南開大學
Nankai University

计算机学院

计算机网络实验报告

配置 Web 服务器，编写简单页面，分析交互过程

姓名：李欣

学号：2011165

专业：计算机科学与技术

2022 年 10 月 29 日

目录

1 实验要求

1. **搭建 Web 服务器**（自由选择系统），并**制作简单的 Web 页面**，包含简单文本信息（至少包含专业、学号、姓名）和自己的 LOGO。
2. 通过浏览器获取自己编写的 Web 页面，使用 **Wireshark** 捕获浏览器与 Web 服务器的交互过程，并进行简单的分析说明。
3. 提交**实验报告**。

2 实验准备——配置 Web 服务器，编写简单页面

2.1 Web 客户/服务器模型

在本次实验中，我们使用的是 Web 客户/服务器模型，其模型图如图所示，

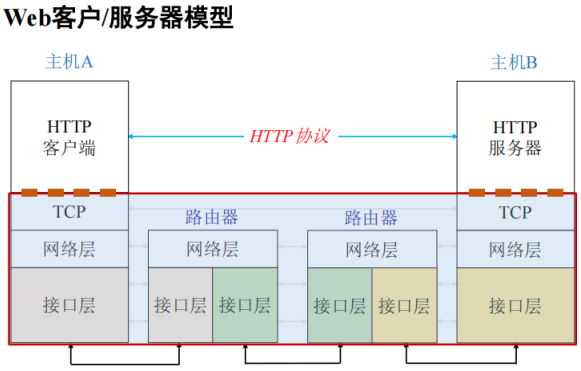

图 2.1: 多线程端口 2

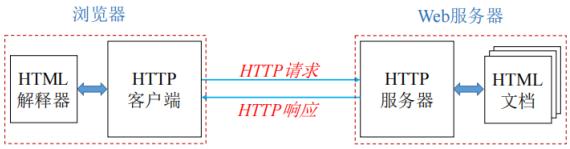

图 2.2: 多线程端口 2

服务器

1. Web 页面（HTML 文档）：包含到多种对象的链接
2. 对象：可以是 HTML 文档、图像文件、视频文件、声音文件、脚本文件等
3. 对象用 URL（统一资源定位符）编址：协议类型://主机名//路径和文件名

客户端

1. 发出请求、接收响应、解释 HTML 文档并显示；
2. 有些对象需要浏览器安装插件

2.2 配置 web 服务器

参考网上的流程，使用 idea 中的 springboot 进行搭建，经过进一步的学习，使用 springboot 直接创建项目时内嵌的服务器是 Tomcat。具体步骤如下：

(1) 手动配置 idea，jdk，marven，在此不再赘述，只给出笔者使用的版本号。

1. **idea 2022.2.3**
2. **maven-3.8.1**
3. **jdk 17.0.4.1**。

(2) 使用使用 springboot 搭建

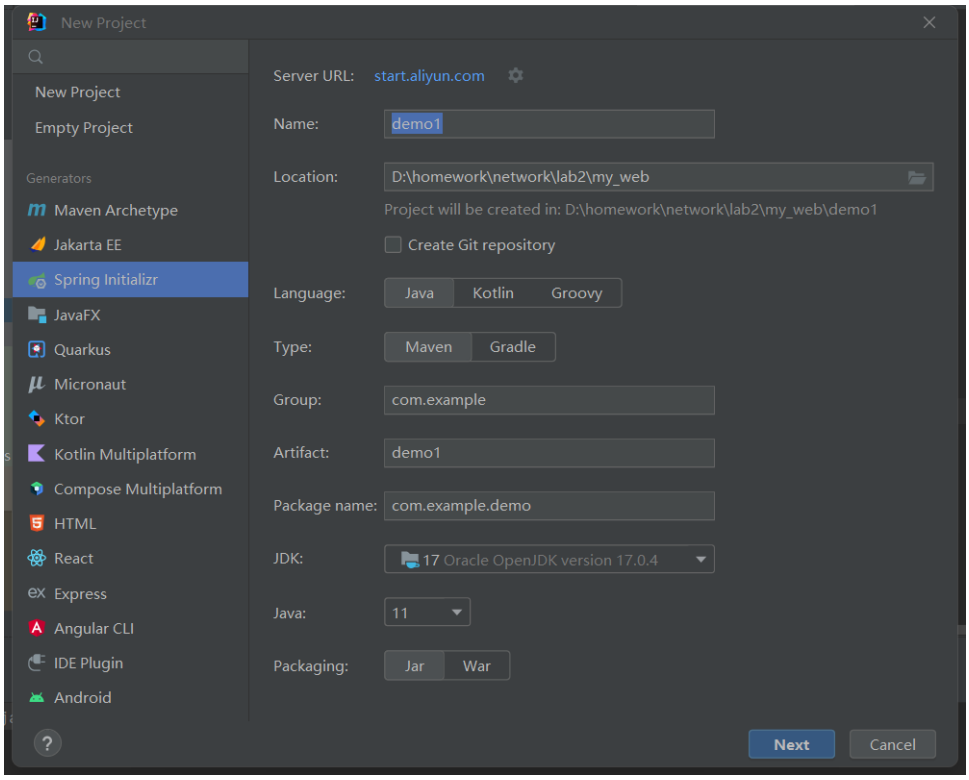

图 2.3: 初始化界面

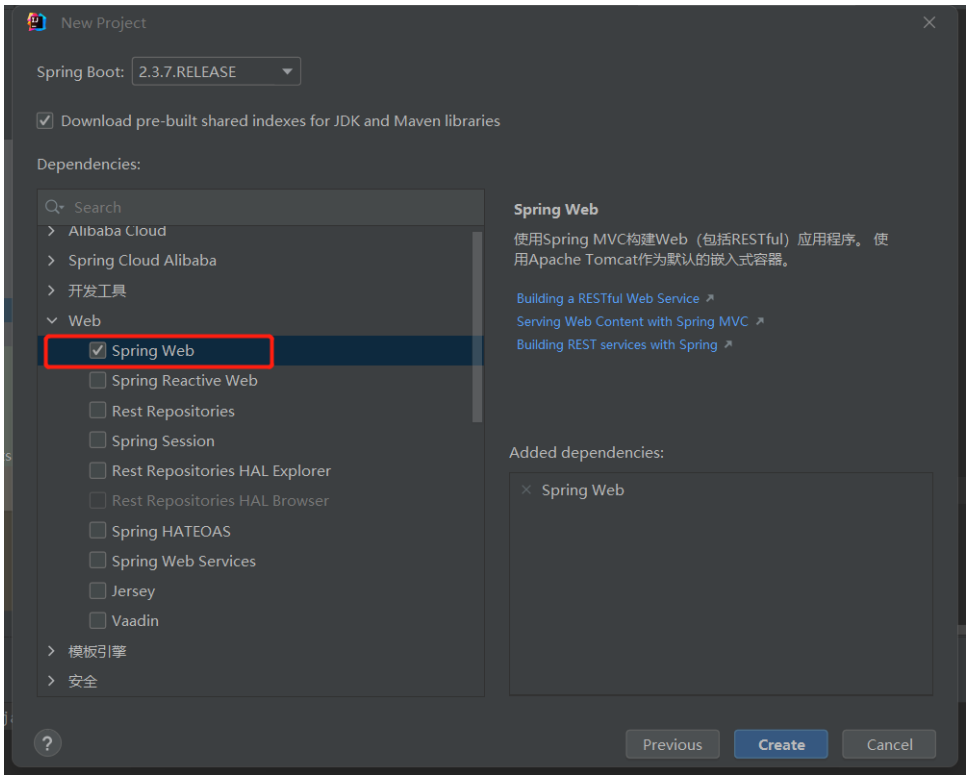

图 2.4: 选择 spring web(必选)

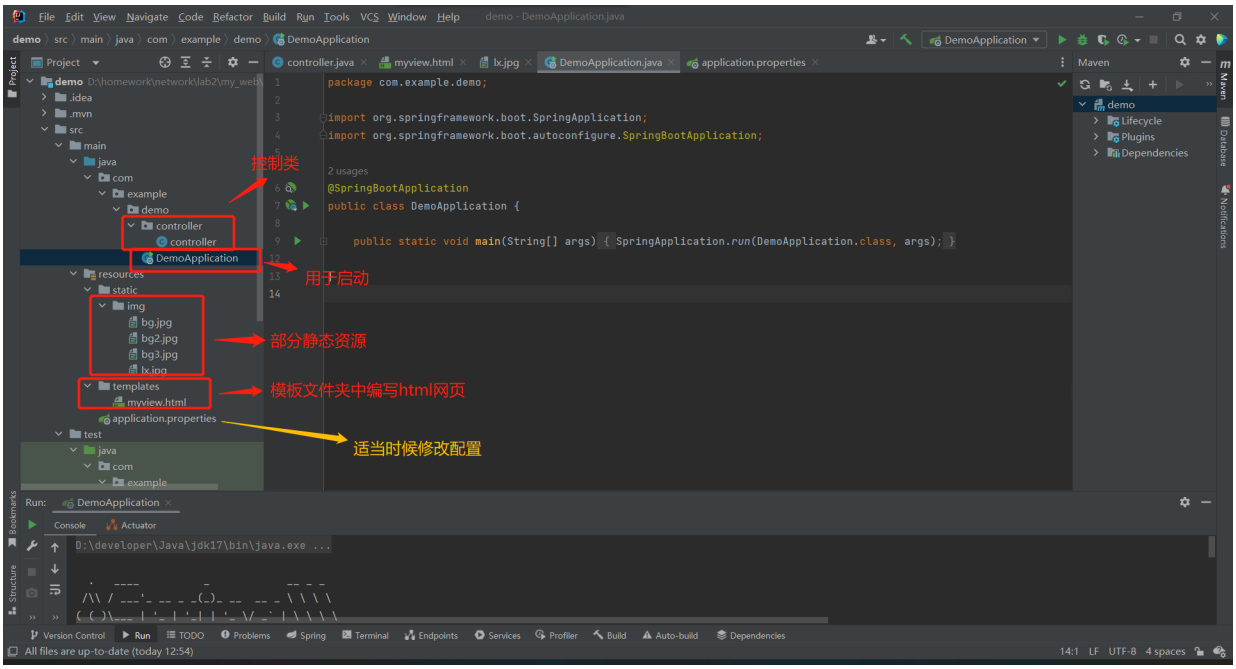

图 2.5: 添加文件

如上述图片所示，即为搭建 web 项目的整个过程，其中，Controller 和 DemoApplication 两个 java 文件的源代码如下：

```
1  package com.example.demo;
```

```

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class DemoApplication {

    public static void main(String[] args) {
        SpringApplication.run(DemoApplication.class, args);
    }

}
```

```
package com.example.demo.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
public class controller {

    @RequestMapping("/myview")
    public String login(){
        return "myview";
    }
}
```

2.3 html 网页文件编写

2.3.1 部分源代码

```
<html>
<head>
  <meta charset="utf-8">
  <title> 本科生简历 </title>
</head>

<body background="/bg3.jpg"/>

<table border="1" align="center" cellpadding="10" width="800">
  <tr>
```

```
11      <th colspan="7" bgcolor="BurlyWood" > 个人简介 </th>
12    </tr>
13
14    <tr>
15      <th bgcolor="BurlyWood"> 姓名:</th>
16      <td> 李欣 </td>
17      <th bgcolor="BurlyWood"> 年龄:</th>
18      <td>21</td>
19      <th bgcolor="BurlyWood"> 民族:</th>
20      <td> 汉 </td>
21      <td rowspan="3" width="100"
22      <p>
23        <img border="0" src="/lx.jpg" alt="" width="140" height="140" />
24      </p>
25      </td>
26    </tr>
27
28      //......
29
30    <tr>
31      <th bgcolor="BurlyWood"> 专业:</th>
32      <td> 计算机科学与技术 </td>
33      <th bgcolor="BurlyWood"> 毕业学校:</th>
34      <td> 南开大学 </td>
35      <th bgcolor="BurlyWood"> 邮编:</th>
36      <td> 300071</td>
37    </tr>
38
39  </table>
40  </body>
41
42  </html>
```

2.3.2 效果图

图 2.6: html 效果图

2.4 遇到的问题

2.4.1 浏览器访问服务器端 html 网页图片位置无法显示图片

解决：在 application.properties 文件中添加配置

```
spring.resources.static-locations=classpath:/templates/,classpath:/static/img/
```

原因分析：

spring.resources.static-location 参数指定了 Spring Boot-web 项目中静态文件存放地址，该参数默认设置为：classpath:\static,classpath:\public,classpath:\resources,classpath:\META-INF\resources, servlet context:\，可以发现这些地址中并没有\templates 也没有\static\img 这个地址。当配置文件中配置此项后，默认配置失效，使用自定义设置！

此处涉及到两个概念：

（1）classpath：通俗来讲 classpath 对应的项目中：web-practice\src\main\resources 文件目录。如："classpath: templates " 即是将 resources 目录下的 templates 文件夹设置为静态文件目录。更深一步讲 classpath 路径为：文件编译后在 target\classes 目录下的文件。

（2）静态文件目录：通俗理解为存放包括：html; jsp; CSS; js; 图片；文本文件等类型文件的目录。这些文件都可以通过浏览器 url 进行访问。同时 controller 中转发的文件目录也必须被设置为静态文件目录，这就是增加了该参数以后就可以正常访问的原因。

2.4.2 无法下载 springboot 相关配置文件

解决：无法访问原网站，切换了阿里云镜像。

3 分析交互过程

3.1 使用 wireshark 捕捉交互的过程

1、首先，我们打开 wireshark，由于我们实验所用的客户端和服务端都是本机，所以选择 loopback 选项

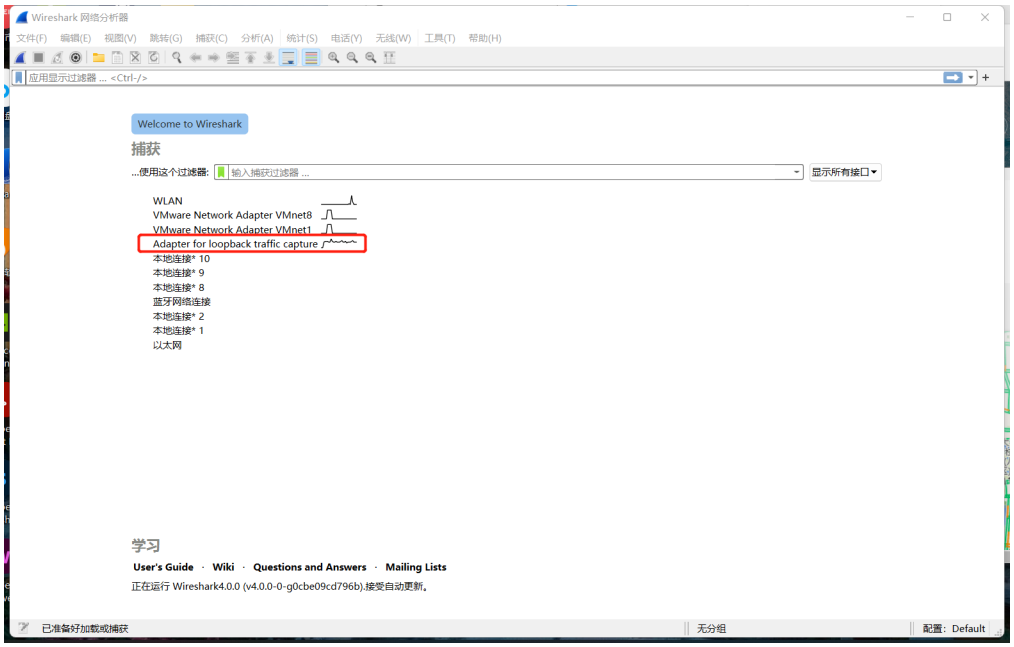

图 3.7: 打开 wireshark

2、其次，我们需要与服务端的 8080 端口进行交互，故筛选条件为 tcp.port=8080

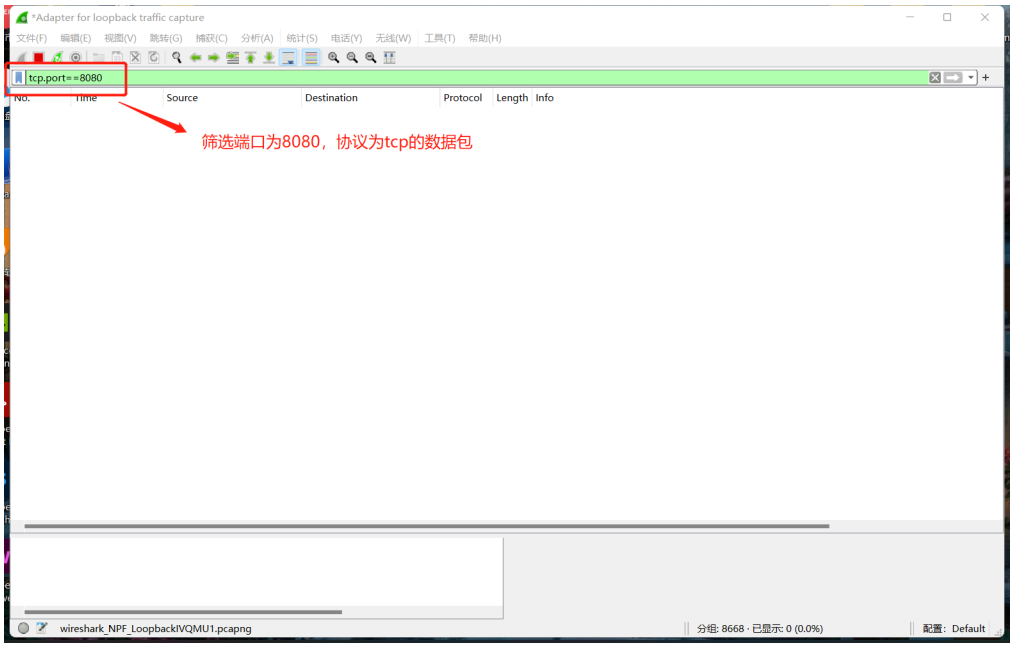

图 3.8: 筛选条件

3、可以看到，此时未捕获到任何信息，这是由于还没有与服务端进行交互，我们打开服务器。

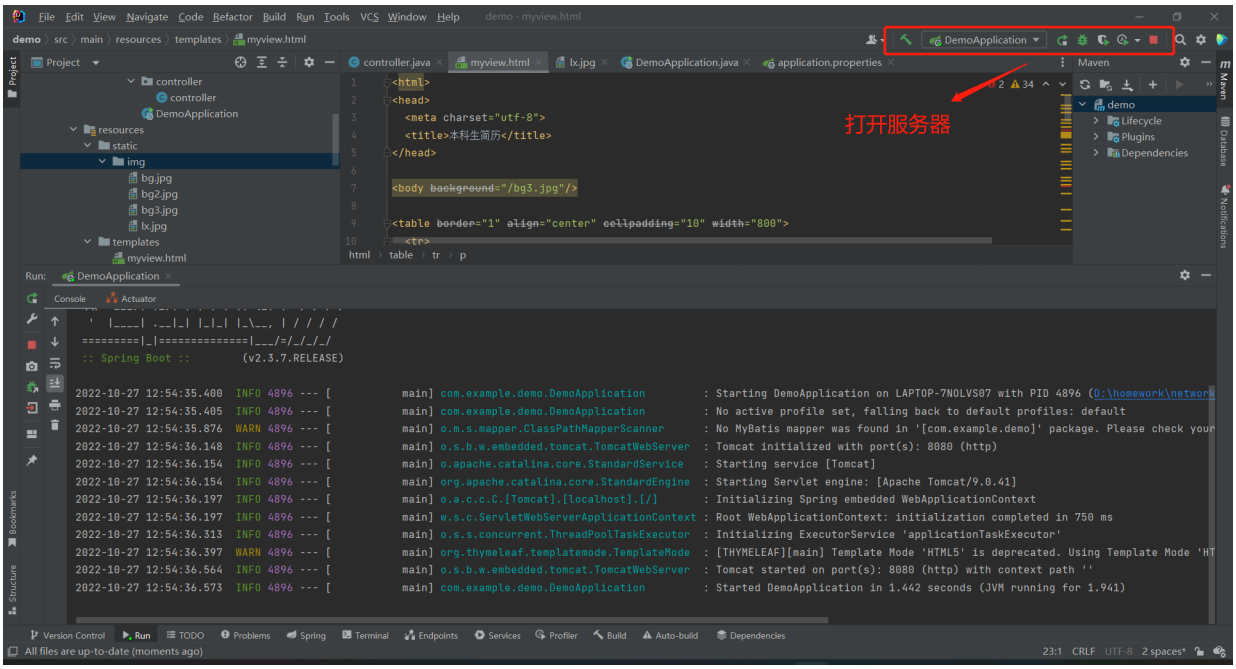

图 3.9: 添加文件

4、打开服务器后，观察 wireshark 能够观察到很多捕获的信息，我们在下一部分进行分析。

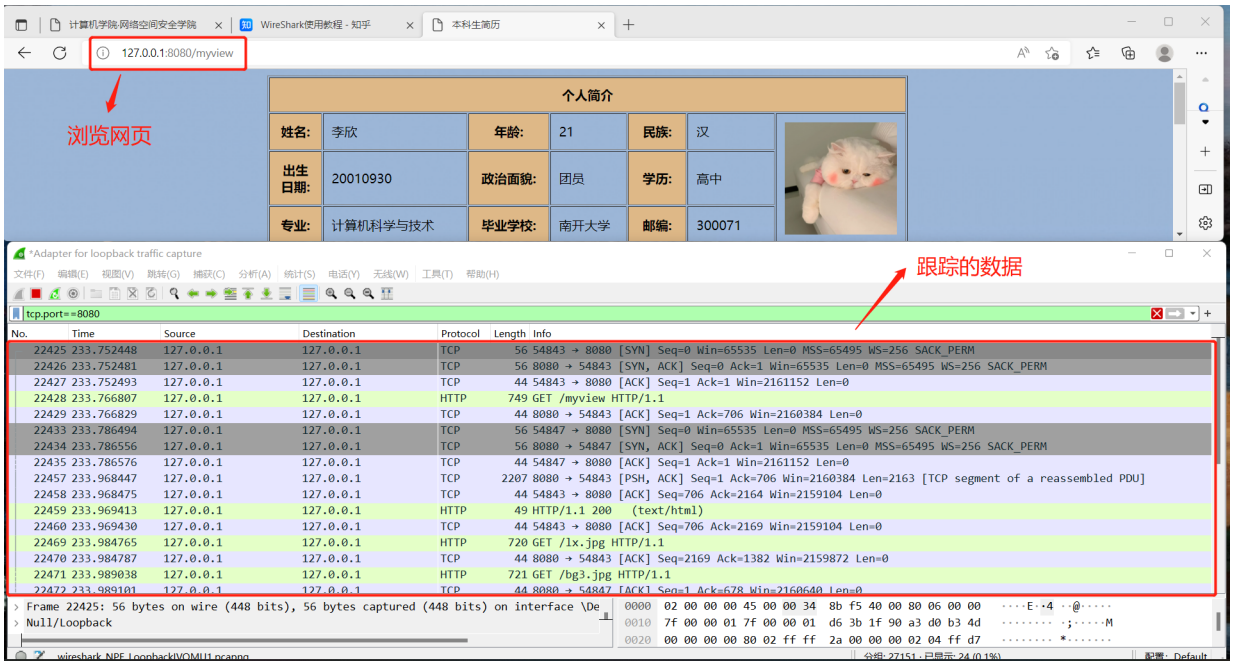

图 3.10: 添加文件

3.2 HTTP 协议和 TCP 协议

在分析交互过程前，我们需要对 tcp 和 http 协议有一定的了解。

TCP 协议是**传输层协议**，主要解决数据如何在网络中**传输**。

HTTP 协议是**应用层协议**，主要解决如何**包装**数据。

3.3 分析交互过程

3.3.1 三次握手建立连接

No.	Time	Source	Destination	Protocol	Length	Info
22425	233.752448	127.0.0.1	127.0.0.1	TCP	56	54843 → 8080 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
22426	233.752481	127.0.0.1	127.0.0.1	TCP	56	8080 → 54843 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
22427	233.752493	127.0.0.1	127.0.0.1	TCP	44	54843 → 8080 [ACK] Seq=1 Ack=1 Win=2161152 Len=0

图 3.11: wireshark 捕捉的三次握手连接

wireshark 捕捉的三次握手连接如上图所示，我们对其进行分析。

第一次握手：54847 端口向 8080 端口发送 syn 包，并令本机初始序列号为 seq=x=0，建立连接时并不传送消息，故报文长度为 0，滑动窗口大小为 65535，可接受的最大 TCO 包为 65495 字节，窗口缩放因子为 256，此时客户端，即浏览器进入 SYN_SEND 状态。

第二次握手：服务端的 8080 端口收到 syn 包后，确认客户端发来的 syn，使得 ack=x+1=1,seq=y=0，即对序列号为 ack(1) 之前的报文进行确认。与此同时发送 (syn，ack) 包给客户端，数据包特性如图所示。

第三次握手：客户端接收服务器端发来的 (syn,ack) 包，客户端返回确认包，ack=y+1=1, 数据包特性如图所示。

至此完成三次握手，客户端和服务端成功建立了连接，由本次实验中于我们使用的是 HTTP1.1，一般建立连接后不轻易断开连接。

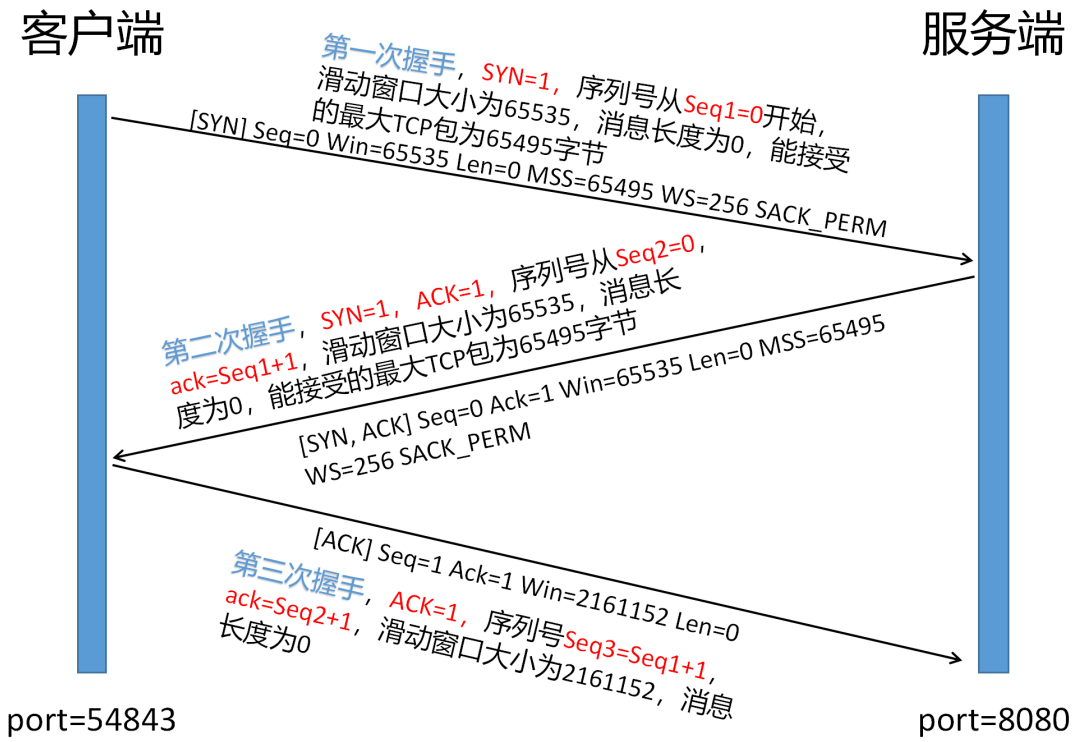

图 3.12: 三次握手连接

如图所示，客户端与服务端通过三次握手建立连接。

- 54843 -> 8080 端口号: 源端口—> 目标端口

- SYN：同步握手信号
- Seq：序列号
- Win: TCP 窗口大小
- Len: 消息长度
- Mss: 最大报文段长度
- Ws: 窗口缩放调整因子
- SACK_PERM：SACK 选项，这里等于 1 表示开启 SACK。

3.3.2 数据传输

HTTP 两种报文：请求 (request)、响应 (response)

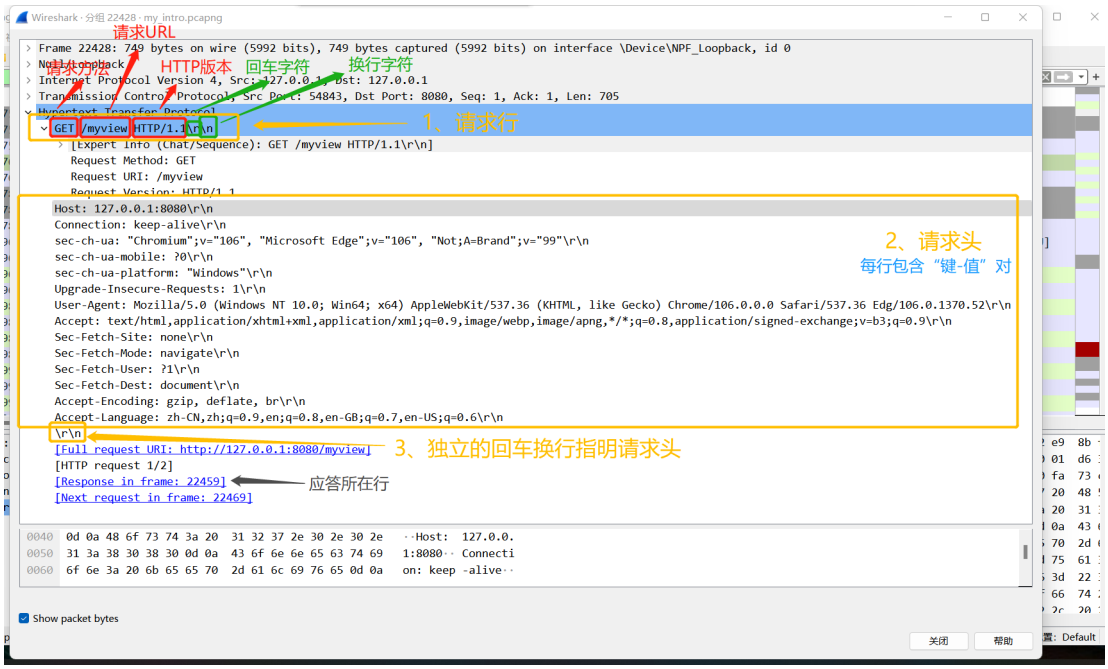

图 3.13: 请求报文

HTTP 请求报文：采用 ASCII，数据部分采用 MIME 格式。

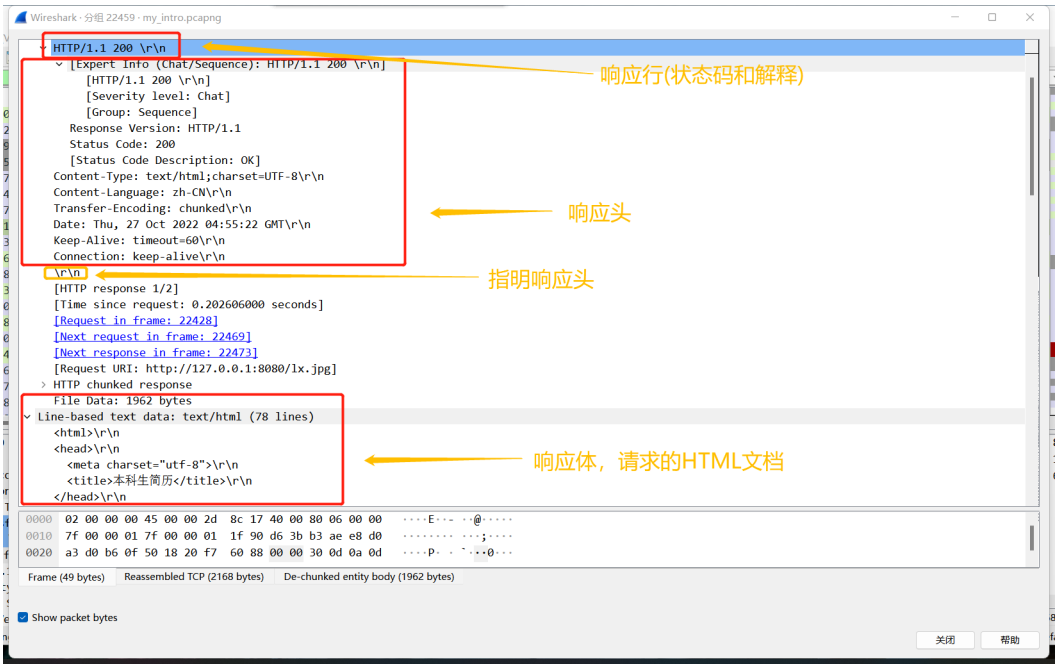

图 3.14: 响应报文

HTTP 响应报文：数据部分采用 MIME 格式。

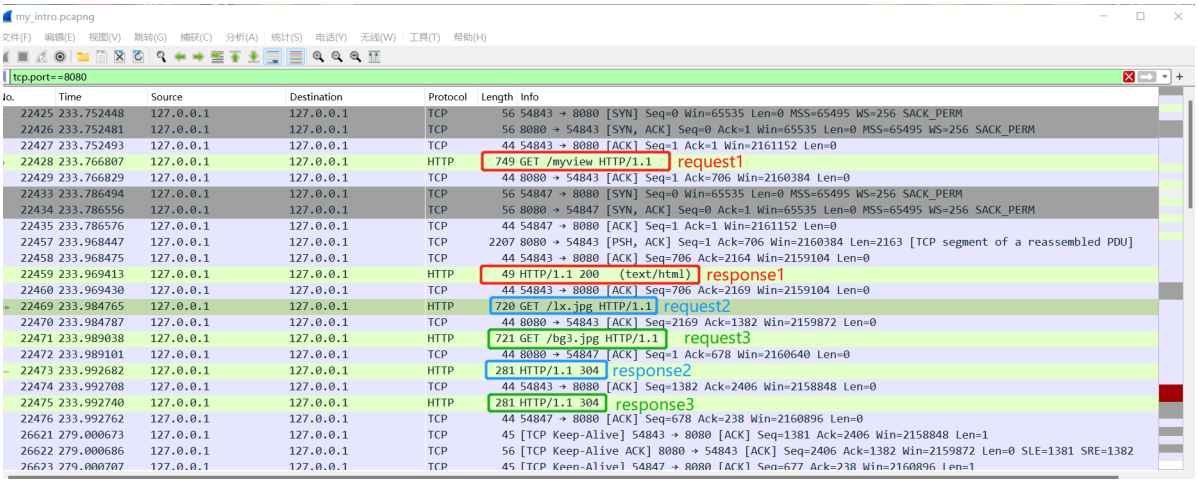

图 3.15: 请求与响应

看整体的请求与响应，发现一共有三次，但图片部分的响应状态码为 304，综合课上所学和学习资料，服务器：如果缓存的对象是最新的，在响应时无需包含该对象，响应头包含 HTTP/1.1 304 Not Modified。由于保存该次 wireshark 捕获的交互过程前，已经做过实验，且没有清除浏览器缓存，图片已经被浏览器缓存了，所以出现了 304，我们将浏览器缓存数据删除后再次实验，发现出现了下面的情况。

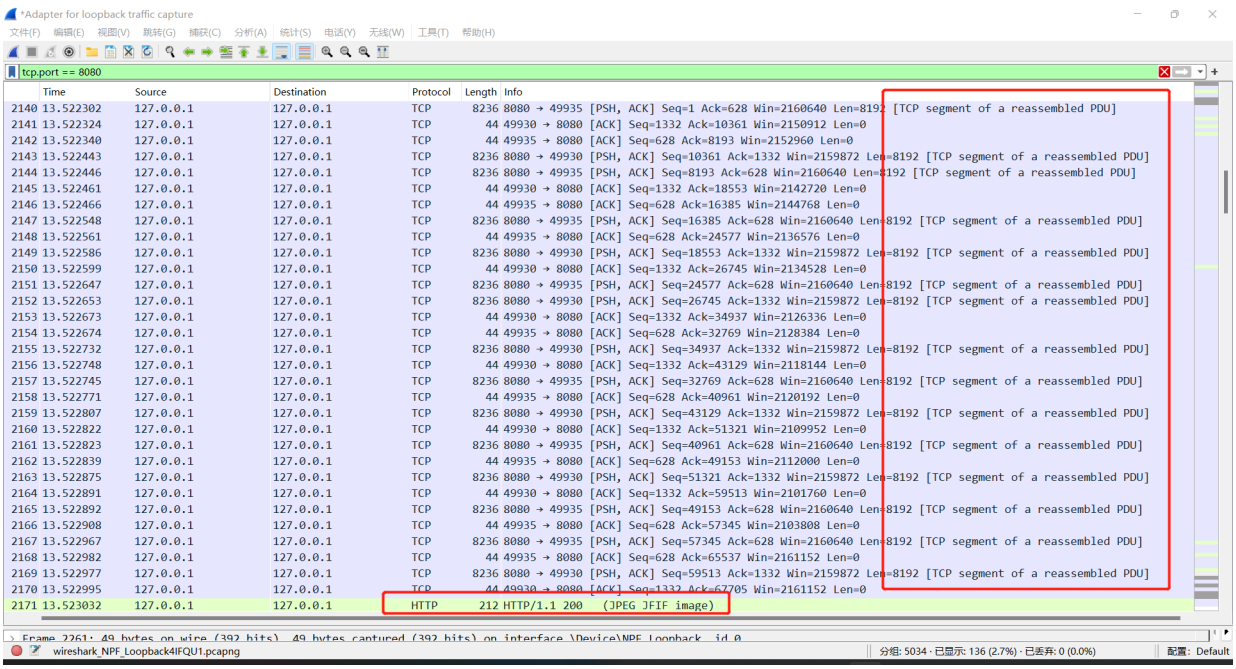

图 3.16: 请求与响应

将缓存数据删除，重新捕获交互过程，发现在 wireshark 抓包过程中出现大量 [TCP segment of a reassembled PDU] 提示信息，当我们基于 TCP 在传输消息时，对于上面的应用层如果出于某些原因（如超过 MSS）TCP Segment 不能一次包含全部的应用层 PDU，而要把一个完整消息分成多个段，就会将除了最后一个分段（segment）的所有其他分段都打上 “TCP segment of a reassembled PDU”。

3.3.3 四次挥手断开连接

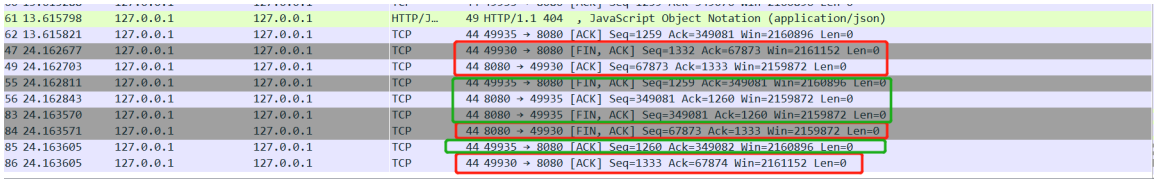

图 3.17: 四次挥手

在这里我们看到，有两个四次挥手断开连接，观察 wireshark 捕获的数据，我们知道浏览器自动开启了多线程，再此，我们仅分析一个四次挥手即可，红色和绿色分别为一个四次挥手断开连接，那么，我们分析红色四次挥手。

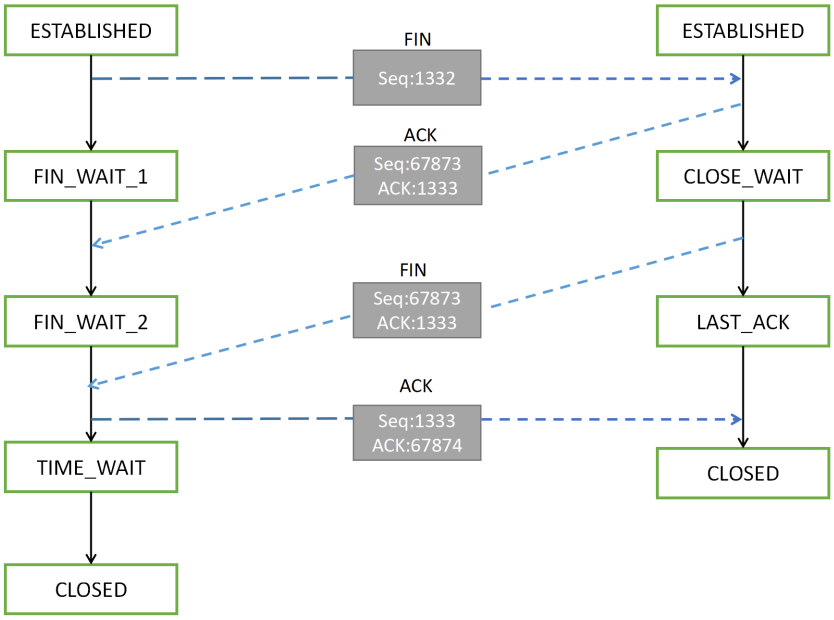

图 3.18: 红色部分四次挥手

第一次挥手:客户端 49930 向服务端 8080 发送一个请求结束的报文 (fin,ack),此时序列号 Seq=X=1332, ack=y=67873 用于关闭客户端到服务端的数据传输，与此同时，客户端进入 FIN_WAIT_1 状态。

第二次挥手: 服务端 8080 收到 fin 之后，发送 ACK 给客户端表明已经收到，seq=67873, 即对收到的 67873 前的序号进行确认，ack=X+1=1333, 服务端进入 CLOSE_WAIT 状态。

第三次挥手: 服务端 8080 端口向客户端发送一个 FIN，用于关闭服务端到客户端的数据传输，服务端进入到 LAST_ACK 状态，seq 和 ack 遇上一部分相同。

第四次握手:客户端对收到的 fin 包进行确认并进入 Time_wait 状态,seq=1333, ack=y+1=67874

3.4 遇到的问题

3.4.1 为什么要进行三次握手连接

解答:

第一次握手使得 **server 端**确认自己的**接收**报文消息功能正常;

第二次握手使得 **client 端**确认自己的**发送和接收**报文消息功能正常;

第三次握手使得 **server 端**确认自己的**接收**报文消息功能正常;

3.4.2 [TCP segment of a reassembled PDU]

解答: 要发送的数据包过大，需要分段。

3.4.3 参数的意义

对 200，304，404 等的参数意义不了解。

状态码	含义
200 OK	请求成功，被请求的对象包含在该响应的数据部分
301 Moved Permanently	请求的对象被移走，新的位置在响应中通过 Location: 给出
400 Bad Request	服务器不能解释请求报文
404 Not Found	服务器中找不到请求的文档
505 HTTP Version Not Supported	服务器不支持相应的 HTTP 版本

表 1: 状态码及其对应的含义

3.4.4 HTTP1.1 协议出现多个三次握手

根据学习的知识基于 HTTP1.1 协议建立连接后，客户端和服务端不会轻易断开，但这里出现了多个三次握手连接

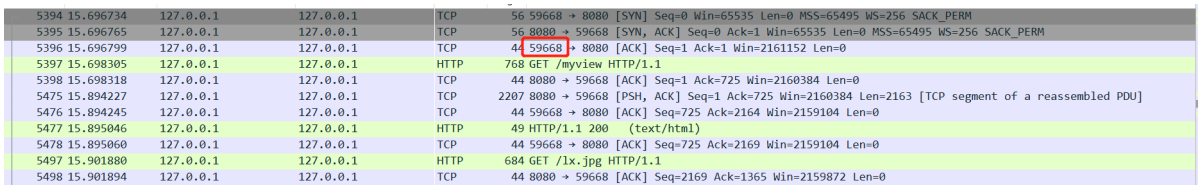

图 3.19: 多线程端口 1

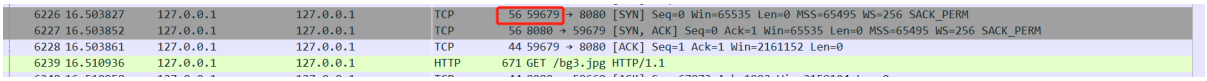

图 3.20: 多线程端口 2

解答：仔细观察端口号的差异，这是由于使用了多线程，观察客户端端口号即可知道，text 文件和 picture:lx.png 是由 59668 端口进行通信的，而 picture2：bg.png 是由 59679 进行通信的。

3.4.5 Keep-alive

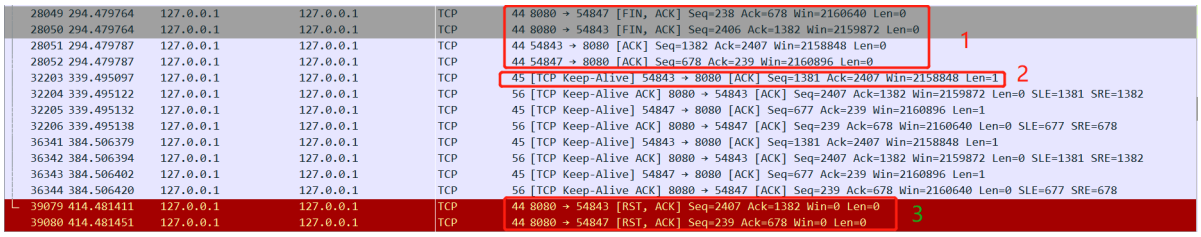

图 3.21: 保持连接

观察上图，发现服务端口 8080 由于长时间未与客户端通信，发送了请求关闭连接的请求，客户端收到服务端的请求并发送 ack(1)，但客户端此时并不想关闭连接，故客户端发送了 Keep-Alive 保持连接的请求 (2)，服务端收到保持连接的请求，但由于已经收到了客户端的 ACK，故启用 RST(3 restart) 机制，