**深 圳 大 学 实 验 报 告**

**课程名称： 互联网编程**

**实验名称： 实验4 传输协议与套接字应用编程**

**学院： 计算机与软件学院**

**专业： 计算机科学与技术**

**指导教师： 毛斐巧**

**报告人： 吴嘉楷 学号： 202215016 班级： 国际班**

**实验时间： 2024.5.15,22,29, 6.5,12,19,26（周三）**

**实验报告提交时间： 2024年6月24日**

**教务处制**

一、实验目的与内容：

**目的：**

1. 掌握服务器端Socket编程开发能力。。

2. 掌握HTTP协议并能用于实际编程开发。

3. 掌握cookie编程技术，会应用于在web应用开发中持久存储和传递客户端状态，会编程管理cookie。

4.掌握URL类的使用，在不考虑所使用协议的细节情况下，会用java编写程序在网络上定位和获取数据。

**内容要求：**

基于java编程实现一个HTTP服务器程序（10分）和HTTP客户端程序（5分），要求

1. 采用多线程技术或线程池编程技术处理客户端请求，支持多客户端同时访问；（10分）
2. 实现GET、HEAD和POST请求，对客户端发送的不同请求给予正确响应；（15分）
3. 在服务器上存放一个文件夹（由HTML文本、图片文件或JS文件等组成），能根据不同请求，返回包括文本和图像2种（及以上）类型的响应，客户端可以正确显示和访问。（10分）
4. HTTP客户端程序能与该HTTP服务器连接并展示响应结果，正确发送不同类型的请求。（10分）
5. 能使用cookie编程技术保存和传递会话状态信息，比如保存用户信息等，需要保存的信息可自行决定。（10分）
6. 对服务器进行性能分析。对服务器进行压力测试，测试可支持多少个客户端同时访问，测试可支持多少个文件同时传输等。（10分）

注意：

1. 实验报告中需要有实验结果的截屏图像。

二、实验过程和代码与结果

**1.给出满足内容要求的HTTP服务器端程序源码和HTTP客户端程序源码，简述设计思路或实验过程。**

1. **实验思路：**

构建一个服务器端应用程序，它能够解析客户端请求、读取所需资源并构造响应消息。该服务器将采用线程池技术来高效地处理并发请求，同时能够识别不同的请求类型并执行相应的资源检索任务。

客户端应用程序将实现与服务器的连接、发送请求、解析响应的功能，并将结果以用户友好的方式展示。对于不适合在控制台展示的内容，如图片，客户端将自动保存这些内容到预设的文件夹中，以便于用户查看和进一步使用。这样的设计旨在提供流畅的用户体验和高效的资源管理。

1. **实验过程：**
2. **服务器端：**
   1. 设计一个服务器端类，该类使用 `THREAD\_COUNT` 作为参数来设定线程池的容量。服务器端持续监听并等待客户端发起的连接请求。连接一旦建立，服务器便立即创建一个处理线程 `Handler`，专门用于处理该客户端的请求。该处理线程负责执行一系列操作，包括解析请求、检索资源以及构建响应。随后，该线程被加入到线程池中，这样可以确保多个客户端能够同时被服务。通过线程池的智能管理，服务器有效地控制了线程的数量，优化了资源的使用，从而提高了并发处理的效率和响应速度。

public class HttpServer {  
 public final static int *THREAD\_COUNT* = 20;  
 public final static int *PORT*=9982;  
  
 public static void main(String []args) throws IOException {  
 ExecutorService pool = Executors.*newFixedThreadPool*(*THREAD\_COUNT*);  
 ServerSocket serverSocket = new ServerSocket(*PORT*);  
 int i = 1;  
  
 while (true) {  
 Socket clientSocket = serverSocket.accept();  
 Runnable task = new Handle(clientSocket, i);  
 pool.submit(task);  
 System.*out*.println(i);  
 i++;  
 }  
 }  
}

* 1. Handle处理线程：

1. 初始化和构造函数：

private final Socket clientSocket;  
private StringBuilder head = new StringBuilder("");  
private final int pos;  
private FileInputStream fr;  
private BufferedInputStream input;  
private BufferedInputStream in;  
private BufferedOutputStream out;  
  
public Handle(Socket s, int i) {  
 clientSocket = s;  
 pos = i;  
}

1. 在 `run` 方法内，我们首先建立与客户端的输入输出流连接。随后，接收并读取客户端发送的请求头信息。通过对这些信息进行细致的解析，我们可以确定请求的性质和类型。基于解析结果，我们将进一步决定将请求分发到 `GET`、`HEAD` 或 `POST` 等相应的处理函数。

public void run() {  
 try {  
 in = new BufferedInputStream(clientSocket.getInputStream());  
 out = new BufferedOutputStream(clientSocket.getOutputStream());  
 Scanner sin = new Scanner(in);  
 String t;  
 while ((t = sin.nextLine()) != null && t.length() != 0) {  
 head.append(t);  
 }  
 System.*out*.println(head);  
 if (head.toString().startsWith("GET"))  
 doGET();  
 else if (head.toString().startsWith("HEAD"))  
 doHEAD();  
 else if (head.toString().startsWith("POST"))  
 doPOST();  
  
 String[] head\_temp = head.toString().split(" ");  
 StringBuilder temp = new StringBuilder("");  
 temp.append(head\_temp[2] + " 200 OK\r\n");  
 Date date = new Date();  
 SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd hh : mm : ss");  
 temp.append("Set-Cookie: " + pos + " " + dateFormat.format(date) + "\r\n\r\n");  
 out.write(temp.toString().getBytes());  
 out.flush();  
 in.close();  
  
 } catch (IOException e) {  
 System.*out*.println(e);  
 }  
}

1. GET 请求处理：`doGET` 方法专门用于响应客户端发起的 GET 请求。该方法首先解析请求头中的字符串，以确定客户端请求访问的资源路径。接着，服务器端利用缓冲输入流从指定路径读取所需资源。最终，通过缓冲输出流，将获取的资源数据高效地传输回客户端。

public void doGET() throws IOException {  
 fr = new FileInputStream("src/resourse/" + head.toString().split(" ")[1]);  
 input = new BufferedInputStream(fr);  
 byte[] buffer = new byte[4096];  
 int len;  
 while ((len = input.read(buffer)) > 0) {  
 out.write(buffer, 0, len);  
 out.flush();  
 }  
 out.close();  
 input.close();  
}

1. HEAD 请求处理：`doHEAD` 方法负责处理客户端的 HEAD 请求，该请求旨在获取资源的元信息而非资源本身。此方法通过解析请求头字符串，首先提取并确认请求行，然后构造响应行，格式为 "HTTP版本 200 OK"，表示请求已被成功接受且资源可访问。接下来，方法将添加包括客户端编号、连接时间等附加信息。最终，服务器使用缓冲输出流将这些响应头信息发送给客户端。

public void doHEAD() throws IOException {  
 String []head\_temp=head.toString().split(" ");  
 StringBuilder t = new StringBuilder("");  
 t.append(head\_temp[2]+" 200 OK\r\n");  
 Date date = new Date();  
 SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");  
 t.append("client=" + pos + "; connect\_time=" + dateFormat.format(date) + "\r\n");  
 t.append("Host:"+clientSocket.getPort());  
 t.append("\r\n");  
 out.write(t.toString().getBytes());  
 out.flush();  
 out.close();  
}

1. POST 请求处理：`doPOST` 方法旨在接收并确认客户端发送的信息。在当前实现中，客户端发送的POST请求仅包含请求头，并未附带任何表单数据。因此，服务器端不会对POST请求中的数据进行解析或处理。为了响应客户端，服务器设置了一条确认信息：“DOING POST operator now！I receive your data！”，表明已成功接收到POST请求。随后，服务器通过缓冲输出流将这条响应消息发送给客户端，确保客户端知晓其请求已被接收。

public void doPOST() throws IOException {  
 StringBuilder t = new StringBuilder("DOING POST operator now!\nI receive your data!\r\n");  
 out.write(t.toString().getBytes());  
 out.flush();  
 out.close();  
}

1. 在服务器端接收到客户端的请求头后，响应头的构造过程开始于将请求头的第一行更新为“HTTP版本 200 OK”，这表明资源可以被正常访问。随后，服务器生成包含客户端编号和连接时间的cookie信息。这些信息随后被整合到响应中，并使用服务器的缓冲输出流发送给客户端，从而完成一次信息交换。

具体代码如下：

String[] head\_temp = head.toString().split(" ");  
StringBuilder temp = new StringBuilder("");  
temp.append(head\_temp[2] + " 200 OK\r\n");  
Date date = new Date();  
SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd hh : mm : ss");  
temp.append("Set-Cookie: " + pos + " " + dateFormat.format(date) + "\r\n\r\n");  
out.write(temp.toString().getBytes());  
out.flush();

1. **客户端：**
2. 通过限制客户端线程的数量上限并同时启动这些线程，我们达成了让多个客户端同时进行访问的目的：

public class Clients {  
 public static void main(String []args){  
 int n=20;  
 for(int i=1;i<=n;i++){  
 new Thread(new Client(i)).start();  
 }  
 }  
}

1. 对于Client类的初始化和构造函数：

private static final int *PORT*=9982;  
private int pos;  
private Socket socket;  
private StringBuilder path\_sin = new StringBuilder("src/client");  
private StringBuilder path\_in = new StringBuilder("src/client\_resourse");  
private StringBuilder head = new StringBuilder("");  
private FileOutputStream fw;  
private BufferedOutputStream output;  
private BufferedInputStream in;  
private BufferedOutputStream out;  
private final String[] type1 = {"GET", "HEAD", "POST"};  
private final String[] type2 = {"file2.jpg", "file1.js", "file1.html"};  
private Random rd = new Random();  
private String receivedCookie;  
Client(int p) {  
 pos = p;  
}

1. 首先，利用随机数生成技术来模拟客户端发送请求，这包括随机确定请求类型和所需资源。根据不同的请求资源，我们将创建相应的文件写入缓冲流，这将用于存储来自服务器端的响应数据。随后，我们将建立与服务器的连接，并设置缓冲输入输出流，以便进行高效的数据交换。通过缓冲输出流，我们将发送经过格式化的请求头到服务器，并通过缓冲输入流来接收并读取服务器的响应。最终，我们将接收到的响应信息通过缓冲输出流写入到客户端本地的文件夹中，这样不仅可以保留通信记录，也便于用户日后的查阅和进一步处理。

public void run() {  
 try {  
 int y1 = rd.nextInt(3);  
 int y2 = rd.nextInt(3);  
 if (y1 == 0) {  
 path\_in.append("/" + pos + "" + type2[y2]);  
 fw = new FileOutputStream(path\_in.toString());  
 output = new BufferedOutputStream(fw);  
 } else {  
 path\_in.append("/" + pos + ".txt");  
 fw = new FileOutputStream(path\_in.toString());  
 output = new BufferedOutputStream(fw);  
 }  
 socket = new Socket("localhost", *PORT*);  
 in = new BufferedInputStream(socket.getInputStream());  
 out = new BufferedOutputStream(socket.getOutputStream());  
 System.*out*.println(pos);  
 head.append(type1[y1]);  
 head.append(" "+type2[y2]+" HTTP/1.1\n\n");  
 out.write(head.toString().getBytes());  
 out.flush();  
 System.*out*.println(pos+":"+type1[y1]+" "+type2[y2]);  
 byte[]byffer=new byte[4096];  
 int len;  
 while((len=in.read(byffer))>0){  
 output.write(byffer,0,len);  
 output.flush();  
 }  
 output.close();  
 } catch (IOException e) {  
 System.*out*.println("Unable to connect to the server");  
 }  
}

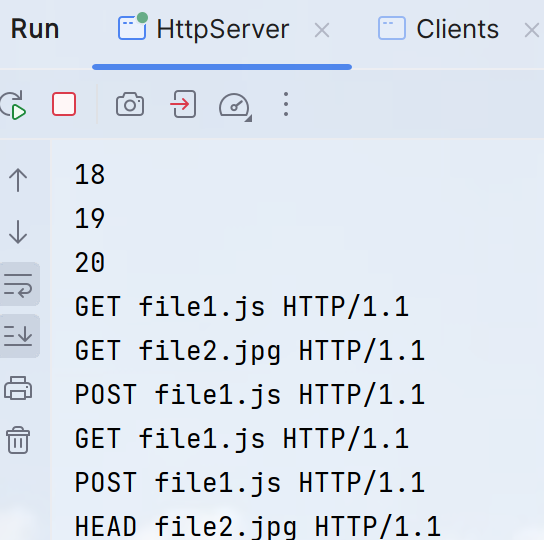
1. 客户端在与服务器端通信时，负责接收并处理由服务器发送的Cookie信息。通过使用缓冲字符流，客户端能够高效地读取服务器的响应信息。一旦检测到服务器端的Cookie信息，客户端将立即将其保存到本地文件夹。这种存储机制确保了Cookie信息可以在随后的请求中被轻松检索和使用，从而维持了会话状态的连续性，并优化了客户端与服务器端之间的交互体验。

private void saveCookieToFile(int clientId, String cookie) {  
 try {  
 path\_sin.append("/"+pos+".txt");  
 File file = new File("client");  
 FileWriter fileWriter = new FileWriter(file, true);  
 BufferedWriter writer = new BufferedWriter(fileWriter);  
 Scanner scanner=new Scanner(in);  
 String str;  
 StringBuilder stringBuilder=new StringBuilder("");  
 while((str=scanner.nextLine())!=null&&str.length()!=0){  
 stringBuilder.append(str);  
 }  
 writer.newLine();  
 writer.flush();  
 writer.close();  
 } catch (IOException e) {  
 System.*out*.println("src.Client " + clientId + " - Unable to save cookie to file");  
 }  
}

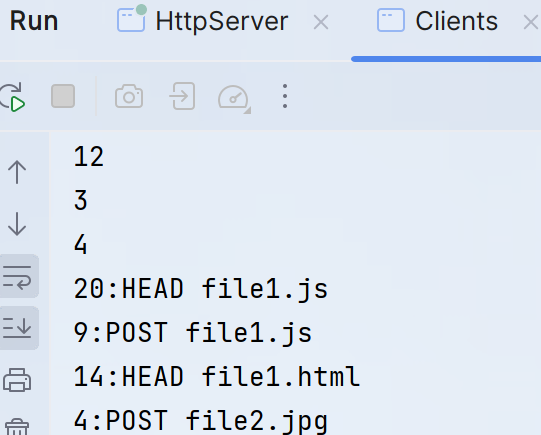
**2. 给出满足不同内容要求所进行测试的运行结果，包括客户端的运行结果及服务器端的运行结果，并简述所进行的测试。**

通过小规模测试程序的运行结果：

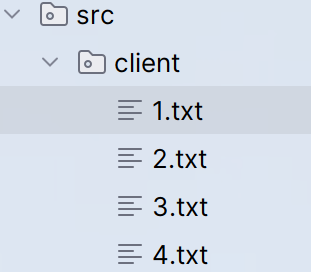
* + - 1. 服务器端控制台输出如下：



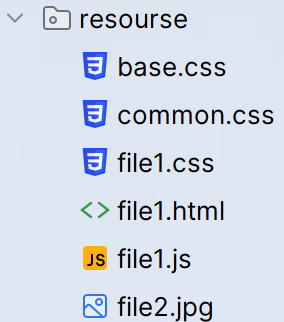
* + - 1. 客户端输出如下：



* + - 1. 生成后文件夹如下：client保存的是客户端接收的cookie文件，client\_resourse保存客户端存储的文件，resourse保存的是服务器端存储的文件：







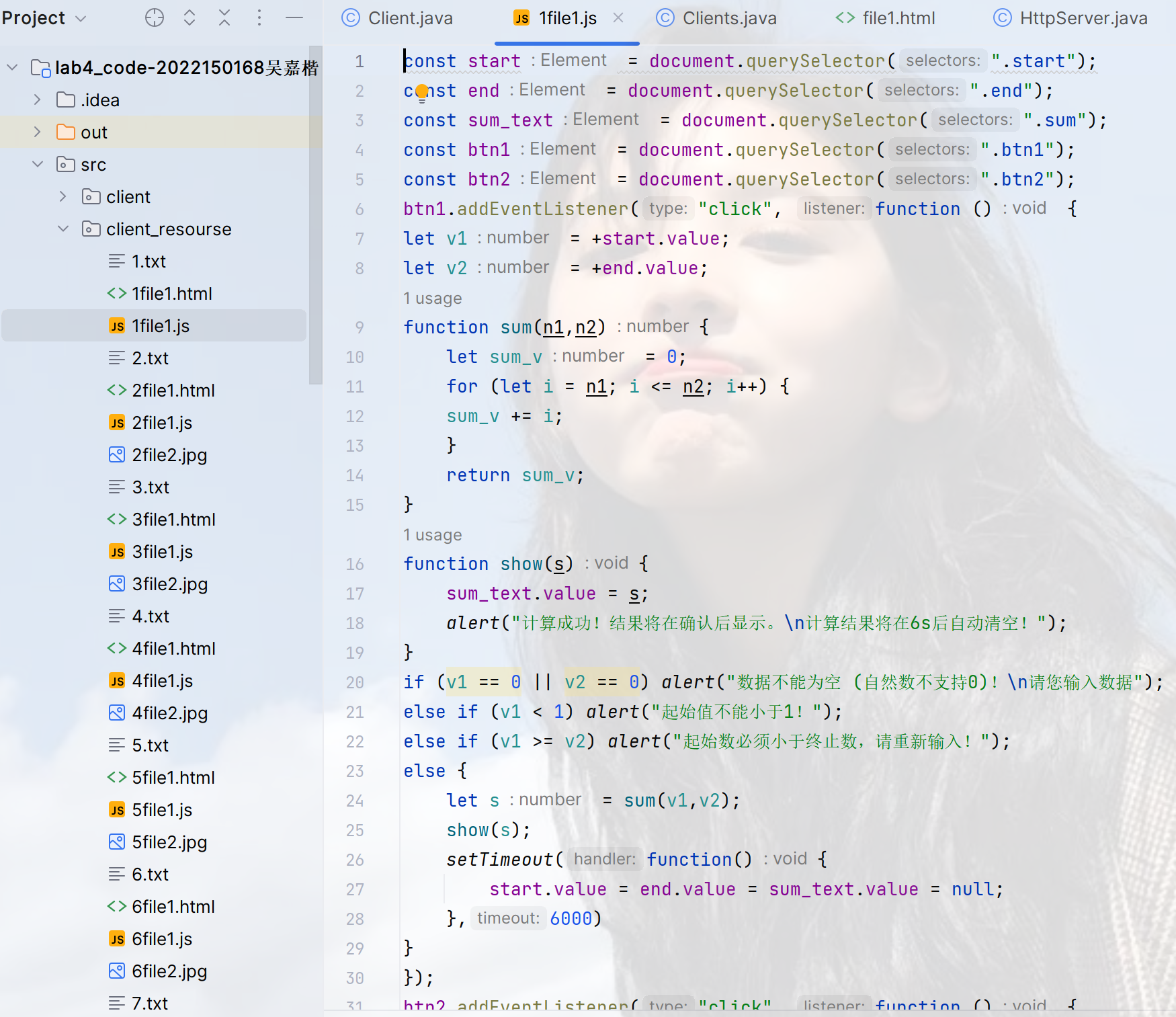
* + - 1. 各个文件传输成功：

在client文件夹中，客户端成功存储服务器端的cookie信息：

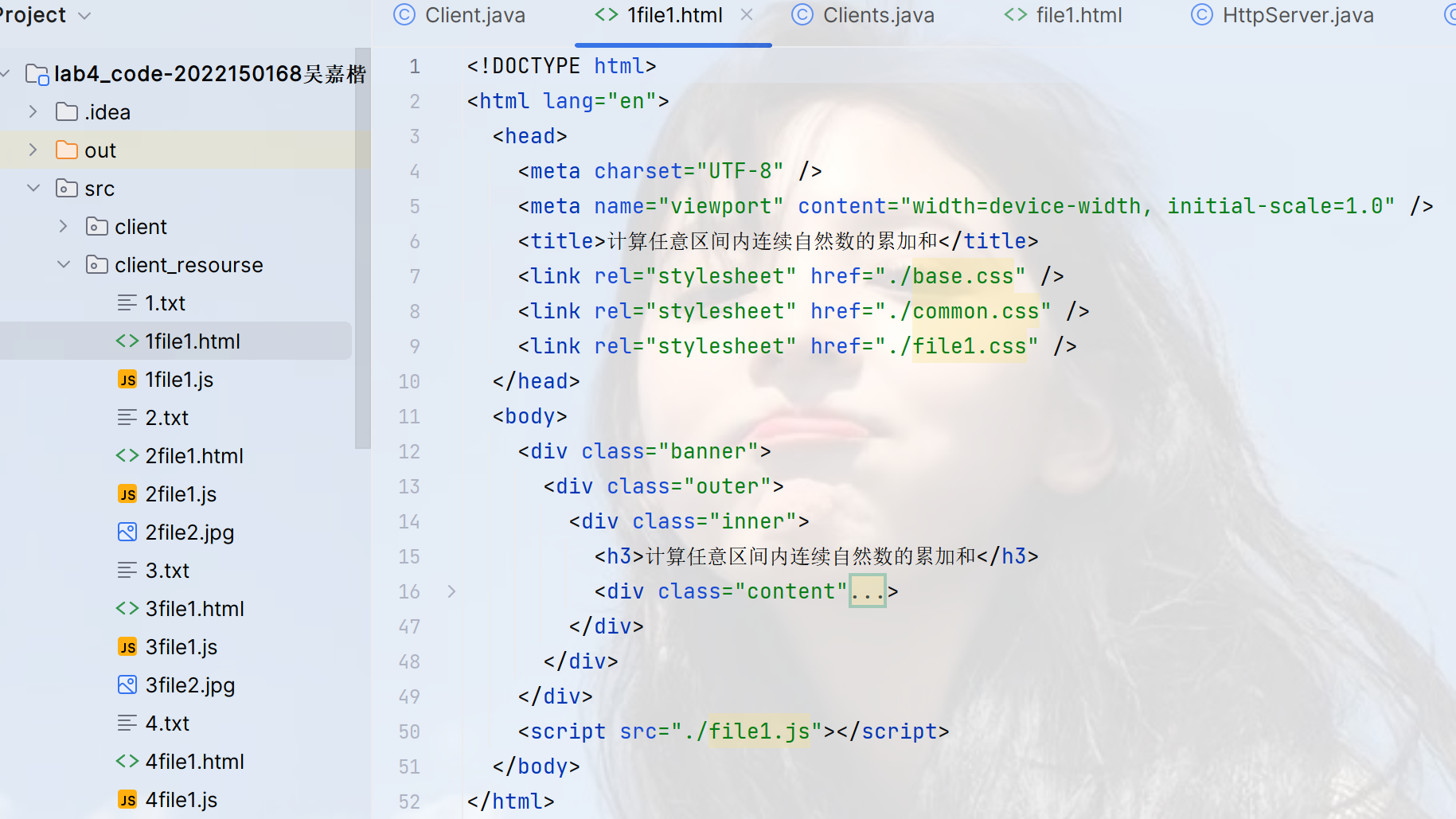


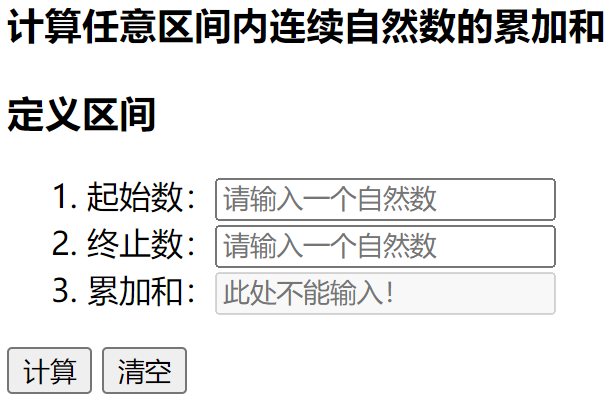
在client\_resourse文件中使用GET接收的资源显示，服务器成功解析请求头，并正确返回对应的信息，客户端成功接收并存储信息：

* 以下是获取到的js文件：（内容与服务器中的js文件一致）



* 以下是获取到的html文件：（由于没获取css文件，所以样式单调）

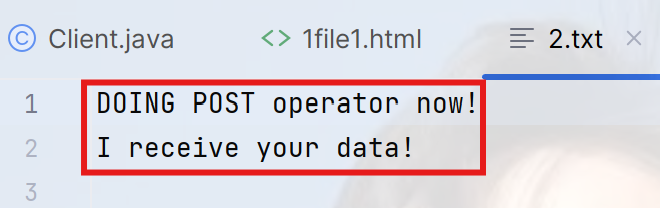




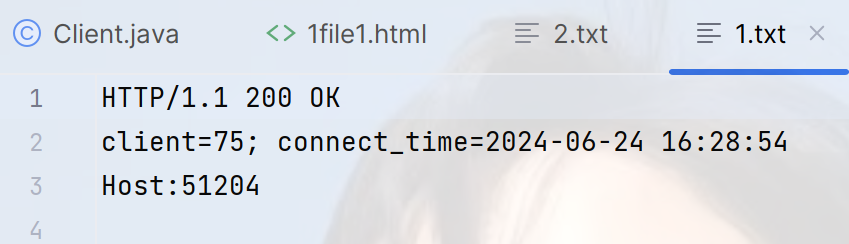
* 以下是获取到的jpg文件：



在client\_resourse中，使用POST指令发送表单，服务器返回信息表明接收表单，正确解析请求头并返回相关信息，客户端正确接收并储存：



在client\_resourse中，使用HEAD请求，服务器正确解析请求头信息，并返回资源信息，同时输出了客户端编号和链接的时间：



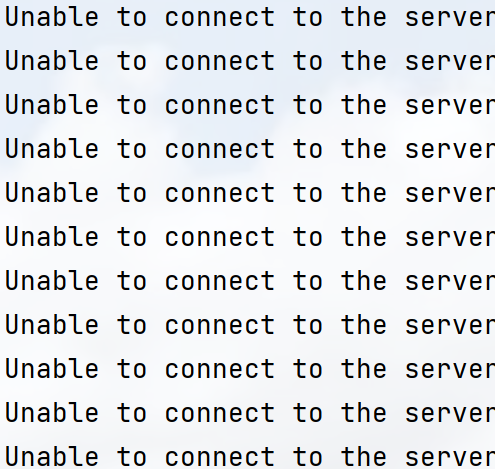
**3.服务器性能分析总结。**

**对服务器进行压力测试，测试可支持多少个客户端同时访问，测试可支持多少个文件同时传输等，描述测试方法与测试结果。**

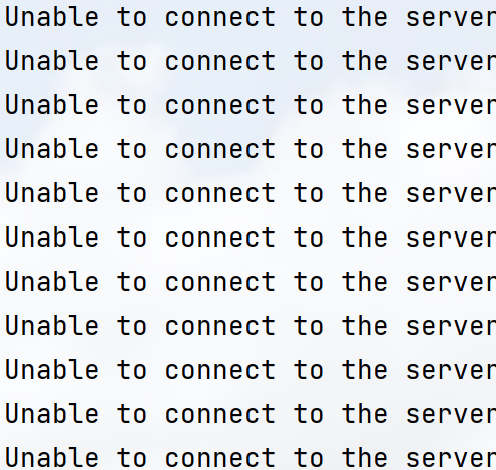
**测试方法：** 通过逐渐增加服务器端线程池的大小，观察到服务器处理客户端连接的能力在什么时候时达到极限。

**测试结果：**

经过测试，在线程池容量为5000时，出现了部分连接不上的情况：



当线程池大小设置为8000时，后续无法连接上的数量增多：



经多次测试，随着线程池大小的增大，当线程池数量增大到8000，后续无法连接上的更多，因此，服务器最多可支持8000个客户端同时访问。

当线程池大小为2000时，每个客户端线程的文件传输数量为3，因此，同时可传输的最大文件个数为6000，如果线程池大小超过2000，则会出现部分文件没有传输的情况，综上所述，同时可传输的文件最大个数为6000。

三、实验总结

（此处写你的过程，比如遇到的错误，以及解决方法，你的所想、所得）

通过本次实验，我深刻理解了HTTP协议的工作机制，并掌握了使用Java编程语言处理HTTP请求与响应的相关技术。在服务器端，我有效利用线程池管理客户端请求，确保了能够应对多客户端的同时访问需求。通过对请求头的解析和根据不同请求类型的区分，我实现了对GET、HEAD及POST请求的准确响应。

服务器能够根据存放在本地文件夹中的资源，成功返回包括HTML页面和图像在内的多样化响应内容，这使得客户端能够准确地展示和访问这些资源。客户端程序不仅能够与服务器建立稳定连接并展示响应结果，还能发送多种类型的请求。整体上，我成功构建了一个基于Java的HTTP服务器和客户端程序，这不仅增强了我对网络通信和Web开发的理解，也提升了相关技术技能。

**难点分析：**

**难点一**：HTTP请求头的解析和处理

**解决方法**：使用Scanner逐行读取输入流并构建HTTP请求头字符串。通过startsWith("GET")等判断请求类型。解析请求头的过程可能出现异常（如请求头格式不规范），需要进行错误处理。

**难点二**：设置Cookie和响应头

**解决方法**：通过SimpleDateFormat格式化日期，设置Set-Cookie头。正确构建和写入HTTP响应头字符串，确保符合HTTP协议规范。

**难点三**：HTTP请求的构建和发送

**解决方法**：通过Random类生成随机数，并根据随机数选择请求类型和资源路径。使用StringBuilder构建HTTP请求头，并通过BufferedOutputStream发送请求头。

**难点四**：响应头的处理

**解决方法**：使用BufferedInputStream读取服务器的响应数据，并逐行解析响应头。检查响应头中的Set-Cookie字段，并调用saveCookieToFile方法保存Cookie。

四、成绩评定

指导教师批阅意见：

成绩评定：

指导教师签字：毛斐巧

年 月 日