

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：

学 院： 计算机学院

专 业： 计算机科学与技术

学 号：

指导教师： 陆魁军

2022 年 12 月 18 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: _____ 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
- 准备好一个图片文件, 命名为 logo.jpg, 放在 img 子目录下
- 写一个 HTML 文件, 命名为 test.html, 放在 html 子目录下, 主要内容为:

```
<html>
  <head><title>Test</title></head>
  <body>
    <h1>This is a test</h1>
    
    <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（该路径和服务端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）。

- 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）

在主线程中通过 while 语句不断循环，持续监听客户端的连接请求，如果有连接请求就创建一个子线程进行处理。

```
public class Main {  
    Run | Debug  
    public static void main(String[] args) {  
        try {  
            ServerSocket serverSocket = new ServerSocket(port: 2635);  
            while (true) {  
                Socket socket = serverSocket.accept();  
                new HttpServer(socket, root: "D:/net/").start();  
            }  
        } catch (Exception e) {  
            e.printStackTrace();  
        }  
    }  
}
```

- 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）

设计的 `HttpServer` 类继承了 `Thread` 类，在构造函数中打开输入流和输出流，并将服务器文件存储的根目录赋值给 `root`。

```
public HttpServer(Socket socket, String root) {
    try {
        req = socket.getInputStream();
        res = socket.getOutputStream();
        this.root = root;
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

之后重写了 `run` 方法，主要逻辑式通过 `GetRequestHeader` 函数来解析请求头，获得一个 `Request` 对象，然后根据 `Request` 中记录的属性来决定调用 `GET` 或者 `POST` 的处理函数进行处理。其他的 `System.out.println` 函数在服务器端输出，主要用来调试。

```
@Override
public void run() {
    try {
        System.out.println(Thread.currentThread().getName() + "start processing request...");
        Request request = getRequestHeader();
        if (request.getMethod() == Request.Method.GET)
            responseGET(request.getURL());
        else if (request.getMethod() == Request.Method.POST)
            responsePOST(request);
        else
            throw new ParseException("Method" + request.getMethod() + "not supported", errorOffset: 0);
        System.out.println(Thread.currentThread().getName() + "execute finish.");
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

`Request` 类的结构如下所示，主要包含 `URL`、`RequestMethod` 和 `params` 三个属性，`URL` 用来记录 `URL` 地址，`RequestMethod` 是枚举类型，分为 `GET` 和 `POST` 两个值，`params` 采用 `hashmap` 来存储一些键值对（对应于后面 `login`, `password` 的处理）

```

public class Request {
    public enum Method {
        GET, POST
    }

    private String URL;
    private Method RequestMethod;
    public Map<String, String> params;
    public Request(String URL, Method mythod) {
        this.URL = URL;
        this.RequestMethod = mythod;
        this.params = new HashMap<>();
    }

    public String getURL() {
        return URL;
    }

    public Method getMethod() {
        return RequestMethod;
    }
}

```

处理 GET 请求的函数 responseGET 如下所示，第二行的打开了请求路径 root 对应的文件，在输出流首先写入 HTTP/1.1 200 OK 表示读取成功。然后根据文件的后缀名判断文件类型以决定写入输出流的 content-type 等内容的类型。之后将文件中的内容转换为字节流传给输出流。另外针对文件不存在的情况也进行了处理，写入 404 not found。

```

private void responseGET(String filepath) throws IOException {
    File file = new File(root, filepath);
    System.out.println("An user request for file: " + file.getName());
    if (!filepath.equals("/") && file.exists()) {
        String extension;
        InputStream in = new FileInputStream(file);
        byte[] bytes = new byte[BUFFER_SIZE];
        int len;
        res.write("HTTP/1.1 200 OK\r\n".getBytes());
        int index = filepath.lastIndexOf(ch: '.');
        extension = filepath.substring(index);
        if (extension.equals(anObject: ".jpg")) {
            res.write("content-type:image/jpg\r\n".getBytes());
        } else if (extension.equals(anObject: ".html")) {
            res.write("content-type:text/html;charset=UTF-8\r\n".getBytes());
        } else {
            res.write("content-type:text/plain;charset=UTF-8\r\n".getBytes());
        }
        res.write("\r\n".getBytes());
        while ((len = in.read(bytes)) != -1) {
            res.write(bytes, off: 0, len);
        }
        res.flush();
        res.close();
        in.close();
    } else { // file not exist
        String err = "HTTP/1.1 404 file not found\r\n" + "Content-Type:text/html;charset=UTF-8\r\n" +
            "Content-Length:22\r\n" + "\r\n" + "<h1>404 not found</h1>";
        res.write(err.getBytes());
        res.flush();
        res.close();
    }
}

```

处理 POST 请求的函数为 responsePOST，如下所示。首先会判断 URL 是否是/dopost

以及 login 和 pass 字段是否存在, 如果不存在会进入 404 界面。如果存在会根据输入的 login 和 pass 和用户名密码是否匹配, 如果匹配输出 login success, 不匹配则输出 login fail。

```
private void responsePOST(Request request) throws IOException {
    if (request.getURL().equals(anObject: "/dopost") && request.params.containsKey(key: "login") && request.params
        String msg;
        String login = (String) request.params.get(key: "login");
        String pass = (String) request.params.get(key: "pass");
        if (login.equals(user) && pass.equals(password)) {
            msg = "<html><head><meta charset=\"UTF-8\"></head><body>login success!</body></html>";
        } else {
            msg = "<html><head><meta charset=\"UTF-8\"></head><body>login fail!</body></html>";
        }
        msg = "HTTP/1.1 200 OK\r\n" + "Content-Type:text/html \r\n" +
            "Content-Length:" + msg.length() + "\r\n" + "\r\n" + msg;
        res.write(msg.getBytes());
        res.flush();
        res.close();
    } else {
        String err = "HTTP/1.1 404 file not found \r\n" + "Content-Type:text/html \r\n" +
            "Content-Length:22 \r\n" + "\r\n" + "<h1>404 not found</h1>";
        res.write(err.getBytes());
        res.flush();
        res.close();
    }
}
```

- 服务器运行后, 用 netstat -an 显示服务器的监听端口

这里可以看出 2635 端口处于监听状态。

PS D:\javacode\webserver> 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	0.0.0.0:445	0.0.0.0:0	LISTENING
TCP	0.0.0.0:902	0.0.0.0:0	LISTENING
TCP	0.0.0.0:912	0.0.0.0:0	LISTENING
TCP	0.0.0.0:2635	0.0.0.0:0	LISTENING
TCP	0.0.0.0:5040	0.0.0.0:0	LISTENING
TCP	0.0.0.0:6646	0.0.0.0:0	LISTENING
TCP	0.0.0.0:7680	0.0.0.0:0	LISTENING
TCP	0.0.0.0:11200	0.0.0.0:0	LISTENING
TCP	0.0.0.0:16422	0.0.0.0:0	LISTENING

- 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。

URL 为 localhost:2635/txt/test.txt, 显示的内容为 hello world



服务器上文件实际存放的路径:

实际存放在 D:\net\txt\test.txt 的位置, 内容为 hello world

此电脑 > Data (D:) > net > txt



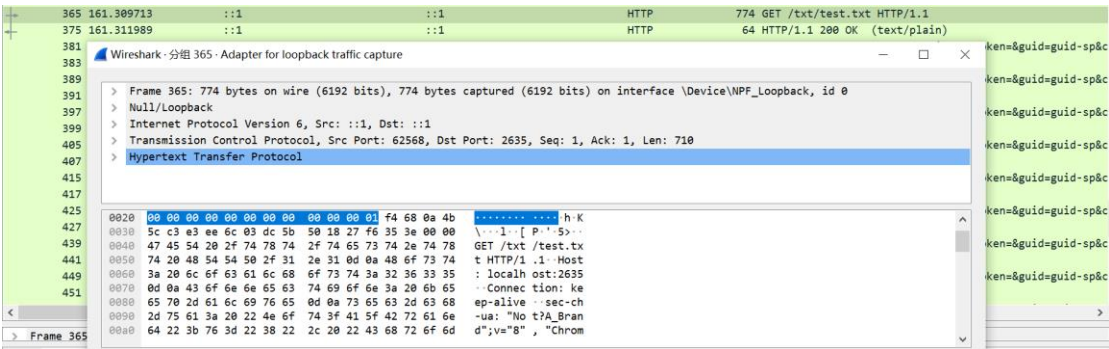
服务器的相关代码片段:

对应的代码为 responseGET 函数中的内容, 判断后缀不属于 .jpg 或者 .html, 所以属

于 else 的部分, 另外后面的 while 循环则是输出文件中所有内容

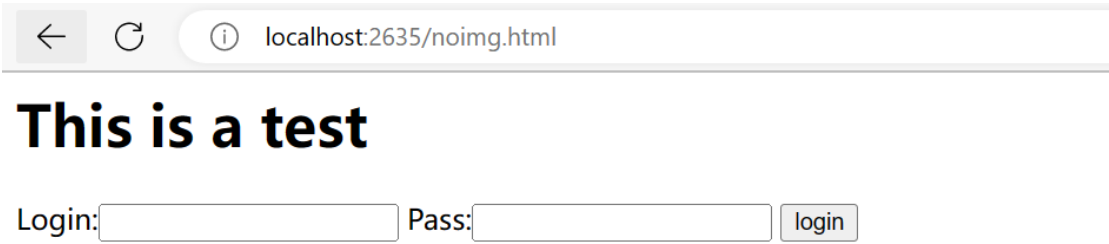
```
private void responseGET(String filepath) throws IOException {
    File file = new File(root, filepath);
    System.out.println("An user request for file: " + file.getName());
    if (!filepath.equals(anObject: "/") && file.exists()) {
        String extension;
        InputStream in = new FileInputStream(file);
        byte[] bytes = new byte[BUFFER_SIZE];
        int len;
        res.write("HTTP/1.1 200 OK\r\n".getBytes());
        int index = filepath.lastIndexOf(ch: '.');
        extension = filepath.substring(index);
        if (extension.equals(anObject: ".jpg")) {
            res.write("content-type:image/jpg \r\n".getBytes());
        } else if (extension.equals(anObject: ".html")) {
            res.write("content-type:text/html;charset=UTF-8\r\n".getBytes());
        } else {
            res.write("content-type:text/plain;charset=UTF-8\r\n".getBytes());
        }
        res.write("\r\n".getBytes());
        while ((len = in.read(bytes)) != -1) {
            res.write(bytes, 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）：



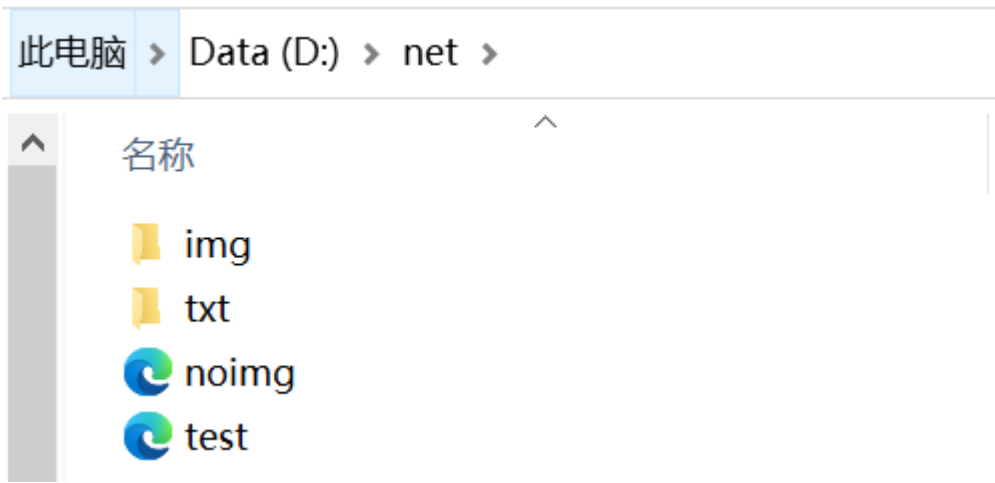
- 浏览器访问只包含文本的 HTML 文件时，浏览器的 URL 地址和显示内容截图。

URL 的内容为 localhost:2635/noimg.html

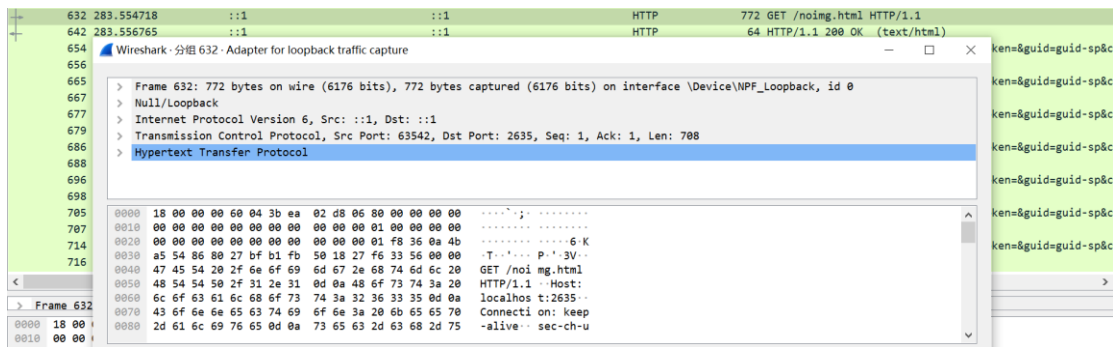


服务器文件实际存放的路径：

实际存储在 D:\net\noimg.html 的位置



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



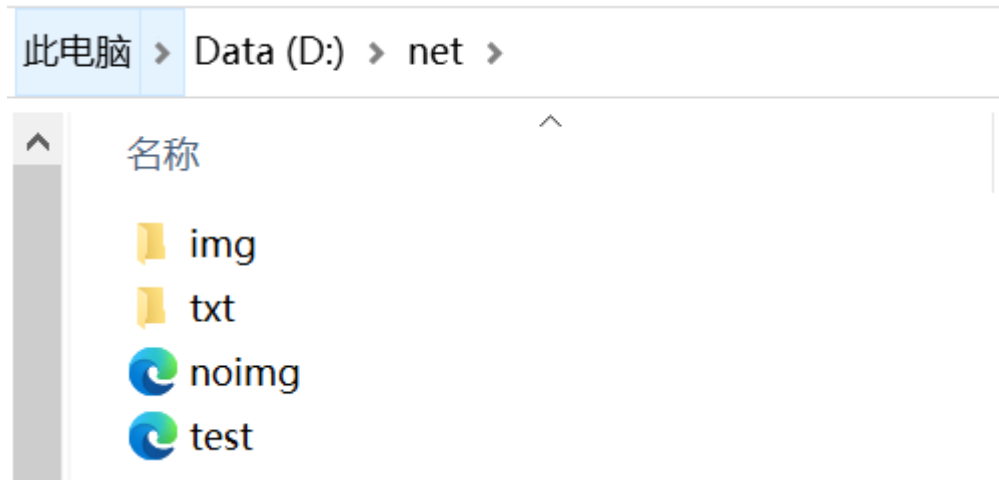
- 浏览器访问包含文本、图片的 HTML 文件时，浏览器的 URL 地址和显示内容截图。

URL 为 localhost:2635/test.html

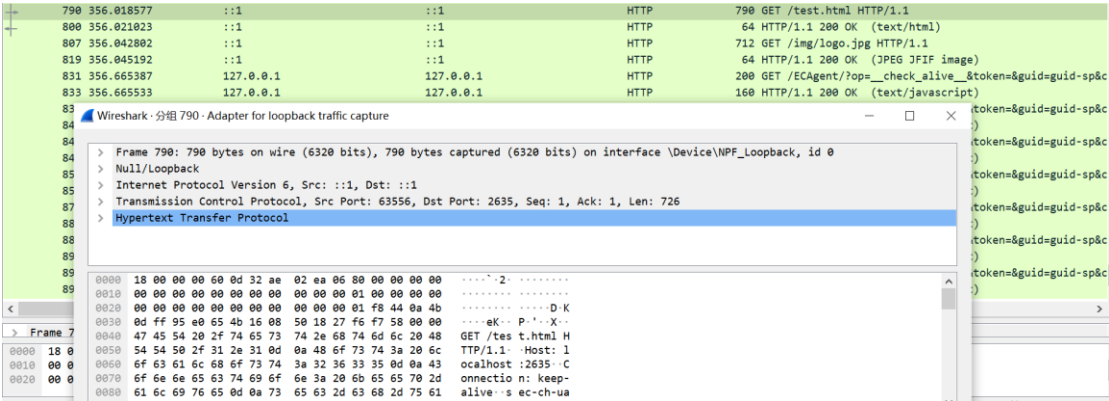


服务器上文件实际存放的路径：

实际存储在 D:\net\test.html 的位置

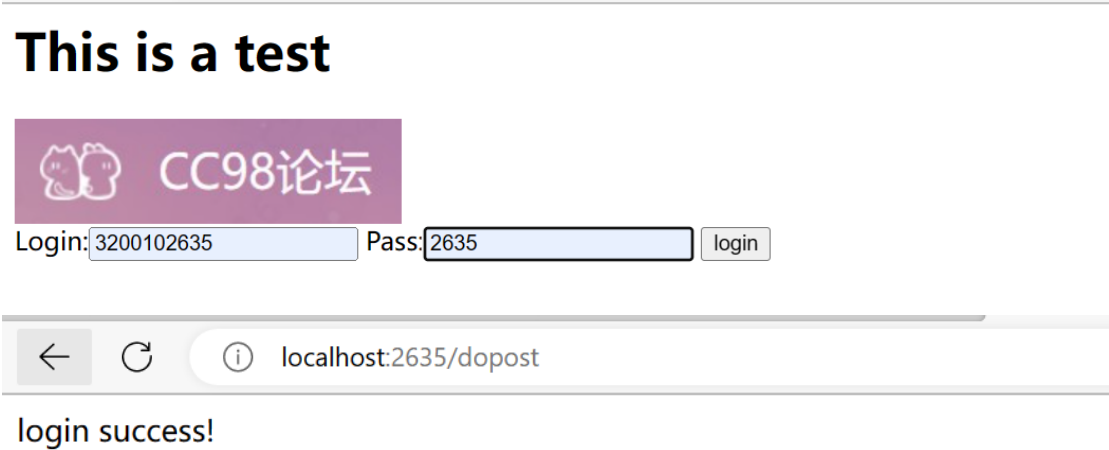


Wireshark 抓取的数据包截图（只截取 HTTP 协议部分，包括 HTML、图片文件的部分内容）：



- 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。

登录成功之后显示 login success！



服务器相关处理代码片段：

在 `getRequestHeader` 中可以解析 POST 请求中的键值对。后面的 `responsePOST` 作用前面已经说明。

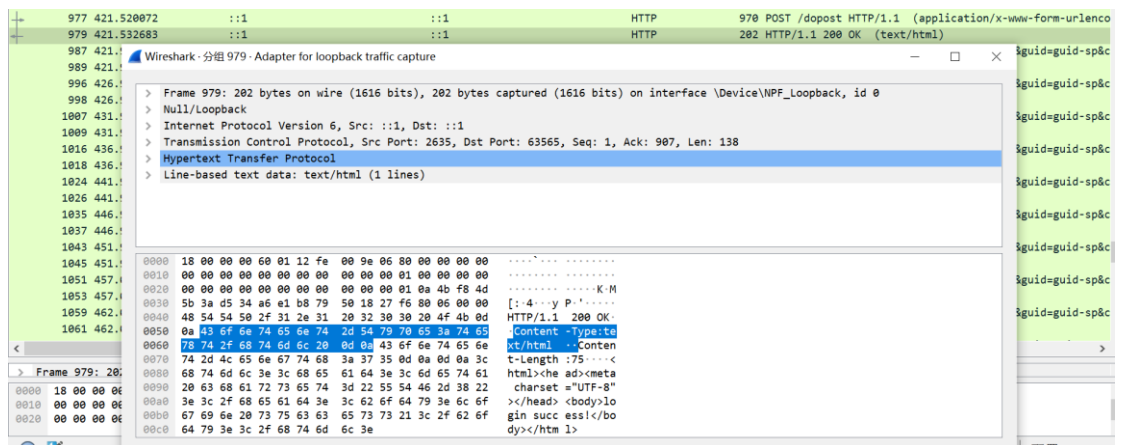
```

private Request getRequestHeader() throws IOException, ParseException, NumberFormatException {
    BufferedReader reader = new BufferedReader(new InputStreamReader(req));
    String line = reader.readLine();
    String[] list = line.split(regex: " ");
    if (list.length != 3)
        throw new ParseException(s: "HTTP RequestHeader parse error", errorOffset: 0);
    Request request = new Request(list[1], Request.Method.valueOf(list[0].trim())); // trim: delete blank at head
    if (request.getMethod() == Request.Method.POST) {
        int contentLength = 0;
        while ((line = reader.readLine()) != null) {
            if ("".equals(line)) { // at the end of the request
                break;
            } else if (line.contains(s: "Content-Length")) { // calculate length
                contentLength = Integer.parseInt(line.substring(line.indexOf(str: "Content-Length") + 16));
            }
        }
        char[] buf;
        if (contentLength != 0) {
            buf = new char[contentLength];
            reader.read(buf, off: 0, contentLength);
            String[] params = new String(buf).split(regex: "&");
            for (String item : params) {
                String key = item.substring(beginIndex: 0, item.indexOf(str: "="));
                String val = item.substring(item.indexOf(str: "=") + 1);
                request.params.put(key, val);
            }
        }
    }
    return request;
}

private void responsePOST(Request request) throws IOException {
    if (request.getURL().equals(anObject: "/dopost") && request.params.containsKey(key: "login") && request.params.containsKey(key: "pass")) {
        String msg;
        String login = (String) request.params.get(key: "login");
        String pass = (String) request.params.get(key: "pass");
        if (login.equals(user) && pass.equals(password)) {
            msg = "<html><head><meta charset='UTF-8'></head><body>login success!</body></html>";
        } else {
            msg = "<html><head><meta charset='UTF-8'></head><body>login fail!</body></html>";
        }
        msg = "HTTP/1.1 200 OK\r\n" + "Content-Type:text/html \r\n" +
            "Content-Length:" + msg.length() + "\r\n" + "\r\n" + msg;
        res.write(msg.getBytes());
        res.flush();
        res.close();
    } else {
        String err = "HTTP/1.1 404 file not found \r\n" + "Content-Type:text/html \r\n" +
            "Content-Length:22 \r\n" + "\r\n" + "<h1>404 not found</h1>";
        res.write(err.getBytes());
        res.flush();
        res.close();
    }
}

```

Wireshark 抓取的数据包截图（HTTP 协议部分）






- 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。

这里输入了错误密码，显示 login fail！

This is a test

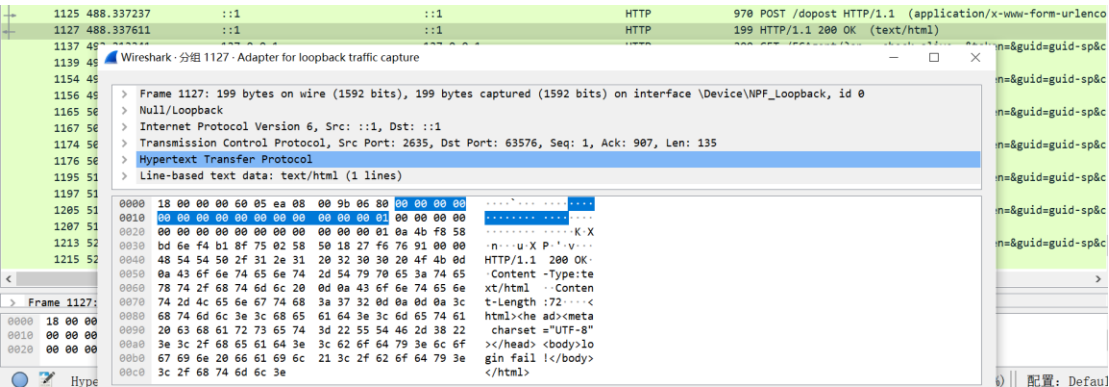
 CC98论坛

Login: Pass:

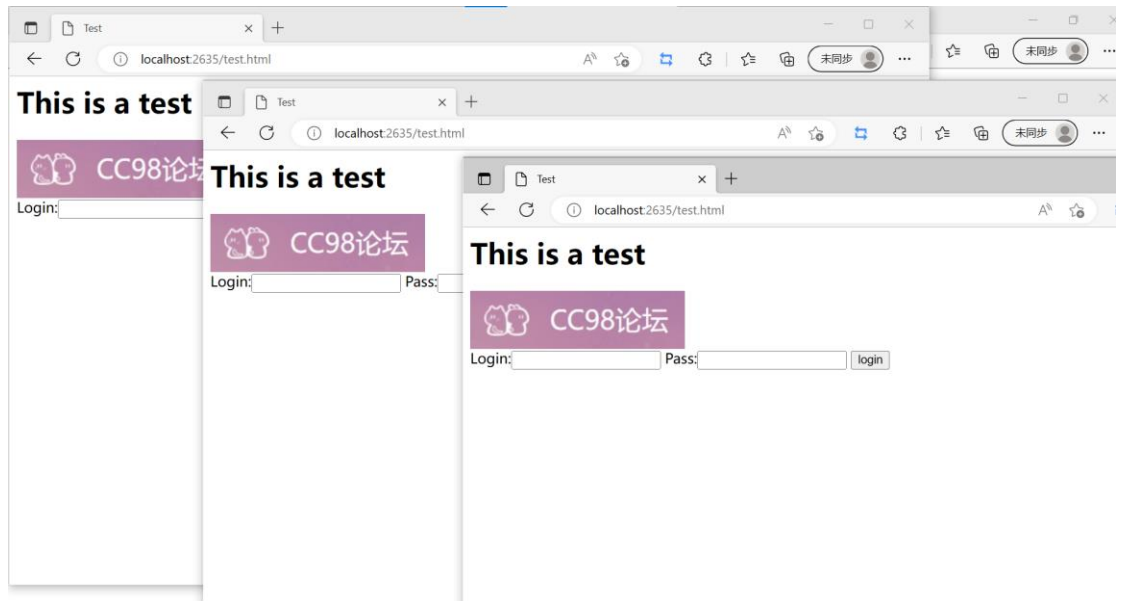
   localhost:2635/dopost

login fail!

- Wireshark 抓取的数据包截图（HTTP 协议部分）



- 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）
发现可以打开多个窗口。



- 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）

```
TCP      [::1]:2635          [::1]:49949          CLOSE_WAIT
TCP      [::1]:2635          [::1]:50074          CLOSE_WAIT
TCP      [::1]:2635          [::1]:50129          CLOSE_WAIT
TCP      [::1]:2635          [::1]:50251          CLOSE_WAIT
TCP      [::1]:2635          [::1]:50262          TIME_WAIT
TCP      [::1]:2635          [::1]:50299          TIME_WAIT
TCP      [::1]:2635          [::1]:50307          TIME_WAIT
TCP      [::1]:2635          [::1]:50308          TIME_WAIT
TCP      [::1]:2635          [::1]:50310          TIME_WAIT
TCP      [::1]:2635          [::1]:50312          TIME_WAIT
```

发现有些处在 CLOSE_WAIT 状态,有些处在 TIME_WAIT 状态。

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?

答: HTTP 的请求头和请求体通过空行进行分割,代码实现为根据 “\r\n” 区分。

- 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?

答：浏览器是根据头部的 Content-Type 字段来判断文件类型的，本实验中需要用到的 Content-Type 包括：

1. text/html: HTML 格式
2. text/plain: 纯文本格式
3. image/jpg: jpg 图片格式

- HTTP 协议的头部是不是一定是文本格式？体部呢？

答：协议的头部是文本格式，体部除了文本格式外，还可以是字节流的方式，比如音频、图片等数据的传输。

- 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？

答：放在体部，如 “login=3200102635&pass=2635”，两个字段通过&符号连接

七、 讨论、心得

这个实验整体来看和 socket 的实验比较像，不过这次我选择了用 java 代替 c++来编写，编写难度上不算很大。实验过程中我遇到过在浏览器上显示的文字是乱码的问题，经过查阅资料后发现可以通过添加 “charset=UTF-8” 字段来解决。

相比于前几个路由器的配置实验，这个实验主要是靠代码实现，可以让我们对逻辑的了解比较清晰，而且不容易出现奇怪的环境配置问题，整体上收获还是很大的。