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**：杨茜雅	**学号**：10215501435
上机实践名称：期末项目实现 Web Server		**上机实践日期**：2023.6.20
上机实践编号：	**组号**：	**上机实践时间**：2023.6.20

一、 题目要求

项目名称：实现 Web Server

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文件；
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。

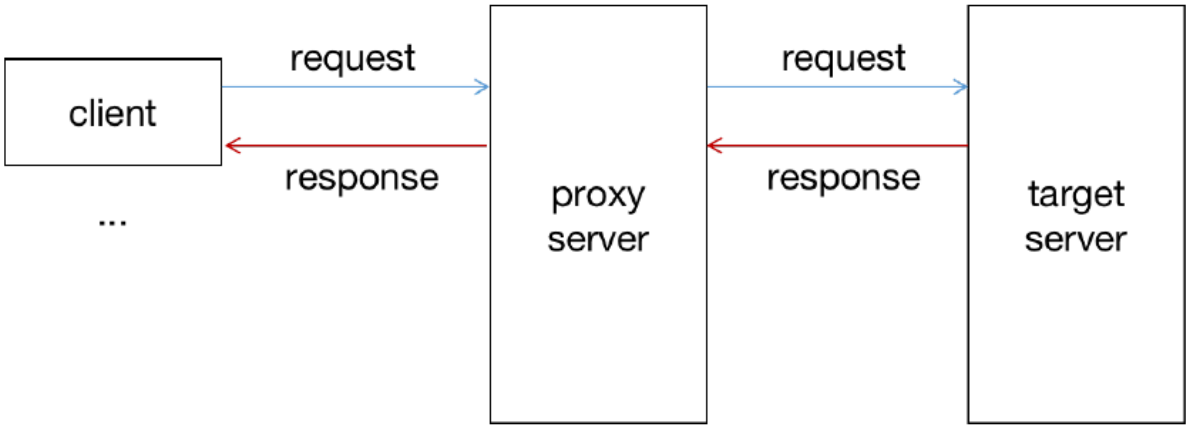
题目1.2：在题目1.1 的基础上实现代理服务器，让浏览器请求经过你的代理来请求 Web 对象。具体而言：

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。

代理服务器图示如下：



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

Webserver:

```
public class WebServer {
    //定义服务器监听的端口号
    private static final int PORT = 8081;
    private static ServerSocket serverSocket;

    public static void main(String[] args) throws IOException {
        //创建一个服务器套接字 serverSocket 并将其绑定到指定的端口号 PORT 上，以便监听客户端的连接请求。
        serverSocket = new ServerSocket(PORT);
        System.out.println("Web server listening on port " + PORT);
        //可以接受客户端的连接请求了！！
```

服务器创建一个 ServerSocket 对象，并将其绑定到指定的端口上，以便监听客户端的连接请求。

```
    while (true) {
        // 接受客户端的连接请求
        Socket clientSocket = serverSocket.accept();
        //accept() 方法是一个阻塞方法，直到有客户端连接请求到达并被接收时才会返回，表示建立了客户端与服务器的TCP连接
        // 为每个连接创建一个新的线程来处理请求
        Thread thread = new Thread(new WebServerRunnable(clientSocket));
        thread.start();
    }
}
```

服务器进入一个无限循环，不断等待客户端的连接请求。当客户端发起 TCP 连接请求时，服务器接受连接请求，并创建一个新的 Socket 对象与客户端建立 TCP 连接。

为每个客户端连接创建一个新的线程，以便并发处理多个客户端的请求。

QUIZ 1：如何解析来自客户端的请求报文的？

```
public void run() {
    try {
        // 获取客户端输入流和输出流
        fromClient = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));
        toClient = clientSocket.getOutputStream();
        String requestLine;
```

在新线程中，服务器通过获取客户端的输入流和输出流来与客户端进行通信。输入流用于接收客户端发送的 HTTP 请求报文，输出流用于发送 HTTP 响应报文给客户端。通过 BufferedReader 从客户端输入流中读取请求报文的每一行，使用 readLine()方法逐行读取。

```
// 读取客户端发送的HTTP请求报文的请求行
while((requestLine = fromClient.readLine())!=null && !requestLine.isEmpty()){
    System.out.println(requestLine);
    //关闭： 1）从客户端读取的数据为 null，即客户端已经关闭连接；2）从客户端读取的数据为空行，即已经读取完了所有的请求头信息。
    //第一行是请求行
    String[] tokens = requestLine.split( regex: "\\s+");
    //用split将请求行分割，得到方法，URI还有版本

    String method = tokens[0];// HTTP请求方法
    String uri = tokens[1]; // 请求的URI（资源标识符）
    String version = tokens[2];// HTTP协议版本
```

根据 HTTP 协议的规范，每个请求报文的第一行是请求行，包含了请求方法、URI 和 HTTP 版本。使用 String 的 split()方法将请求行按空格进行分割，得到三个部分：方法、URI 和版本。

```
// 检查请求行是否符合HTTP协议的格式，开始解析请求行了！！
if (tokens.length != 3) {
    // 如果请求行不符合 HTTP 协议的格式，返回 400 错误
    sendError(toClient, status: "400 Bad Request", message: "Invalid request line.");
    return;
}
```

如果请求行不符合 HTTP 请求报文的格式那就返回 400 错误

```
if ("/".equals(uri)) {
    // 如果请求的是根目录，指定返回的文件路径
    uri = "C:\\Users\\86138\\Desktop\\final_project\\src\\Test\\index.html";
} else if(uri.contains("index.html")){
    // 如果请求的是 index.html，指定返回的文件路径
    uri = "C:\\Users\\86138\\Desktop\\final_project\\src\\Test\\index.html";
}
```

根据请求的 URI 来确定要返回的文件路径。在代码中，如果 URI 是根目录/，则将文件路径设置为特定的目录下的 index.html 文件，如果 URI 包含 index.html，则同样设置文件路径为 index.html。这部分逻辑可以根据实际需求进行扩展。

```
                //shutdown功能实现
//支持关闭服务器的功能，可以在请求 URI 中添加 /shutdown 来关闭服务器
                //具体实现为调用 ServerSocket 对象的 close() 方法来关闭服务器端口。
                if ("/shutdown".equals(uri)) {
                    // 如果请求的是关闭服务器命令，返回 503 错误，并关闭服务器
                    System.out.println("Server shutting down...");
                    sendError(toClient,  status: "503 Service Unavailable",  message: "shutdown");
                    serverSocket.close();
                    System.exit( status: 0);//退出程序
                    return;
                }
```

如果请求的 URI 是/shutdown，则表示请求关闭服务器。服务器将返回 503 错误，并关闭 ServerSocket，然后通过调用 System.exit(0)来退出程序。

```
String filename = uri;
File file = new File(filename);

//不是shutdown也不是Index
if (!file.exists()) {
    //如果请求的文件不存在，将返回404错误。
    sendError(toClient,  status: "404 Not Found",  message: "404 Not Found.");
    return;
}
```

检查文件是否存在。使用 File 类的实例化对象，传入文件路径，然后调用 exists()方法来检查文件是否存在。如果文件不存在，则返回 404 错误。

QUIZ 2：WebServer 是如何构建响应报文的？

```
//FileInputStream 类和 read() 方法来读取指定文件的二进制数据，然后将其转换成字符串类型的数据进行返回。
FileInputStream fis = new FileInputStream(file);
//读取文件内容
byte[] fileContent = new byte[(int) file.length()];
//存在fileContent里面
//读完了!
fis.read(fileContent);
//在读取完成之后，需要显式地调用 fis.close() 方法关闭文件输入流，以释放系统资源和避免资源泄漏等问题
fis.close();
String content = new String(fileContent).trim();
//将文件内容转化为字符串
String contentType = Files.probeContentType(file.toPath());
//获取文件内容类型（根据文件的拓展名来判断内容类型
```

如果文件存在，使用 FileInputStream 来读取文件内容。通过 FileInputStream 读取文件的字节内容，并存储在 byte 数组 fileContent 中。

将文件内容转换为字符串，并获取文件的内容类型。使用 Files 类的 probeContentType()方法来获取文件的内容类型，该方法根据文件的扩展名来判断内容类型。

```
// 发送HTTP响应报文的头部
//头部包含了状态行 xxok!!
toClient.write(("HTTP/1.1 200 OK\r\n").getBytes());
//指定了返回的类型和字符
toClient.write(("Content-Type: " + contentType + "; charset=utf-8\r\n").getBytes());
//主体内容的长度
toClient.write(("Content-Length: " + content.getBytes().length + "\r\n").getBytes());
//表示结束，分隔响应头和响应主体
toClient.write(("\r\n").getBytes());
```

在发送响应之前，服务器构建 HTTP 响应报文的头部，包括状态行、Content-Type 字段和 Content-Length 字段，并发送给客户端。将响应头部写入输出流中，然后写入空行，表示响应头部结束。

```
// 发送HTTP响应报文的主体内容，将文件内容作为响应主体发送给客户端
toClient.write(content.getBytes());
toClient.flush();
//通过输出流的flush()方法将响应发回客户端
System.out.println("send response");
```

最后，将文件内容作为响应主体写入输出流。

通过输出流的 flush()方法将响应发送给客户端，确保数据被完全发送。

```
        }
    } catch (IOException e) {
        //打印错误信息
        System.err.println("Error handling request: " + e.getMessage());
    } finally {
        try {
            // 关闭流和套接字
            if(fromClient!=null) fromClient.close();
            if(toClient!=null) toClient.close();
            if(clientSocket!=null) clientSocket.close();
        } catch (IOException e) {
            // 处理关闭流和套接字时可能发生的异常
        }
    }
}
```

在异常处理方面，如果发生了 IOException 或其他错误，服务器打印错误信息，并关闭相关的流和套接字。

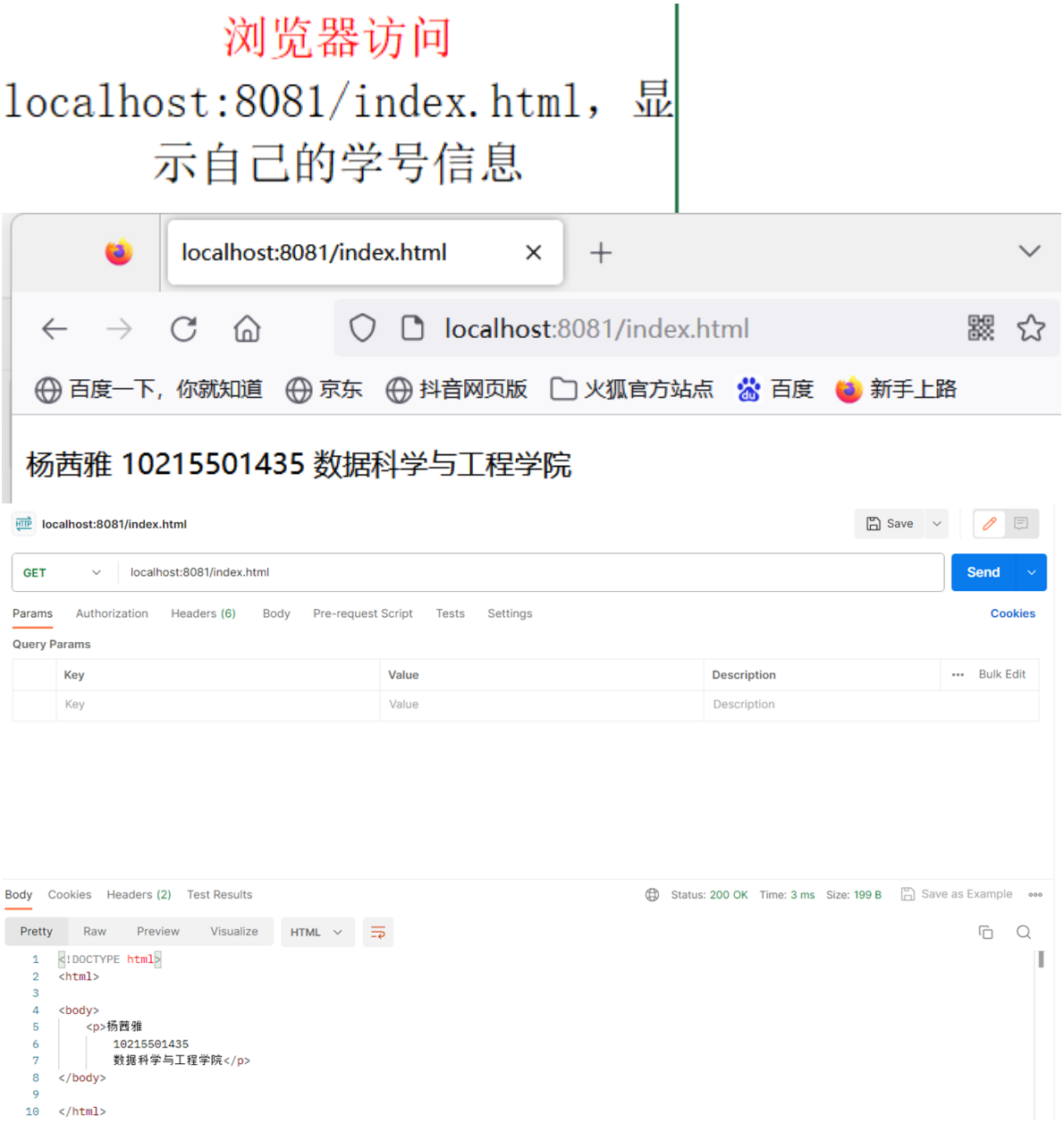
```
    private void sendError(OutputStream out, String status, String message) throws IOException {
        // 发送HTTP错误响应，根据输出流、状态码、错误消息这些参数来构建错误相应报文，发回客户端
        out.write(("HTTP/1.1 " + status + "\r\n").getBytes());
        out.write(("Content-Type: text/plain; charset=utf-8\r\n").getBytes());
        out.write(("Content-Length: " + message.getBytes().length + "\r\n").getBytes());
        out.write("\r\n".getBytes());
        out.write(message.getBytes());
    }
}
```

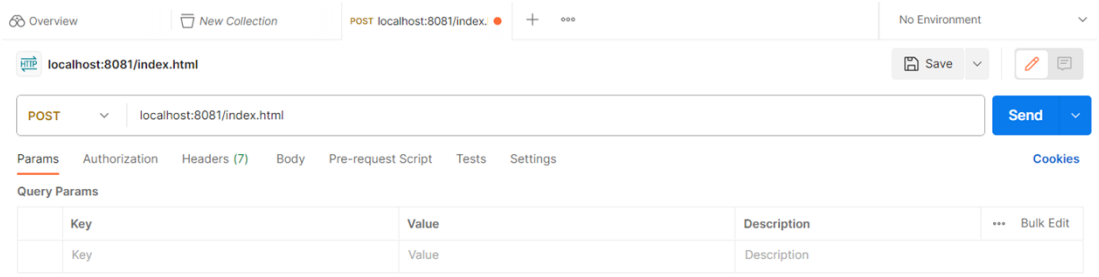
最后，定义了一个 sendError()方法，用于发送错误响应给客户端。它根据指定的状态码、消息内容和响应头部将错误信息发送给客户端。

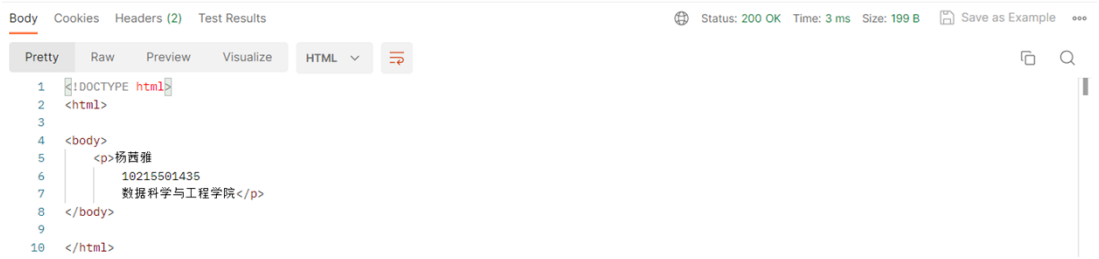
三、 性能测试情况

功能 1：

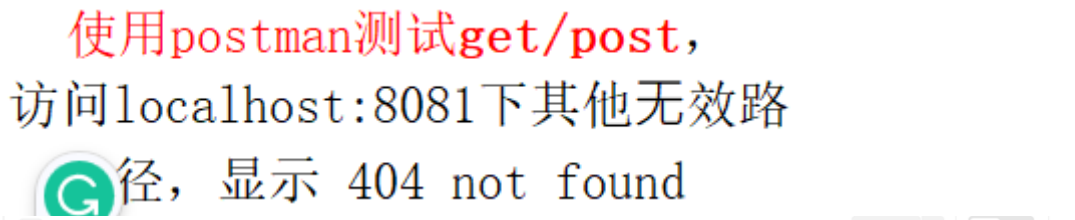
浏览器访问 localhost:8081/index.html，显示自己的学号信息

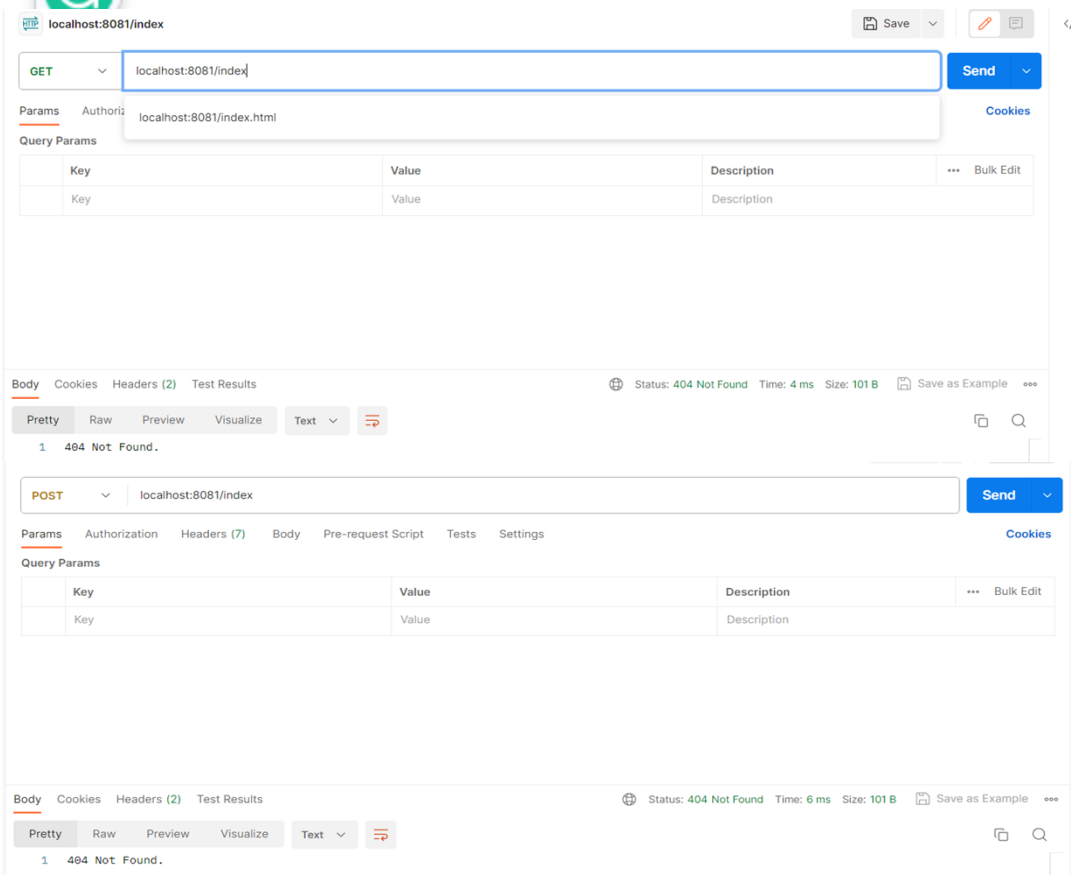



Overview
New Collection
POST localhost:8081/index.
No Environment
localhost:8081/index.html
Save
POST
localhost:8081/index.html
Send
Params
Authorization
Headers (7)
Body
Pre-request Script
Tests
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
Key
Value
Description


Body
Cookies
Headers (2)
Test Results
Status: 200 OK
Time: 3 ms
Size: 199 B
Save as Example
Pretty
Raw
Preview
Visualize
HTML
<!DOCTYPE html>
<html>
<body>
杨茜雅
10215501435
数据科学与工程学院
</body>
</html>

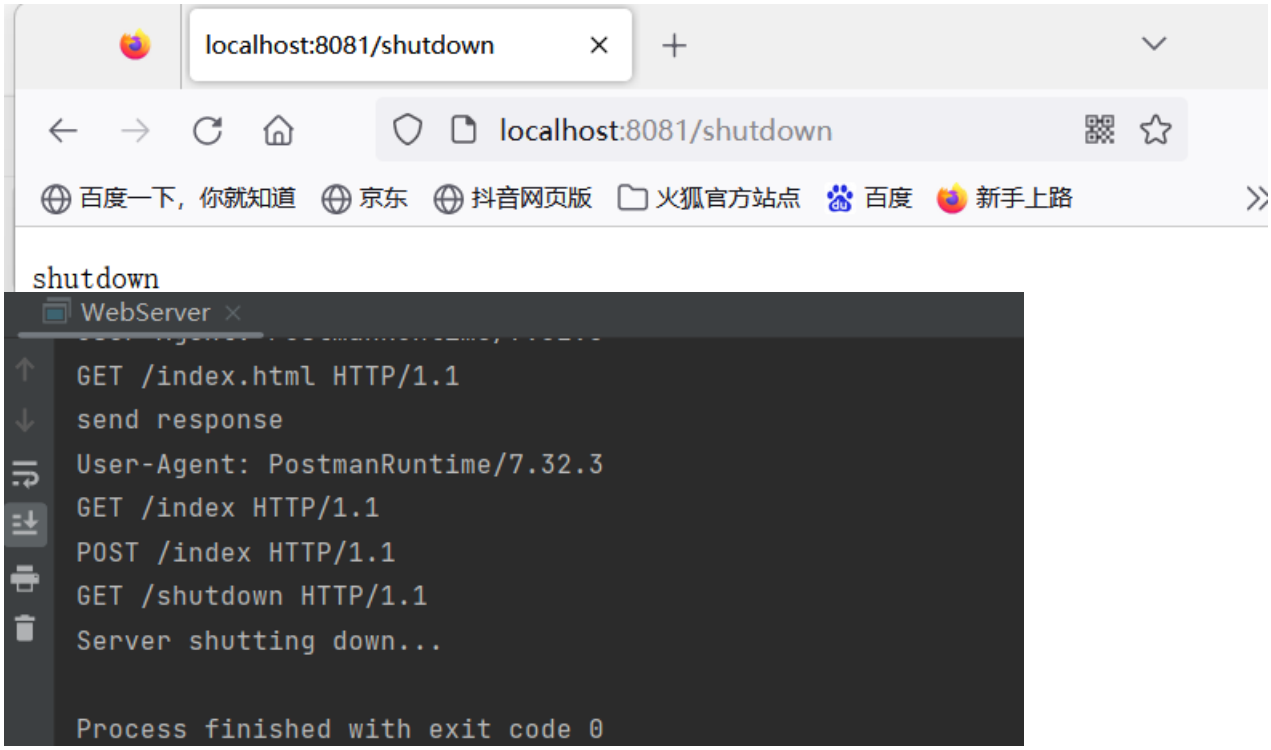
功能 2：


使用postman测试get/post，
访问localhost:8081下其他无效路
径，显示 404 not found


localhost:8081/index
Save
GET
localhost:8081/index
Send
localhost:8081/index.html
Params
Cookies
Query Params
Key
Value
Description
Bulk Edit
Key
Value
Description
Body
Cookies
Headers (2)
Test Results
Status: 404 Not Found
Time: 4 ms
Size: 101 B
Save as Example
Pretty
Raw
Preview
Visualize
Text
1 404 Not Found.
POST
localhost:8081/index
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 (2)
Test Results
Status: 404 Not Found
Time: 6 ms
Size: 101 B
Save as Example
Pretty
Raw
Preview
Visualize
Text
1 404 Not Found.

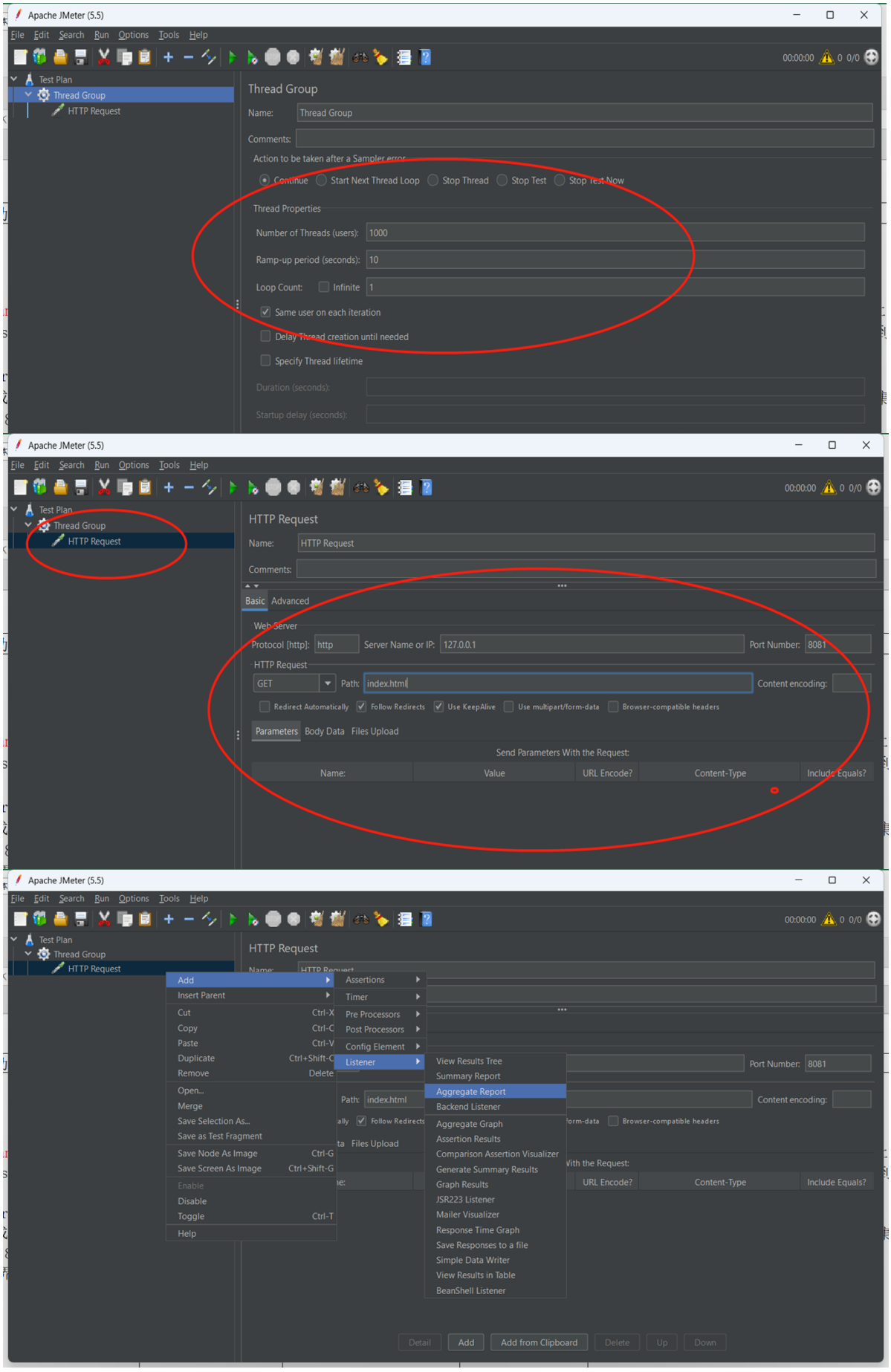
功能 3：

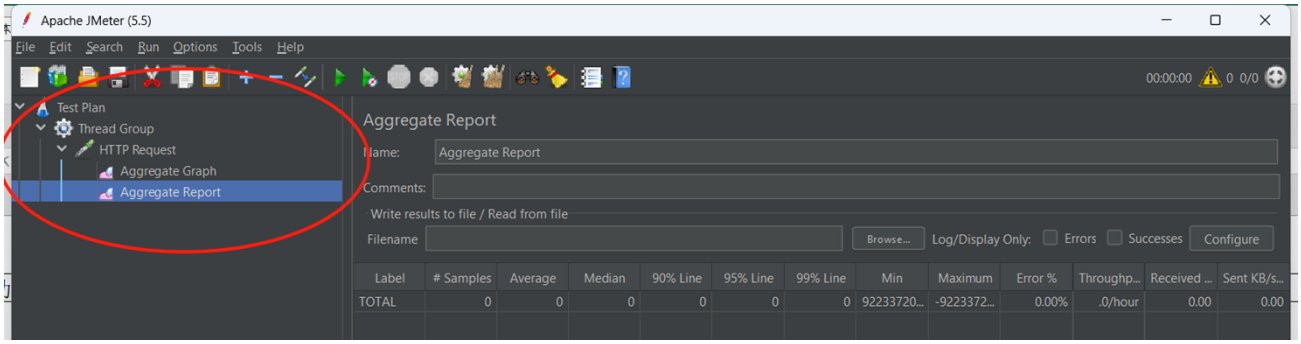
浏览器访问 localhost:8081/shutdown能使服务器关闭



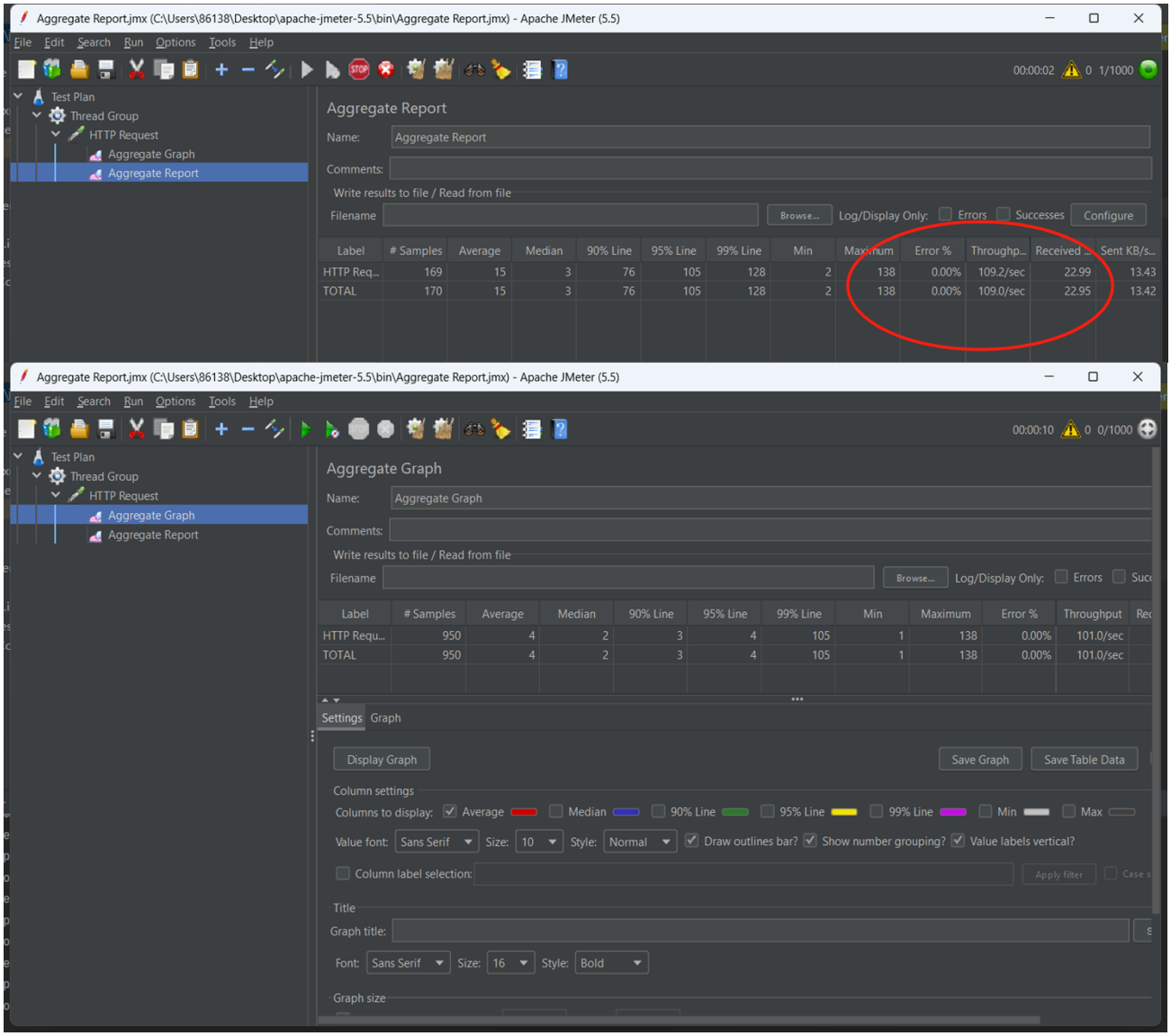
性能测试：

使用JMeter测试，得出聚合结果	【基本要求】参数为Number of Threads (users)=1000, Ramp-up period=10时，吞吐为100/sec左右 ，错误率0~10%(10')；仅达到一项要求(5')；均未达到要求(0')。 【Bonus要求】参数为Number of Threads (users)=5000, Ramp-up period=5时，增加集合点设置 Number of Simulated Users to Group by=1000，Timeout in milliseconds=0，吞吐为500/sec左右及以上，错误率0~10% (10'+5')。


Apache JMeter (5.5)
File Edit Search Run Options Tools Help
Test Plan
Thread Group
HTTP Request
Thread Group
Name: Thread Group
Comments:
Action to be taken after a Sampler error
Continue Start Next Thread Loop Stop Thread Stop Test Stop Test Now
Thread Properties
Number of Threads (users): 1000
Ramp-up period (seconds): 10
Loop Count: Infinite 1
Same user on each iteration
Delay Thread creation until needed
Specify Thread lifetime
Duration (seconds):
Startup delay (seconds):
HTTP Request
Name: HTTP Request
Comments:
Basic Advanced
Web Server
Protocol [http]: http
Server Name or IP: 127.0.0.1
Port Number: 8081
HTTP Request
GET
Path: index.html
Content encoding:
Redirect Automatically Follow Redirects Use KeepAlive Use multipart/form-data Browser-compatible headers
Parameters Body Data Files Upload
Send Parameters With the Request:
Name: Value URL Encode? Content-Type Include Equals?
Add
Insert Parent
Cut Ctrl-X
Copy Ctrl-C
Paste Ctrl-V
Duplicate Ctrl+Shift-C
Remove Delete
Open...
Merge
Save Selection As...
Save as Test Fragment
Save Node As Image Ctrl-G
Save Screen As Image Ctrl+Shift-G
Enable
Disable
Toggle Ctrl-T
Help
Assertions
Timer
Pre Processors
Post Processors
Config Element
Listener
View Results Tree
Summary Report
Aggregate Report
Backend Listener
Aggregate Graph
Assertion Results
Comparison Assertion Visualizer
Generate Summary Results
Graph Results
JSR223 Listener
Mailer Visualizer
Response Time Graph
Save Responses to a file
Simple Data Writer
View Results in Table
BeanShell Listener
Detail Add Add from Clipboard Delete Up Down



测试结果：



四、总结

本次实验因我个人的时间管理问题，只实现了第一个任务 Webserver，总体而言 Web 服务器的实现思路还是比较简单的，尤其是开始理论课的复习之后再实现该代码时会更加熟练，有一种学以致用的感觉，在性能测试中也有不错的表现。没能实现代理服务器是比较遗憾的，希望可以在之后时间允许的情况下带着继续学习的态度将其完成。