LẬP TRÌNH MẠNG

HTTP – SMTP - FTP

hungdn@ptit.edu.vn

Nội dung

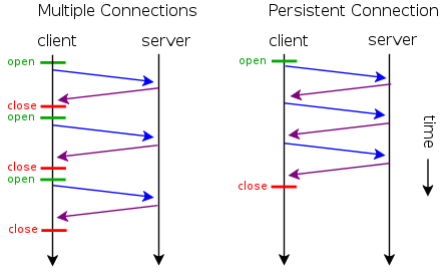
- Giao thức HTTP (Hypertext Transfer Protocol)
- Giao thức SMTP (Simple Mail Transfer Protocol)
- Giao thức FTP (File transfer Protocol)
- ~~Giao thức SNMP (Simple Network Management Protocol)~~

Đều sử dụng giao thức TCP để gửi/nhận thông điệp

1. HTTP

HTTP (Hypertext Transfer Protocol)

- **HTTP là một giao thức client/server ở tầng ứng dụng**
  - Định nghĩa cấu trúc của các thông điệp được trao đổi và cách chúng được trao đổi qua mạng
  - Sử dụng giao thức TCP tầng vận chuyển
  - Đảm bảo tin cậy truyền tải thông điệp
- **Phiên bản**
  - Phiên bản HTTP 1.0, mọi đối tượng trên trang (hình ảnh, đồ họa, ...) yêu cầu tạo một kết nối để truyền tải
  - Phiên bản HTTP 1.1, bổ sung *persistence,* cho phép client và server duy trì một kết nối cho tới khi tất cả các đối tượng trên trang được truyền tải xong (# **state**)
- **Giao thức phi trạng thái (stateless)**
  - Client và Server không lưu trữ thông tin về đối tượng còn lại trong suốt phiên làm việc
  - Client kết nối -> Server đáp trả thông tin yêu cầu -> đóng kết nối.



Ex: request / response

- **Request**

```
GET /somedir/index.html  HTTP/ 1.1
Host: www.ptit.edu.vn
Connection: close
User-Agent : Mozilla/4.0
Accept-language :en-Us,en
```

- **Respone**

```
HTTP/1.1 200 OK
Date: Mon, 23 May 2005 22:38:34 GMT
Content-Type: text/html; charset=UTF-8
Content-Length: 138
Last-Modified: Wed, 08 Jan 2003 23:11:55 GMT
Server: Apache/1.3.3.7 (Unix) (Red-Hat/Linux)
Accept-Ranges: bytes
Connection: close

<html>
      ...
</html>
```

Ex: HTTP Client

- HTTP Client sử dụng Java Socket
  - Bước 1: Tạo socket (Socket vs SocketChanel)
  - Bước 2: Tạo luồng I/O tới socket (luồng input và output)
  - Bước 3: Đọc và ghi vào luồng tuân theo giao thức của máy chủ (HTTP)
  - Bước 4: Đóng luồng
  - Bước 5: Đóng socket
- Lưu ý:
  - Tại bước 3, thực hiện việc khởi tạo và ghi HTTP request ra luồng output

Ex: HTTP Server

- HTTP Server sử dụng Java Socket
  - Bước 1: Tạo ServerSocket lắng nghe tại port xác định
  - Bước 2: Tạo luồng I/O tới socket kết nối
  - Bước 3: Đọc và ghi vào luồng theo giao thức HTTP
  - Bước 4: Đóng luồng
  - Bước 5: Đóng Socket
- Lưu ý:
  - Tại bước 3, thực hiện xử lý yêu cầu và ghi HTTP Response ra luồng output

Ex: HTTP Server

```
public class HttpSimpleServer {
        public static void main(String args[]) throws IOException {
                ServerSocket server = new ServerSocket(8080);
                System.out.println("Listening for connection on port 8080 ....");
                while (true) {
                        try (Socket socket = server.accept()) {
                                InputStreamReader isr = new
                                                InputStreamReader(socket.getInputStream());
                                BufferedReader reader = new BufferedReader(isr);
                                String line = reader.readLine();
                                while (!line.isEmpty()) {
                                        System.out.println(line);
                                        line = reader.readLine();
                                }
                                Date today = new Date();
                                String httpResponse = "HTTP/1.1 200 OK\r\n\r\n" + today;
                                socket.getOutputStream().write(httpResponse.getBytes("UTF-8"));
                                socket.close();
                        }
                }
        }
}
```

Ex: HTTP Client

```
public class HttpSimpleClient {
        public static void main(String[] args) {
                Socket s;
                try {
                        s = new Socket(InetAddress.getByName("localhost"), 8080);
                        PrintWriter out = new PrintWriter(s.getOutputStream(), true);
                        String request = "GET " + " HTTP/1.1\r\n"
                                        + "User-Agent: HttpSimple/1.0\r\n"
                                        + "Accept: text/*\r\n"
                                        + "Host: " + "localhost:8080" + "\r\n" + "\r\n";
                        out.println(request);
                        InputStream in = s.getInputStream();
                        int read = 0;
                        byte[] bytes = new byte[1024];
                        while ((read = in.read(bytes)) != -1) {
                                in.read(bytes);
                        }
                        System.out.println(new String(bytes));
                        s.close();
                } catch (Exception ex) {
                        Logger.getLogger(HttpSimpleClient.class.getName()).log(Level.SEVERE, null, ex);
                }
        }
}
```

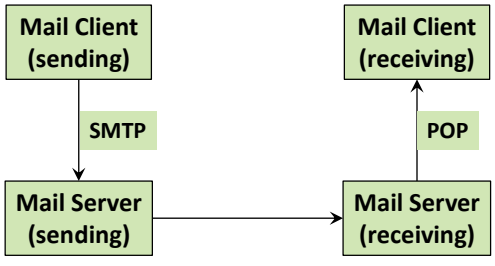
Một số lớp hữu ích

- **Lớp URLConnection cung cấp các phương thức cho việc truyền thông sử dụng URL**
  - Có thể sử đọc trực tiếp từ một URL connection sử dụng phương thức openStream()
- **Lớp HttpURLConnection**
  - Cung cấp nhiều phương thức hữu ích khi làm việc với Http Url
- **Lớp HttpServer**
  - Xây dựng một web server đơn giản (light-weight HTTP server)
- **Jsoup**
  - Vấn đề: truy cập vào các thành phần của một tài liệu html
  - Jsoup là một thư viện Java cung cấp API cho phép parse HTML thành cây phân cấp đối tượng (tương tự như trình duyệt làm việc với tài liệu html -> DOM)
  - Ứng dụng
    - Tìm kiếm (select) và trích xuất dữ liệu
    - Tương tác với các phần tử, thuộc tính và nội dung trong tài liệu HTML

2. SMTP

Sending/Receiving Email

- **SMTP (Simple Mail Transfer Protocol) 25**
  - Được sử dụng để gửi thư điện tử từ máy chủ mail này đến máy chủ mail khác
- **POP (Post Office Protocol) 110**
  - Được sử dụng để lấy thư điện tử từ máy chủ mail
  - Truyền tải toàn bộ thư tử máy chủ mail về mail client
- **IMAP(Internet Message Access Protocol)**
  - Được sử dụng bởi các dịch vụ mail trên web như Yahoo, Gmail
  - Cho phép một trình duyệt web (mail client) có thể đọc các thư điện tử chứa trên máy chủ mail



SMTP

- Giao thức cho phép gửi/nhận thư điện tử host-to-host sử dụng giao thức TCP, cổng 25
- Gửi lệnh theo mã ASCII, kết thúc bằng dòng mới
- Thực hiện truyền request và response giữa hai đối tượng Sender-SMTP và Recipient-SMTP
- Đối tượng nhận có thể là máy chủ đích hoặc qua máy chủ trung gian relay SMTP-server
- Lệnh và trả lời không phân biệt chữ hoa chữ thường

Ex: SMTP Session

```
S: 220 smtp.example.com ESMTP Postfix
C: HELO relay.example.org
S: 250 Hello relay.example.org, I am glad to meet you
C: MAIL FROM:<bob@example.org>
S: 250 Ok
C: RCPT TO:<alice@example.com>
S: 250 Ok
C: RCPT TO:<theboss@example.com>
S: 250 Ok
C: DATA
S: 354 End data with <CR><LF>.<CR><LF>
C: From: "Bob Example" <bob@example.org>
C: To: "Alice Example" <alice@example.com>
C: Cc: theboss@example.com
C: Date: Tue, 15 Jan 2008 16:02:43 -0500
```

Code 250 means everything is OK

Source: Wikipedia

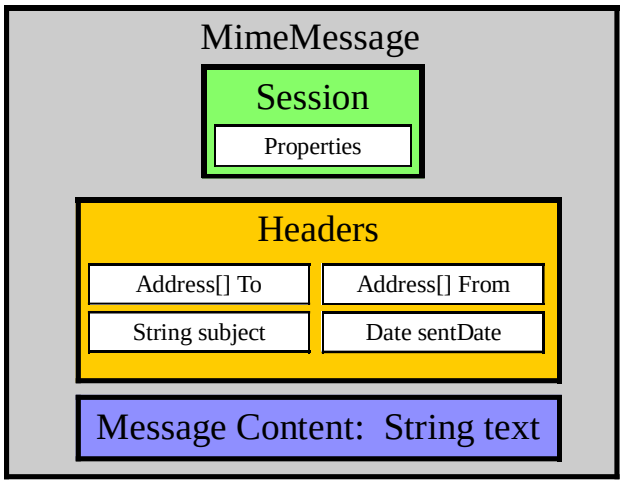
Ex: SMTP Session

```
C: Subject: Test message
C:
C: Hello Alice.
C: This is a test message with 5 header fields and 4 lines.
C: Your friend,
C: Bob
C: .
S: 250 Ok: queued as 12345
C: QUIT
S: 221 Bye
{The server closes the connection}
```

Source: Wikipedia

Các thành phần trong mail

- **Body**
  - Các đoạn text mã NVT ASCII
- **Multipurpose Internet Mail Extension (MIME)**
  - Thường được gửi dưới dạng nhị phân (attachments, embedded graphics, or non-ASCII characters)
  - Ex:
    - Text
    - Message
    - Image
    - Video
    - Multipart
    - Application
    - Audio



Một số lớp java làm việc với mail

- `Address (package javax.mail)`
- `InternetAddress (package javax.mail.internet)`
  - sử dụng định dạng "user@host.domain".
- `Message (package javax.mail)`
- `MimeMessage (package javax.mail.internet)`
  - Làm việc với các kiểu MIME trong thư.
  - Kế thừa lớp trừu tượng Message và cài đặt interface MimePart

Một số lớp java làm việc với mail

- `Transport` (package `javax.mail`) – represents a transport protocol that allows e-mail messages to be sent.
  - implemented by a specific protocol
- `Session` (package `javax.mail`) – provides an interface between the e-mail client and the network, supporting the creation of and access to Store and Transport objects.
- `Authenticator` (package `javax.mail`) – an object that knows how to obtain authentication for a network connection.
- `PasswordAuthentication` (package `javax.mail`) – a data holder that is used by Authenticator for a user name and a password.

Cài đặt JavaMail

- Download JavaMail.
  - http://www.oracle.com/technetwork/java/javamail/index.html
  - Phiên bản mới nhất JavaMail 1.x.x
  - Tên file is javax.mail.jar
- Thêm thư viện javax.mail.jar vào project.

Gửi mail

- B1: Xác định giao thức

```
props.setProperty("xxxx", value)
```

- B2: Tạo session để gửi mail

```
Session session = Session.getInstance(props, new Authenticator()...);
```

- B3: Tạo mail qua lớp Message

```
Message mess = new MimeMessage(sess);
mess.setFrom(new InternetAddress("yourmail@gmail.com"));
...
```

- B4: Gửi mail

```
Message mess = send.createMessage(sess);
Transport.send(mess);
```

Nhận mail

- B1: Xác định giao thức (imap hoặc pop)

```
props.setProperty("mail.store.protocol", protocol)
```

- B2: Tạo session để đọc mail

```
Session session = Session.getInstance(props, null);
```

- B3: Truy cập mail sử dụng lớp Store

```
Store store = session.getStore();
store.connect("imap.gmail.com", "yourEmailId@gmail.com", "password");
Folder inbox = store.getFolder("INBOX");
inbox.open(Folder.READ_ONLY);
```

- B4: Đọc inbox

```
Message msg = inbox.getMessage(1);
Message msg = inbox.getMessage(inbox.getMessageCount());
```

MessageSend.java

```
public Session createSession() {
        final String username = "yourmail@gmail.com";
        final String password = "yourpass";

        Properties props = new Properties();
        props.put("mail.smtp.host", "smtp.gmail.com");
        props.put("mail.smtp.socketFactory.port", "465");
        props.put("mail.smtp.socketFactory.class", "javax.net.ssl.SSLSocketFactory");
        props.put("mail.smtp.auth", "true");
        props.put("mail.smtp.port", "465");

        Session session = Session.getDefaultInstance(props, new Authenticator() {
                protected PasswordAuthentication getPasswordAuthentication() {
                        return new PasswordAuthentication(username, password);
                }
        });
        return session;

}
```

MessageSend.java (Cont.)

```
public Message createMessage(Session sess) throws MessagingException {
        Message mess = new MimeMessage(sess);
        mess.setFrom(new InternetAddress("yourmail@gmail.com"));
        mess.setRecipients(Message.RecipientType.TO, InternetAddress.parse("
                                                    yourmail@gmail.com ", false));
        mess.setSubject("Test");
        mess.setText("This is a test of JavaMail's functionality.");
        mess.setSentDate(new Date());
        return mess;
}
```

```
public static void main(String[] args) {
        MessageSend send = new MessageSend();
        Session sess = send.createSession();
        try {
                Message mess = send.createMessage(sess);
                Transport.send(mess);
        } catch (MessagingException e) {
                System.out.println("Messaging Exception:  " + e);
        }
}
```

ReadingMail.java

```
public static void main(String[] args) {
        Properties props = new Properties();
        props.setProperty("mail.store.protocol", "pop3");
        try {
                Session session = Session.getInstance(props, null);
                Store store = session.getStore();
                store.connect("imap.gmail.com", "yourmail@gmail.com", "yourpass");
                Folder inbox = store.getFolder("INBOX");
                inbox.open(Folder.READ_ONLY);
                Message msg = inbox.getMessage(inbox.getMessageCount());
                Address[] in = msg.getFrom();
                System.out.println(in.length);
                for (Address address : in)
                        System.out.println("FROM:" + address.toString());
                Multipart mp = (Multipart) msg.getContent();
                BodyPart bp = mp.getBodyPart(0);
                System.out.println("SENT DATE:" + msg.getSentDate());
                System.out.println("SUBJECT:" + msg.getSubject());
                System.out.println("CONTENT:" + bp.getContent());
        } catch (Exception ex) {         ex.printStackTrace();    }
}
```

3. FTP

Thư viện Apache Commons Net API

- Thư viện hỗ trợ xây dựng ứng dụng upload/download file
- Lớp FTPClient: storeXXX chuyển file từ local → server

```
boolean storeFile(String remote, InputStream local)
OutputStream storeFileStream(String remote)
boolean storeUniqueFile(InputStream local)
boolean storeUniqueFile(String remote, InputStream local)
OutputStream storeUniqueFileStream()
OutputStream storeUniqueFileStream(String remote)
```

- Thiết lập kết nối
  - *Local active mode:* Mở cổng trên client để server kết nối → thường bị chặn bởi tường lửa
  - *Local passive mode:* Mở công trên server để client kết nối → không bị chặn

Upload file

- Kết nối và login tới server.
- Thiết lập kết nối kiểu *local passive*.
- Thiết lập kiểu dữ liệu truyền là nhị phân.
- Tạo InputStream cho file local.
- Xây dựng đường dẫn file từ xa cho server.
- Gọi các phương thức storeXXX() để truyền, có 2 cách:
  - Dựa trên InputStream
  - Dựa trên OutputStream
- Đóng luồng.
- Gọi phương thức completePendingCommand() để hoàn thành công việc.
- Thoát và đóng kết nối.

FTPUploadDemo

```
FTPClient ftpClient = new FTPClient();
try {

        ftpClient.connect(server, port);
        ftpClient.login(user, pass);
        ftpClient.enterLocalPassiveMode();
        ftpClient.setFileType(FTP.BINARY_FILE_TYPE);

        // APPROACH #1: uploads first file using an InputStream
        File firstLocalFile = new File("D:/Test/Projects.zip");

        String firstRemoteFile = "ProjectsRemote.zip";
        InputStream inputStream = new FileInputStream(firstLocalFile);

        System.out.println("Start uploading first file");
        boolean done = ftpClient.storeFile(firstRemoteFile, inputStream);
        inputStream.close();
        if (done) {
                System.out.println("The first file is uploaded successfully.");
        }
```

FTPUploadDemo (Cont.)

```
// APPROACH #2: uploads second file using an OutputStream
File secondLocalFile = new File("E:/Test/Report.doc");
String secondRemoteFile = "test/Report.doc";
inputStream = new FileInputStream(secondLocalFile);

System.out.println("Start uploading second file");
OutputStream outputStream = ftpClient.storeFileStream(secondRemoteFile);
byte[] bytesIn = new byte[4096];
int read = 0;
while ((read = inputStream.read(bytesIn)) != -1) {
        outputStream.write(bytesIn, 0, read);
}
inputStream.close();
outputStream.close();
boolean completed = ftpClient.completePendingCommand();
if (completed) {
        System.out.println("The second file is uploaded successfully.");
}
```

```
if (ftpClient.isConnected()) {
        ftpClient.logout();
        ftpClient.disconnect();
}
```

Download file

- Kết nối và login tới server.
- Thiết lập kết nối kiểu *local passive*.
- Thiết lập kiểu dữ liệu truyền là nhị phân.
- Thiết lập đường dẫn file từ xa
- Tạo OutputStream để ghi file local.
- Nhận file
  - retrieveFile() :
  - retrieveFileStream()
- Đóng luồng.
- Thoát và đóng kết nối.

FTPDownloadDemo

```
FTPClient ftpClient = new FTPClient();
try {

        ftpClient.connect(server, port);
        ftpClient.login(user, pass);
        ftpClient.enterLocalPassiveMode();
        ftpClient.setFileType(FTP.BINARY_FILE_TYPE);

        // APPROACH #1: using retrieveFile(String, OutputStream)
        String remoteFile1 = "/test/video.mp4";
        File downloadFile1 = new File("D:/Downloads/video.mp4");
        OutputStream outputStream1 = new BufferedOutputStream(new
FileOutputStream(downloadFile1));
        boolean success = ftpClient.retrieveFile(remoteFile1, outputStream1);
        outputStream1.close();

        if (success) {
                System.out.println("File #1 has been downloaded successfully.");
        }
```

FTPDownloadDemo (Cont.)

```
// APPROACH #2: using InputStream retrieveFileStream(String)
String remoteFile2 = "/test/song.mp3";
File downloadFile2 = new File("D:/Downloads/song.mp3");
OutputStream outputStream2 = new BufferedOutputStream(new FileOutputStream(downloadFile2));
InputStream inputStream = ftpClient.retrieveFileStream(remoteFile2);
byte[] bytesArray = new byte[4096];
int bytesRead = -1;
while ((bytesRead = inputStream.read(bytesArray)) != -1) {
        outputStream2.write(bytesArray, 0, bytesRead);
}

success = ftpClient.completePendingCommand();
if (success) {
        System.out.println("File #2 has been downloaded successfully.");
}
outputStream2.close();
inputStream.close();
```

```
if (ftpClient.isConnected()) {
        ftpClient.logout();
        ftpClient.disconnect();
}
```

Demo

- **Http Server**
- **Http Client**
- **HttpURLConnection**
- **SMTP**
- **FTP server/client**

Exercise

- **Java.NIO**
  - SocketChanel
- **Jsoup**
- **HttpServer**
- **HTTP**
- **SMTP**
- **FTP**

Chủ đề

- **Simple web server**
  - Validate HTTP Request
  - Handle Path/URL
  - ...
- **Simple FTP Server/Client**
- **Simple mail server**
- **Mô phỏng / xây dựng các ứng dụng sử dụng các giao thức tầng ứng dụng**
  - HTTP
  - SMTP
  - FTP

Q & A