

华东师范大学数据科学与工程学院期末项目报告

课程名称：计算机网络与编程	年级：21级	上机实践成绩：
指导教师：张召	姓名：彭一坤	学号：10215501412
上机实践名称：实现Web Server	上机实践日期：2023.6.14	
上机实践编号：	组号：	上机实践时间：

一、 题目要求

题目1.1: 请使用Java语言开发一个简单的Web服务器，它能处理HTTP请求。具体而言，你的Web服务器将：

1. 当一个客户端（浏览器）联系时创建一个连接套接字
2. 从这个连接套接字接受HTTP请求
3. 解释该请求以确定所请求的特定文件
4. 从文件系统中获得请求的
5. 创建一个由请求的文件组成的HTTP响应报文
6. 经TCP连接向请求的浏览器返回响应。

功能要求:

- 请使用ServerSocket和Socket进行代码实现；
- 请使用多线程接管连接；
- 在浏览器中输入localhost:8081/index.html能显示自己的学号信息（编写简单的index.html）；
- 在浏览器中输入localhost:8081下其他无效路径，浏览器显示 404 not found；
- 在浏览器中输入localhost:8081/shutdown能使服务器关闭；
- 使用postman进行测试，测试get和post两种请求方法。

题目1.2: 在题目1.1的基础上实现代理服务器，让浏览器请求经过你的代理来请求Web对象。具体而言：

1. 当你的代理服务器从一个浏览器接收到对某个对象的HTTP请求时，它生成对相同对象的一个新的HTTP请求并向初始服务器发送；
2. 当该代理从初始服务器接收到具有该对象的HTTP相应时，它生成一个包括该对象的新的HTTP响应，并发送给该客户。

功能要求:

- 请在题1.1的代码上进行修改，使用ServerSocket和Socket进行代码实现；
- 请分别使用浏览器（可设置浏览器代理）和postman，并进行代理测试。

性能测试

使用JMeter进行压测，在保证功能完整的前提下测试每秒响应的请求数。

Bonos (optional): 分析当前能支持同时连接的最大数，使用学习过的NIO修改代码使服务器能同时支持并发的1000个连接。（注意JMeter中的集合点设置）

二、 功能实现情况

题目1.1

首先，实现一个普通的服务器，可以解析HTTP请求。实现一个SimpleWebServer类，在循环中创建进程，接管客户端的连接。

```
1 public class SimpleWebServer {
2     private static final int PORT = 8081;
3     private ServerSocket serverSocket;
4
5     public static void main(String[] args) {
6         SimpleWebServer server = new SimpleWebServer();
7         server.start();
8     }
9
10    public void start() {
11        try {
12            serverSocket = new ServerSocket(PORT); //创建服务器套接字
13            System.out.println("Server started. Listening at port " + PORT);
14            while (true) {
15                Socket clientSocket = serverSocket.accept(); //处理客户端连接
16                new Thread(new ClientHandler(clientSocket)).start();
17            }
18        } catch (IOException e) {
19            e.printStackTrace();
20        }
21    }
22 }
```

为了处理客户端连接，定义ClientHandler类，实现Runnable接口，表示该类是可运行的，可以被放到新的线程中执行。在该类的构造函数中，会传入一个Socket对象，即当前客户端的套接字，用于之后与客户端进行通信。

在run()方法中，首先通过输入流读取从客户端发送过来的HTTP请求信息，并解析出请求方法、请求资源等相关参数。接着，判断请求方法为GET或POST时，根据请求资源的不同，返回相应的HTTP响应。

如果请求资源是/index.html，则读取本地文件index.html并返回给客户端；如果请求资源是/shutdown，则在页面中显示Server is shutting down...信息，并关闭客户端套接字和服务端程序；其他情况在页面中显示404 not found.信息。如果请求方法不是GET或POST，则返回405 Method Not Allowed信息。最后，关闭输入输出流和客户端套接字，并捕获可能出现的异常。

```
1 class ClientHandler implements Runnable {
2     private Socket clientSocket;
3
4     public ClientHandler(Socket clientSocket) {
5         this.clientSocket = clientSocket;
6     }
7
8     @Override
9     public void run() {
10        try {
11            //解析客户端的请求行
12            BufferedReader in = new BufferedReader(new
13                InputStreamReader(clientSocket.getInputStream()));
14            OutputStream out = clientSocket.getOutputStream();
```

```

14
15     String requestLine = in.readLine();
16     System.out.println(requestLine);
17
18     String[] tokens = requestLine.split(" ");
19     String method = tokens[0].toUpperCase();//方法名
20     String resource = tokens[1];//请求的资源参数
21
22     if (method.equals("GET") || method.equals("POST")) {
23         if (resource.contains("/index.html")) {
24             File file = new
File("D:/Javaprojects/networkFinal/src/index.html");
25             if (file.exists()) {
26                 //文件路径正确，返回状态码200
27                 out.write("HTTP/1.1 200 OK\r\n".getBytes());
28                 out.write("Content-Type: text/html; charset=UTF-
8\r\n".getBytes());
29                 out.write("\r\n".getBytes());
30
31                 FileInputStream fileInputStream = new
FileInputStream(file);
32                 byte[] buffer = new byte[1024];
33                 int len;
34                 while ((len = fileInputStream.read(buffer)) != -1) {
35                     out.write(buffer, 0, len);
36                 }
37                 fileInputStream.close();
38             } else {
39                 //没有找到文件路径
40                 out.write("HTTP/1.1 404 Not Found\r\n".getBytes());
41                 out.write("Content-Type: text/plain; charset=UTF-
8\r\n".getBytes());
42                 out.write("\r\n".getBytes());
43                 out.write("File not found.".getBytes());
44             }
45         } else if (resource.equals("/shutdown")) {
46             //关闭服务器
47             out.write("HTTP/1.1 200 OK\r\n".getBytes());
48             out.write("Content-Type: text/plain; charset=UTF-
8\r\n".getBytes());
49             out.write("\r\n".getBytes());
50             out.write("Server is shutting down...".getBytes());
51             clientSocket.close();
52             System.exit(0);
53         } else { //请求其他资源
54             out.write("HTTP/1.1 404 Not Found\r\n".getBytes());
55             out.write("Content-Type: text/plain; charset=UTF-
8\r\n".getBytes());
56             out.write("\r\n".getBytes());
57             out.write("404 not found.".getBytes());
58         }
59     }
60     else { //使用GET和POST以外的其他方法
61         out.write("HTTP/1.1 405 Method Not Allowed\r\n".getBytes());

```

```

62         out.write("Content-Type: text/plain; charset=UTF-
8\r\n".getBytes());
63         out.write("\r\n".getBytes());
64         out.write("Method not allowed.".getBytes());
65     }
66
67     in.close();
68     out.flush();
69     out.close();
70     clientSocket.close();
71
72     } catch (IOException e) {
73         e.printStackTrace();
74     }
75 }
76 }

```

使用postman测试得到如下结果：

GET http://localhost:8081/ir + ... No Envir

HTTP http://localhost:8081/index.html Save

GET http://localhost:8081/index.html

Params Authorization Headers (7) Body Pre-request Script Tests Settings

Query Params

Key	Value	Description
Key	Value	Description

Body Cookies Headers (1) Test Results Status: 200 OK Time: 6 ms Size: 254 B

Pretty Raw Preview Visualize HTML

```

1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head>
5   <meta charset="UTF-8">
6   <title>学号信息</title>
7 </head>
8
9 <body>
10  <h1 style="text-align:center">10215501412 彭一珊</h1>
11 </body>
12
13 </html>

```

POST http://localhost:8081/

No Environment

http://localhost:8081/index.html

Save

POST

http://localhost:8081/index.html

Send

Params

Authorization

Headers (8)

Body

Pre-request Script

Tests

Settings

Cook

Query Params

	Key	Value	Description	...	Bulk Ed
	Key	Value	Description		

Body

Cookies

Headers (1)

Test Results

Status: 200 OK

Time: 16 ms

Size: 254 B

Save as Example

Pretty

Raw

Preview

Visualize

HTML

1

<!DOCTYPE html>

2

<html lang="en">

3

4

<head>

5

<meta charset="UTF-8">

6

<title>学号信息</title>

7

</head>

8

9

<body>

10

<h1 style="text-align:center">10215501412 彭一琿</h1>

11

</body>

12

13

</html>

localhost:8081/111

Save

GET

localhost:8081/111

Send

Params

Authorization

Headers (7)

Body

Pre-request Script

Tests

Settings

Cookies

Query Params

	Key	Value	Description	...	Bulk Edit
	Key	Value	Description		

Body

Cookies

Headers (1)

Test Results

Status: 404 Not Found

Time: 9 ms

Size: 81 B

Save as Example

Pretty

Raw

Preview

Visualize

Text

1

404 not found.

使用浏览器测试得到如下结果：

localhost:8081/index.html

10215501412 彭一琿

127.0.0.1:8081

404 not found.

127.0.0.1:8081/111

404 not found.

Server is shutting down...

题目1.2

首先定义了一个名为 `ProxyServer` 的类，其中包含了一个私有常量 `PORT`，表示代理服务器监听的端口号。还定义了一个私有成员变量 `serverSocket`，表示服务器的套接字。

在 `main` 方法中，创建了一个 `ProxyServer` 实例 `server`，并调用了其 `start` 方法。在 `start` 方法内部，通过 `ServerSocket` 类创建了一个服务器套接字，并将其绑定到指定的端口号上。然后通过一个无限循环来不断等待客户端连接，每当客户端连接时，就会创建一个新的线程 `Thread` 来处理该客户端请求，具体处理逻辑由 `ProxyClientHandler` 类实现。

异常处理代码 `catch(IOException e)` 会捕获可能抛出的 `IOException` 异常，并打印出错误信息。

```
1 public class ProxyServer {
2     private static final int PORT = 8082;
3     private ServerSocket serverSocket;
4
5     public static void main(String[] args) {
6         ProxyServer server = new ProxyServer();
7         server.start();
8     }
9
10    public void start() {
11        try {
12            serverSocket = new ServerSocket(PORT);
13            System.out.println("Proxy server started. Listening at port " +
14                                PORT);
15            while (true) {
16                Socket clientSocket = serverSocket.accept();
17                new Thread(new ProxyClientHandler(clientSocket)).start();
18            }
19        } catch (IOException e) {
20            e.printStackTrace();
21        }
22    }
23 }
```

随后创建一个 `ProxyClientHandler` 类，通过 `clientSocket.getInputStream()` 获取到客户端发来的请求消息，将其封装为 `BufferedReader` 对象，以便逐行读取并进行解析。然后，通过 `clientSocket.getOutputStream()` 获取客户端的输出流，以便向客户端发送响应消息。创建一些局部变量用于记录请求的内容。

```

1 public void run() {
2     try {
3         InputStream clientIn = clientSocket.getInputStream();
4         BufferedReader in = new BufferedReader(new
InputStreamReader(clientIn));
5         OutputStream out = clientSocket.getOutputStream();
6         StringBuilder head = new StringBuilder();
7         String requestLine;
8         String host = null;
9         String method="";
10        int len;
11        int port = 80;

```

接着，在循环中逐行读取客户端发来的消息，并且检查是否包含请求头 "Host"，如果有，则提取出其中的主机名和端口号，并将其存储在变量 `host` 和 `port` 中。同时，将请求消息头存储在 `head` 变量中，直到读取到空行表示消息头结束。

```

1 while ((requestLine = in.readLine()) != null) {
2     head.append(requestLine).append("\r\n");
3     if (requestLine.length() == 0) {
4         break;
5     }
6     else {
7         String[] temp = requestLine.split(" ");
8         if (temp[0].contains("host") || temp[0].contains("Host")) {
9             host = temp[1];
10            if (host.contains(":")){
11                String[] tmp=host.split(":");
12                host=tmp[0];
13                port= Integer.parseInt(tmp[1]);
14            }
15        }
16    }
17    if (requestLine.trim().isEmpty()) {
18        break;
19    }
20 }
21 head.append("\r\n");
22 try{
23     method = head.substring(0, head.indexOf(" "));
24 }
25 catch(StringIndexOutOfBoundsException e){
26     System.out.println("requestLine=NULL");
27 }

```

然后，根据请求方式判断是否需要建立与目标服务器的连接。如果是 "CONNECT" 方式，则向客户端发送握手成功的消息，此时客户端会发来新的请求，因此调用 `request_handler` 方法，在新线程中处理客户端和目标服务器之间的数据传输。否则，直接将收到的完整请求消息转发到目标服务器。

```

1 Socket serverSocket = new Socket(host, port);
2 System.out.println("连接到服务器: " + serverSocket.getInetAddress() + ":" +
  serverSocket.getPort());
3 OutputStream serverOut = serverSocket.getOutputStream();
4 InputStream serverIn = serverSocket.getInputStream();
5 if (method.equals("CONNECT")) {
6     out.write("HTTP/1.1 200 ConnectionEstablished\r\n\r\n".getBytes());
7     out.flush();
8     new request_handler(clientIn, serverOut).start();
9 } else {
10     serverOut.write(head.toString().getBytes());
11 }

```

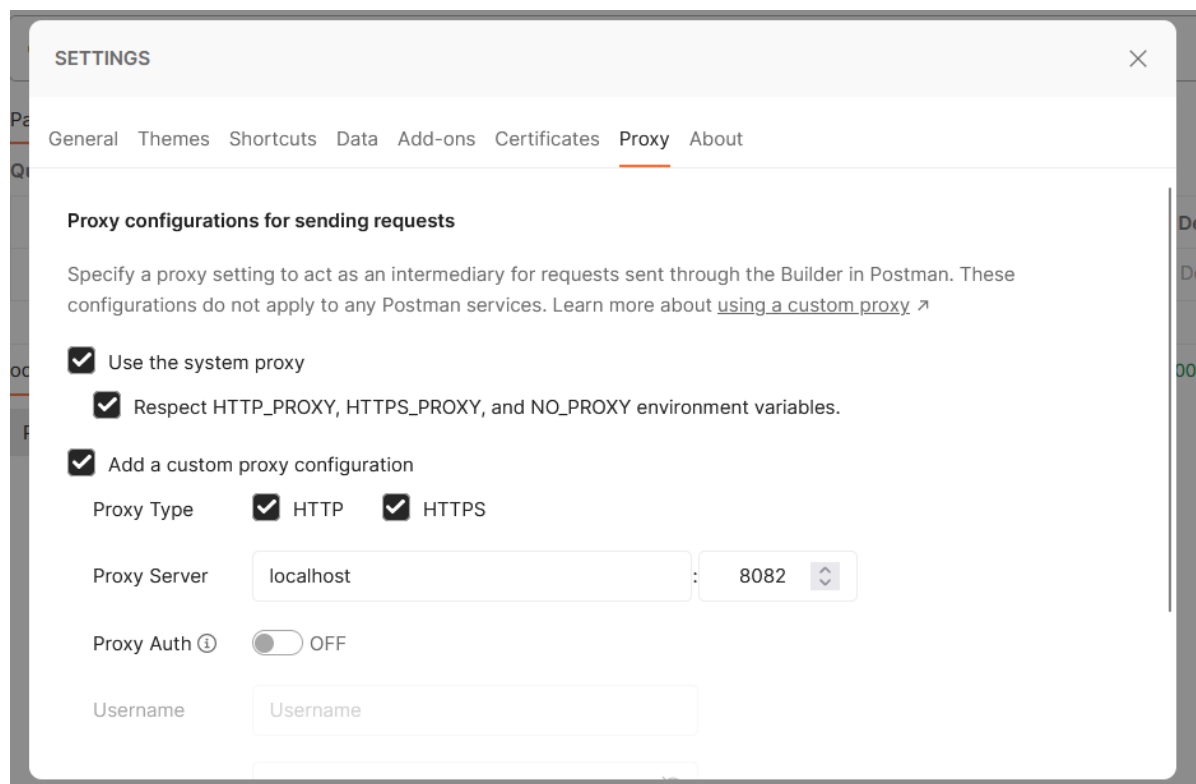
最后，从目标服务器读取响应消息，并将其写入客户端的输出流中。在整个过程结束后，关闭相关的流和套接字。

```

1 byte[] returnBuffer = new byte[102400];
2 while ((len = serverIn.read(returnBuffer)) != -1) {
3     out.write(returnBuffer, 0, len);
4 }
5 in.close();
6 out.flush();
7 out.close();
8 serverIn.close();
9 serverOut.flush();
10 serverOut.close();
11 serverSocket.close();
12 clientSocket.close();

```

然后测试代理服务器的功能。首先开启postman的代理服务，转接实现的web服务器，端口号是8082：



然后打开SimpleWebServer和ProxyServer，对地址index.html进行访问：

GET http://localhost:8081/ir

http://localhost:8081/index.html

GET http://localhost:8081/index.html

Params Authorization Headers (7) Body Pre-request Script Tests Settings

Query Params

Key	Value	Description
Key	Value	Description

Body Cookies Headers (1) Test Results

Status: 200 OK Time: 7 ms Size: 254 B

Pretty Raw Preview Visualize HTML

```
1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head>
5   <meta charset="UTF-8">
6   <title>学号信息</title>
7 </head>
8
9 <body>
10  <h1 style="text-align:center">10215501412 彭一珊</h1>
11 </body>
12
13 </html>
```

访问题目给出的网址：

GET http://www.baidu.com/

http://www.baidu.com/search/error.html

GET http://www.baidu.com/search/error.html

Params Authorization Headers (8) Body Pre-request Script Tests Settings

Query Params

Key	Value	Description
Key	Value	Description

Body Cookies (3) Headers (11) Test Results

Status: 200 OK Time: 88 ms Size: 5.66 KB

Pretty Raw Preview Visualize HTML

```
1 <!DOCTYPE html>
2 <!--STATUS OK-->
3 <html>
4
5 <head>
6   <meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1">
7   <meta http-equiv="content-type" content="text/html; charset=utf-8">
8   <meta content="always" name="referrer">
9   <script src="https://ss1.bdstatic.com/5eN1bjq8AAUYm2zgoY3K/r/www/nocache/imgdata/seErrorRec.js"></script>
10  <title>页面不存在_百度搜索</title>
11  <style data-for="result">
12    body {
13      color: #333;
14      background: #fff;
15      padding: 0;
16      margin: 0;
```

然后设置系统代理进行测试：

编辑代理服务器

使用代理服务器

☒ 开

代理 IP 地址

127.0.0.1

端口

8082

请勿对以下列条目开头的地址使用代理服务器。若有多个条目，请使用英文分号 (;) 来分隔。

☒ 请勿将代理服务器用于本地(Intranet)地址

保存

取消

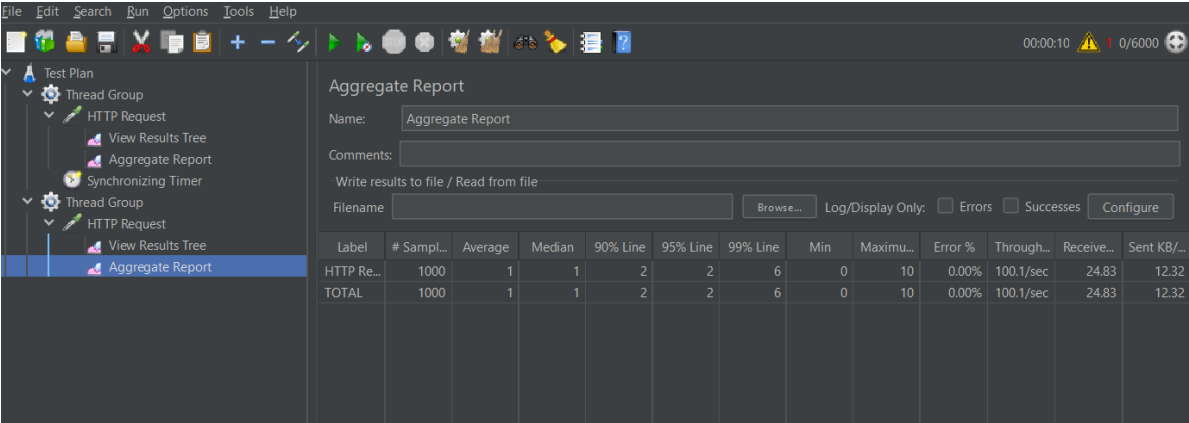
连接代理服务器后，可以正常访问所有网站，

ProxyServer × SimpleWebServer (1) ×

```
"C:\Program Files (x86)\Java\jdk-19\bin\java.exe" "-javaagent:D
Proxy server started. Listening at port 8082
连接到服务器: bizapi.csdn.net/123.249.99.208:443
连接到服务器: g.csdnimg.cn/121.194.10.213:443
连接到服务器: sb.firefox.com.cn/123.56.1.57:443
连接到服务器: download.csdn.net/120.46.209.149:443
连接到服务器: ev.csdn.net/49.4.45.146:443
requestLine=NULL
连接到服务器: localhost/127.0.0.1:80
连接到服务器: csdnimg.cn/121.194.10.213:443
连接到服务器: s3a.pstatp.com/222.192.187.38:443
连接到服务器: zhanzhang.toutiao.com/222.192.187.39:443
连接到服务器: img-home.csdnimg.cn/120.221.143.6:443
```

三、性能测试情况

首先，设置参数为Number of Threads (users)=1000, Ramp-up period=10，访问localhost:8081，得到如下运行结果，可见错误率为0.00%，吞吐率是100.1/sec



四、总结

本次实验旨在通过使用Java语言开发一个简单的Web服务器和代理服务器，来实现处理HTTP请求并进行响应的功能。

在实现过程中，我按照题目要求使用了Socket来处理连接，并通过解析HTTP请求的方式来确定所请求的特定文件。我进一步熟悉了HTTP请求和响应的报文格式，练习了输入输出流读写的编程方式，对java网络编程有了更深刻的理解。

在实现代理服务器部分，我利用了与Web服务器相似的技术来实现代理服务器。当代理服务器接收到客户端请求时，它首先解析请求的头信息，获取到初始服务器的ip地址和端口号，然后使用Socket连接这个服务器。代理服务器需要生成一个相同对象的新的HTTP请求并发送给初始服务器。然后代理服务器再将从初始服务器接收到的HTTP响应返回给客户端。通过这种方式，我实现了代理服务器与Web服务器之间的连接和数据交换，进一步深入了解了代理服务器的工作原理。另外，由于HTTPS协议需要先建立CONNECTION再发送请求，因此代理服务器建立好连接后，要给客户端发送回连接建立成功的回复报文，然后重新接收GET或POST等请求才行。

总之，本次实验不仅让我更加深入地了解了Web服务器和代理服务器的实现原理和工作方式，同时也帮助我提升了Java编程和性能优化的技能，对计算机网络的学习有了更加深入的探索。