浙江大学

本科实验报告

课程名称：	计算机网络基础
实验名称：	实现一个轻量级的 WEB 服务器
姓　　名：	张 溢 弛
学　　院：	计算机学院
系：	软件工程
专　　业：	软件工程
学　　号：	3180103772
指导教师：	邱劲松

2020 年　　12 月　　30 日

浙江大学实验报告

实验名称：实现一个轻量级的 WEB 服务器　实验类型：编程实验

同组学生：单人完成　实验地点：计算机网络实验室

一、 实验目的

深入掌握 HTTP 协议规范，学习如何编写标准的互联网应用服务器。

二、 实验内容

- 服务程序能够正确解析 HTTP 协议，并传回所需的网页文件和图片文件
- 使用标准的浏览器，如 IE、Chrome 或者 Safari，输入服务程序的 URL 后，能够正常显示服务器上的网页文件和图片
- 服务端程序界面不做要求，使用命令行或最简单的窗体即可
- 功能要求如下：
  1. 服务程序运行后监听在 80 端口或者指定端口
  2. 接受浏览器的 TCP 连接（支持多个浏览器同时连接）
  3. 读取浏览器发送的数据，解析 HTTP 请求头部，找到感兴趣的部分
  4. 根据 HTTP 头部请求的文件路径，打开并读取服务器磁盘上的文件，以 HTTP 响应格式传回浏览器。要求按照文本、图片文件传送不同的 Content-Type，以便让浏览器能够正常显示。
  5. 分别使用单个纯文本、只包含文字的 HTML 文件、包含文字和图片的 HTML 文件进行测试，浏览器均能正常显示。
- 本实验可以在前一个 Socket 编程实验的基础上继续，也可以使用第三方封装好的 TCP 类进行网络数据的收发
- 本实验要求不使用任何封装 HTTP 接口的类库或组件，也不使用任何服务端脚本程序如 JSP、ASPX、PHP 等

三、 主要仪器设备

联网的 PC 机、Wireshark 软件、Visual Studio、gcc 或 Java 集成开发环境。

四、 操作方法与实验步骤

- 阅读 HTTP 协议相关标准文档，详细了解 HTTP 协议标准的细节，有必要的话使用 Wireshark 抓包，研究浏览器和 WEB 服务器之间的交互过程
- 创建一个文档目录，与服务器程序运行路径分开
- 准备一个纯文本文件，命名为 test.txt，存放在 txt 子目录下面
- 准备好一个图片文件，命名为 logo.jpg，放在 img 子目录下面
- 写一个 HTML 文件，命名为 test.html，放在 html 子目录下面，主要内容为：

```
<html>
    <head><title>Test</title></head>
    <body>
        <h1>This is a test</h1>
        <img src="img/logo.jpg">
        <form action="dopost" method="POST">
            Login:<input name="login">
            Pass:<input name="pass">
            <input type="submit" value="login">
        </form>
    </body>
</html>
```

- 将 test.html 复制为 noimg.html，并删除其中包含 img 的这一行。
- 服务端编写步骤（**需要采用多线程模式**）

a) 运行初始化，打开 Socket，监听在指定端口（**请使用学号的后 4 位作为服务器的监听端口**）

b) 主线程是一个循环，主要做的工作是等待客户端连接，如果有客户端连接成功，为该客户端创建处理子线程。该子线程的主要处理步骤是：

1. 不断读取客户端发送过来的字节，并检查其中是否连续出现了 2 个回车换行符，如果未出现，继续接收；如果出现，按照 HTTP 格式解析第 1 行，分离出方法、文件和路径名，其他头部字段根据需要读取。

✧ **如果解析出来的方法是 GET**

2. 根据解析出来的文件和路径名，读取响应的磁盘文件（该路径和服务器程序可能不在同一个目录下，需要转换成绝对路径）。如果文件不存在，第 3 步的响应消息的状态设置为 404，并且跳过第 5 步。
3. 准备好一个足够大的缓冲区，按照 HTTP 响应消息的格式先填入第 1 行（状态码=200），加上回车换行符。然后模仿 Wireshark 抓取的 HTTP 消息，填入必要的几行头部（需要哪些头部，请试验），其中不能缺少的 2 个头部是 Content-Type 和 Content-Length。Content-Type 的值要和文件类型相匹配（请通过抓包确定应该填什么），Content-Length 的值填写文件的字节大小。
4. 在头部行填完后，再填入 2 个回车换行
5. 将文件内容按顺序填入到缓冲区后面部分。

✧ **如果解析出来的方法是 POST**

6. 检查解析出来的文件和路径名，如果不是 dopost，则设置响应消息的状态为 404，然后跳到第 9 步。如果是 dopost，则设置响应消息的状态为 200，并继续下一步。
7. 读取 2 个回车换行后面的体部内容（长度根据头部的 Content-Length 字段的指示），并提取出登录名（login）和密码（pass）的值。如果登录名是你的学号，密码是学号的后 4 位，则将响应消息设置为登录成功，否则将响应消息设置为登录失败。
8. 将响应消息封装成 html 格式，如

<html><body>响应消息内容</body></html>

9. 准备好一个足够大的缓冲区，按照 HTTP 响应消息的格式先填入第 1 行（根据前面的情况设置好状态码），加上回车换行符。然后填入必要的几行头部，其中不能缺少的 2 个头部是 Content-Type 和 Content-Length。Content-Type 的值设置为 text/html，如果状态码=200，则 Content-Length 的值填写响应消息的字节大小，并将响应消息填入缓冲区的后面部分，否则填写为 0。
10. 最后一次性将缓冲区内的字节发送给客户端。
11. 发送完毕后，关闭 socket，退出子线程。

c) 主线程还负责检测退出指令（如用户按退出键或者收到退出信号），检测到后即通知并等待各子线程退出。最后关闭 Socket，主程序退出。

- 编程结束后，将服务器部署在一台机器上（本机也可以）。在服务器上分别放置纯文本文件（.txt）、只包含文字的测试 HTML 文件（将测试 HTML 文件中的包含 img 那一行去掉）、包含文字和图片的测试 HTML 文件（以及图片文件）各一个。
- 确定好各个文件的 URL 地址，然后使用浏览器访问这些 URL 地址，如 http://x.x.x.x:port/dir/a.html，其中 port 是服务器的监听端口，dir 是提供给外部访问的路径，请设置为与文件实际存放路径不同，通过服务器内部映射转换。
- 检查浏览器是否正常显示页面，如果有问题，查找原因，并修改，直至满足要求
- 使用多个浏览器同时访问这些 URL 地址，检查并发性

五、 实验数据记录和处理

请将以下内容和本实验报告一起打包成一个压缩文件上传：

- **源代码：需要说明编译环境和编译方法，如果不能编译成功，将影响评分**
- **可执行文件：可运行的.exe 文件或 Linux 可执行文件**

以下实验记录均需结合屏幕截图（截取源代码或运行结果），进行文字标注（看完请删除本句）。

本实验代码采用 Java+JDK1.8 写成

- 服务器的主线程循环关键代码截图（解释总体处理逻辑，省略细节部分）

```
public class main {
    public static void main(String[] args) throws IOException {
        ServerSocket serverSocket = new ServerSocket( port: 3772);
        while (true) {
            Socket socket = serverSocket.accept();
            new WebServer(socket,  path: "./static").start();
        }
    }
}
```

主线程的循环代码主要是利用 Java 的 ServerSocket 初始化并监听 3772 端口，一旦监听

到了新的 HTTP 请求，就新建一个 WebServer 类的对象来处理 HTTP 请求，而 WebServer 类采用了多线程的方式来实现。

- 服务器的客户端处理子线程关键代码截图（解释总体处理逻辑，省略细节部分）

```
public class WebServer extends Thread {

    private InputStream input;
    private OutputStream output;
    private String path;
    private static final int BUFFER_SIZE = 4096;
    private static final String account = "3180103772";
    private static final String password = "3772";
    private Request request;

    public WebServer(Socket socket, String path) throws IOException {
        this.input = socket.getInputStream();
        this.output = socket.getOutputStream();
        this.path = path;
    }
```

```
@Override
public void run() {
    System.out.println(Thread.currentThread().getName() + " start!");
    try {
        this.request = HTTPParser();
        if (request.getRequest().equals("GET")) {
            System.out.println("A GET Request!!!");
            // 处理get请求
            responseGetRequest();
        } else if (request.getRequest().equals("POST")) {
            System.out.println("A POST Request!!!");
            // 处理post请求
            responsePostRequest();
        } else {
            System.out.println("Unknown Request!!");
            // 本简易web服务器不提供其他的HTTP请求的服务
        }
        System.out.println(Thread.currentThread().getName() + " End!");
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

子线程通过继承 Thread 类的方式实现了多线程的运行模式，通过一个 socket 对象和资源所在的路径来初始化，并且覆写了父类 Thread 类中的 run 方法，首先解析 HTTP 请求的头部，并对 GET 和 POST 以及其他类型分别做不同的处理

- 服务器运行后，用 netstat –an 显示服务器的监听端口

  我的学号后面四位是 3772，本次作业中学号后四位作为监听端口，打开服务器之后，在 cmd 中运行 netstat –an 的结果如下：

  （注：下面这张是截图，不是复制的文本框，因为我的 cmd 设置成了灰色）

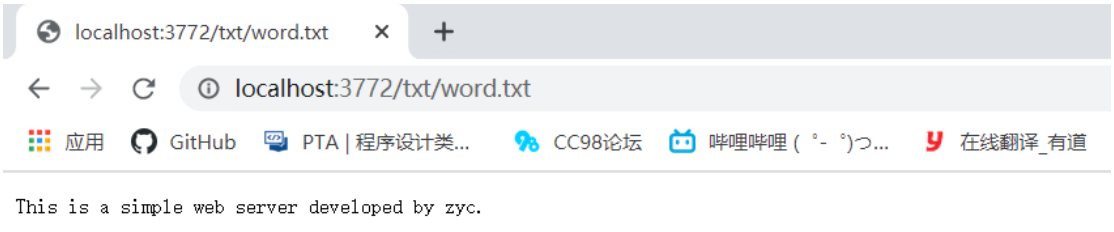
```
C:\Users\74096>netstat -an

活动连接

  协议   本地地址          外部地址        状态
  TCP    0.0.0.0:135            0.0.0.0:0              LISTENING
  TCP    0.0.0.0:443            0.0.0.0:0              LISTENING
  TCP    0.0.0.0:445            0.0.0.0:0              LISTENING
  TCP    0.0.0.0:903            0.0.0.0:0              LISTENING
  TCP    0.0.0.0:913            0.0.0.0:0              LISTENING
  TCP    0.0.0.0:3306           0.0.0.0:0              LISTENING
  TCP    0.0.0.0:3772           0.0.0.0:0              LISTENING
  TCP    0.0.0.0:5040           0.0.0.0:0              LISTENING
  TCP    0.0.0.0:7680           0.0.0.0:0              LISTENING
  TCP    0.0.0.0:33060          0.0.0.0:0              LISTENING
  TCP    0.0.0.0:49664          0.0.0.0:0              LISTENING
  TCP    0.0.0.0:49665          0.0.0.0:0              LISTENING
  TCP    0.0.0.0:49666          0.0.0.0:0              LISTENING
  TCP    0.0.0.0:49667          0.0.0.0:0              LISTENING
  TCP    0.0.0.0:49668          0.0.0.0:0              LISTENING
  TCP    0.0.0.0:49669          0.0.0.0:0              LISTENING
  TCP    10.185.229.23:139      0.0.0.0:0              LISTENING
```

  我们发现 TCP 协议的 3772 端口正在进行监听

- 浏览器访问纯文本文件（.txt）时，浏览器的 URL 地址和显示内容截图。

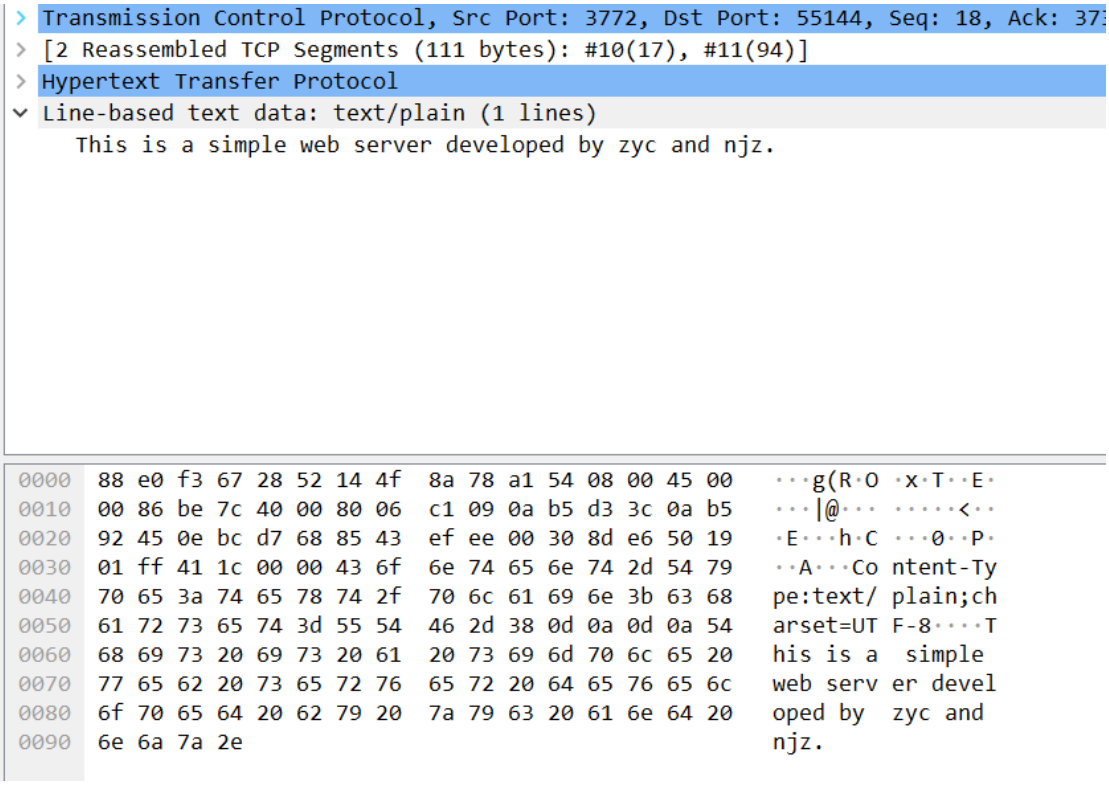


服务器上文件实际存放的路径：

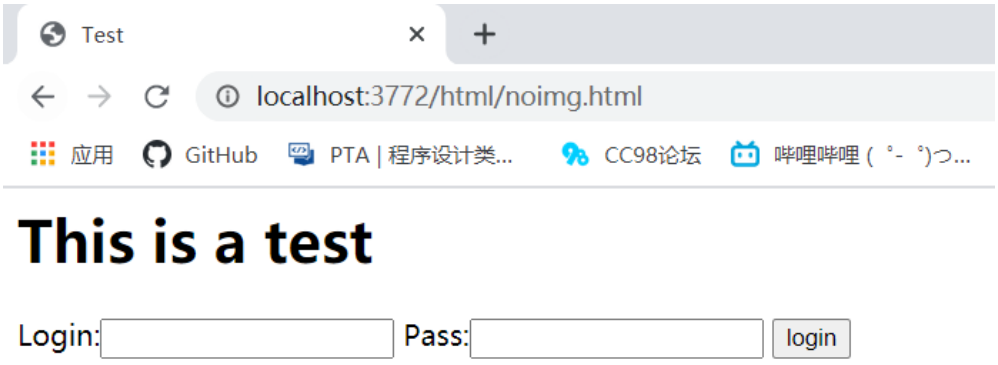
服务器的相关代码片段：

```
} else {
    // 传输的是纯文本时候的情况
    output.write("Content-Type:text/plain;charset=UTF-8\r\n".getBytes());
}
output.write("\r\n".getBytes());
// 写入文件中的数据
while ((len = fileInput.read(buffer)) != -1) {
    output.write(buffer, off: 0, len);
}
```

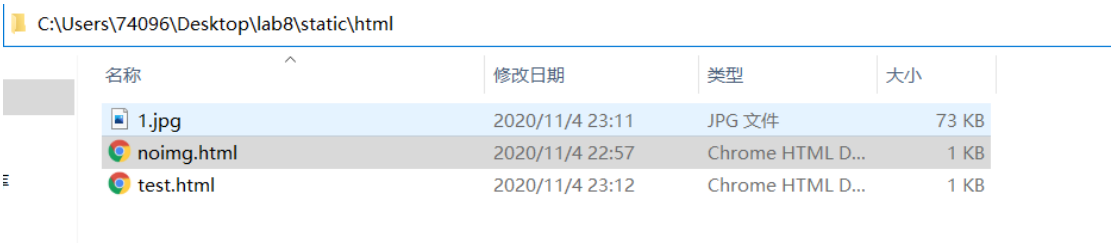
Wireshark 抓取的数据包截图（通过跟踪 TCP 流，只截取 HTTP 协议部分）：



- 浏览器访问只包含文本的 HTML 文件时，浏览器的 URL 地址和显示内容截图。

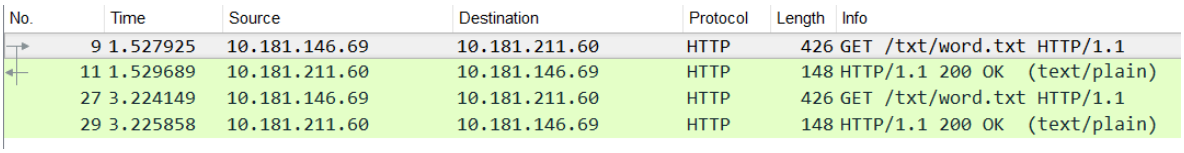


服务器文件实际存放的路径：



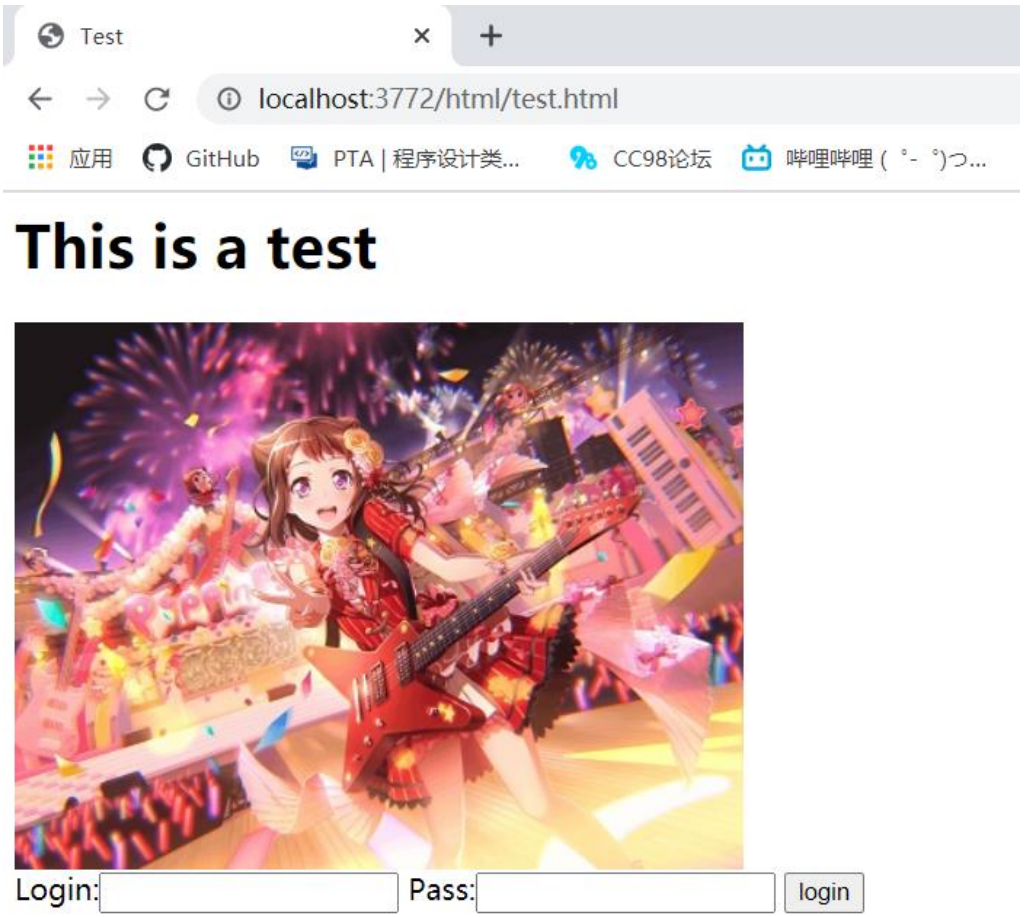
Wireshark 抓取的数据包截图（只截取 HTTP 协议部分，包括 HTML 内容）：

这是在 wireshark 中过滤 HTTP 协议之后得到的内容，可以很清楚地看到 HTTP 的 GET 请求

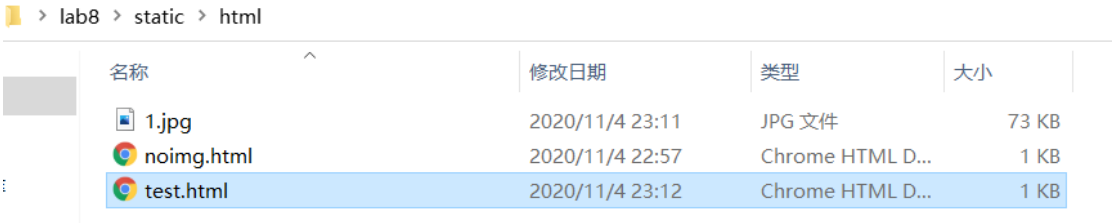


No.	Time	Source	Destination	Protocol	Length	Info
9	1.527925	10.181.146.69	10.181.211.60	HTTP	426	GET /txt/word.txt HTTP/1.1
11	1.529689	10.181.211.60	10.181.146.69	HTTP	148	HTTP/1.1 200 OK (text/plain)
27	3.224149	10.181.146.69	10.181.211.60	HTTP	426	GET /txt/word.txt HTTP/1.1
29	3.225858	10.181.211.60	10.181.146.69	HTTP	148	HTTP/1.1 200 OK (text/plain)

- 浏览器访问包含文本、图片的 HTML 文件时，浏览器的 URL 地址和显示内容截图。

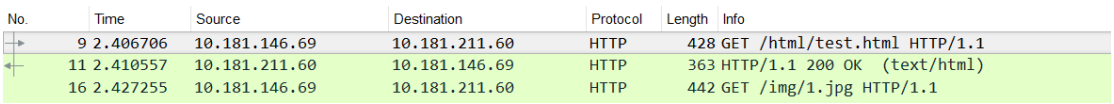


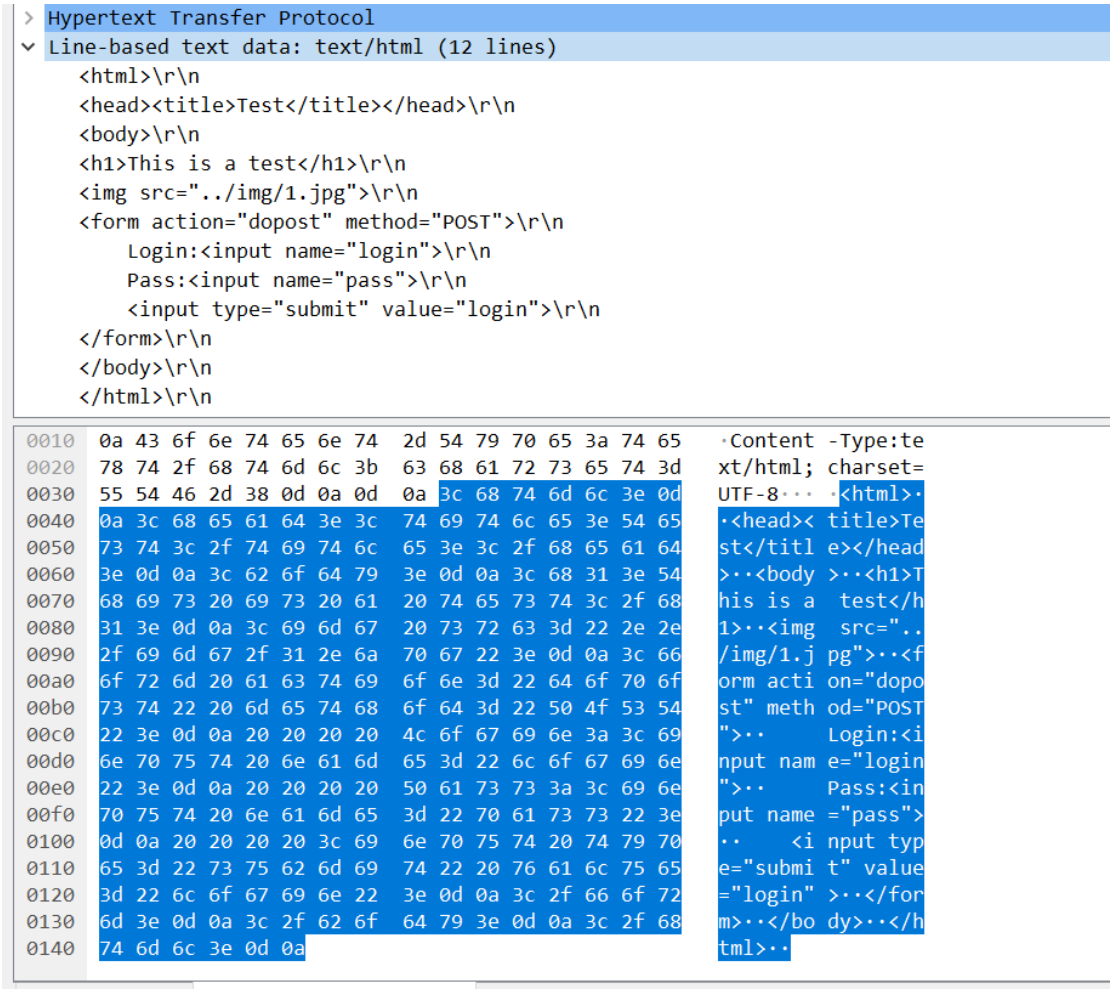
服务器上文件实际存放的路径：



Wireshark 抓取的数据包截图（只截取 HTTP 协议部分，包括 HTML、图片文件的部分内容）：

发现服务器发送了 html 文件的同时也发送的一张 jpg 图片





- 浏览器输入正确的登录名或密码，点击登录按钮（login）后的显示截图。

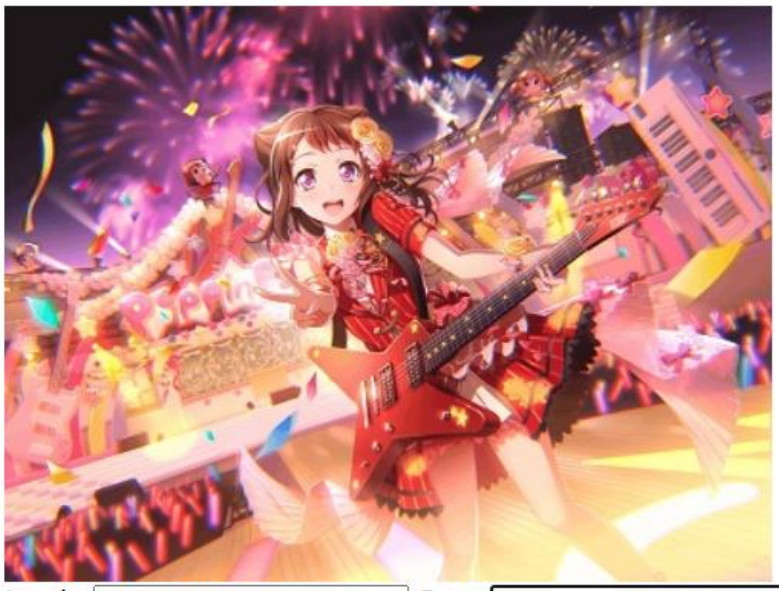
先输入账号和密码进行登录

Test

localhost:3772/html/test.html

应用 GitHub PTA | 程序设计类... CC98论坛 哔哩哔哩 (゜- ゜)つ

This is a test



Login: 3180103772 Pass: 3772 login

localhost:3772/html/dopost

localhost:3772/html/dopost

应用 GitHub PTA | 程序设计类... CC98论坛 哔哩哔哩 (゜- ゜)つ...

Login Success!

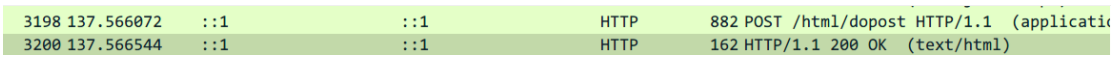
服务器相关处理代码片段：

```
public void responsePostRequest() throws IOException {
    System.out.println(request.getId());
    System.out.println(request.getPassword());
    System.out.println(request.getPath());
    if (request.getPath().equals("/html/dopost") && !request.getId().isEmpty()
            && !request.getPassword().isEmpty()) {
        String message;
        if (request.getId().equals(account) && request.getPassword().equals(password)) {
            message = "<h1>Login Success!</h1>";
        } else {
            message = "<h1>Login Fail!</h1>";
        }
        String result = "HTTP/1.1 200 OK\r\n" + "Content-Type:text/html;charset=UTF-8\n" + "Content-Length:"
                + message.length() + "\r\n" + "\r\n" + message;
        output.write(result.getBytes());
```

这里服务器会先检测要访问的路径、输入的账号密码是否匹配，再返回相应的内容，登陆成功、账号密码错误和未填写密码或者路径错误看到的是不同的内容。

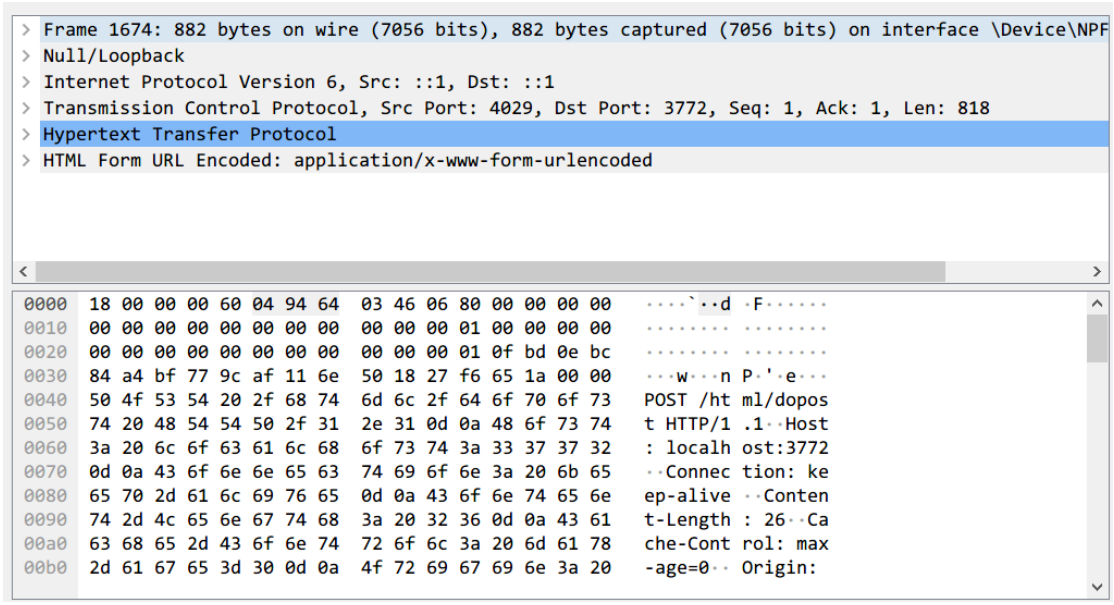
Wireshark 抓取的数据包截图（HTTP 协议部分）

请求数据包和响应数据包



POST 请求数据包的具体内容

这张图截的不太好，没有标出账号密码，不过下一节登陆失败的这一部分有类似的内容

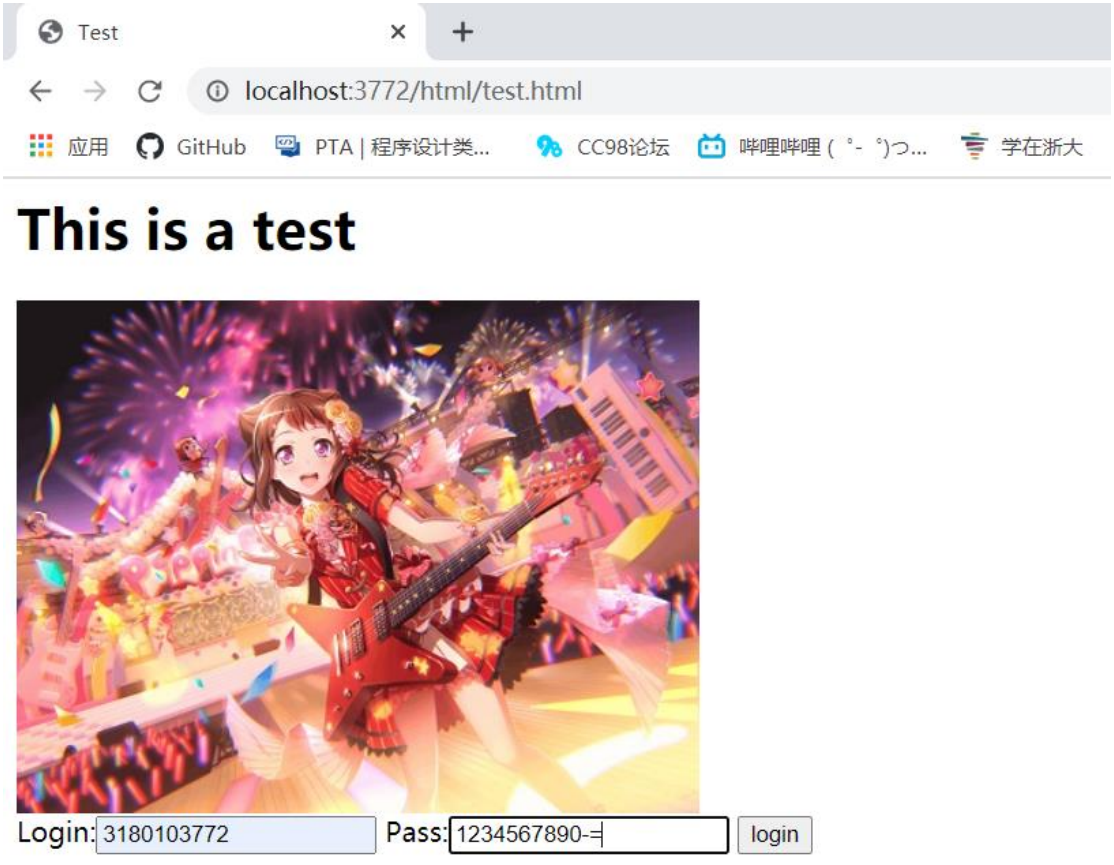


响应数据包的具体内容

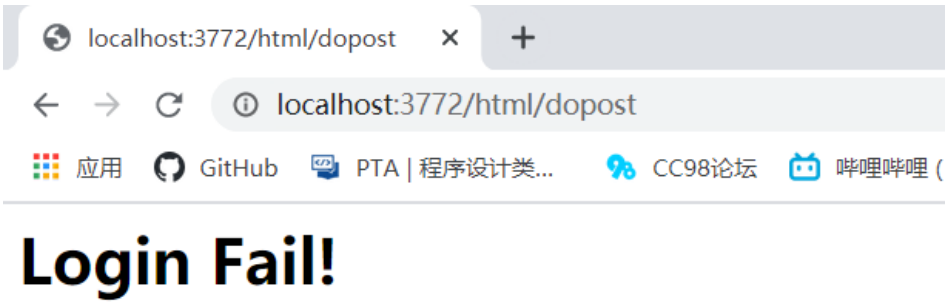
```
> Frame 3200: 162 bytes on wire (1296 bits), 162 bytes captured (1296 bits) on interface \Device\NPF
> Null/Loopback
> Internet Protocol Version 6, Src: ::1, Dst: ::1
> Transmission Control Protocol, Src Port: 3772, Dst Port: 16144, Seq: 1, Ack: 819, Len: 98
> Hypertext Transfer Protocol
v Line-based text data: text/html (1 lines)
    <h1>Login Success!</h1>
```

```
0000  18 00 00 00 60 08 e8 46  00 76 06 80 00 00 00 00   ····`··F ·v······
0010  00 00 00 00 00 00 00 00  00 00 00 01 00 00 00 00   ···· ···· ····
0020  00 00 00 00 00 00 00 00  00 00 00 01 0e bc 3f 10   ···· ···· ····?·
0030  ab 1f 54 c0 6d 5d 9c cd  50 18 27 f6 ea 54 00 00   ··T·m]·· P·'··T··
0040  48 54 54 50 2f 31 2e 31  20 32 30 30 20 4f 4b 0d   HTTP/1.1  200 OK·
0050  0a 43 6f 6e 74 65 6e 74  2d 54 79 70 65 3a 74 65   ·Content -Type:te
0060  78 74 2f 68 74 6d 6c 3b  63 68 61 72 73 65 74 3d   xt/html; charset=
0070  55 54 46 2d 38 0a 43 6f  6e 74 65 6e 74 2d 4c 65   UTF-8·Co ntent-Le
0080  6e 67 74 68 3a 32 33 0d  0a 0d 0a 3c 68 31 3e 4c   ngth:23· ···<h1>L
0090  6f 67 69 6e 20 53 75 63  63 65 73 73 21 3c 2f 68   ogin Suc cess!</h
00a0  31 3e                                             1>
```

- 浏览器输入错误的登录名或密码，点击登录按钮（login）后的显示截图。

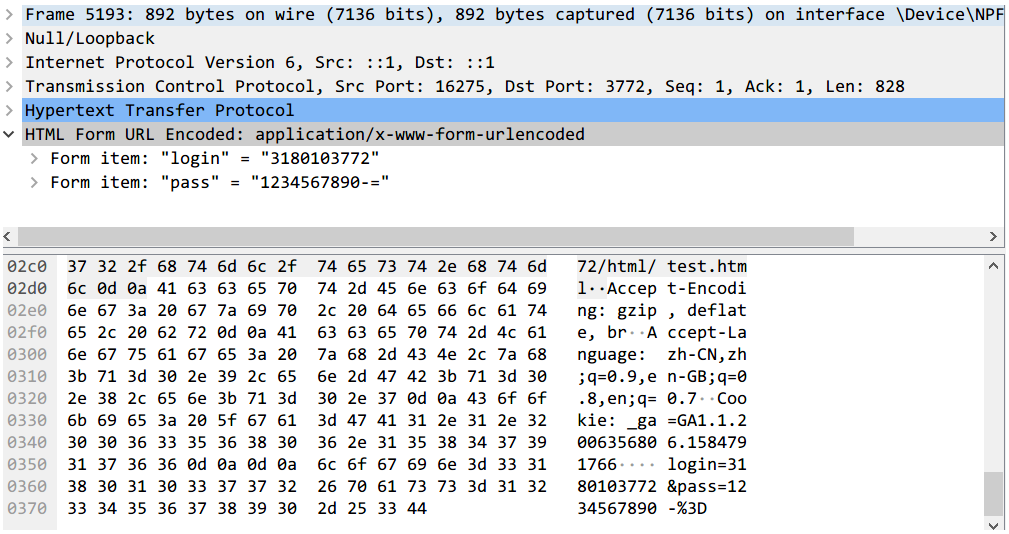


登陆后的结果



- Wireshark 抓取的数据包截图（HTTP 协议部分）

  请求数据包的具体内容，发现 POST 包中有账号密码



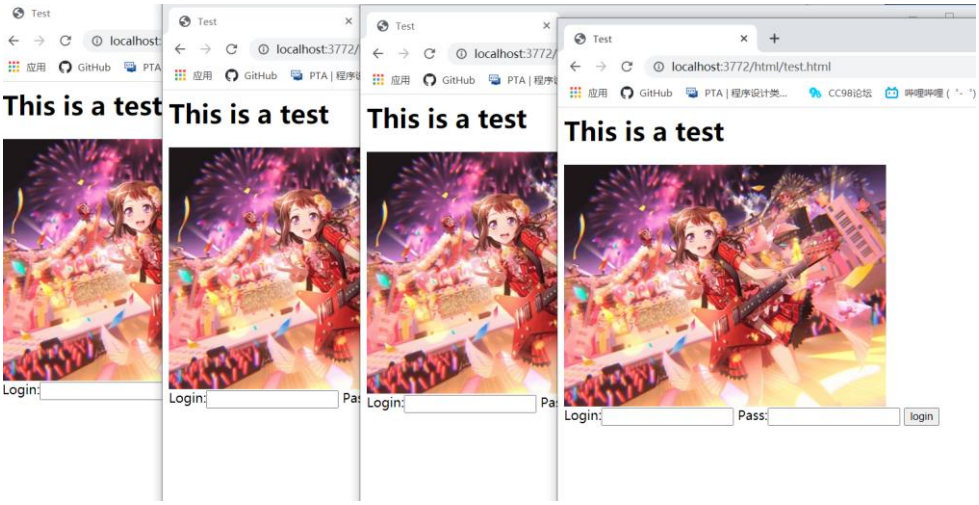
响应数据包的具体内容，返回了 Login Fail

```
> Frame 5195: 159 bytes on wire (1272 bits), 159 bytes captured (1272 bits) on interface \Device\NPF
> Null/Loopback
> Internet Protocol Version 6, Src: ::1, Dst: ::1
> Transmission Control Protocol, Src Port: 3772, Dst Port: 16275, Seq: 1, Ack: 829, Len: 95
> Hypertext Transfer Protocol
v Line-based text data: text/html (1 lines)
    <h1>Login Fail!</h1>
```

```
0000  18 00 00 00 60 0d 7f 1d  00 73 06 80 00 00 00 00   ····`··· ·s·····
0010  00 00 00 00 00 00 00 00  00 00 00 01 00 00 00 00   ········ ········
0020  00 00 00 00 00 00 00 00  00 00 00 01 0e bc 3f 93   ········ ·····?·
0030  d3 91 0d c1 fd 8a 47 63  50 18 27 f6 ee de 00 00   ·····Gc P·'·····
0040  48 54 54 50 2f 31 2e 31  20 32 30 30 20 4f 4b 0d   HTTP/1.1  200 OK·
0050  0a 43 6f 6e 74 65 6e 74  2d 54 79 70 65 3a 74 65   ·Content -Type:te
0060  78 74 2f 68 74 6d 6c 3b  63 68 61 72 73 65 74 3d   xt/html; charset=
0070  55 54 46 2d 38 0a 43 6f  6e 74 65 6e 74 2d 4c 65   UTF-8·Co ntent-Le
0080  6e 67 74 68 3a 32 30 0d  0a 0d 0a 3c 68 31 3e 4c   ngth:20· ···<h1>L
0090  6f 67 69 6e 20 46 61 69  6c 21 3c 2f 68 31 3e      ogin Fai l!</h1>
```

- 多个浏览器同时访问包含图片的 HTML 文件时，浏览器的显示内容截图（将浏览器窗口缩小并列）

  下图证明服务器是多线程的程序，支持并发的访问



- 多个浏览器同时访问包含图片的 HTML 文件时，使用 netstat –an 显示服务器的 TCP 连接（截取与服务器监听端口相关的）

有一部分端口的状态是 LISTENING,有一部分是 FIN_WAIT

```
活动连接

  协议   本地地址          外部地址          状态
  TCP    0.0.0.0:135      0.0.0.0:0        LISTENING
  TCP    0.0.0.0:443      0.0.0.0:0        LISTENING
  TCP    0.0.0.0:445      0.0.0.0:0        LISTENING
  TCP    0.0.0.0:903      0.0.0.0:0        LISTENING
  TCP    0.0.0.0:913      0.0.0.0:0        LISTENING
  TCP    0.0.0.0:3306     0.0.0.0:0        LISTENING
  TCP    0.0.0.0:3772     0.0.0.0:0        LISTENING
  TCP    0.0.0.0:5040     0.0.0.0:0        LISTENING
  TCP    0.0.0.0:7680     0.0.0.0:0        LISTENING
  TCP    0.0.0.0:33060    0.0.0.0:0        LISTENING
  TCP    0.0.0.0:49664    0.0.0.0:0        LISTENING
  TCP    0.0.0.0:49665    0.0.0.0:0        LISTENING
  TCP    0.0.0.0:49666    0.0.0.0:0        LISTENING
  TCP    0.0.0.0:49667    0.0.0.0:0        LISTENING
  TCP    0.0.0.0:49668    0.0.0.0:0        LISTENING
  TCP    0.0.0.0:49669    0.0.0.0:0        LISTENING

  TCP    [::1]:57059      [::1]:3772       FIN_WAIT_2
  TCP    [::1]:63122      [::1]:3772       FIN_WAIT_2
```

六、 实验结果与分析

根据你编写的程序运行效果，分别解答以下问题（看完请删除本句）：

- HTTP 协议是怎样对头部和体部进行分隔的？

  HTTP 协议通过两个空行\n 来对 head 和 body 进行分隔

- 浏览器是根据文件的扩展名还是根据头部的哪个字段判断文件类型的？

  浏览器是根据头部的 Content-Type 来判断文件类型的。主要的类型有 HTML 文件，纯文本，XML 文件， jpg 图片，png 图片等等格式

- HTTP 协议的头部是不是一定是文本格式？体部呢？

  HTTP 协议的头部一定是文本格式，但是 body 在传输音视频的时候可以是字节流的形式，而 Java 的 socket 中也需要以字节流的方式写入

- POST 方法传递的数据是放在头部还是体部？两个字段是用什么符号连接起来的？

  放在 body 中，用&将多个字段连接起来

七、 讨论、心得

本次实验是一个比较友好的编程实验，相比于之前的 lab3-6 更像是一个计算机学院课程的 lab 了，我们主要采用了 Java 和 Java 自带的 socket 库来进行编程，但是我们在整个实验的过程中也遇到了不少困难，比如：

- 对 Java 的 Socket API 不熟悉，经常出现 flush 之后忘记 close 的情况，导致测试的时候发现网页一直处于加载状态，这是因为 socket 通信如果没有 close 就会一直尝试收发数据，导致网页还在不断加载
- 解析 HTTP 请求的时候对请求的格式不太熟悉，尤其是处理 head 和 body 的空行的时候花了一点时间
- wireshark 是抓不到本地回路的数据包的，经过老师提醒，可以用两台电脑连接同一个 WLAN，就可以用另一台电脑的浏览器向服务器发送 HTTP 请求，这时候的数据包是可以被捕捉到的。