tảng Spring framework.

11.5. BÀI TẬP

1. Xây dựng một ứng dụng web với Spring để giới thiệu các địa điểm vui chơi giải trí theo các thành phố, điểm du lịch của Việt Nam. Với các chức năng:

   - Bên phía người quản trị: có thể thêm, sửa, xóa một địa chỉ vui chơi giải trí, ăn uống
   - Bên người sử dụng phổng thông, có thể vào tìm kiếm các địa chỉ vui chơi, giải trí, ăn uống theo địa điểm.

2. Xây dựng một ứng dụng web với Spring để giới thiệu các món ăn của một nhà hàng. Với các chức năng:

   - Bên phía người quản trị: có thể thêm, sửa, xóa một món ăn trong menu
   - Bên người sử dụng phổng thông, có thể vào tìm kiếm và xem các món ăn.

3. Xây dựng một ứng dụng web với Spring để làm một diễn đàn trao đổi kinh nghiệm học tập, vui chơi, giải trí, làm thêm của sinh viên. Với các chức năng:

- Người dùng có thể vào đăng bài mới
- Người dùng vào xem một bài đăng và bình luận vào bài đăng đó

BÀI TẬP DỰ ÁN

Nội dung phần này sẽ trình bày anh sách một số đề tài bài tập dự án cho môn học lập trình mạng. Với mỗi dự án, yêu cầu chung như sau:

- Phần quản lí thông tin và thống kê người dùng: thiết kế và cài đặt trên web, theo 4 mô hình ứng dụng web đã học: JSP-Tomcat, Struts 1, Struts 2, Spring.

- Phần tương tác giữa các người dùng (chát, chơi, thi đấu): thiết kế và cài đặt trên 2 công nghệ: phiên bản dùng socket (giao thức TCP hoặc UDP), và phiên bản tương tác trên giao diện web (dùng 1 trong 4 mô hình trên).

1. Game oẳn tù tì thi đấu đối kháng online

- Hệ thống có một server và nhiều client. Server lưu toàn bộ thông tin và dữ liệu.
- Để chơi, người chơi phải login vào tài khoản của mình từ một máy client. Sau khi login thành công, giao diện hiện lên một danh sách các người chơi đang online, mỗi người chơi có các thông tin: tên, tổng số điểm hiện có của người chơi, trạng thái (hoặc đang bận nếu đang chơi với người khác, hoặc đang rỗi nếu không chơi với ai)
- Muốn mời (thách đấu) ai thì người chơi click vào tên của đối thủ đó trong danh sách online.
- Khi bị thách đấu, người chơi có thể chấp nhận (OK), hoặc từ chối (Reject).
- Khi chấp nhận, 2 người chơi sẽ vào chơi với nhau, và server sẽ làm trọng tài. Giao diện chơi gồm 3 lựa chọn: giấy, kéo, búa, và 2 nút: send, thoát.
- Mỗi người chơi chọn 1 trong 3 vật và click vào nút Send. Mỗi nước đi mỗi người có 15s để chọn và gửi lên server.
- Khi cả 2 đối thủ đã click Send, server sẽ chấm điểm xem ai thắng và gửi kết quả về cho cả 2 đối thủ: thắng 1 điểm, hòa 0.5 điểm, thua 0 điểm.
- Nếu muốn kết thúc trò chơi với đối thủ hiện tại, người chơi click vào nút Thoát. Hệ thống sẽ thông báo với người còn lại.
- Kết quả các trận đấu được lưu vào server. Mỗi người chơi đều có thể vào xem bảng xếp hạng các người chơi trong toàn bộ hệ thống, theo lần lượt các tiêu chí: tổng số điểm (giảm dần), tổng số trận thắng (giảm dần).

2. Game cờ caro thi đấu đối kháng online

- Hệ thống có một server và nhiều client. Server lưu toàn bộ thông tin và dữ liệu.
- Để chơi, người chơi phải login vào tài khoản của mình từ một máy client. Sau khi login thành công, giao diện hiện lên một danh sách các người chơi đang online, mỗi người chơi

có các thông tin: tên, tổng số điểm hiện có của người chơi, trạng thái (hoặc đang bận nếu đang chơi với người khác, hoặc đang rỗi nếu không chơi với ai)

- Muốn mời (thách đấu) ai thì người chơi click vào tên của đối thủ đó trong danh sách online.
- Khi bị thách đấu, người chơi có thể chấp nhận (OK), hoặc từ chối (Reject).
- Khi chấp nhận, 2 người chơi sẽ vào chơi với nhau, và server sẽ làm trọng tài. Giao diện chơi gồm bàn cờ và nút thoát.
- Người thách đấu sẽ nhận quân O và đi sau, người bị thách đấu sẽ nhận quân X và đi trước. Sau mỗi ván, thứ tự đi sẽ đổi ngược lại. Mỗi nước đi mỗi người có 15s để đi.
- Sau mỗi nước đi, server sẽ kiểm tra xem ai thắng và gửi kết quả về cho cả 2 đối thủ: thắng 1 điểm, hòa 0.5 điểm, thua 0 điểm. Nếu chưa có ai thắng thì để cho 2 đối thủ chơi tiếp.
- Sau mỗi ván, đều có dialog hỏi mỗi người chơi có muốn tiếp tục không. Nếu cả hai tiếp tục thì chơi tiếp, nếu một trong hai đối thủ dừng chơi thì thoát ra và server báo cho người chơi còn lại.
- Kết quả các trận đấu được lưu vào server. Mỗi người chơi đều có thể vào xem bảng xếp hạng các người chơi trong toàn bộ hệ thống, theo lần lượt các tiêu chí: tổng số điểm (giảm dần), trung bình điểm của các đối thủ đã gặp (giảm dần), trung bình số nước đi trong các trận thắng (tăng dần), trung bình số nước đi trong các trận thua (giảm dần).

3. Game dò mìn thi đấu đối kháng online

- Hệ thống có một server và nhiều client. Server lưu toàn bộ thông tin và dữ liệu.
- Để chơi, người chơi phải login vào tài khoản của mình từ một máy client. Sau khi login thành công, giao diện hiện lên một danh sách các người chơi đang online, mỗi người chơi có các thông tin: tên, tổng số điểm hiện có của người chơi, trạng thái (hoặc đang bận nếu đang chơi với người khác, hoặc đang rỗi nếu không chơi với ai)
- Muốn mời (thách đấu) ai thì người chơi click vào tên của đối thủ đó trong danh sách online.
- Khi bị thách đấu, người chơi có thể chấp nhận (OK), hoặc từ chối (Reject).
- Khi chấp nhận, 2 người chơi sẽ vào chơi với nhau, và server sẽ làm trọng tài. Giao diện chơi gồm bàn dò mìn, ô hiện thời gian và nút thoát.
- Server sẽ sinh tự động cùng một đề bài và gửi về cho cả hai đối thủ chơi. Ai dò nhanh hơn thì người đó giành chiến thắng.
- Sau mỗi ván, server sẽ kiểm tra xem ai thắng và gửi kết quả về cho cả 2 đối thủ: thắng 1 điểm, hòa 0.5 điểm, thua 0 điểm.

- Sau mỗi ván, đều có dialog hỏi mỗi người chơi có muốn tiếp tục không. Nếu cả hai tiếp tục thì chơi tiếp, nếu một trong hai đối thủ dừng chơi thì thoát ra và server báo cho người chơi còn lại.
- Kết quả các trận đấu được lưu vào server. Mỗi người chơi đều có thể vào xem bảng xếp hạng các người chơi trong toàn bộ hệ thống, theo lần lượt các tiêu chí: tổng số điểm (giảm dần), trung bình điểm của các đối thủ đã gặp (giảm dần), trung bình thời gian kết thúc trong các trận thắng và hòa (tăng dần).

4. Game dò số thi đấu đối kháng online

- Hệ thống có một server và nhiều client. Server lưu toàn bộ thông tin và dữ liệu.
- Để chơi, người chơi phải login vào tài khoản của mình từ một máy client. Sau khi login thành công, giao diện hiện lên một danh sách các người chơi đang online, mỗi người chơi có các thông tin: tên, tổng số điểm hiện có của người chơi, trạng thái (hoặc đang bận nếu đang chơi với người khác, hoặc đang rỗi nếu không chơi với ai)
- Muốn mời (thách đấu) ai thì người chơi click vào tên của đối thủ đó trong danh sách online.
- Khi bị thách đấu, người chơi có thể chấp nhận (OK), hoặc từ chối (Reject).
- Khi chấp nhận, 2 người chơi sẽ vào chơi với nhau, và server sẽ làm trọng tài. Giao diện chơi gồm bàn dò số kích thước 20x20 số, ô thời gian và nút thoát.
- Server sẽ sinh tự động cùng một đề bài và gửi về cho cả hai đối thủ chơi. Mỗi người chơi phải click các số trong ma trận theo thứ tự giá trị số tăng dần. Nếu click sai tại bước bất kì thì bị tính là không hoàn thành. Ai hoàn thành dò nhanh hơn thì người đó giành chiến thắng. Nếu cả hai đối thủ đều không hoàn thành thì thì tính hòa.
- Sau mỗi ván, server sẽ kiểm tra xem ai thắng và gửi kết quả về cho cả 2 đối thủ: thắng 1 điểm, hòa 0.5 điểm, thua 0 điểm.
- Sau mỗi ván, đều có dialog hỏi mỗi người chơi có muốn tiếp tục không. Nếu cả hai tiếp tục thì chơi tiếp, nếu một trong hai đối thủ dừng chơi thì thoát ra và server báo cho người chơi còn lại.
- Kết quả các trận đấu được lưu vào server. Mỗi người chơi đều có thể vào xem bảng xếp hạng các người chơi trong toàn bộ hệ thống, theo lần lượt các tiêu chí: tổng số điểm (giảm dần), trung bình điểm của các đối thủ đã gặp (giảm dần), trung bình thời gian kết thúc trong các trận thắng (tăng dần).

5. Game lật ảnh thi đấu đối kháng online

- Hệ thống có một server và nhiều client. Server lưu toàn bộ thông tin và dữ liệu.

- Để chơi, người chơi phải login vào tài khoản của mình từ một máy client. Sau khi login thành công, giao diện hiện lên một danh sách các người chơi đang online, mỗi người chơi có các thông tin: tên, tổng số điểm hiện có của người chơi, trạng thái (hoặc đang bận nếu đang chơi với người khác, hoặc đang rỗi nếu không chơi với ai)
- Muốn mời (thách đấu) ai thì người chơi click vào tên của đối thủ đó trong danh sách online.
- Khi bị thách đấu, người chơi có thể chấp nhận (OK), hoặc từ chối (Reject).
- Khi chấp nhận, 2 người chơi sẽ vào chơi với nhau, và server sẽ làm trọng tài. Giao diện chơi gồm bàn lật ảnh kích thước 5x5 ô ảnh, ô thời gian và nút thoát.
- Server sẽ sinh tự động cùng một đề bài và gửi về cho cả hai đối thủ chơi. Mỗi người chơi phải click các ô trong ma trận theo cặp: nếu cùng ảnh thì lật, nếu không cùng thì cả hai lại bị úp lại. Ai hoàn thành lật nhanh hơn thì người đó giành chiến thắng.
- Sau mỗi ván, server sẽ kiểm tra xem ai thắng và gửi kết quả về cho cả 2 đối thủ: thắng 1 điểm, hòa 0.5 điểm, thua 0 điểm.
- Sau mỗi ván, đều có dialog hỏi mỗi người chơi có muốn tiếp tục không. Nếu cả hai tiếp tục thì chơi tiếp, nếu một trong hai đối thủ dừng chơi thì thoát ra và server báo cho người chơi còn lại.
- Kết quả các trận đấu được lưu vào server. Mỗi người chơi đều có thể vào xem bảng xếp hạng các người chơi trong toàn bộ hệ thống, theo lần lượt các tiêu chí: tổng số điểm (giảm dần), trung bình điểm của các đối thủ đã gặp (giảm dần), trung bình thời gian kết thúc trong các trận thắng (tăng dần).

6. Game ghép ảnh thi đấu đối kháng online

- Hệ thống có một server và nhiều client. Server lưu toàn bộ thông tin và dữ liệu.
- Để chơi, người chơi phải login vào tài khoản của mình từ một máy client. Sau khi login thành công, giao diện hiện lên một danh sách các người chơi đang online, mỗi người chơi có các thông tin: tên, tổng số điểm hiện có của người chơi, trạng thái (hoặc đang bận nếu đang chơi với người khác, hoặc đang rỗi nếu không chơi với ai)
- Muốn mời (thách đấu) ai thì người chơi click vào tên của đối thủ đó trong danh sách online.
- Khi bị thách đấu, người chơi có thể chấp nhận (OK), hoặc từ chối (Reject).
- Khi chấp nhận, 2 người chơi sẽ vào chơi với nhau, và server sẽ làm trọng tài. Giao diện chơi gồm 1 ảnh mẫu và 1 bàn ghép ảnh kích thước 5x5 ô ảnh, ô thời gian và nút thoát.
- Server sẽ sinh tự động cùng một đề bài và gửi về cho cả hai đối thủ chơi. Mỗi người chơi phải click các ô và thả vào vị trí tương ứng trong ma trận cho khớp với nhau thành ảnh mẫu. Ai hoàn thành ghép nhanh hơn thì người đó giành chiến thắng.

- Sau mỗi ván, server sẽ kiểm tra xem ai thắng và gửi kết quả về cho cả 2 đối thủ: thắng 1 điểm, hòa 0.5 điểm, thua 0 điểm.
- Sau mỗi ván, đều có dialog hỏi mỗi người chơi có muốn tiếp tục không. Nếu cả hai tiếp tục thì chơi tiếp, nếu một trong hai đối thủ dừng chơi thì thoát ra và server báo cho người chơi còn lại.
- Kết quả các trận đấu được lưu vào server. Mỗi người chơi đều có thể vào xem bảng xếp hạng các người chơi trong toàn bộ hệ thống, theo lần lượt các tiêu chí: tổng số điểm (giảm dần), trung bình điểm của các đối thủ đã gặp (giảm dần), trung bình thời gian kết thúc trong các trận thắng (tăng dần).

7. Game Pikachu thi đấu đối kháng online

- Hệ thống có một server và nhiều client. Server lưu toàn bộ thông tin và dữ liệu.
- Để chơi, người chơi phải login vào tài khoản của mình từ một máy client. Sau khi login thành công, giao diện hiện lên một danh sách các người chơi đang online, mỗi người chơi có các thông tin: tên, tổng số điểm hiện có của người chơi, trạng thái (hoặc đang bận nếu đang chơi với người khác, hoặc đang rỗi nếu không chơi với ai)
- Muốn mời (thách đấu) ai thì người chơi click vào tên của đối thủ đó trong danh sách online.
- Khi bị thách đấu, người chơi có thể chấp nhận (OK), hoặc từ chối (Reject).
- Khi chấp nhận, 2 người chơi sẽ vào chơi với nhau, và server sẽ làm trọng tài. Giao diện chơi gồm một ma trận10x10 ô ảnh các nhân vật trong pikachu, ô thời gian và nút thoát.
- Server sẽ sinh tự động cùng một đề bài và gửi về cho cả hai đối thủ chơi. Mỗi người chơi phải click các ô theo cặp: nếu nối với nhau bởi tối đa 3 đoạn thẳng thì đúng và 2 ô tương ứng sẽ biết mất. Khi không còn ô nào trên bàn thì hoàn thành. Ai hoàn thành ghép nhanh hơn thì người đó giành chiến thắng.
- Sau mỗi ván, server sẽ kiểm tra xem ai thắng và gửi kết quả về cho cả 2 đối thủ: thắng 1 điểm, hòa 0.5 điểm, thua 0 điểm.
- Sau mỗi ván, đều có dialog hỏi mỗi người chơi có muốn tiếp tục không. Nếu cả hai tiếp tục thì chơi tiếp, nếu một trong hai đối thủ dừng chơi thì thoát ra và server báo cho người chơi còn lại.
- Kết quả các trận đấu được lưu vào server. Mỗi người chơi đều có thể vào xem bảng xếp hạng các người chơi trong toàn bộ hệ thống, theo lần lượt các tiêu chí: tổng số điểm (giảm dần), trung bình điểm của các đối thủ đã gặp (giảm dần), trung bình thời gian kết thúc trong các trận thắng (tăng dần).

8. Game giải toán nhanh thi đấu đối kháng online

- Hệ thống có một server và nhiều client. Server lưu toàn bộ thông tin và dữ liệu.
- Để chơi, người chơi phải login vào tài khoản của mình từ một máy client. Sau khi login thành công, giao diện hiện lên một danh sách các người chơi đang online, mỗi người chơi có các thông tin: tên, tổng số điểm hiện có của người chơi, trạng thái (hoặc đang bận nếu đang chơi với người khác, hoặc đang rỗi nếu không chơi với ai)
- Muốn mời (thách đấu) ai thì người chơi click vào tên của đối thủ đó trong danh sách online.
- Khi bị thách đấu, người chơi có thể chấp nhận (OK), hoặc từ chối (Reject).
- Khi chấp nhận, 2 người chơi sẽ vào chơi với nhau, và server sẽ làm trọng tài. Giao diện chơi gồm một danh sách N câu hỏi toán nhanh theo dạng trắc nghiệm, ô thời gian và nút thoát.
- Server sẽ sinh tự động cùng một đề bài và gửi về cho cả hai đối thủ chơi. Mỗi người chơi phải click chọn các đáp án của các câu hỏi. Khi hoàn thành thì click nút submit bài làm. Ai đúng hết và nhanh hơn thì giành chiến thắng. Nếu cả hai đều có ít nhất 1 câu sai thì hòa.
- Sau mỗi trận, server sẽ kiểm tra xem ai thắng và gửi kết quả về cho cả 2 đối thủ: thắng 1 điểm, hòa 0.5 điểm, thua 0 điểm.
- Sau mỗi ván, đều có dialog hỏi mỗi người chơi có muốn tiếp tục không. Nếu cả hai tiếp tục thì chơi tiếp, nếu một trong hai đối thủ dừng chơi thì thoát ra và server báo cho người chơi còn lại.
- Kết quả các trận đấu được lưu vào server. Mỗi người chơi đều có thể vào xem bảng xếp hạng các người chơi trong toàn bộ hệ thống, theo lần lượt các tiêu chí: tổng số điểm (giảm dần), trung bình điểm của các đối thủ đã gặp (giảm dần), trung bình thời gian kết thúc trong các trận thắng (tăng dần).

9. Game trắc nghiệm nhanh thi đấu đối kháng online

- Hệ thống có một server và nhiều client. Server lưu toàn bộ thông tin và dữ liệu.
- Để chơi, người chơi phải login vào tài khoản của mình từ một máy client. Sau khi login thành công, giao diện hiện lên một danh sách các người chơi đang online, mỗi người chơi có các thông tin: tên, tổng số điểm hiện có của người chơi, trạng thái (hoặc đang bận nếu đang chơi với người khác, hoặc đang rỗi nếu không chơi với ai)
- Muốn mời (thách đấu) ai thì người chơi click vào tên của đối thủ đó trong danh sách online.
- Khi bị thách đấu, người chơi có thể chấp nhận (OK), hoặc từ chối (Reject).
- Khi chấp nhận, 2 người chơi sẽ vào chơi với nhau, và server sẽ làm trọng tài. Giao diện chơi gồm một danh sách N câu hỏi theo dạng trắc nghiệm, ô thời gian và nút thoát.

- Server sẽ sinh tự động cùng một đề bài và gửi về cho cả hai đối thủ chơi. Mỗi người chơi phải click chọn các đáp án của các câu hỏi. Khi hoàn thành thì click nút submit bài làm. Ai đúng hết và nhanh hơn thì giành chiến thắng. Nếu cả hai đều có ít nhất 1 câu sai thì hòa.
- Sau mỗi trận, server sẽ kiểm tra xem ai thắng và gửi kết quả về cho cả 2 đối thủ: thắng 1 điểm, hòa 0.5 điểm, thua 0 điểm.
- Sau mỗi ván, đều có dialog hỏi mỗi người chơi có muốn tiếp tục không. Nếu cả hai tiếp tục thì chơi tiếp, nếu một trong hai đối thủ dừng chơi thì thoát ra và server báo cho người chơi còn lại.
- Kết quả các trận đấu được lưu vào server. Mỗi người chơi đều có thể vào xem bảng xếp hạng các người chơi trong toàn bộ hệ thống, theo lần lượt các tiêu chí: tổng số điểm (giảm dần), trung bình điểm của các đối thủ đã gặp (giảm dần), trung bình thời gian kết thúc trong các trận thắng (tăng dần).

10. Game cờ vua thi đấu đối kháng online

- Hệ thống có một server và nhiều client. Server lưu toàn bộ thông tin và dữ liệu.
- Để chơi, người chơi phải login vào tài khoản của mình từ một máy client. Sau khi login thành công, giao diện hiện lên một danh sách các người chơi đang online, mỗi người chơi có các thông tin: tên, tổng số điểm hiện có của người chơi, trạng thái (hoặc đang bận nếu đang chơi với người khác, hoặc đang rỗi nếu không chơi với ai)
- Muốn mời (thách đấu) ai thì người chơi click vào tên của đối thủ đó trong danh sách online.
- Khi bị thách đấu, người chơi có thể chấp nhận (OK), hoặc từ chối (Reject).
- Khi chấp nhận, 2 người chơi sẽ vào chơi với nhau, và server sẽ làm trọng tài. Giao diện chơi gồm bàn cờ và nút thoát.
- Người thách đấu sẽ nhận quân đen và đi sau, người bị thách đấu sẽ nhận quân trắng và đi trước. Sau mỗi ván, thứ tự đi sẽ đổi ngược lại. Mỗi nước đi mỗi người có 30s để đi.
- Sau mỗi nước đi, server sẽ kiểm tra xem ai thắng và gửi kết quả về cho cả 2 đối thủ: thắng 1 điểm, hòa 0.5 điểm, thua 0 điểm. Nếu chưa có ai thắng thì để cho 2 đối thủ chơi tiếp.
- Sau mỗi ván, đều có dialog hỏi mỗi người chơi có muốn tiếp tục không. Nếu cả hai tiếp tục thì chơi tiếp, nếu một trong hai đối thủ dừng chơi thì thoát ra và server báo cho người chơi còn lại.
- Kết quả các trận đấu được lưu vào server. Mỗi người chơi đều có thể vào xem bảng xếp hạng các người chơi trong toàn bộ hệ thống, theo lần lượt các tiêu chí: tổng số điểm (giảm dần), trung bình điểm của các đối thủ đã gặp (giảm dần), trung bình số nước đi trong các trận thắng (tăng dần), trung bình số nước đi trong các trận thua (giảm dần).

11. Game cờ tướng thi đấu đối kháng online

- Hệ thống có một server và nhiều client. Server lưu toàn bộ thông tin và dữ liệu.
- Để chơi, người chơi phải login vào tài khoản của mình từ một máy client. Sau khi login thành công, giao diện hiện lên một danh sách các người chơi đang online, mỗi người chơi có các thông tin: tên, tổng số điểm hiện có của người chơi, trạng thái (hoặc đang bận nếu đang chơi với người khác, hoặc đang rỗi nếu không chơi với ai)
- Muốn mời (thách đấu) ai thì người chơi click vào tên của đối thủ đó trong danh sách online.
- Khi bị thách đấu, người chơi có thể chấp nhận (OK), hoặc từ chối (Reject).
- Khi chấp nhận, 2 người chơi sẽ vào chơi với nhau, và server sẽ làm trọng tài. Giao diện chơi gồm bàn cờ và nút thoát.
- Người thách đấu sẽ đi sau, người bị thách đấu sẽ đi trước. Sau mỗi ván, thứ tự đi sẽ đổi ngược lại. Mỗi nước đi mỗi người có 30s để đi.
- Sau mỗi nước đi, server sẽ kiểm tra xem ai thắng và gửi kết quả về cho cả 2 đối thủ: thắng 1 điểm, hòa 0.5 điểm, thua 0 điểm. Nếu chưa có ai thắng thì để cho 2 đối thủ chơi tiếp.
- Sau mỗi ván, đều có dialog hỏi mỗi người chơi có muốn tiếp tục không. Nếu cả hai tiếp tục thì chơi tiếp, nếu một trong hai đối thủ dừng chơi thì thoát ra và server báo cho người chơi còn lại.
- Kết quả các trận đấu được lưu vào server. Mỗi người chơi đều có thể vào xem bảng xếp hạng các người chơi trong toàn bộ hệ thống, theo lần lượt các tiêu chí: tổng số điểm (giảm dần), trung bình điểm của các đối thủ đã gặp (giảm dần), trung bình số nước đi trong các trận thắng (tăng dần), trung bình số nước đi trong các trận thua (giảm dần).

12. Game cờ vây thi đấu đối kháng online

- Hệ thống có một server và nhiều client. Server lưu toàn bộ thông tin và dữ liệu.
- Để chơi, người chơi phải login vào tài khoản của mình từ một máy client. Sau khi login thành công, giao diện hiện lên một danh sách các người chơi đang online, mỗi người chơi có các thông tin: tên, tổng số điểm hiện có của người chơi, trạng thái (hoặc đang bận nếu đang chơi với người khác, hoặc đang rỗi nếu không chơi với ai)
- Muốn mời (thách đấu) ai thì người chơi click vào tên của đối thủ đó trong danh sách online.
- Khi bị thách đấu, người chơi có thể chấp nhận (OK), hoặc từ chối (Reject).
- Khi chấp nhận, 2 người chơi sẽ vào chơi với nhau, và server sẽ làm trọng tài. Giao diện chơi gồm bàn cờ và nút thoát.
- Người thách đấu sẽ nhận quân đen và đi sau, người bị thách đấu sẽ nhận quân trắng và đi trước. Sau mỗi ván, thứ tự đi sẽ đổi ngược lại. Mỗi nước đi mỗi người có 15s để đi.

- Sau mỗi nước đi, server sẽ kiểm tra xem ai thắng và gửi kết quả về cho cả 2 đối thủ: thắng 1 điểm, hòa 0.5 điểm, thua 0 điểm. Nếu chưa có ai thắng thì để cho 2 đối thủ chơi tiếp.
- Sau mỗi ván, đều có dialog hỏi mỗi người chơi có muốn tiếp tục không. Nếu cả hai tiếp tục thì chơi tiếp, nếu một trong hai đối thủ dừng chơi thì thoát ra và server báo cho người chơi còn lại.
- Kết quả các trận đấu được lưu vào server. Mỗi người chơi đều có thể vào xem bảng xếp hạng các người chơi trong toàn bộ hệ thống, theo lần lượt các tiêu chí: tổng số điểm (giảm dần), trung bình điểm của các đối thủ đã gặp (giảm dần), trung bình điểm thu được trong các trận thắng (giảm dần), trung bình điểm thu được trong các trận thua (giảm dần).

13. Game tá lả online

- Hệ thống có một server và nhiều client. Server lưu toàn bộ thông tin và dữ liệu.
- Để chơi, người chơi phải login vào tài khoản của mình từ một máy client. Sau khi login thành công, giao diện hiện lên hai danh sách: một danh sách các người chơi đang online, mỗi người chơi có các thông tin: tên, tổng số điểm hiện có của người chơi, trạng thái (hoặc đang bận nếu đang chơi với người khác, hoặc đang rỗi nếu không chơi với ai). Danh sách thứ hai là danh sách các bàn đang thi đấu: mỗi bàn hiện mã màn, số lượng người chơi trong bàn.
- Muốn mời (thách đấu) ai thì người chơi click vào tên của đối thủ đó trong danh sách online.
- Khi bị thách đấu, người chơi có thể chấp nhận (OK), hoặc từ chối (Reject).
- Khi chấp nhận, 2 người chơi sẽ vào chơi với nhau và tạo thành một bàn, và server sẽ làm trọng tài. Giao diện chơi gồm bàn tá lả và nút thoát khỏi bàn.
- Người chơi cũng có thể click vào một bàn chưa đủ 4 người để chơi.
- Sau mỗi nước đi, server sẽ kiểm tra xem ai thắng và gửi kết quả về cho các đối thủ: nhất 4 điểm, nhì 2 điểm, ba 1 điểm, bét 0 điểm. Nếu chưa có ai thắng thì để cho các đối thủ chơi tiếp. Nếu trận đấu chỉ có 2 người thì điểm là (2-0), nếu trận đấu chỉ có 3 người thì điểm là (3-1-0).
- Sau mỗi ván, đều có dialog hỏi mỗi người chơi có muốn tiếp tục không. Nếu cả hai tiếp tục thì chơi tiếp, nếu một trong hai đối thủ dừng chơi thì thoát ra và server báo cho người chơi còn lại.
- Kết quả các trận đấu được lưu vào server. Mỗi người chơi đều có thể vào xem bảng xếp hạng các người chơi trong toàn bộ hệ thống, theo lần lượt các tiêu chí: tổng số điểm (giảm dần), tổng số lần đứng nhất (giảm dần), tổng số lần đứng nhì (giảm dần), tổng số lần đứng ba (giảm dần), tổng số lần đứng bét (giảm dần).

14. Game ba cây online

- Hệ thống có một server và nhiều client. Server lưu toàn bộ thông tin và dữ liệu.
- Để chơi, người chơi phải login vào tài khoản của mình từ một máy client. Sau khi login thành công, giao diện hiện lên hai danh sách: một danh sách các người chơi đang online, mỗi người chơi có các thông tin: tên, tổng số điểm hiện có của người chơi, trạng thái (hoặc đang bận nếu đang chơi với người khác, hoặc đang rỗi nếu không chơi với ai). Danh sách thứ hai là danh sách các bàn đang thi đấu: mỗi bàn hiện mã màn, số lượng người chơi trong bàn (tối đa 4 người).
- Muốn mời (thách đấu) ai thì người chơi click vào tên của đối thủ đó trong danh sách online.
- Khi bị thách đấu, người chơi có thể chấp nhận (OK), hoặc từ chối (Reject).
- Khi chấp nhận, 2 người chơi sẽ vào chơi với nhau và tạo thành một bàn, và server sẽ làm trọng tài. Giao diện chơi gồm bàn chơi và nút thoát khỏi bàn.
- Người chơi cũng có thể click vào một bàn chưa đủ 4 người để chơi.
- Mỗi người được bốc 3 quân bài trên bàn, server sẽ kiểm tra xem ai thắng và gửi kết quả về cho các đối thủ: nhất 4 điểm, nhì 2 điểm, ba 1 điểm, bét 0 điểm. Nếu chưa có ai thắng thì để cho các đối thủ chơi tiếp. Nếu trận đấu chỉ có 2 người thì điểm là (2-0), nếu trận đấu chỉ có 3 người thì điểm là (3-1-0).
- Sau mỗi ván, đều có dialog hỏi mỗi người chơi có muốn tiếp tục không. Nếu cả hai tiếp tục thì chơi tiếp, nếu một trong hai đối thủ dừng chơi thì thoát ra và server báo cho người chơi còn lại.
- Kết quả các trận đấu được lưu vào server. Mỗi người chơi đều có thể vào xem bảng xếp hạng các người chơi trong toàn bộ hệ thống, theo lần lượt các tiêu chí: tổng số điểm (giảm dần), tổng số lần đứng nhất (giảm dần), tổng số lần đứng nhì (giảm dần), tổng số lần đứng ba (giảm dần), tổng số lần đứng bét (giảm dần).

15. Game tiến lên online

- Hệ thống có một server và nhiều client. Server lưu toàn bộ thông tin và dữ liệu.
- Để chơi, người chơi phải login vào tài khoản của mình từ một máy client. Sau khi login thành công, giao diện hiện lên hai danh sách: một danh sách các người chơi đang online, mỗi người chơi có các thông tin: tên, tổng số điểm hiện có của người chơi, trạng thái (hoặc đang bận nếu đang chơi với người khác, hoặc đang rỗi nếu không chơi với ai). Danh sách thứ hai là danh sách các bàn đang thi đấu: mỗi bàn hiện mã màn, số lượng người chơi trong bàn.
- Muốn mời (thách đấu) ai thì người chơi click vào tên của đối thủ đó trong danh sách online.

- Khi bị thách đấu, người chơi có thể chấp nhận (OK), hoặc từ chối (Reject).
- Khi chấp nhận, 2 người chơi sẽ vào chơi với nhau và tạo thành một bàn, và server sẽ làm trọng tài. Giao diện chơi gồm bàn chơi và nút thoát khỏi bàn.
- Người chơi cũng có thể click vào một bàn chưa đủ 4 người để chơi.
- Sau mỗi nước đi, server sẽ kiểm tra xem ai thắng và gửi kết quả về cho các đối thủ: nhất 4 điểm, nhì 2 điểm, ba 1 điểm, bét 0 điểm. Nếu chưa có ai thắng thì để cho các đối thủ chơi tiếp. Nếu trận đấu chỉ có 2 người thì điểm là (2-0), nếu trận đấu chỉ có 3 người thì điểm là (3-1-0).
- Sau mỗi ván, đều có dialog hỏi mỗi người chơi có muốn tiếp tục không. Nếu cả hai tiếp tục thì chơi tiếp, nếu một trong hai đối thủ dừng chơi thì thoát ra và server báo cho người chơi còn lại.
- Kết quả các trận đấu được lưu vào server. Mỗi người chơi đều có thể vào xem bảng xếp hạng các người chơi trong toàn bộ hệ thống, theo lần lượt các tiêu chí: tổng số điểm (giảm dần), tổng số lần đứng nhất (giảm dần), tổng số lần đứng nhì (giảm dần), tổng số lần đứng ba (giảm dần), tổng số lần đứng bét (giảm dần).

16. Game solitaire thi đấu đối kháng online

- Hệ thống có một server và nhiều client. Server lưu toàn bộ thông tin và dữ liệu.
- Để chơi, người chơi phải login vào tài khoản của mình từ một máy client. Sau khi login thành công, giao diện hiện lên hai danh sách: một danh sách các người chơi đang online, mỗi người chơi có các thông tin: tên, tổng số điểm hiện có của người chơi, trạng thái (hoặc đang bận nếu đang chơi với người khác, hoặc đang rỗi nếu không chơi với ai). Danh sách thứ hai là danh sách các bàn đang thi đấu: mỗi bàn hiện mã màn, số lượng người chơi trong bàn.
- Muốn mời (thách đấu) ai thì người chơi click vào tên của đối thủ đó trong danh sách online.
- Khi bị thách đấu, người chơi có thể chấp nhận (OK), hoặc từ chối (Reject).
- Khi chấp nhận, 2 người chơi sẽ vào chơi với nhau và tạo thành một bàn, và server sẽ làm trọng tài. Giao diện chơi gồm bàn chơi và nút thoát khỏi bàn.
- Server sẽ ra cùng một đề bài và gửi đến cả 2 người chơi. Ai giải được ván bài nhanh hơn thì giành thắng lợi và được 1 điểm, người thua 0 điểm. Nếu cả hai người chơi đều không giải được thì ván đấu hòa, mỗi bên được 0.5 điểm.
- Sau mỗi ván, đều có dialog hỏi mỗi người chơi có muốn tiếp tục không. Nếu cả hai tiếp tục thì chơi tiếp, nếu một trong hai đối thủ dừng chơi thì thoát ra và server báo cho người chơi còn lại.

- Kết quả các trận đấu được lưu vào server. Mỗi người chơi đều có thể vào xem bảng xếp hạng các người chơi trong toàn bộ hệ thống, theo lần lượt các tiêu chí: tổng số điểm (giảm dần), trung bình điểm của các đối thủ đã gặp (giảm dần), trung bình thời gian giải trong các trận thắng (tăng dần).

17. Game spider solitaire thi đấu đối kháng online

- Hệ thống có một server và nhiều client. Server lưu toàn bộ thông tin và dữ liệu.
- Để chơi, người chơi phải login vào tài khoản của mình từ một máy client. Sau khi login thành công, giao diện hiện lên hai danh sách: một danh sách các người chơi đang online, mỗi người chơi có các thông tin: tên, tổng số điểm hiện có của người chơi, trạng thái (hoặc đang bận nếu đang chơi với người khác, hoặc đang rỗi nếu không chơi với ai). Danh sách thứ hai là danh sách các bàn đang thi đấu: mỗi bàn hiện mã màn, số lượng người chơi trong bàn.
- Muốn mời (thách đấu) ai thì người chơi click vào tên của đối thủ đó trong danh sách online.
- Khi bị thách đấu, người chơi có thể chấp nhận (OK), hoặc từ chối (Reject).
- Khi chấp nhận, 2 người chơi sẽ vào chơi với nhau và tạo thành một bàn, và server sẽ làm trọng tài. Giao diện chơi gồm bàn chơi và nút thoát khỏi bàn.
- Server sẽ ra cùng một đề bài và gửi đến cả 2 người chơi. Ai giải được ván bài nhanh hơn thì giành thắng lợi và được 1 điểm, người thua 0 điểm. Nếu cả hai người chơi đều không giải được thì ván đấu hòa, mỗi bên được 0.5 điểm.
- Sau mỗi ván, đều có dialog hỏi mỗi người chơi có muốn tiếp tục không. Nếu cả hai tiếp tục thì chơi tiếp, nếu một trong hai đối thủ dừng chơi thì thoát ra và server báo cho người chơi còn lại.
- Kết quả các trận đấu được lưu vào server. Mỗi người chơi đều có thể vào xem bảng xếp hạng các người chơi trong toàn bộ hệ thống, theo lần lượt các tiêu chí: tổng số điểm (giảm dần), trung bình điểm của các đối thủ đã gặp (giảm dần), trung bình thời gian giải trong các trận thắng (tăng dần).

18. Chát online

- CSDL về người dùng được lưu trên server. Mỗi người dùng có một danh sách bạn bè. Hai người dùng là bạn bè thì có thể chát trực tiếp với nhau.
- Người dùng cũng có thể vào các phòng chat để chát với nhiều người cùng lúc.
- Khi một người dùng login thành công vào hệ thống, giao diện chính sẽ hiện lên danh sách các ban bè của người đó: mỗi người hiện nick name của họ. Những ai đang online thì hiện ở chế độ active (click vào được), những ai không online ở thời điểm đó thì hiện ở chế độ

inactive (không click vào được). Đồng thời, trạng thái của người dùng mới online này sẽ được cập nhật thành online trong tất cả danh sách bạn bè của bạn bè người này đang online.

- Người dùng có thể click vào nick của một người bạn đang active, khi đó server sẽ kết nối hai người thành một kênh riêng chát với nhau. Mỗi người có thể gõ các dòng chát gửi đi, cả hai bên đều nhận được dòng chát trên cửa sổ riêng với bạn đó. Server đồng thời cũng lưu lại lịch sử chát giữa hai người.
- Người dùng cũng có thể tạo phòng chát và mời bạn bè vào phòng chát đó. Khi được mời, một người có thể từ chối hoặc chấp nhận tham gia chát với cả phòng. Người dùng cũng có thể nhảy vào danh sách xác phòng chát công cộng mà người khác đã tạo ra.
- Khi chát trong chế độ chát cả phòng, mọi người đều có thể gõ dòng chát để gửi đi, và tất cả người tham gia đều nhận được dòng chát. Thứ tự xuất hiện các dngf chát ở các người chát là thứ tự server nhận được dòng chát đấy.
- Khi muốn thoát khỏi chương trình, người dùng click vào nút logout. Khi đó, trạng thái của người dùng đó trong danh sách bạn bè của bạn bè người đó đều tự động chuyển thành inactive.

19. Mạng xã hội online

- Mạng có nhiều thành viên, thông tin các thành viên đều được lưu trên server.
- Mỗi thành viên có thể có nhiều bạn bè. Thành viên là bạn bè thì có thể đăng trạng thái trên tường của nhau, hoặc bình luận vào các trạng thái trên tường của nhau.
- Mỗi thành viên có một trang chủ riêng của mình, gọi là tường nhà mình. Sau khi login thành công, thành viên sẽ vào tường nhà mình. Trên đó sẽ có 2 danh sách. Thứ nhất là danh sách các trạng thái mới nhất của thành viên đó, xếp theo thứ tự thời gian đăng bài. Thứ hai là danh sách các trạng thái mới nhất của các bạn bè của thành viên đó, xếp theo thứ tự thời gian của bình luận muộn nhất trong bài đăng.
- Mỗi thành viên có thể click vào một trong các bài đăng ở hai danh sách trên để xem. Khi xem có thể thêm bình luận vào bài đăng. Khi thêm bình luận thì chủ bài đăng và chủ tường tương ứng sẽ có thêm một thông báo có bình luận mới trong inbox thông báo của mình.
- Mỗi thành viên cũng có thể đăng mới trạng thái, hoặc đăng mới bài lên tường của bạn bè. Khi đó, bạn bè của người có tường bị đăng sẽ được tự động cập nhất thông tin mới của bài đăng.
- Chủ một tường có quyền xóa các bài đăng do mình đăng (ở tất cả các tường), các bình luận do mình đăng (ở tất cả các bài đăng), hay các bài đăng và bình luận của bất kì ai trên

tường của mình. Khi một bài đăng hay một bình luận bị xóa thì nó sẽ cập nhật lại thứ tự hiển thị của bài đăng/bình luận đó.

- Khi thành viên logout, mà các thành viên bạn bè vẫn có hoạt động đăng bài, bình luận liên quan đến thành viên đấy thì hệ thống sẽ thêm các thông báo cập nhật vào inbox cho thành viên đấy. Lần login tiếp théo thành viên đấy sẽ nắm được các thông tin mới cập nhật liên quan đến mình.

20. Diễn đàn kín online

- Diễn đàn có nhiều thành viên. Thông tin các thành viên được lưu trên server.
- Chỉ có các thành viên đã login vào hệ thống mới được đăng bài và đọc bài của người khác. Những người dùng khác, chưa đăng kí hoặc chưa login thì không được tham gia vào diễn đàn.
- Diễn đàn có nhiều chuyên mục. Mỗi chuyên mục có thể có nhiều chuyên mục con. Mỗi chuyên mục có thể có nhiều bài đăng.
- Mỗi thành viên sau khi login thì sẽ ở trang chủ diễn đàn: hiển thị tên các chuyên mục mức cao nhất của diễn đàn. Mỗi chuyên mục hiện tên và hiện bài đăng mới nhất trong chuyên mục đó. Thời gian tính là thời điểm gần nhất một bài đăng được đăng hoặc một bình luận của bài đăng đó được đăng.
- Người dùng có thể click vào một chuyên mục hoặc vào các chuyên mục con. Sau đó có thể chọn đăng bài mới trong chuyên mục đó, hoặc chọn xem một bài đăng trong chuyên mục đó. Trong khi xem một bài đăng, người dùng có thể thêm bình luận trong bài đăng đó.
- Khi một bài đăng mới được tạo ra, hoặc một bình mới được thêm vào, hệ thống sẽ tự động cập nhật đến toàn bọ người dùng đnag online và đnag ở giao diện trang chủ diễn đàn.