实践任务书
网络应用程序设计

东北大學

东北大学软件学院

2017年2月

目录

实践要求

《网络应用程序设计》实践课程开设的目的是为了让学生加深理解计算机网络的工作方式，计算机网络的体系结构和 TCP/IP协议栈，学会使用常用的网络命令并能够利用这些命令进行网络故障的检测；该实践课程还让学生在熟悉 Eclipse开发环境的基础上，使学生掌握Java网络编程、多线程编程，能够熟练掌握面向对象的方法，使用Java的编码规则，能够利用所学的计算机网络知识和Java编程语言知识根据需求进行程序设计，独立完成程序设计，培养学生理论联系实际的能力。

在《网络应用程序设计实践》的实习过程中，要求学生做到：

（1）预习实验指导书有关部分，认真做好实验内容的准备， 就实验可能出现的情况提前做出思考和分析。

（2）遵守机房纪律，服从辅导教师指挥，爱护实验设备。

（3）上机不迟到。不允许无故缺席（确实有特殊情况的除外，但需要有证明）。

（4）仔细观察程序调试过程中出现的各种问题，并对主要问题进行记录，给出必要说明和分析。

（5）认真书写实践报告。实践报告模板另附。

实践的验收将分为三个部分。第一部分是平时上机操作的验收，包括检查程序运行和即时提问。第二部分是最终答辩考核情况。第三部分是提交的书面实践报告。

实践内容概述

(一)实践的任务

本次实践课的内容大体上分为三部分内容：

（1） 使用HTTP通信

（2） 使用FTP通信

（3） 使用Socket实现即时通信系统

（4） 基于RMI的分布式会议议程服务（选做）

这三个实验目的是为了让学生加深理解计算机网络的工作方式，计算机网络的体系结构和 TCP/IP协议栈，学会使用常用的网络命令并能够利用这些命令进行网络故障的检测；该实践课程还让学生在熟悉 Eclipse开发环境的基础上，使学生掌握Java网络编程、多线程编程和 RMI编程，能够熟练掌握面向对象的方法，使用Java的编码规则，能够利用所学的计算机网络知识和 Java编程语言知识根据需求进行程序设计，独立完成程序设计，培养学生理论联系实际的能力。

注意：要求学生最终提交完整的源代码，并提交实验报告，参与最终的答辩。

（二）本次实践规定

（1） 所有程序保存的位置：要求建立一个以“自己的姓名＋学号后四位”命名的文件夹。所有和程序实践相关的文档和源代码都存放在该文件夹中。

（2） 使用良好的程序设计方法，利用面向对象的方法设计和实现系统。利用所学的软件工程、数据结构等知识指导编程。

（3） 对变量的命名要尽量有意义，例如定义一个变量：

```
private double salary;
```

（4）代码应该符合 Java的编码规则，代码中含有适当的注释。例如：

```
/*!Begin Snippet:file*/
/**
 * This class models a person.
 *
 * @author author name
 * @version 1.0.0
 */
public class Person  {
            /* Name of the person */
            private String  name;
            /* Address of the person */
            private String  address;
            /**
             * Constructs a <code>Person</code> object.
             *
             * @param initialName  the name of the person.
             * @param initialAddress  the address of the person.
             */
            public Person (String initialName, String initialAddress) {
                        name = initialName;
                        address = initialAddress;
            }
            /**
             * Returns the name of this person.
             *
             * @return the name of this person.
             */
            public String getName() {
                        return this.name;
            }
            /**
             * Returns the address of this person.
             *
             * @return the address of this person.
             */
            public String getAddress() {
                        return this.address;
            }
}
/*!End Snippet:file*/
```

（5）按照课时安排完成相应实践内容并按时提交实践成果物。

实验 1 使用 FTP 实现网络通信（8 学时）

(一) 实验目的

（1）理解FTP通信原理；

（2）学习使用Socket和多线程编写简单的FTP服务器；

（3）学习使用Sokcet编写简单的FTP客户端；

(二) 实验内容

本实验要求完成以下功能（使用PASV模式）：

功能名称	功能说明	优先级
获取文件列表	给定用户工作目录,获得文件列表,包括文件名称,文件大小,文件创建时间	高
文件上传	上传文件到用户指定工作目录中	高
	上传文件夹到用户指定工作目录中	低
	支持断点续传	低
文件下载	给定用户工作目录,下载文件	高
	给定用户工作目录,下载文件夹	低
	支持断点续传	低

(三) 预习内容

（1）FTP通信流程和通信命令

FTP 使用2个端口，一个数据端口和一个命令端口（也叫做控制端口）。这两个端口一般是21（命令端口）和 20（数据端口）。

1> 命令端口

一般来说，客户端有一个 Socket 用来连接 FTP 服务器的相关端口，它负责 FTP 命令的发送和接收返回的响应信息。一些操作如“登录”、“改变目录”、“删除文件”，依靠这个连接发送命令就可完成。

2> 数据端口

对于有数据传输的操作，主要是显示目录列表，上传、下载文件，我们需要依靠另一个 Socket 来完成。

如果使用被动模式，通常服务器端会返回一个端口号。客户端需要另开一个 Socket 来连接这个端口，然后数据会通过这个新开的端口传输。

如果使用主动模式，通常客户端会发送一个端口号给服务器端，并在这个端口监听。服务器需要连接到客户端开启的这个数据端口，并进行数据的传输。

3> 主动模式 (PORT)

主动模式下，客户端随机打开一个大于 1024 的端口（我们称这个端口为 N）向服务器的命令端口，即 21 端口，发起连接，同时开放另一个端口监听（我们称这个端口为 N+1），并向服务器发出 “port N+1” 命令，由服务器从它自己的数据端口 (20) 主动连接到客户端指定的数据端口 (N+1)。

FTP 的客户端只是告诉服务器自己的端口号，让服务器来连接客户端指定的端口。对于客户端的防火墙来说，这是从外部到内部的连接，可能会被阻塞。

4> 被动模式 (PASV)

为了解决服务器发起到客户的连接问题，有了另一种 FTP 连接方式，即被动方式。命令连接和数据连接都由客户端发起,这样就解决了从服务器到客户端的数据端口的连接被防火墙过滤的问题。被动模式下，当开启一个 FTP 连接时，客户端打开两个任意的本地端口 (N > 1024 和 N+1) 。

第一个端口连接服务器的 21 端口，提交 PASV 命令。然后，服务器会开启一个任意的端口 (P > 1024)，返回如“227 entering passive mode (127,0,0,1,4,18)”。 它返回了 227 开头的信息，在括号中有以逗号隔开的六个数字，前四个指服务器的地址，最后两个，将倒数第二个乘 256 再加上最后一个数字，这就是 FTP 服务器开放的用来进行数据传输的端口。如得到 227 entering passive mode (h1,h2,h3,h4,p1,p2)，那么端口号是 p1*256+p2，ip 地址为 h1.h2.h3.h4。这意味着在服务器上有一个端口被开放。客户端收到命令取得端口号之后，会通过 N+1 号端口连接服务器的端口 P，然后在两个端口之间进行数据传输。

5> 主要用到的 FTP 命令

FTP 每个命令都有 3 到 4 个字母组成，命令后面跟参数，用空格分开。每个命令都以"\r\n"结束。

要下载或上传一个文件，首先要登入 FTP 服务器，然后发送命令，最后退出。这个过程中，主要用到的命令有 USER、PASS、SIZE 、CWD、RETR、PASV、PORT、REST、QUIT。

USER: 指定用户名。通常是控制连接后第一个发出的命令。“USER gaoleyi\r\n”： 用户名为 gaoleyi 登录。

PASS: 指定用户密码。该命令紧跟 USER 命令后。“PASS gaoleyi\r\n”：密码为 gaoleyi。

SIZE: 从服务器上返回指定文件的大小。“SIZE file.txt\r\n”：如果 file.txt 文件存在，则返回该文件的大小。

CWD: 改变工作目录。如：“CWD dirname\r\n”。

PASV: 让服务器在数据端口监听，进入被动模式。如：“PASV\r\n”。

PORT: 告诉 FTP 服务器客户端监听的端口号,让 FTP 服务器采用主动模式连接客户端。如：“PORT h1,h2,h3,h4,p1,p2”。

RETR: 下载文件。“RETR file.txt \r\n”：下载文件 file.txt。

STOR: 上传文件。“STOR file.txt\r\n”：上传文件 file.txt。

REST: 该命令并不传送文件，而是略过指定点后的数据。此命令后应该跟其它要求文件传输的 FTP 命令。“REST 100\r\n”：重新指定文件传送的偏移量为 100 字节。

QUIT: 关闭与服务器的连接。

6> FTP 响应码

客户端发送 FTP 命令后，服务器返回响应码。

响应码用三位数字编码表示：

第一个数字给出了命令状态的一般性指示，比如响应成功、失败或不完整。

第二个数字是响应类型的分类，如 2 代表跟连接有关的响应，3 代表用户认证。

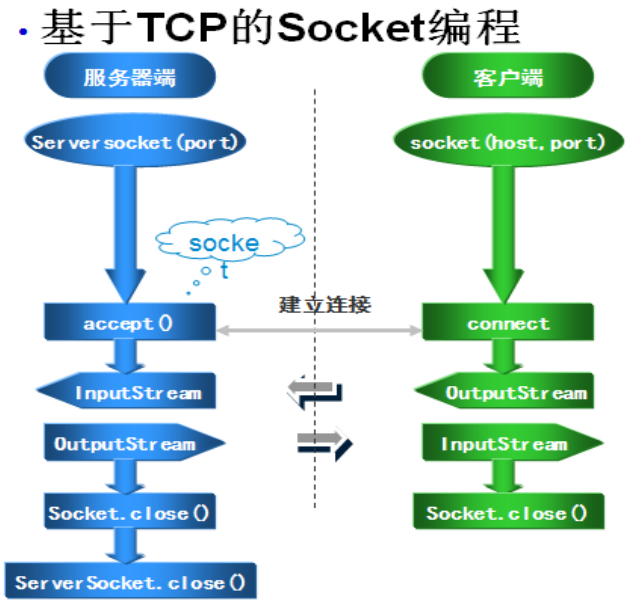
第三个数字提供了更加详细的信息。如：

227 entering passive mode (127,0,0,1,4,18)

230 Logged on

250 CWD successful

（2）Socket编程



- 利用TCP协议进行通信两个应用程序，有主从之分一个称为服务器程(Server)，另外一个称为客户机程(Client)
- 交互过程

(1) 服务器程序创建一个ServerSocket,然后调用accept方法等待客户来连接

(2) 客户端程序创建一个socket并请求与服务器建立连接

(3) 刚才建立了连接的两个socket在一个单独的线程上对话

(4) 服务器开始等待新的连接请求

典型的Socket通信代码如下：

```
public class TCPServer {
    public static void main(String args[]) throws
IOException{
        ServerSocket ss = new ServerSocket(3336);
        while(true){
        Socket s = ss.accept();
       DataInputStream dis = new
      DataInputStream(s.getInputStream());
        System.out.println(dis.readUTF());
     System.out.println("Hello,client:"+s.getInetAddress()
+":"+s.getPort());
        dis.close();
        s.close();
        }
    }
}
```

TCPServer.java

```
public class TCPClient {
    public static void main(String args[]) throws
UnknownHostException, IOException{
        Socket s = new Socket("127.0.0.1",3336);
    OutputStream os = s.getOutputStream();
        DataOutputStream dos = new
DataOutputStream(os);
        dos.writeUTF("Hello Server!");
        dos.flush();
        dos.close();
        s.close();
    }
}
```

TCPClient.java

（3）多线程编程

为了保证FTP服务器能够同时处理多个客户端的请求，服务器为每个客户端分配一个线程，参考代码如下：

```
try{
    ServerSocket s = new ServerSocket(PORT);
    logger.info("Connecting to server A...");
    logger.info("Connected Successful! Local Port:"+s.getLocalPort()+". Default Directory:'"+F_DIR+"'.");

    while( true ){
        //接受客户端请求
        Socket client = s.accept();
        //创建服务线程
        new ClientThread(client, F_DIR).start();
    }
} catch(Exception e) {
    logger.error(e.getMessage());
    for(StackTraceElement ste : e.getStackTrace()){
        logger.error(ste.toString());
    }
}
```

（四） 实验基本步骤

（1）编写FTP客户端

1> 获得文件列表

- 客户端和 FTP 服务器建立 Socket 连接。

```
socket = new Socket(host, port);
reader = new BufferedReader(new InputStreamReader(socket.getInputStream()));
writer = new PrintWriter(new OutputStreamWriter(socket.getOutputStream()));
```

- 向服务器发送 USER、PASS 命令登录 FTP 服务器。

```
writer.println(“USER root”);
writer.flush();
reponse = reader.readLine();
writer.println(“PASS root”);
writer.flush();
reponse = reader.readLine();
```

- 使用 PASV 命令得到服务器监听的端口号，建立数据连接。

```
writer.println(“PASV”);
writer.flush();
reponse = reader.readLine(); // 227 entering passive mode  (h1,h2,h3,h4,p1,p2)
```

- 使用 p1*256+p2 计算出数据端口，连接数据端口，准备接收数据。

```
Socket dataSocket = new Socket(ip, port1);
```

- 使用 List 命令获得文件列表。

```
writer.println(“List”);
writer.flush();
reponse = reader.readLine();
```

- 从数据端口中接收数据

```
DataInputStream dis = new DataInputStream(dataSocket.getInputStream());
        String s = "";
        while ((s = dis.readLine()) != null) {
            String l = new String(s.getBytes("ISO-8859-1"), "utf-8");
            System.out.println(l);
        }
```

- 在下载完毕后断开数据连接并发送 QUIT 命令退出。

```
writer.println(“QUIT”);
writer.flush();
reponse = reader.readLine();
```

2> 下载文件

- 客户端和 FTP 服务器建立 Socket 连接。
- 向服务器发送 USER、PASS 命令登录 FTP 服务器。
- 使用 PASV 命令得到服务器监听的端口号，建立数据连接。
- 使用 RETR 命令下载文件。

```
writer.println(“RETR filename”);
writer.flush();
reponse = reader.readLine();
```

从数据端口中接收数据，保存到本地磁盘

在支持断点续传的情况下，如果本地已经存在文件，先发送 REST Size 命令指定远程文件偏移量，再发送 RETR 命令

- 在下载完毕后断开数据连接并发送 QUIT 命令退出。

3> 上传文件

- 客户端和 FTP 服务器建立 Socket 连接。
- 向服务器发送 USER、PASS 命令登录 FTP 服务器。
- 使用 PASV 命令得到服务器监听的端口号，建立数据连接。
- 使用 STOR 命令上传文件。

```
writer.println(“STOR filename”);
writer.flush();
reponse = reader.readLine();
```

使用数据端口向服务器发送文件数据

在支持断点续传的情况下，先发送 Size 命令查看文件大小，如果返回的 Size>0, 说明已经上传过，使用 InputStream.skip(length)方法指定本地文件偏移量。

- 在下载完毕后断开数据连接并发送 QUIT 命令退出。

(2) 编写服务器端程序

1. 服务器每接收一个客户端连接，为该客户端分配一个单独的子线程。
2. 子线程接收客户端命令，做相应处理

1> User 命令

2> Pass 命令

使用用户名和密码检验用户合法性

3> PASV 命令

获得服务器端空闲端口，发送给客户端，作为数据传输的端口

```
ServerSocket ss = null;
while( true ){
    //获取服务器空闲端口
    port_high = 1 + generator.nextInt(20);
    port_low = 100 + generator.nextInt(1000);
    try {
        //服务器绑定端口
        ss = new ServerSocket(port_high * 256 + port_low);
        break;
    } catch (IOException e) {
        continue;
    }
}
System.out.println("用户"+clientIp+"："+username+"执行PASV命令");
InetAddress i = null;
try {
    i = InetAddress.getLocalHost();
} catch (UnknownHostException e1) {
    e1.printStackTrace();
}
pw.println("227Entering Passive Mode ("+i.getHostAddress().replace(".", ",")+","+port_high+","+port_low+")");
//pw.println("227 Entering Passive Mode ("+i.getHostAddress().replace(".", ",")+","+port_high+","+port_low+")");
pw.flush();
```

4> Size 命令

使用命令端口返回文件大小，

5> REST 命令

记录文件偏移量，后续客户端下载文件时使用。

6> RETR 命令

通过数据端口发送文件

7> STOR 命令

通过数据端口接收文件

8> QUIT 命令

断开该客户端的连接

9> CWD 命令

设置用户的工作目录，即上传和下载文件的位置。

10> LIST 命令

列出工作目录下的文件信息，包括文件大小，文件创建时间，文件名称。

（五）实验提交内容

提交整个 FTP 客户端工程

提交整个 FTP 服务器端工程

（六）实验思考题

FTP 的工作流程和原理

实验 2 使用 HTTP 实现网络通信（8 学时）

(一) 实验目的

（1）理解HTTP通信原理；

（2）学习使用Socket编写HTTP服务器；

（3）学习使用Socket编写HTTP客户端；

(二) 实验内容

本实验要求完成以下功能（使用GET方式）：

功能名称	功能说明	优先级
服务器功能	提供 HTML,JPG 等 MIME 类型的资源	高
客户端功能	访问服务器，获取 HTML 和 JPG 资源，保存到本地磁盘	高
	访问服务器，获取 WMV 等其他资源类型，保存到本地磁盘	低

（1） 编写简单的 HTTP 1.1 客户端程序

要求：

- 通过命令行接口构建简单的HTTP1.1客户端程序；
- 简单的HTTP1.1客户端程序要求能够和Internet上的Web服务器建立TCP连接；
- Internet上的 Web服务器能够处理使用该客户端程序发送的简单请求(例如 HTTP的GET请求)；
- 简单的HTTP1.1 客户端程序能够接受服务器响应，在命令行上显示出服务器的应答头部，把服务器的应答内容保存在一个文件中。

（2） 编写简单的 HTTP 1.1 服务器程序

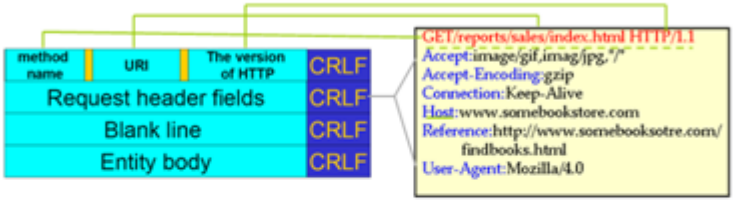
要求：

- 构建端口为8888的简单HTTP1.1服务器程序；
- 该服务器能够对所有的请求（不论是否合法）都给予应答，其中要求对HTTP的GET方法根据RFC规定给予成功的响应；
- 在任何情况下，服务器都要能够提供服务；
- 通过给命令行传递参数指定服务器的默认访问目录(例如把 c:\www作为命令行参数传递给服务器，那么如果客户发送请求 GET /test/index.html，那么服务器应该把文件C:\www\test\index.html 发送给客户端，如果文件不存在应该提示出错)；(可选)
- 服务器在成功地给予响应的同时，应该能够根据文件的扩展名，推断出至少 2种文件的 MIME类型（例如如果文件的扩展名为.htm或者.html，那么文件的 MIME类型为text/html)；
- 服务器能够响应嵌入JPEG文件的HTML页面；
- 能够通过 Netscape or Internet Explorer浏览器访问HTTP1.1服务器中存放的Web资源；

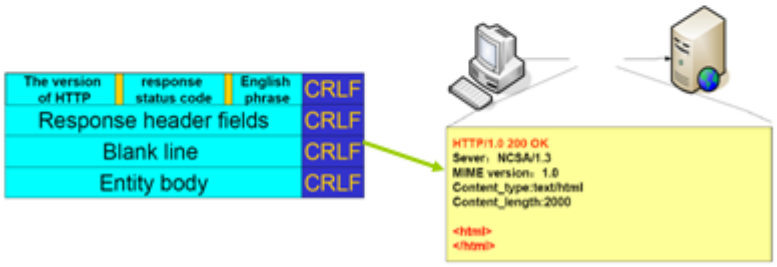
(三) 预习内容

HTTP通信原理

- HTTP请求格式



- HTTP应答格式



（四） 实验基本步骤

(1) 搭建 HTTP 服务器

1. 启动 ServerSocket,监听到客户端连接，分配子线程处理客户端请求；

```
ServerSocket ss=null;
try {
    ss = new ServerSocket(8888);
} catch (IOException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
};
while (true) {

    try {

          Socket s = ss.accept();
          TaskThread t = new TaskThread(s);
          t.start();
    } catch (IOException e1) {
        // TODO Auto-generated catch block
        e1.printStackTrace();
    }

}
```

2. 根据客户端请求文件的后缀名，发送响应类型；

```
reader = new BufferedReader(new InputStreamReader(s.getInputStream()));
  String firstLineOfRequest;
  firstLineOfRequest = reader.readLine();

  String uri = firstLineOfRequest.split(" ")[1];

    writer = new PrintStream(s.getOutputStream());
    writer.println("HTTP/1.1 200 OK");// 返回应答消息,并结束应答
    if(uri.endsWith(".html"))
    {
          writer.println("Content-Type:text/html");
    }
    else if(uri.endsWith(".jpg"))
    {
         writer.println("Content-Type:image/jpeg");

    }
    else
    {
        writer.println("Content-Type:application/octet-stream");
    }

    in = new FileInputStream("c:/workspace"+uri);

    //发送响应头
    writer.println("Content-Length:" + in.available());// 返回内容字节数
    writer.println();// 根据HTTP 协议, 空行将结束头信息

    writer.flush();
```

3. 发送响应数据；

```
//发送响应体
                byte[] b = new byte[1024];
                int len = 0;
                len = in.read(b);
                while(len!=-1)
                {
                   os.write(b, 0, len);
                   len =  in.read(b);
                }
            os.flush();
```

4. 请求资源不存在的情况下，发送纯文本的响应信息。

```
//发送响应头
                 writer.println("HTTP/1.1 404 Not Found");
                 writer.println("Content-Type:text/plain");
                 writer.println("Content-Length:7");
                 writer.println();

                  //发送响应体
                  writer.print("访问内容不存在");
              writer.flush();
```

服务器代码编写成功后，在浏览器中测试：

如：输入 http://localhost:8888/login.html

输入 http://localhost:8888/ios.jpg

(2) 搭建 HTTP 客户端

1. 连接服务器

```
Socket s = new Socket(host, port);
```

2. 发送请求头

```
PrintStream writer = new PrintStream(s.getOutputStream());
writer.println("GET /"+filename+" HTTP/1.1");
writer.println("Host:localhost");
writer.println("connection:keep-alive");
writer.println();
writer.flush();
```

3. 发送请求体（GET 方式中，请求数据挂在 URL 后，POST 方式中，请求数据放在请求体中）

4. 接收响应状态

响应成功（状态码 200） - 保存资源到本地磁盘

```
//跳过响应中的前四行，开始读取响应数据
InputStream in = s.getInputStream();
BufferedReader reader = new BufferedReader(new InputStreamReader(in));
String firstLineOfResponse = reader.readLine();//HTTP/1.1 200 OK
String secondLineOfResponse = reader.readLine();//Content-Type:text/html
String threeLineOfResponse = reader.readLine();//Content-Length:
String fourLineOfResponse = reader.readLine();//blank line
//读取响应数据，保存文件
//success
byte[] b = new byte[1024];
OutputStream out = new FileOutputStream(savelocation+"/"+filename);
int len = in.read(b);
while(len!=-1)
{
    out.write(b, 0, len);
    len = in.read(b);
}
in.close();
out.close();
```

响应失败（状态码 404） - 将响应信息打印在控制台上

```
//output error message
StringBuffer result = new StringBuffer();
String line;
while ((line = reader.readLine()) != null) {
        result.append(line);
}
reader.close();
System.out.println(result);
```

（五） 实验提交内容

提交整个 HTTP 客户端工程

提交整个 HTTP 服务器端工程

（六） 实验思考题

对比 HTTP1.0与HTTP1.1的差异

参考 http://www.ietf.org/rfc/rfc1945.txt RFC 1945 (HTTP/1.0)

参考 http://www.ietf.org/rfc/rfc2068.txt RFC 2068 (HTTP/1.1)

实验 3 使用 WebSocket 编写聊天室（12 学时）

（一） 实验目的

（1） 理解WebSocket的通信原理；

（2） 使用tomcat服务器搭建多人聊天室；

（二） 实验内容

本实验要求完成以下功能：

功能名称	功能说明	优先级
多人聊天	在网页上实现多人聊天	高
跟某个人聊天	用户登录，显示在线列表，指定和某个人聊天	中
在线客服	在线客服功能	低

（三） 预习内容

(1)WebSocket概念

现很多网站为了实现即时通讯，所用的技术都是轮询(polling)。轮询是在特定的的时间间隔（如每1秒），由浏览器对服务器发出HTTP request，然后由服务器返回最新的数据给客户端的浏览器。这种传统的HTTP request 的模式带来很明显的缺点——浏览器需要不断的向服务器发出请求，然而HTTP request 的header是非常长的，里面包含的有用数据可能只是一个很小的值，这样会占用很多的带宽。

在 WebSocket API，浏览器和服务器只需要做一个握手的动作，然后，浏览器和服务器之间就形成了一条快速通道。两者之间就直接可以数据互相传送。在此 WebSocket 协议中，为我们实现即时服务带来了两大好处：

1. Header

互相沟通的 Header 是很小的-大概只有 2 Bytes

2. Server Push

服务器的推送，服务器不再被动的接收到浏览器的 request 之后才返回数据，而是在有新数据时就主动推送给浏览器。

浏览器请求

GET /webfin/websocket/ HTTP/1.1
Host: localhost
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: xqBt3ImNzJbYqRINxEFlkg==
Origin: http://服务器地址
Sec-WebSocket-Version: 13

服务器回应

HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: K7DJLdLooIwIG/MOpvWFB3y3FE8=

实现 WebSocket 的浏览器

Chrome	Supported in version 4+
Firefox	Supported in version 4+

Internet Explorer	Supported in version 10+
Opera	Supported in version 10+
Safari	Supported in version 5+

实现了 WebSocket 的服务器

jetty 7.0.1 版本

resin

pywebsocket, apache http server 扩展

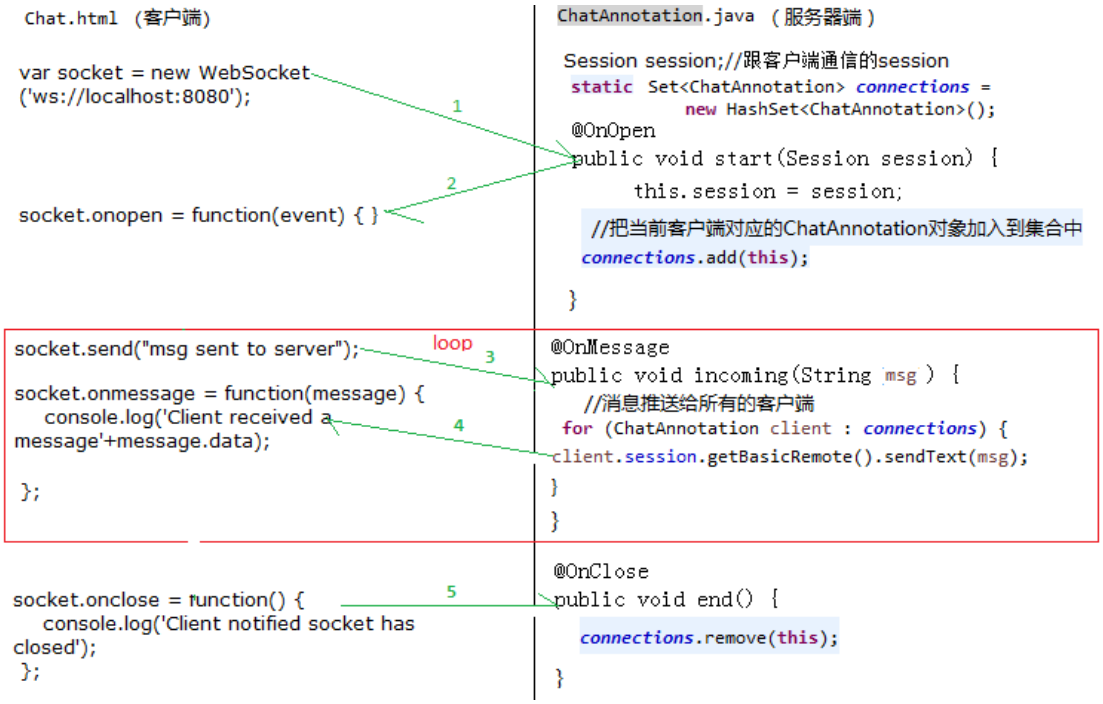
apache tomcat 7.0.27 版本

Nginx 1.3.13 版本

jWebSocket java 实现版

值得注意的是服务器端没有标准的 api, 各个实现都有自己的一套 api,所以使用 websocket 开发服务器端有一定的风险.可能会被锁定在某个平台上或者将来被迫升级.

(2)WebSocket通信规则



1) 客户端发起请求，服务器端创建一个 ChatAnnotation 对象，自动运行 start 方法，将 Websocket 会话对象 session 作为入参。服务器端将当前 ChatAnnotation 对象加入到 static 的集合中。
2) 客户端 onopen 方法自动运行。
3) 客户端通过事件触发 socket.send()方法，服务器端 incoming 方法自动运行，客户端发送的信息自动作为方法入参，服务器端可以遍历 static 的集合，把消息推送给所有的客户端。
4) 收到推送消息后，客户端的 onmessage 方法自动运行。
5) 客户端如果推出，客户端的 onclose 方法自动运行，接着服务器端的 end 方法自动运行，在 end 方法中，从 static 的集合中删除当前 ChatAnnotation 对象。

（四） 实验基本步骤

(1) 编写客户端代码：

1） 客户端发起请求，考虑浏览器兼容性：

```
Var host = “ws://localhost:8080/ProjectName/ChatServlet”;
if ('WebSocket' in window) {
    Chat.socket = new WebSocket(host);
} else if ('MozWebSocket' in window) {
    Chat.socket = new MozWebSocket(host);
}
```

2） 在客户端的输入框 chat 中添加回车事件，在回车事件中给服务器发送消息。

```
document.getElementById('chat').onkeydown = function(event) {
 if (event.keyCode == 13) {
        socket.send(document.getElementById('chat').value);
  }
};
```

3） 监听服务器端的反馈，有消息时打印消息在文本域 console 中。

```
socket.onmessage = function (message) {
          var console = document.getElementById('console');
          var p = document.createElement('p');
          p.innerHTML = message.data;
          console.appendChild(p);
          };
```

(2)编写服务器端代码

1） 编写 start 方法（@OnOpen）,将当前服务器端对象放入集合，通知所有客户端有新客户端连接。

```
@OnOpen
public void start(Session session) {
    this.session = session;
    connections.add(this);
    String message = String.format("* %s %s", nickname, "has joined.");
    broadcast(message);
}
```

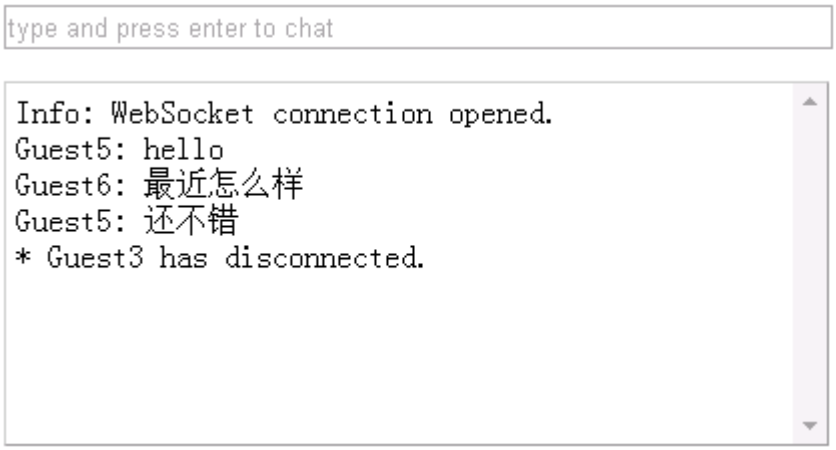
2） 编写 incoming 方法（@OnMessage），接收客户端消息，向其他客户端广播。

```
@OnMessage
public void incoming(String message) {
    broadcast(message);
}
```

3） 编写 end 方法（@OnClose）,移除当前服务器端对象。

```
@OnClose
public void end() {
    connections.remove(this);
}
```

(3)启动 tomcat,在浏览器中运行：



（五） 实验提交内容

提交多人聊天项目

（六） 实验思考题

WebSocket相对于Http协议的优势在哪？

实验 4 基于 RMI 的分布式议程服务（选做）

(一)实验目的

（1） 理解并掌握分布式系统的基本概念；

（2） 掌握分布式系统应用程序的开发方法；

（3） 掌握 RMI工作原理；

（4） 能够用 Java语言的RMI机制，编写RMI程序；

（二）实验内容

（1） 编写 RMI程序

用 RMI 构建一个分布式共享会议议程服务。不同的客户应该能够使用共享会议议程服务，该服务提供会议的查询、增加和删除功能。会议议程服务器有允许用户注册和撤销会议的功能。

- 用户注册功能：新的用户必须注册，注册时必须提供一个用户名和一个密码。如果新用户提供的用户名已经被其他人使用了，要求提示一个出错信息。如果新用户注册成功，也要输出一个提示信息。

register [username] [password]

- 增加会议：注册用户可以在他自己的会议议程上增加会议。会议必须在两个注册用户之间召开。一个注册用户如果没有其他可用的注册用户将不能召开会议。增加会议时，需要提供会议的开始和结束时间、会议的标注、召开会议的用户名称。会议增加后，该会议要在增加会议的用户的会议议程中显示，同时要在另一个参加会议的用户的会议议程中显示。如果一个用户已有的会议和新增的会议冲突，应该提示出错信息，同时该会议将不能被增加到会议议程中。不管会议能否被增加，程序都要给用户提示信息。

add [username] [password] [otherusername] [start] [end] [title]

- 查询会议：注册的用户如果给定一个时间间隔可以查询自己在某个时间段的会议议程上的所有会议（既包括注册用户召开的会议，也包括注册用户被邀请参加的会议）。查询会议提供的参数包括查询时间段的起始和结束时间。查询结束后，输出在特定时间段内所有会议实体列表。该列表包括会议的开始时间、结束时间、会议的标注、召开会议的用户的名字。

query [username] [password] [start] [end]

- 删除会议：注册用户可以删除已经创建的会议。删除会议应该提供的参数包含删除会议的执行用户和唯一能够标识会议的标识符。

delete [username] [password] [meetingid]

- 清除会议：注册用户清除自己召开的所有的会议。

clear [username] [password]

本实验不要求实现服务端对象的持久性。学有余力的同学可以在完成本实验的规定项目后，尝试以关系数据库或文件系统实现服务端对象的持久性。

（三）预习内容

（1） 复习 Java网络编程相关知识；

（2） 复习 RMI机制；

（四）实验基本步骤

【RMI（Remote Method Invocation，远程函数调用）的概念】

想要跨越网络，在其他机器上执行程序，传统做法似乎容易让人混淆，不仅恼人而且实现上容易发生问题。思考这个问题的最好方式，便是通过某种存活于其他机器上的对象，使你可以发送消息给远端对象，并取得其执行结果，整个过程就好像该对象存活于本机上一样。

这种简化手法，正是 RMI允许你做的方式。

用 RMI编写一个分布式应用，核心有以下三方面：

（1） 定位远程对象

1. 一个应用可以利用 RMI的名字服务功能注册器远程对象。

2. 可以象操作普通对象一样传送并返回一个远程对象的引用(指针)。

（2） 与远程对象通信

底层的通信由 RMI实现，对于系统开发人员来说，远程调用和标准的 Java方法调用没有什么区别。

（3）为需要传递的对象装载类的字节码

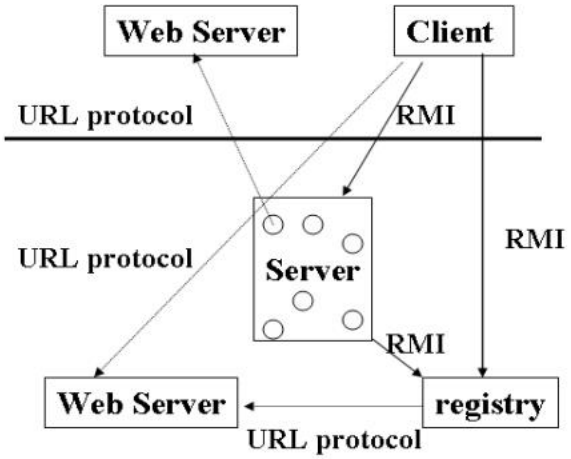
RMI允许调用者向远程对象传递一个对象，因此RMI提供这种装载对象的机制。

【RMI（Remote Method Invocation，远程函数调用）的工作过程】

使用 RMI通信的系统一般分为两类客户端和服务器，服务器提供 RMI服务，客户端调用服务对象的方法。RMI服务器必须通过 loopup服务注册，这样客户才能找到RMI服务器提供的服务。一旦 RMI服务器注册，该服务器就等待客户端发送的 RMI请求。

RMI客户发送 RMI消息调用远端的对象的方法，在调用任何远端对象之前，客户端必须有一个远端对象的索引,一旦获得远端对象的索引,客户端就可以和远端服务交互了。

客户/服务器 RMI交互过程如图所示。



客户端和远端服务的交互是通过 RMI系统中的 stub和 skeleton对象来实现的。Stub对象像一个代理对象，传递客户请求给 RMI服务器。skeleton对象负责监听传递过来的RMI请求并且把这些请求传递给 RMI服务。stub 对象和 skeleton 对象之间的通信通过TCP的 Socket进行。

【编写 RMI程序步骤】

（1） 定义 RMI服务接口

扩展 java.rmi.Remote接口，通过该接口提供远程客户可以访问的方法

例如：

```
public class RMILightBulb extends java.rmi.Remote{
        public void on() throws java.rmi.RemoteException;
        public void off() throws java.rmi.RemoteException;
        public Boolean isOn() throws java.rmi.RemouteException;
}
```

2） 实现 RMI服务接口

例如：

```
public class RMILightBulbImpl
extends java.rmi.server.UnicastRemoteObject
implements RMILightBulb{
        public RMILightBulbImpl() throws java.rmi.RemoteException{
                setBulb(false);
        }
        private Boolean lightOn;
        public void on() throws java.rmi.RemoteException{
                setBulb(true);
        }
        public boolean off() throws java.rmi.RemoteException{
                setBulb(false);
        }
        public Boolean isOn() throws java.rmi.RemouteException{
                return getBulb();
        }
        public void setBulb(boolean value){
                lightOn=value;
        }
        public Boolean getBulb(){
                return lightOn;
        }
}
```

（3） 创建 Stub和 Skeleton类

Stub类和 Skeleton类分别负责分发和处理 RMI请求，但是开发者不能创建这些类。一旦远程服务存在，可以使用 rmic工具来创建 Stub类和 Skeleton类。

例如：

```
rmic RMILightBulbImpl
```

这样就会产生如下文件：

RMILightBulbImpl_Stub.class

RMILightBulbImpl_Skeleton.class

（4） 创建 RMI服务器程序

RMI服务器负责产生RMI服务的实例。

例如：

```
import java.rmi.*;
import java.rmi.server.*;
public class LightBulbServer{
        public static void main(String args[]){
                System.out.println("Loading RMI service" );
                try{
                        RMILightBulbImpl bulbService=new RMILightBulbImpl();
                        RemoteRef location=bulbService.getRef();
                        System.out.println(location.remoteToString());
                        String registry="localhost";
                        if(args.length>=1){
                                registry=args[0];
                        }
                        String registration="rmi://"+registry+"/RMILightBulb";
                        Naming.rebind(registration,bulbService);
                }catch(RemoteException re){
                        System.err.println("Remote Error - "+re);
                }catch(Exception e){
                        System.err.println("Error - "+e);
                }
        }
}
```

（5） 创建 RMI客户端程序

```
import java.rmi.*;
public class LightBulbClient{
        public static void main(String args[]){
                System.out.println("Looking for light bulb service" );
                try{
                        String registry="localhost";
                        if(args.length>=1){
                                registry=args[0];
                        }
                        String registration="rmi://"+registry+"/RMILightBulb";
                        Remote remoteService=Naming.lookup(registration);
                        RMILightBulb bulbService=(RMILightBulb)remoteService;
                        System.out.println("Invoking bulbservice.on()");
                        bulbService.on();
                        System.out.println("Bulb state :+ "+bulbService.isOn());
                }catch(NotBoundException nbe){
                        System.out.pirntln("No light bulb service available in registry!");
                }catch(RemoteException re){
                        System.err.println("Remote Error - "+re);
                }catch(Exception e){
                        System.err.println("Error - "+e);
                }
        }
}
```

【RMI程序运行步骤】

（1） 编译成功所有 java文件

（2） 调用 rmiregistry命令

（3） 在独立的控制台上，运行服务器程序

例如：

java LightBulbServer hostname

（4） 在独立的控制台上，运行客户端程序

（五）实验提交内容

1．应提交内容：

所有必要的能够运行会议议程服务的文件

readme.txt——每个程序编译和执行的指令

2．将程序的实现过程写在实验报告中，并提交实践报告。（可以利用 UML的类图，对象图，顺序图，用例图，状态图或结构化的程序流程图来描述程序的实现过程，可列出程序实现的关键源代码）。

（六）实验思考题

思考编写 RMI程序的步骤。