**武汉大学国际软件学院**

**实验报告**

**课程名称 网络及分布式计算实验**

**专业年级 2014级软件工程**

**姓 名**

**学 号**

**实验学期 2016-2017　 学年 第1 学期**

**课堂时数 课外时数**

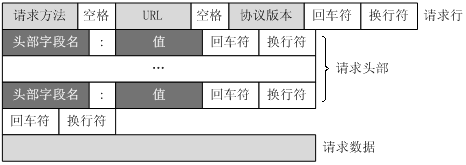
**填写时间 2017 年 5 月 12 日**

**实验任务一**

1. **首先来看一下http的报文结构**

**1、请求报文**

一个HTTP请求报文由请求行（request line）、请求头部（header）、空行和请求数据4个部分组成，下图给出了请求报文的一般格式。



**(1)请求行**

请求行由请求方法字段、URL字段和HTTP协议版本字段3个字段组成，它们用空格分隔。例如，GET /index.html HTTP/1.1。

HTTP协议的请求方法有GET、POST、HEAD、PUT、DELETE、OPTIONS、TRACE、CONNECT。这里介绍最常用的GET方法和POST方法。

GET：当客户端要从服务器中读取文档时，使用GET方法。GET方法要求服务器将URL定位的资源放在响应报文的数据部分，回送给客户端。使用GET方法时，请求参数和对应的值附加在URL后面，利用一个问号（“?”）代表URL的结尾与请求参数的开始，传递参数长度受限制。例如，/index.jsp?id=100&op=bind。

POST：当客户端给服务器提供信息较多时可以使用POST方法。POST方法将请求参数封装在HTTP请求数据中，以名称/值的形式出现，可以传输大量数据。

**(2)请求头部**

请求头部由关键字/值对组成，每行一对，关键字和值用英文冒号“:”分隔。请求头部通知服务器有关于客户端请求的信息，典型的请求头有：

User-Agent：产生请求的浏览器类型。

Accept：客户端可识别的内容类型列表。

Host：请求的主机名，允许多个域名同处一个IP地址，即虚拟主机。

**(3)空行**

最后一个请求头之后是一个空行，发送回车符和换行符，通知服务器以下不再有请求头。

(**4)请求数据**

请求数据不在GET方法中使用，而是在POST方法中使用。POST方法适用于需要客户填写表单的场合。与请求数据相关的最常使用的请求头是Content-Type和Content-Length。

**2、响应报文**

响应报文的格式大体上和请求报文类似，只是第一行有所不同，读者可以自己在网上查找这方面的介绍，这里不再赘述。

**二、程序的实现**

一个简单web服务器的实现原理，它本身只能处理某个目录下的静态资源文件（文本、图片等）。采用**[Java](http://lib.csdn.net/base/javase" \t "_blank" \o "Java SE知识库)**来实现的话，可以含以下几个模块，而且各个模块间的关系如图1所示。

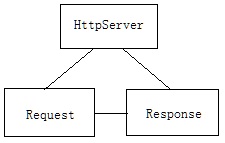


图1、简单web服务器的模块

* WebServer即为服务器，它用于服务器的启动及接收用户的请求并返回结果；
* Request：对用户请求进行分析，解析请求串，并获取对应的访问url；
* Response：根据用户请求生成响应结果，并将结果输出给客户端。

下面通过代码来具体的看下该服务器的实现过程。

WebServer是服务器的主实现类，用于关联Request及Response，源代码如下所示：

import java.io.\*;  
import java.net.InetAddress;  
import java.net.ServerSocket;  
import java.net.Socket;  
import java.net.UnknownHostException;  
  
*/\*\*  
 \* 一个简单的web服务器  
 \* 不是什么特别难的  
 \* 以前都是用python写服务器  
 \* 第一次用java  
 \* 所以参考部分API及网络资料  
 \* Created by BG on 2017/4/18.  
 \*/*public class WebServer {  
  
 public static final String *WEB\_ROOT* = System.*getProperty*("user.dir") + File.*separator* + "htmlfile";  
 private static final String *SHUTDOWN\_COMMAND* = "/SHUTDOWN";  
 private boolean shutdown = false;  
  
 public static void main(String[] args) {  
 WebServer server = new WebServer();  
 server.start();  
 }  
  
 //启动服务器，并接收用户请求进行处理  
 public void start() {  
 ServerSocket serverSocket = null;  
 // 常用端口可能会冲突，但是简易服务器用用也无所谓吧  
 int port = 1234;  
 try {  
 serverSocket = new ServerSocket(port, 1, InetAddress.*getByName*("127.0.0.1"));  
 } catch (UnknownHostException e) {  
 e.printStackTrace();  
 System.*exit*(-1);  
 } catch (IOException e) {  
 e.printStackTrace();  
 System.*exit*(-1);  
 }  
  
 //若请求的命令不为SHUTDOWN时，循环处理请求  
 while(!shutdown) {  
  
 Socket socket = null;  
 InputStream input = null;  
 OutputStream output = null;  
  
 try {  
 //创建socket进行请求处理  
 socket = serverSocket.accept();  
 input = socket.getInputStream();  
 output = socket.getOutputStream();  
 //接收请求  
 Request request = new Request(input);  
 request.parser();  
 //处理请求并返回结果  
 Response response = new Response(output);  
 response.setRequest(request);  
 response.sendStaticResource();  
 //关闭socket  
 socket.close();  
 //若请求命令为关闭，则关闭服务器  
 shutdown = request.getUri().equals(*SHUTDOWN\_COMMAND*);  
 } catch (IOException e) {  
 e.printStackTrace();  
 continue;  
 }  
 }  
 }  
}

该类本身是一个应用程序，包含main方法，直接通过java命令即可运行。在运行时，它本身会启动一个ServerSocket类用于监听服务器的某个端口。当接收到的命令不是停止服务器的SHUTDOWN时，它会创建一个Socket套接字，用于接收请求及返回响应结果。

        Request类则用于请求的接收，对于Http协议来讲，通过浏览器向服务器发送请求有一定的格式，其实Request也就是接收这些请求信息，并对其进行分析，抽取出所需的信息，包括cookie、url等。其中http发送的请求包括三部分：

* 请求方法 统一资源标识符 协议/版本
* 请求头
* 请求实体

Request类用于接收socket套接字发送过来的字节流，并按照http协议请求的格式进行解析。对于简单的web服务器而言，我们只需要解析出它的uri即可，这样即可以通过通过文件匹配的方式找到对应的资源。具体的实现代码如下所示：

import java.io.IOException;  
import java.io.InputStream;  
  
*/\*\*  
 \* 接收请求  
 \* Created by BG on 2017/4/18.  
 \*/*public class Request {  
  
 private InputStream input;  
 private String uri;  
  
 public Request(InputStream input) {  
 this.input = input;  
 }  
  
 public void parser() {  
  
 StringBuffer request = new StringBuffer();  
 byte[] buffer = new byte[1024]; // 应该够用了吧  
 int i = 0;  
  
 try {  
 i = input.read(buffer);  
 } catch (IOException e) {  
 e.printStackTrace();  
 i = -1;  
 }  
  
 for(int k = 0; k < i; k++) {  
 request.append((char)buffer[k]);  
 }  
  
 uri = parserUri(request.toString());  
  
 }  
  
 private String parserUri(String requestData) {  
 int index1, index2;  
 index1 = requestData.indexOf(' ');  
 if(index1 != -1) {  
 index2 = requestData.indexOf(' ', index1 + 1);  
 if(index2 > index1) {  
 return requestData.substring(index1 + 1, index2);  
 }  
 }  
 return null;  
 }  
  
 public String getUri() {  
 return uri;  
 }  
}

通过上述代码可以看出parser方法用于解析具体请求，它接收socket的字节流，对其进行处理，获取其中的uri信息。

        当获取到uri信息后，即可将uri对应到服务器的某个应用或目录中。本文只是实现了一个简单的静态资源服务器，即将uri对应到某个目录下的文件，若文件存在则打开并读取文件信息; 若不存在则直接返回一段错误信息。Response类即用于处理该逻辑，同时它会将文件流写回至socket套接字中，由socket套接字将响应结果返回给客户端。

        对于http协议而言，响应也是有一定的格式要求的，不能发送任意格式的信息，否则浏览器是无法接收并处理的。Http响应结果也包括三部分：

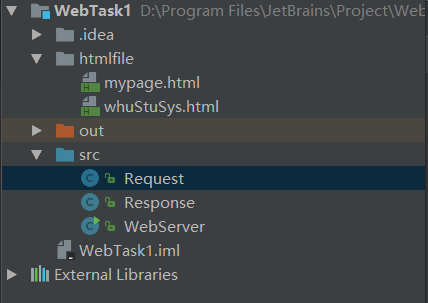
* 协议 状态码 描述
* 响应头
* 响应实体段

Response类中当访问请求的文件不存在时，需要发送一段固定的响应文本给客户端，该段响应文档的格式必须严格按照http响应格式进行组织，否则客户端接收不到。具体的实现源码如下所示：

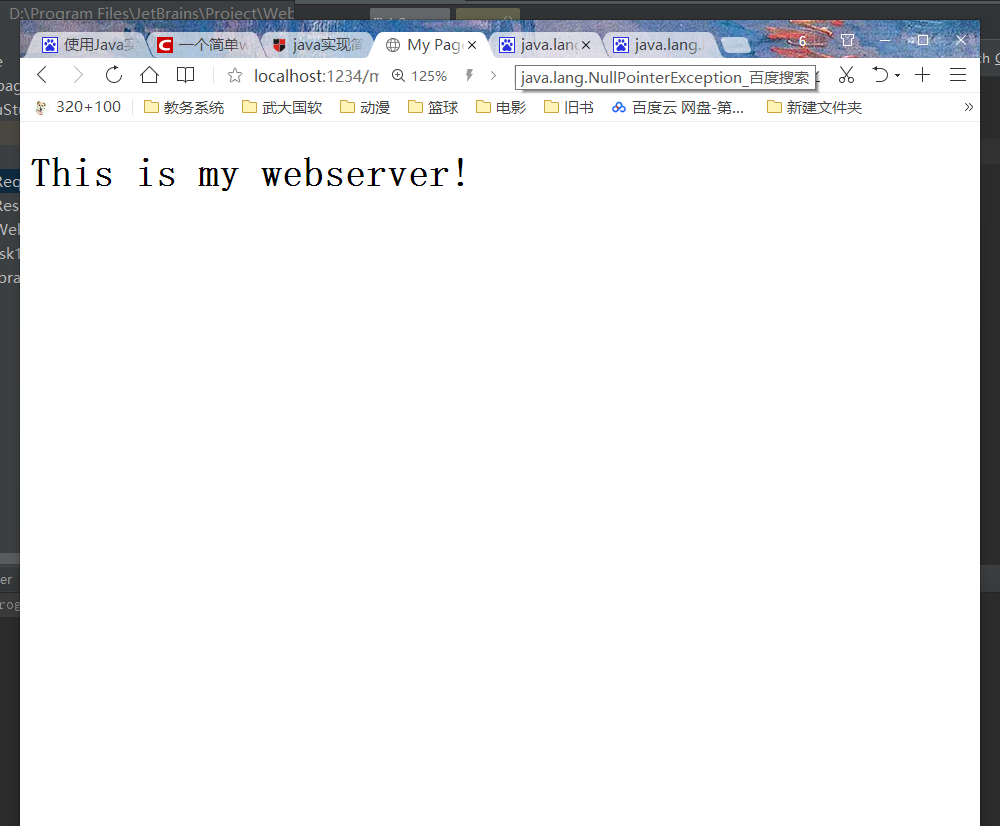
import java.io.File;  
import java.io.FileInputStream;  
import java.io.IOException;  
import java.io.OutputStream;  
  
*/\*\*  
 \* Created by BG on 2017/4/18.  
 \*/*public class Response {  
  
 private OutputStream output;  
 private Request request;  
 private static final int *BUFFER\_SIZE* = 1024;  
  
 public Response(OutputStream output) {  
 this.output = output;  
 }  
  
 public void setRequest(Request request) {  
 this.request = request;  
 }  
  
 // 发送一个静态资源给客户端，若本地服务器有对应的文件则返回，否则返回404页面  
 public void sendStaticResource() {  
 byte[] buffer = new byte[*BUFFER\_SIZE*];  
 int ch;  
 FileInputStream fis = null;  
 try {  
 File file = new File(WebServer.WEB\_ROOT, request.getUri());  
 if(file.exists()) {  
 fis = new FileInputStream(file);  
 ch = fis.read(buffer);  
 while(ch != -1) {  
 output.write(buffer, 0, ch);  
 ch = fis.read(buffer, 0, *BUFFER\_SIZE*);  
 }  
 } else {  
 String errorMessage = "HTTP/1.1 404 File Not Found \r\n" +  
 "Content-Type: text/html\r\n" +  
 "Content-Length: 24\r\n" +  
 "\r\n" +  
 "<h1>404 Not Found<br></h1>";  
  
 if (request.getUri().equals("/SHUTDOWN")){  
 String endMessage = "HTTP/1.1 Close Server \r\n" +  
 "Content-Type: text/html\r\n" +  
 "Content-Length: 24\r\n" +  
 "\r\n" +  
 "<h1>Close Server<br></h1>";  
  
 output.write(endMessage.getBytes());  
 }else {  
 output.write(errorMessage.getBytes());  
 }  
 }  
 } catch (Exception e) {  
 System.*out*.println(e.toString());  
 } finally {  
 if(fis != null) {  
 try {  
 fis.close();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
 }  
}

分析Response类，sendStaticResource方法中根据请求的uri去目录下查找是否有对应的文件存在，若有则直接读入文件并返回；否则返回一段错误消息给客户端。

当然若要运行该程序，则还需要在该工程对应的目录下创建一个htmlfile目录，其中可以存放一个html文件。

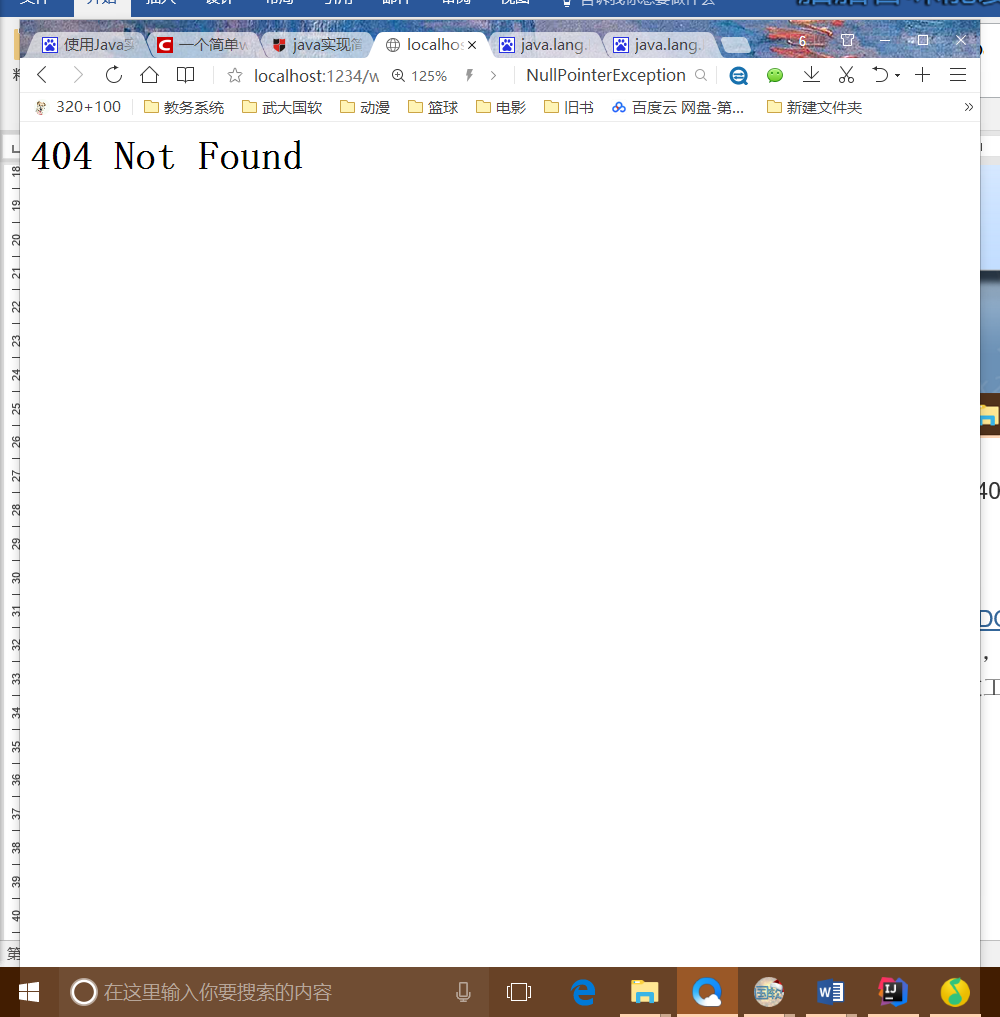


通过浏览器访问url地址：[http://127.0.0.1:8080/\*.htm](http://127.0.0.1:8080/*.htm)l 或 [http://localhost:8080/\*.htm](http://localhost:8080/*.htm)l

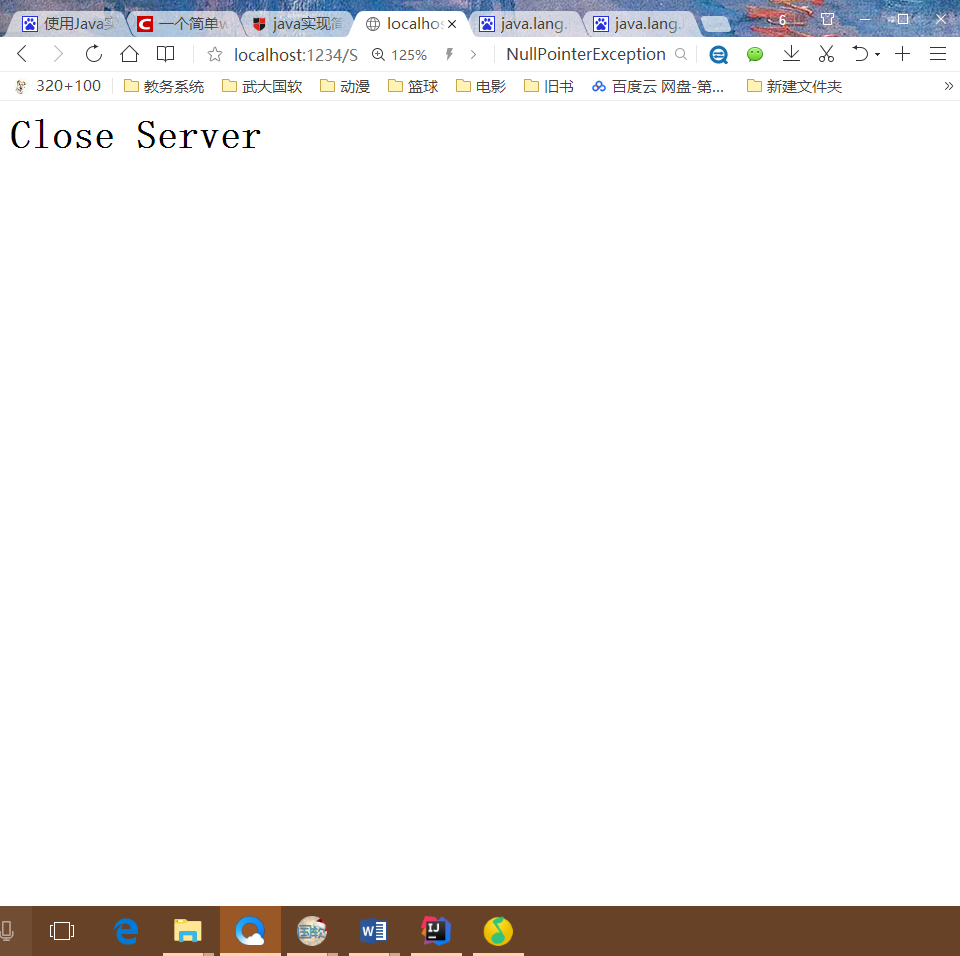




若访问index1.htm由于不存在则直接返回404 File Not Found信息。



可以直接输入<http://127.0.0.1:8080/SHUTDOWN> 即可关闭服务器。



至此一个简单的web服务器就实现完了，它本身的功能不太完整，但对于初学者了解http请求的处理、响应及web服务器的大致工作流是有帮助的。