课程设计——Java 实现 Web Server

霰佳铭 2017141491010

一、Requirements:

1、The web server should response the http request from user agent correctly.

实现结果：Web Server 可以正常响应 http 请求，将请求报文输出在控制台，能够正确发回 http 响应报文与请求文件。

2、The web server should be capable to serve at least 2 http user agent simultaneously.

实现结果：采用 Java 多线程，可以同时响应多个用户的访问

3、Users can set up the working port of this web server.

实现结果：用户可以手动设置 Web Server

二、开发工具&环境

操作系统：Window 10 家庭版 64Bit
处理器：Intel Core i7-7700HQ CPu @ 2.80GHz 2.80GHz
内存(RAM)：8.00GB

编辑器：IntellJ IDEA 2019.1.3 x64
开发语言：java
Java SE 运行环境：1.8.0_221-b11
Java JDK：1.8.0_221
JVM 虚拟机：HotSpot 64-Bit Server VM(25.211-b11,mixed mode)

三、设计模型

三、代码：

1. WebServer 类

```
package httpServer;
import java.net.ServerSocket;
import java.net.Socket;
/*
 * @author Xianjiaming
 * @StudentID 2017141491010
 * @version v20191123
 * */
public class WebServer {
    public static void main(String[] args){
        int port = 12345;        //用户可以自己设定 Server 的端口号
        ServerSocket server;    //Server 的套接字
        Socket client;            //Client 的套接字
        try{
            //以端口号作为参数，建立 Server 套接字的对象
            server = new ServerSocket(port);
            System.out.println("The WebServer is listening on port "+server.getLocalPort());
            while(true){
                //server 监听等待连接，一旦有连接便创建 Socket 实例，client 引用到该对象
                client = server.accept();
                //启动通信线程
                new communicateThread(client).start();
            }
        }
        catch(Exception e){
            System.out.println(e.getMessage());
        }
    }
}
```

重要函数：

server = new ServerSocket(port);//以端口号作为参数，建立 Server 套接字的对象

client = server.accept();//server 监听等待连接，一旦有连接便创建 Socket 实例，client 引用到该对象

new communicateThread(client).start();//启动通信线程

2. conmmunicateThread 类

```
package httpServer;
import java.io.*;
import java.net.Socket;
/*
* @author Xianjiaming
* @StudentID 2017141491010
* @version v20191123
* */
public class communicateThread extends Thread {
    private static Object lock = new Object(); //锁
        private Socket client; //客户端的套接字
        //构造函数
        public communicateThread(Socket s){
            client = s;
        }
        //获取请求报文
        private String getRequest(InputStream in){
        StringBuffer request = new StringBuffer();
        byte[] buffer = new byte[1024];
        //将用户发送的 Http 请求报文写进 buffer，返回值是 buffer 的有效长度
        int len;
        try{
            len = in.read(buffer);
        }
        catch(IOException e){
            e.printStackTrace();
            len = -1;
        }
        for(int i = 0; i < len; i++){
            request.append((char)buffer[i]);
        }
        System.out.print(request.toString());
        return request.toString();
    }
     //获取请求文件名称
    private String getResourcePath(String s){
        //结合 HTTP 请求报文来看，第一行为请求方法+请求 URL+HTTP 协议版本(以空格分隔)，故第一行第二个位置为请求 URL
```

```
        int index1,index2;
        String resourcePath = null;
        index1 = s.indexOf(" ");
        if(index1!=-1){
            index2 = s.indexOf(" ",index1+1);
            if(index2>index1){
                //提取出请求文件的名称，在第一个空格和第二个空格之间(去除'/')
                resourcePath = s.substring(index1+2,index2);
                //默认加载主页
                if(resourcePath.equals("")){
                    resourcePath = "index.html";
                }
            }
        }
        return resourcePath;
    }
     //把响应报文以及请求的文件写进输出流
     private void sendResponse(String fileName,OutputStream out) throws IOException {
        File file = null;
        FileInputStream fis = null;
        byte buf[] = new byte[1024];
        try {
            file = new File(fileName);
            if (file.exists()) {
                fis = new FileInputStream(file);
                //将响应报文头写进输出流
                out.write("HTTP/1.1 200 OK\n".getBytes());
                out.write("Content-Type: text/html; charset=UTF-8\n\n".getBytes());
                //将请求的文件写进输出流
                int readLength = fis.read(buf);
                if(readLength > 0){
                    out.write(buf,0,readLength);
                }
            }
            else {
                System.out.println("404 Not Found");
                System.out.println("请检查路径是否正确!\n");
                String errMsg = "HTTP/1.1 404 Not Found\r\n"+
                        "Content-Type:text/html\r\n"+
                        "Content-Length:23\r\n"+
                        "\r\n"+
                        "<h1>File Not Found</h1>";
                out.write(errMsg.getBytes());
            }
        }
```

```
catch(Exception e){
                    e.getMessage();
            }
            finally {
                //关闭 FileInputStream
                if(fis!=null) {
                    fis.close();
                }
            }
        }
        //运行通信线程
        public void run(){
            InputStream in = null;
            OutputStream out = null;
            try {
                //read 方法是阻塞方法，后续线程会被阻塞，而 Socket 的输入流没有明显的结束语句，
                //直到输出流被关闭，后续进程才能继续通行
                synchronized (lock) {
                    in = client.getInputStream();
                    out = client.getOutputStream();
                    //获取 Http 请求报文
                    String request = getRequest(in);
                    //获得请求的资源路径
                    String fileName = getResourcePath(request);
                    //发送 Http 响应报文
                    sendResponse(fileName, out);
                    //关闭输入输出流
                    out.flush();
                    out.close();
                       in.close();
                }
            }
            catch(Exception e){
                e.printStackTrace();
            }
                finally {
                    try{
                        client.close();
                    }
                    catch (Exception e){
                        e.printStackTrace();
                    }
                }
                    }
                }
```

重要函数：

Private String getRequest(InputStream in):获取请求报文

调用了 InputStream 类的 read(byte[],off,len)方法，将请求报文写入 byte[] buffer。

创建了 StringBuffer 类对象，将 buffer 中的字符遍历并写入 StringBuffer。使用 StringBuffer 既可以连续写入，获得更高的效率，同时可以保证线程安全。

最后调用 toString()方法，以字符串形式返回请求报文

private String getResourcePath(String s)：获取请求文件的名称

图 15-4　HTTP 请求报文

根据 HTTP 请求报文的格式，可以看到，第一行为请求方法+请求 URL+HTTP 协议版本(以空格分隔)，故第一行第二个位置为请求 URL。

在第一个空格和第二个空格之间(去除'/')，提取出请求文件的名称，同时默认加载主页

private void sendResponse(String fileName,OutputStream out)：把响应报文以及请求的文件写进输出流

创建 File 类文件，参数为 String fileName

调用 OutputStream 类 write()方法，将响应报文头写进输出流。

若当文件存在，创建 FileInputStream 类对象，调用 read()方法，将 file 写进 buf 中，再通过 write()方法将 buf 写进输出流。

若当文件不存在，返回 404 Not Found 报文。

最后关闭 FileInputStream 对象。

public void run()：重写 Thread 父类的 run()方法

运行通信线程。按照流程，先获取 http 请求报文，获得请求的资源路径，再发送 http 响应报文，最后关闭 IO 流以及套接字 client。依次调用 getRequest()、getResourcePath()、sendResponce()、close()。

这里要加锁的原因：InputStream 类 read 方法是阻塞方法，并且 Socket 的输入流没有明显的结束语句，所以第一个线程直到关闭 IO 流之前，后续线程会被阻塞。但是这时如果第二个线程阻塞在 read 语句上，由于访问的是 InputStream 类同一对象，第一个线程已经读到了末尾，第二个线程只能读到 null，导致报文被吞。加上锁之后，保证从获取请求报文到返回响应报文这一系列操作具有原子性，针对小文件具有可行性；如果是较大的文件，可能导致访问过慢。更好的方法可能是使用 NIO 类。

3. index.html

```
<!DOCTYPE html>
<html xmlns:font-family="http://www.w3.org/1999/xhtml">
<head>
    <meta charset="UTF-8">
    <title>Web Server</title>
</head>
<body>
    <h1 style="text-align:center;font-size:30px;">计算机网络 Course Project<hr></h1>
    <p style="text-align:center;font-family:arial;font-size:20px;">Java 实现 Web Server</p>
    <p style="text-align:center;font-family:arial;font-size:20px;">霰佳铭 2017141491010</p>
</body>
</html>
```

四、运行结果：

先运行 java 程序，打开浏览器在浏览器上输入 localhost:12345/index.html
浏览器可以正确返回主页页面

控制台上打印出 http 请求报文

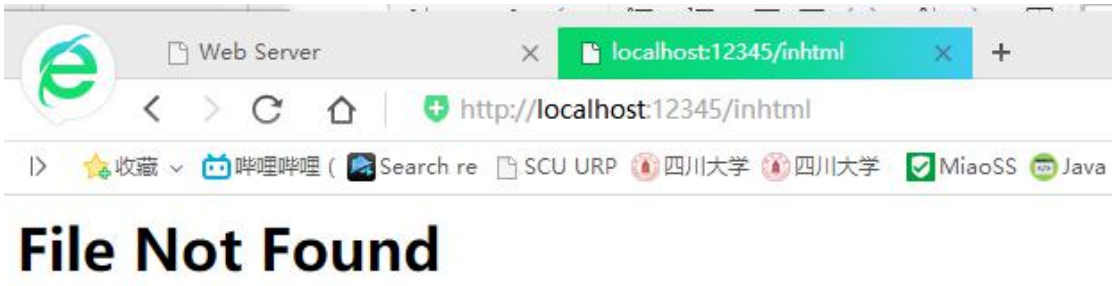
```
"C:\Program Files\Java\jdk1.8.0_221\bin\java.exe" ...
The WebServer is listening on port 12345
GET /index.html HTTP/1.1
Host: localhost:12345
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/63.0.3239.132 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
Accept-Encoding: gzip, deflate, br
Accept-Language: zh-CN,zh;q=0.9
Cookie: __guid=111872281.544542865798384200.1574516096628.5134; monitor_count=414
```

如果输入 localhost:12345/，会默认访问主页

如果输入不存在的地址 如 localhost:12345/inhtml，会返回 404 Not Found 响应报文，同时显示错误页面 (某些浏览器会请求 favicon.ico，即地址栏左侧小图标，在这里因没有图标文件，所以应当返回 404 Not Found)

```
GET /inhtml HTTP/1.1
Host: localhost:12345
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/63.0.3239.132 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
Accept-Encoding: gzip, deflate, br
Accept-Language: zh-CN,zh;q=0.9
Cookie: __guid=111872281.544542865798384200.1574516096628.5134; monitor_count=706

404 Not Found
请检查路径是否正确!
```



可以实现多个用户同时访问 Server，实现并发

五、提交 jar 包：

一个 index.html 文件和一个可运行.jar 文件
在 Windows 下可以通过命令行，先切换到当前目录，
然后输入命令：java -jar Web_Server.jar

```
E:\IdeaProjects\Web Server\out\artifacts\Web_Server_jar>java -jar Web_Server.jar
The WebServer is listening on port 12345
GET /favicon.ico HTTP/1.1
Host: localhost:12345
Connection: keep-alive
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Ch
36
Accept-Encoding: gzip, deflate, br
Accept-Language: zh-CN,zh;q=0.9
Cookie: __guid=111872281.5445428657983842000.1574516096628.5134; monitor_count=788
```

（某些浏览器会请求 favicon.ico，即地址栏左侧小图标，在这里因没有图标文件，所以应当返回 404 Not Found）

打开浏览器输入 localhost:12345 即可实现访问页面

同时提交源代码。