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：	郑乔尹
学　　院：	计算机学院
系：	
专　　业：	计算机科学与技术
学　　号：	3210104169
指导教师：	陆魁军

2023 年　12　月　12　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：蒋奕　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**

- 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）

```
class WebServer {
public:
    explicit WebServer(int port) : port(port), serverSocket(0) {}

    ~WebServer() {
        Stop();
    }

    void Start() {
        Initialize();
        std::cout << "Server started on port " << port << std::endl;

        while (!shouldStop) {
            SOCKET clientSocket = AcceptConnection();
            if (clientSocket != INVALID_SOCKET) {
                auto *param = new ThreadParam{ webServer: this, clientSocket}; // Pass the pointer to the ThreadParam struct
                CreateThread( lpThreadAttributes: nullptr, dwStackSize: 0, lpStartAddress: HandleClient, lpParameter: reinterpret_cast<LPVOID>(param), dwCreationFlags: 0, lpThreadId: nullptr);
            }
        }

        std::cout << "Server stopped" << std::endl;
    }
```

WebServer 类的 Start 成员函数用于启动主线程，主线程启动后，如果 shouldStop 信号始终未被置 1，则一直进行循环，如果有浏览器成功连接服务器，则创建一个子线程与之交互。

- 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）

```
static DWORD WINAPI HandleClient(LPVOID param) {
    char buffer[1024 * 1024] = {0};
    // reload the param
    auto *threadParam = static_cast<ThreadParam*>(param); //ThreadParam*
    auto *webServerParam = static_cast<WebServer*>(threadParam->webServer);
    SOCKET clientSocket = threadParam->clientSocket;
    struct sockaddr_in HTTP_addr = {0};
    int HTTP_addr_len = sizeof(HTTP_addr);
    if (getpeername( s: clientSocket, name: (struct sockaddr *)&HTTP_addr, namelen: &HTTP_addr_len) == SOCKET_ERROR) {
        perror( ErrMsg: "Error getting client address");
        closesocket( s: clientSocket);
        return -1;
    }
    std::cout << "Client connected: " << inet_ntoa( in: HTTP_addr.sin_addr) << ":" << HTTP_addr.sin_port << std::endl;

    while (true) {
        int bytesRead = recv( s: clientSocket, buf: buffer, len: sizeof(buffer) - 1, flags: 0);

        if (bytesRead <= 0) {
            perror( ErrMsg: "Error reading from client");
            closesocket( s: clientSocket);
            return -1;
        }

        std::string request( s: buffer);

//          std::cout << "Request: " << std::endl << request << std::endl;
        std::istringstream iss( str: request);
        std::string method, file, headers;
        iss >> method >> file >> headers;
```

```
//          std::cout << "Method: " << method << std::endl;
//          std::cout << "File: " << file << std::endl;
//          std::cout << "Headers: " << headers << std::endl;

        if (method == "GET") {
            webServerParam->HandleGet(clientSocket, url: file);
        } else if (method == "POST") {
            webServerParam->HandlePost(clientSocket, file, request);
        } else {
            webServerParam->SendNotFoundResponse(clientSocket);
        }
    }

    closesocket( s: clientSocket);
    return 0;
}
```

将传入的参数重新转化为结构体，并从中取出 `this` 指针和当前连接的浏览器的 `socket` 编号，接收其发送的请求，并从请求中解析出请求方法（`method`），请求的文件（`file`），请求头（`headers`），根据请求方法调用对应的处理函数。

GET 请求的处理：

1. 解析请求的文件路径为服务器端的路径，为其添加父目录

```
155     void HandleGet(SOCKET clientSocket, const std::string& url) {
156 //        std::cout << "Handling GET request" << std::endl;
157 
158         std::string type;
159         std::string name = ".";
160 
161         if (url.find( s: ".txt") != std::string::npos) {
162             type = "text/plain";
163             name += "/txt";
164         } else if (url.find( s: ".html") != std::string::npos) {
165             name += "/html";
166             type = "text/html";
167         } else if (url.find( s: ".png") != std::string::npos) {
168             type = "image/png";
169             name += "/img";
170         } else if (url.find( s: ".ico") != std::string::npos) {
171             type = "image/ico";
172             name += "/img";
173         }
174 
175         name += url;
176 //        std::cout << "Finding the file in " << name << std::endl;
```

2. 通过解析出的路径访问服务器上对应的文件并发送响应：

```
178         std::ifstream fileStream( s: name,  mode: std::ios::binary);
179
180         if (fileStream.is_open()) {
181 //            std::cout << "File found" << std::endl;
182
183             fileStream.seekg(0, std::ios::end);
184             std::streampos filelen = fileStream.tellg();
185             fileStream.seekg(0, std::ios::beg);
186
187             std::ostringstream contentStream;
188             contentStream << fileStream.rdbuf();
189             std::string content = contentStream.str();
190
191             char len[10];
192             _ltoa_s( Value: static_cast<long>(filelen),  &: len,  Radix: 10);
193
194             std::string response = "HTTP/1.1 200 OK\r\n";
195             response += "Connection: keep-alive\r\n";
196             response += "Content-Length:" + std::string( s: len) + "\r\n";
197             response += "Server: csr_http1.1\n";
198
199             if (url.find( s: ".png") != std::string::npos) {
200                 response += "Accept-Ranges: bytes\r\n";
201             }
202
203             response += "Content-Type: " + type + "\r\n";
204             response += "\r\n" + content;
205
206             SendHttpResponse(clientSocket, response);
207 //            std::cout << "Send successfully" << std::endl;
```

POST 请求的处理：

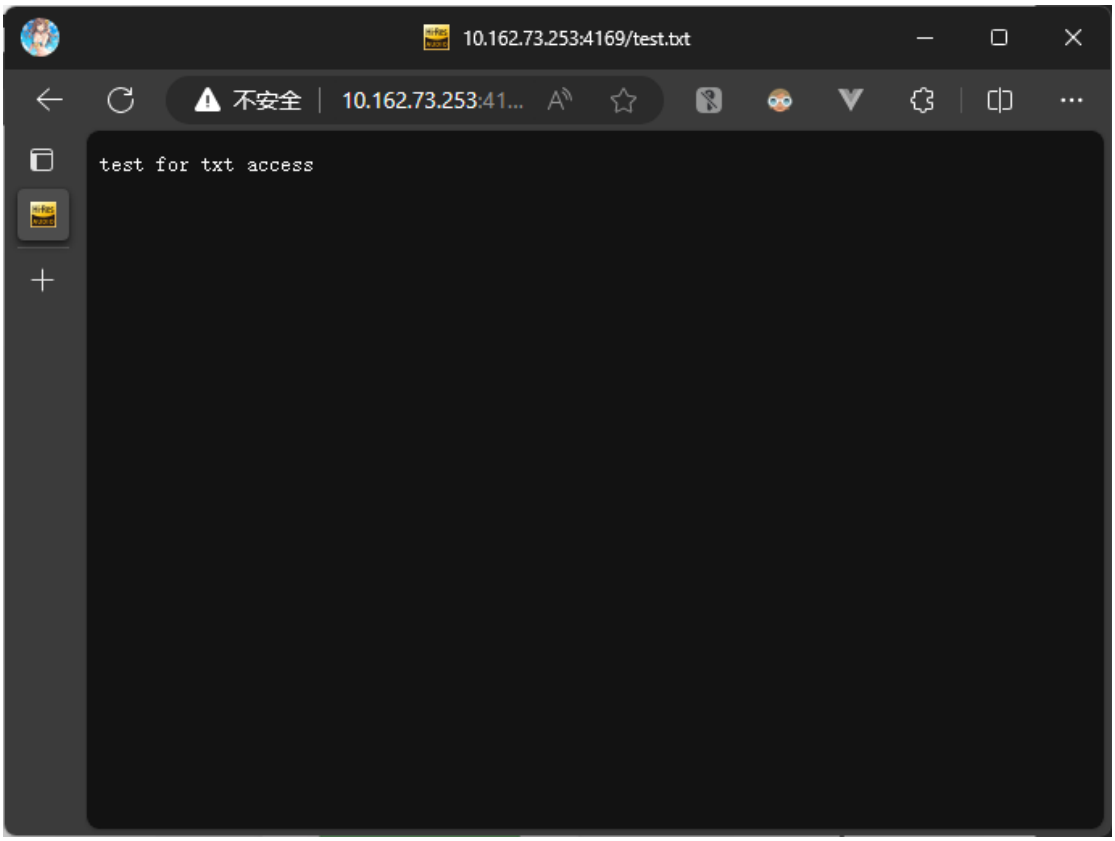
```cpp
void HandlePost(SOCKET clientSocket, const std::string& file, const std::string& request) {
    std::cout << "Handling POST request" << std::endl;
    std::string successMsg = "<html><body><h1>Login Success</h1><p>From server: Login successfully.</p></body></html>\r\n";
    std::string failedMsg = "<html><body><h1>Login Failed</h1><p>From server: Login failed.</p></body></html>\r\n";
    // parse "login=xxx&pass=xxx"
    std::string login = request.substr( pos: request.find( s: "login=") + strlen( Str: "login="));
    login = login.substr( pos: 0, n: login.find( s: "&"));
    std::string password = request.substr( pos: request.find( s: "pass=") + strlen( Str: "pass="));
    password = password.substr( pos: 0, n: password.find( s: "&"));

    std::string response = "HTTP/1.1 200 OK\r\n";
    response += "Content-Type: text/html\r\n";
    response += "Content-Length: ";

    if (login == "3210104169" && password == "4169") {
        response += std::to_string( val: successMsg.size());
        response += "\r\n\r\n";
        response += successMsg;
        std::cout << "Login successfully" << std::endl;
    } else {
        response += std::to_string( val: failedMsg.size());
        response += "\r\n\r\n";
        response += failedMsg;
        std::cout << "Login failed" << std::endl;
    }

    SendHttpResponse(clientSocket, response);
}
```

从 POST 请求中解析出用户输入的账号和密码，检查是否为 3210104169 与 4169，如果正确，则响应内容为 Login Success，否则为 Login Failed。

- 服务器运行后，用 netstat –an 显示服务器的监听端口

```
命令提示符

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    0.0.0.0:49670          0.0.0.0:0              LISTENING
TCP    0.0.0.0:49672          0.0.0.0:0              LISTENING
TCP    0.0.0.0:49673          0.0.0.0:0              LISTENING
TCP    0.0.0.0:49705          0.0.0.0:0              LISTENING
TCP    0.0.0.0:55409          0.0.0.0:0              LISTENING
TCP    10.162.73.253:139      0.0.0.0:0              LISTENING
TCP    10.162.73.253:4169     10.162.73.253:50256    ESTABLISHED
TCP    10.162.73.253:49225    117.135.154.112:443    ESTABLISHED
TCP    10.162.73.253:49410    20.198.162.76:443      ESTABLISHED
TCP    10.162.73.253:50063    13.69.109.130:443      TIME_WAIT
TCP    10.162.73.253:50165    110.72.100.117:20702   TIME_WAIT
TCP    10.162.73.253:50171    110.72.100.117:20702   TIME_WAIT
TCP    10.162.73.253:50173    110.72.100.117:20702   TIME_WAIT
TCP    10.162.73.253:50175    110.72.100.117:20702   TIME_WAIT
TCP    10.162.73.253:50181    110.72.100.117:20702   TIME_WAIT
TCP    10.162.73.253:50183    110.72.100.117:20702   TIME_WAIT
TCP    10.162.73.253:50193    117.135.154.108:443    TIME_WAIT
TCP    10.162.73.253:50199    110.72.100.117:20702   ESTABLISHED
TCP    10.162.73.253:50201    110.72.100.117:20702   TIME_WAIT
TCP    10.162.73.253:50210    223.109.146.194:443    TIME_WAIT
TCP    10.162.73.253:50216    13.107.5.80:443        ESTABLISHED
TCP    10.162.73.253:50223    20.189.173.11:443      ESTABLISHED
TCP    10.162.73.253:50224    10.162.73.253:4169     TIME_WAIT
```

- 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：

`./txt/test.txt`

服务器的相关代码片段：

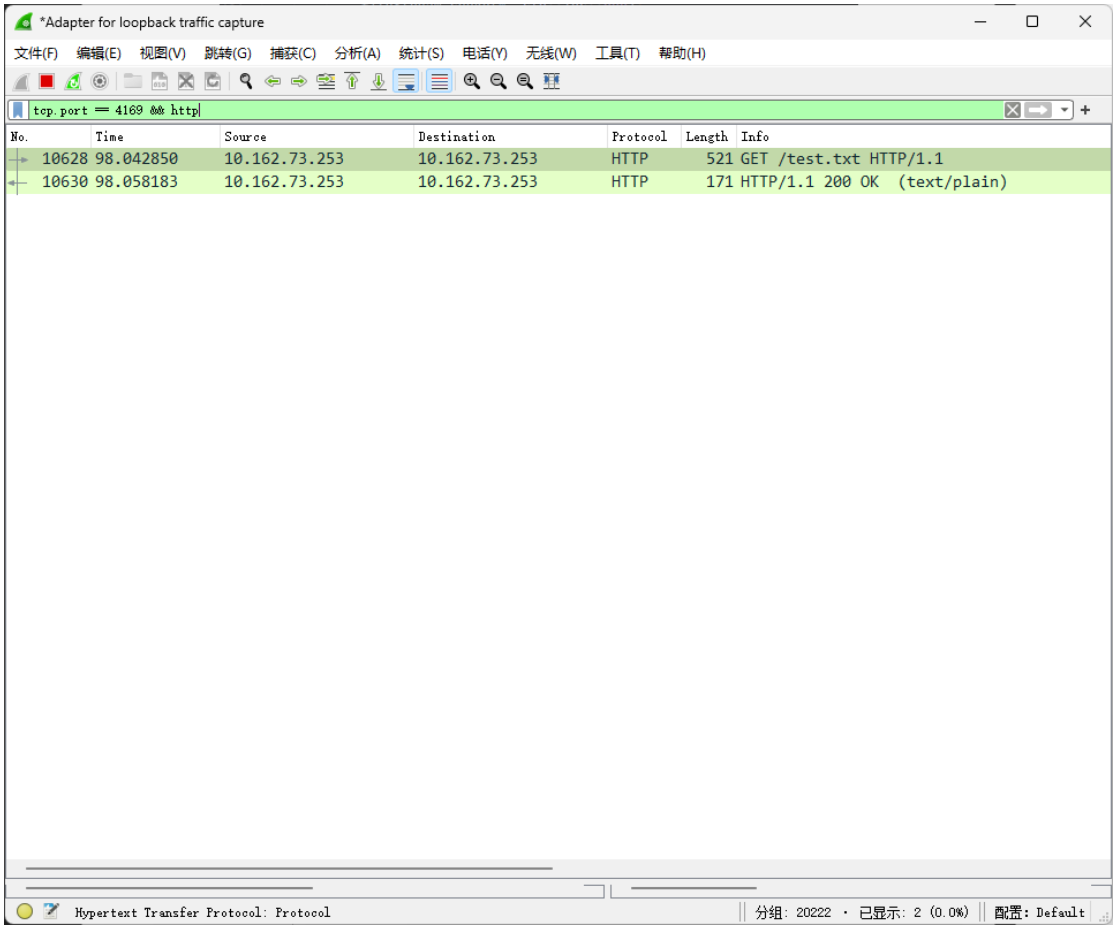
构造文件在服务器上的路径：

```
157     void HandleGet(SOCKET clientSocket, const std::string& url) {
158 //        std::cout << "Handling GET request" << std::endl;
159
160         std::string type;
161         std::string name = ".";
162
163         if (url.find( s: ".txt") != std::string::npos) {
164             type = "text/plain";
165             name += "/txt";
166         } else if (url.find( s: ".html") != std::string::npos) {
167             name += "/html";
168             type = "text/html";
169         } else if (url.find( s: ".png") != std::string::npos) {
170             type = "image/png";
171             name += "/img";
172         } else if (url.find( s: ".ico") != std::string::npos) {
173             type = "image/ico";
174             name += "/img";
175         }
176
177         name += url;
178 //        std::cout << "Finding the file in " << name << std::endl;
179
```

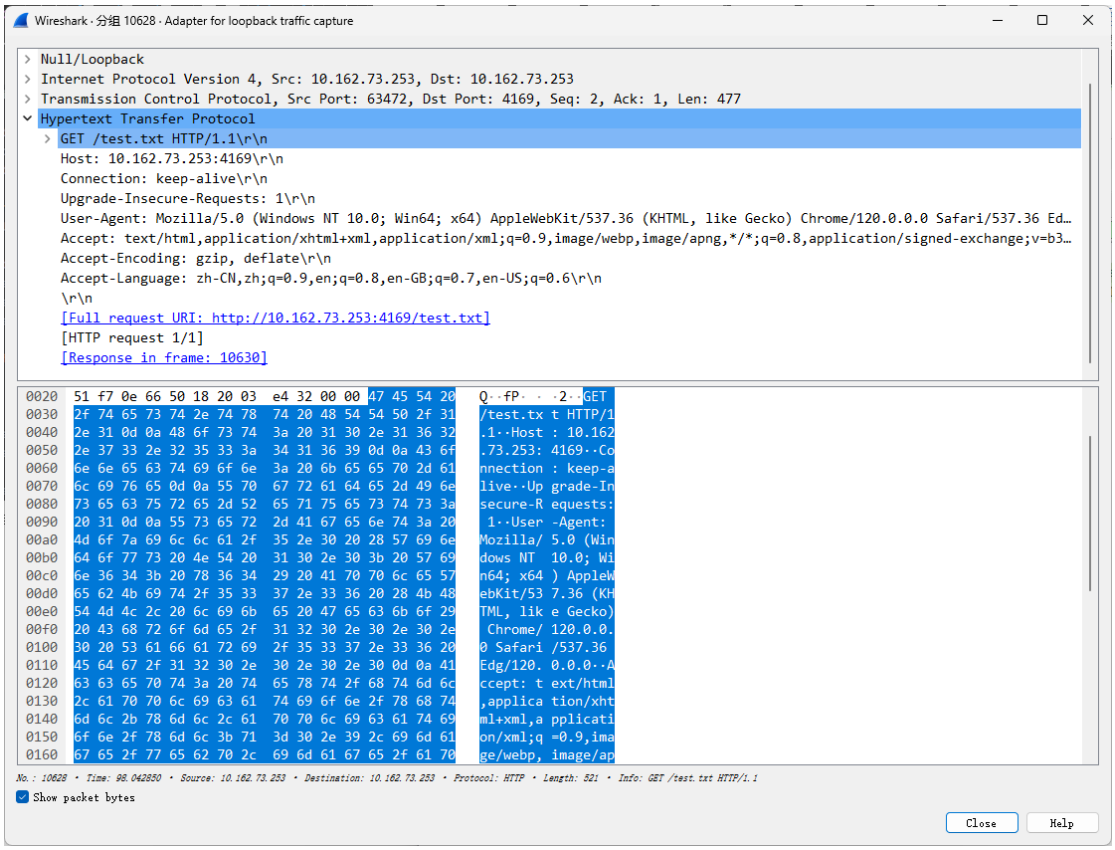
访问对应的 test.txt 文件并构造响应内容，发送给浏览器：

```
182         if (fileStream.is_open()) {
183 //            std::cout << "File found" << std::endl;
184
185             fileStream.seekg(0, std::ios::end);
186             std::streampos filelen = fileStream.tellg();
187             fileStream.seekg(0, std::ios::beg);
188
189             std::ostringstream contentStream;
190             contentStream << fileStream.rdbuf();
191             std::string content = contentStream.str();
192
193             char len[10];
194             _ltoa_s( Value: static_cast<long>(filelen), &: len, Radix: 10);
195
196             std::string response = "HTTP/1.1 200 OK\r\n";
197             response += "Connection: keep-alive\r\n";
198             response += "Content-Length:" + std::string( s: len) + "\r\n";
199             response += "Server: csr_http1.1\n";
200
201             if (url.find( s: ".png") != std::string::npos) {
202                 response += "Accept-Ranges: bytes\r\n";
203             }
204
205             response += "Content-Type: " + type + "\r\n";
206             response += "\r\n" + content;
207
208             SendHttpResponse(clientSocket, response);
```

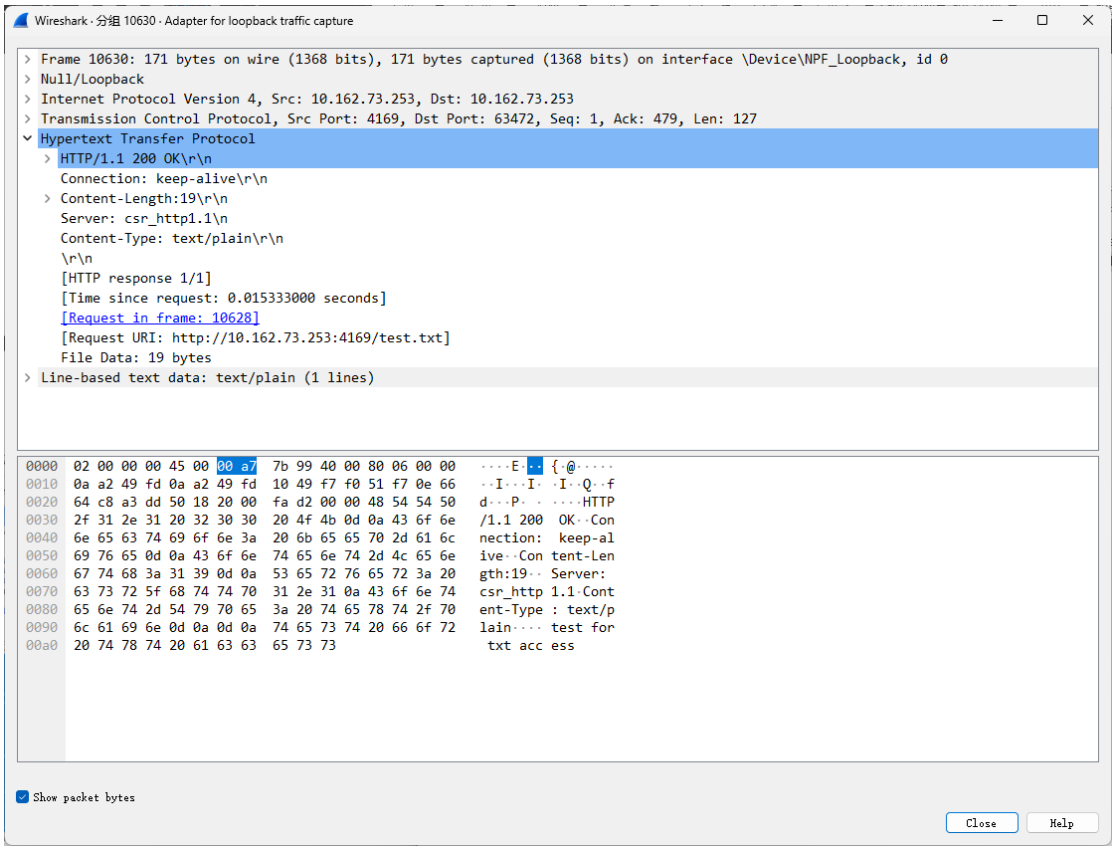
Wireshark 抓取的数据包截图（通过跟踪 TCP 流，只截取 HTTP 协议部分）：


*Adapter for loopback traffic capture
文件(F) 编辑(E) 视图(V) 跳转(G) 捕获(C) 分析(A) 统计(S) 电话(Y) 无线(W) 工具(T) 帮助(H)
tcp.port == 4169 && http
No. Time Source Destination Protocol Length Info
10628 98.042850 10.162.73.253 10.162.73.253 HTTP 521 GET /test.txt HTTP/1.1
10630 98.058183 10.162.73.253 10.162.73.253 HTTP 171 HTTP/1.1 200 OK (text/plain)
Hypertext Transfer Protocol: Protocol
分组: 20222 · 已显示: 2 (0.0%)
配置: Default

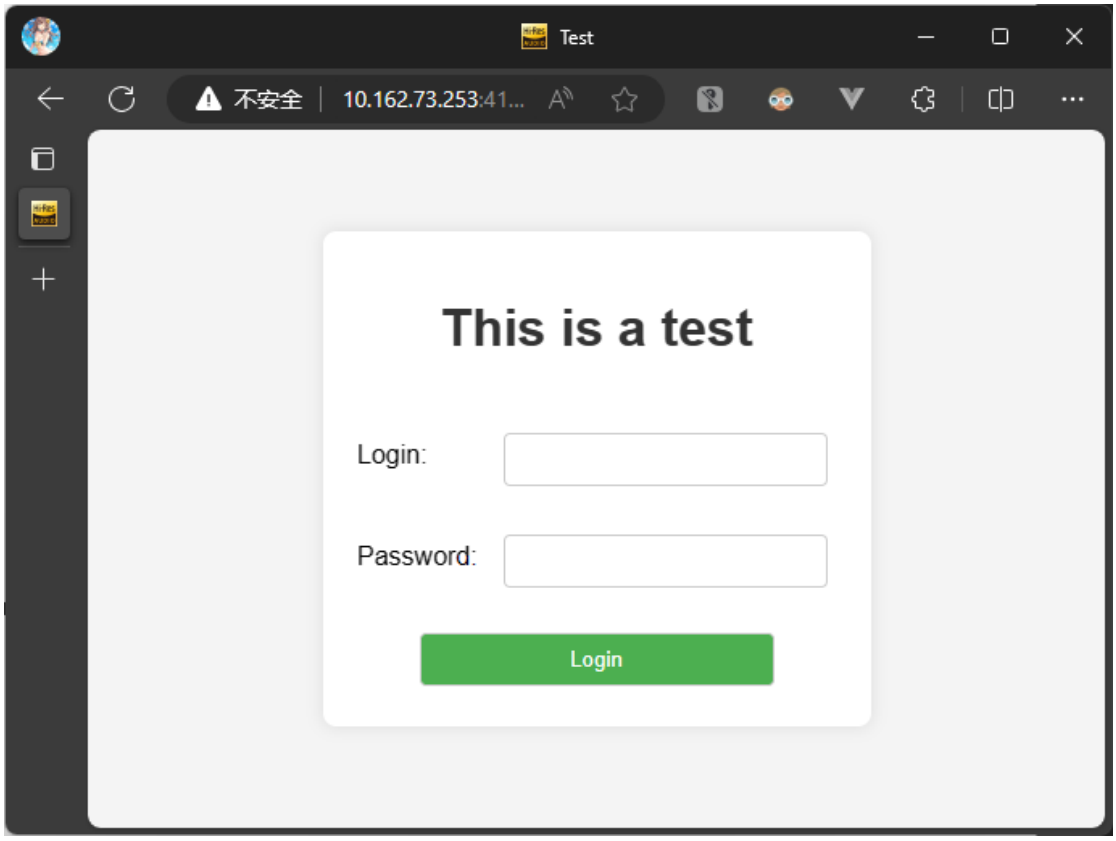
请求：



响应：



- 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。



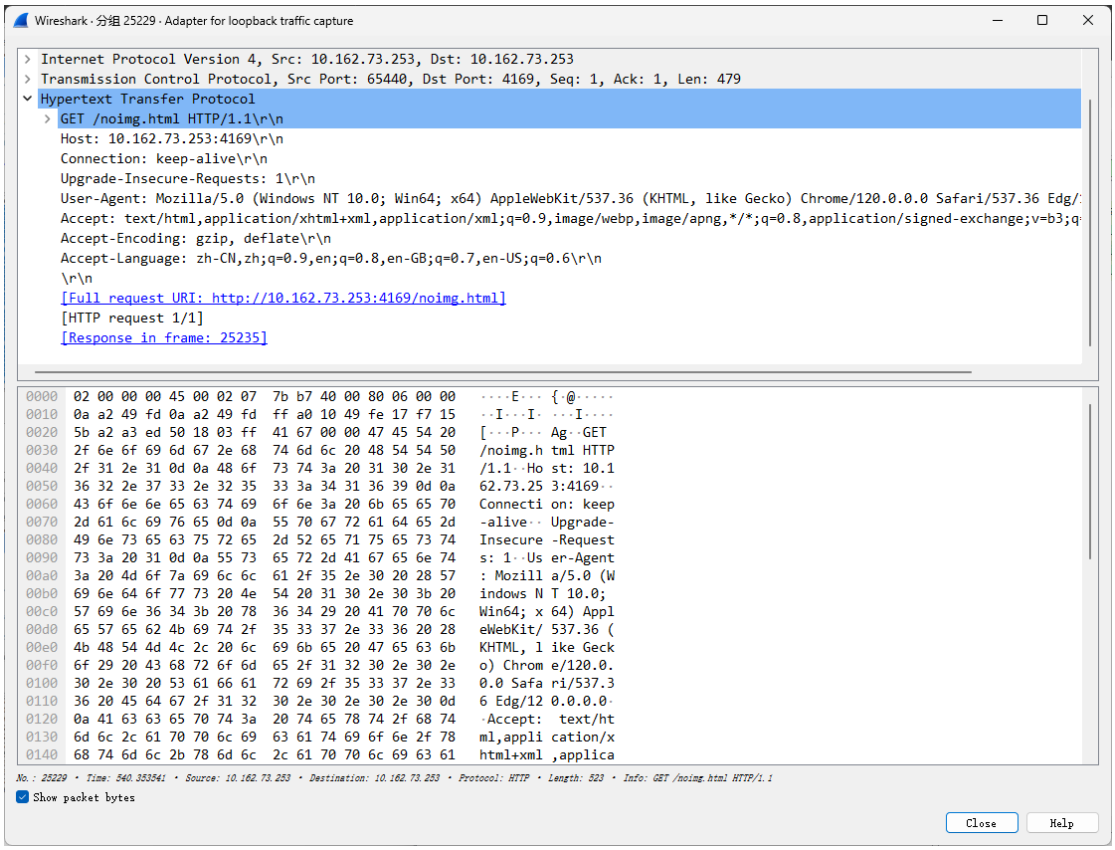
服务器文件实际存放的路径：

`./html/noimg.html`

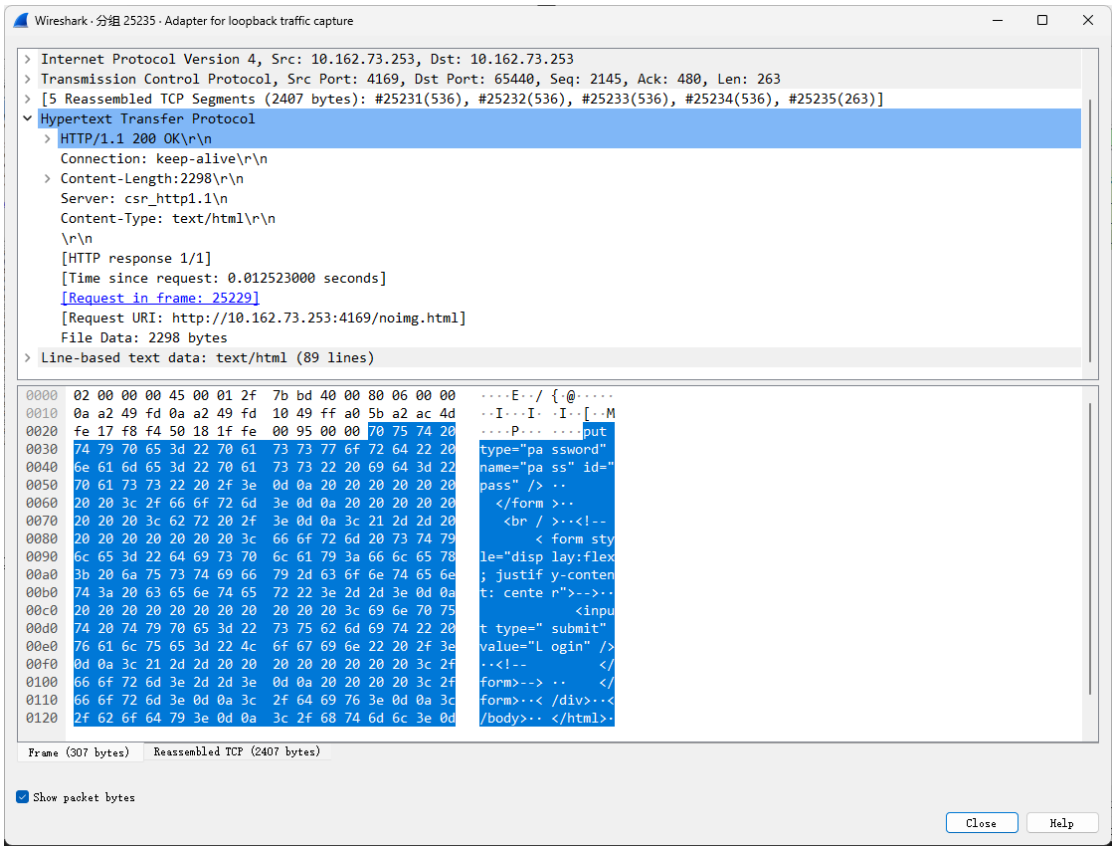
Wireshark 抓取的数据包截图（只截取 HTTP 协议部分，包括 HTML 内容）：

25229	540.353541	10.162.73.253	10.162.73.253	HTTP	523 GET /noimg.html HTTP/1.1
25235	540.366064	10.162.73.253	10.162.73.253	HTTP	307 HTTP/1.1 200 OK (text/html)

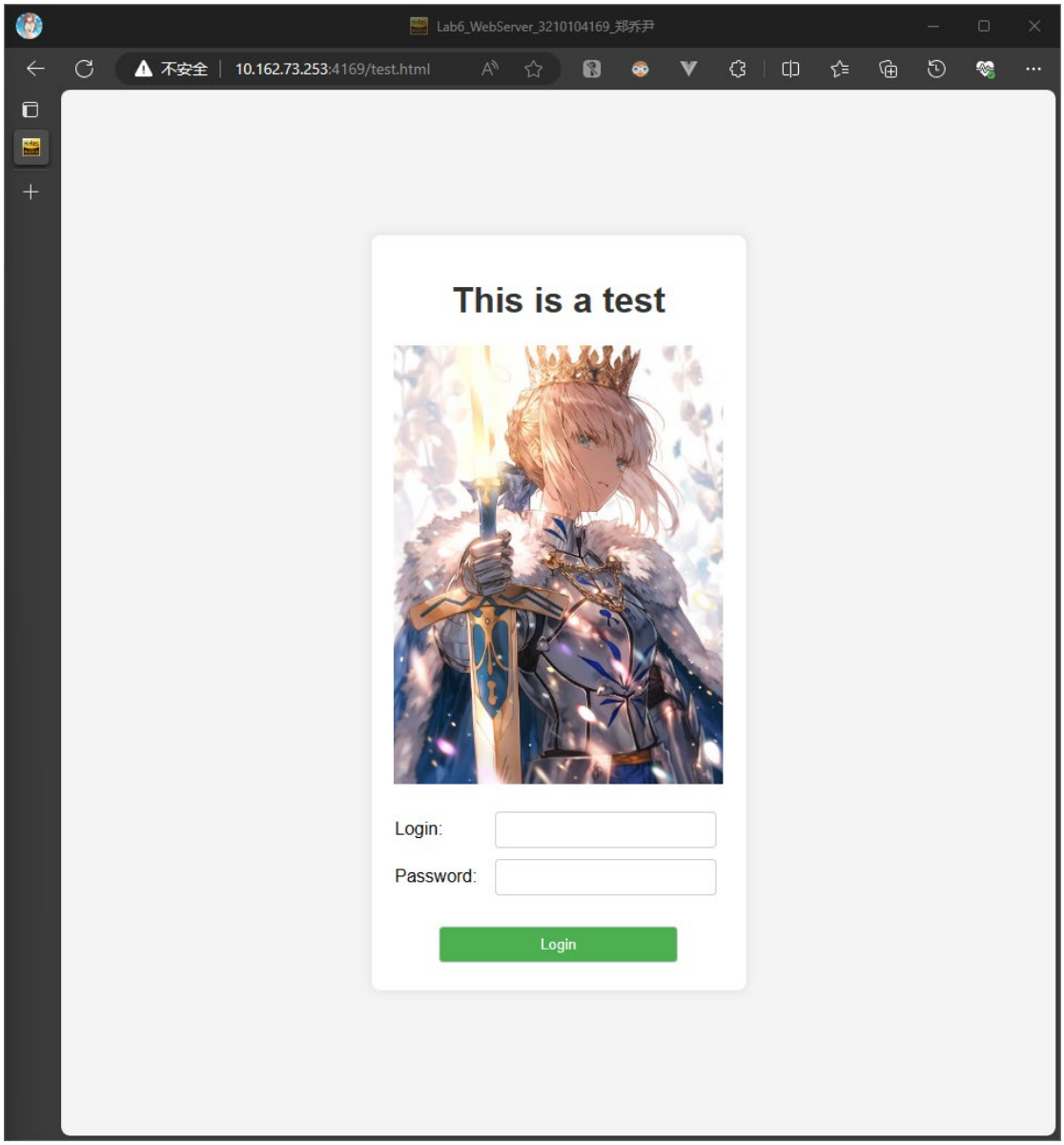
请求：



响应：



- 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：

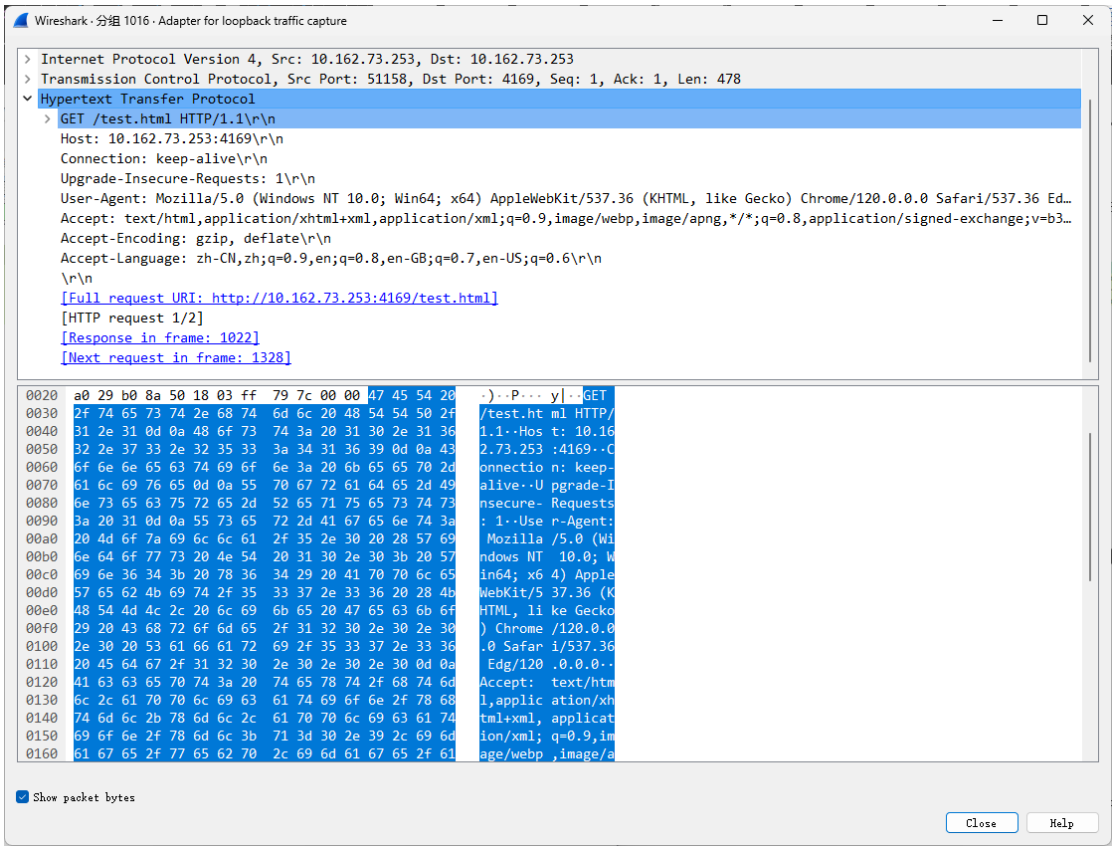
`./html/test.html`

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）：

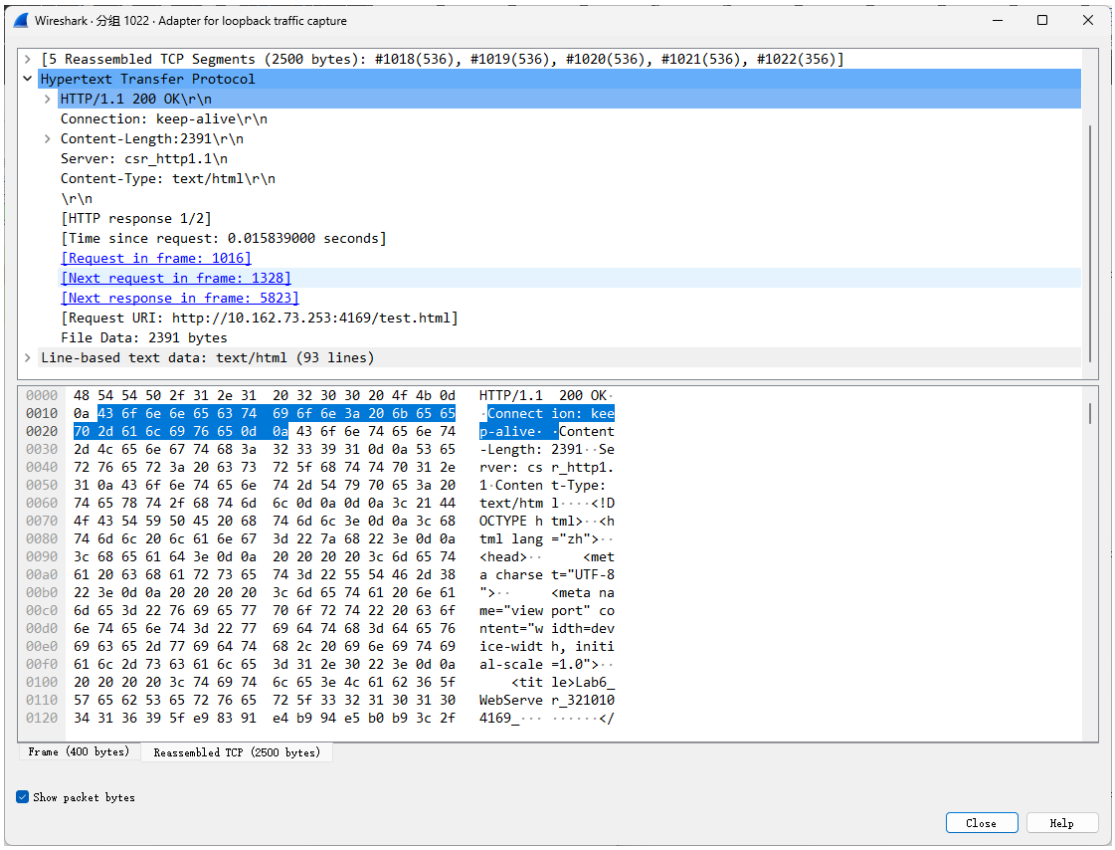
1016	23.604382	10.162.73.253	10.162.73.253	HTTP	522	GET /test.html HTTP/1.1
1022	23.620221	10.162.73.253	10.162.73.253	HTTP	400	HTTP/1.1 200 OK (text/html)
1328	23.859557	10.162.73.253	10.162.73.253	HTTP	466	GET /logo.png HTTP/1.1
5823	24.022204	10.162.73.253	10.162.73.253	HTTP	92	HTTP/1.1 200 OK (PNG)

HTML:

请求：

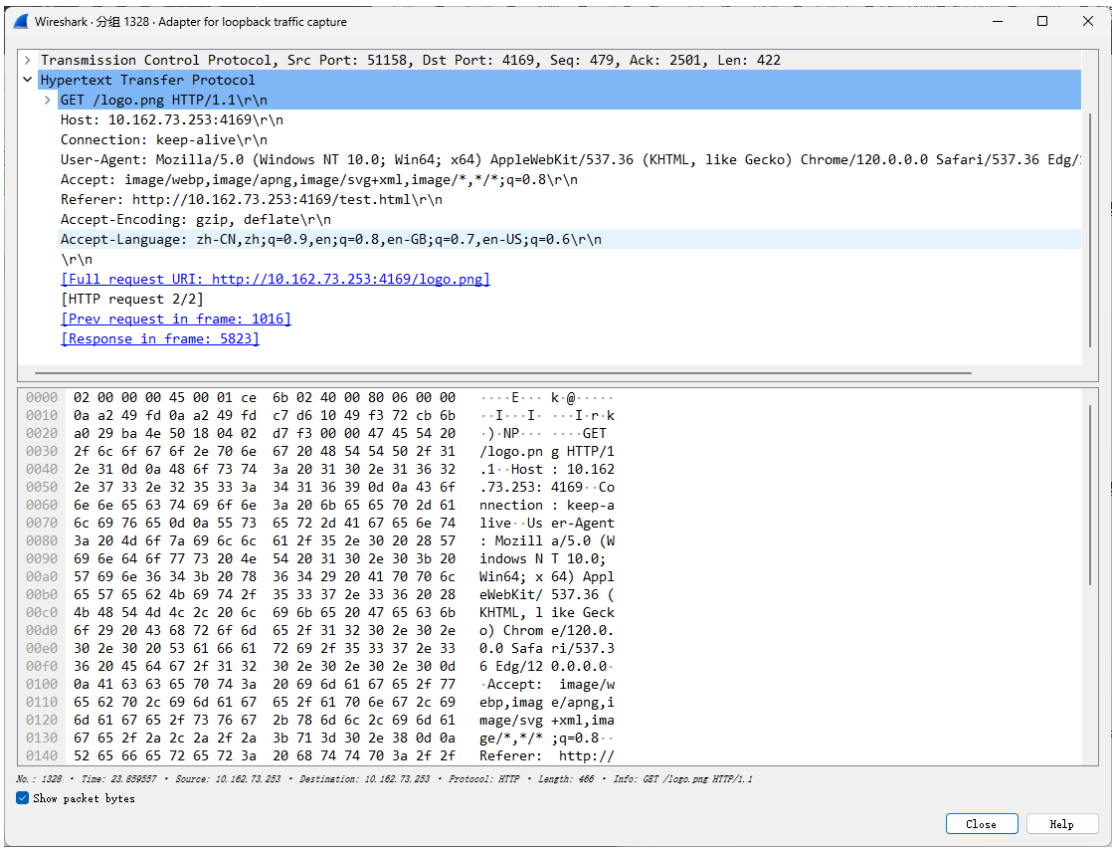

Wireshark · 分组 1016 · Adapter for loopback traffic capture
Internet Protocol Version 4, Src: 10.162.73.253, Dst: 10.162.73.253
Transmission Control Protocol, Src Port: 51158, Dst Port: 4169, Seq: 1, Ack: 1, Len: 478
Hypertext Transfer Protocol
GET /test.html HTTP/1.1\r\n
Host: 10.162.73.253:4169\r\n
Connection: keep-alive\r\n
Upgrade-Insecure-Requests: 1\r\n
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/120.0.0.0 Safari/537.36 Ed…
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3…
Accept-Encoding: gzip, deflate\r\n
Accept-Language: zh-CN,zh;q=0.9,en;q=0.8,en-GB;q=0.7,en-US;q=0.6\r\n
\r\n
[Full request URI: http://10.162.73.253:4169/test.html]
[HTTP request 1/2]
[Response in frame: 1022]
[Next request in frame: 1328]
0020 a0 29 b0 8a 50 18 03 ff 79 7c 00 00 47 45 54 20 ·)··P··· y|··GET
0030 2f 74 65 73 74 2e 68 74 6d 6c 20 48 54 54 50 2f /test.ht ml HTTP/
0040 31 2e 31 0d 0a 48 6f 73 74 3a 20 31 30 2e 31 36 1.1··Hos t: 10.16
0050 32 2e 37 33 2e 32 35 33 3a 34 31 36 39 0d 0a 43 2.73.253 :4169··C
0060 6f 6e 6e 65 63 74 69 6f 6e 3a 20 6b 65 65 70 2d onnectio n: keep-
0070 61 6c 69 76 65 0d 0a 55 70 67 72 61 64 65 2d 49 alive··U pgrade-I
0080 6e 73 65 63 75 72 65 2d 52 65 71 75 65 73 74 73 nsecure- Requests
0090 3a 20 31 0d 0a 55 73 65 72 2d 41 67 65 6e 74 3a : 1··Use r-Agent:
00a0 20 4d 6f 7a 69 6c 6c 61 2f 35 2e 30 20 28 57 69 Mozilla /5.0 (Wi
00b0 6e 64 6f 77 73 20 4e 54 20 31 30 2e 30 3b 20 57 ndows NT 10.0; W
00c0 69 6e 36 34 3b 20 78 36 34 29 20 41 70 70 6c 65 in64; x6 4) Apple
00d0 57 65 62 4b 69 74 2f 35 33 37 2e 33 36 20 28 4b WebKit/5 37.36 (K
00e0 48 54 4d 4c 20 6c 69 6b 65 20 47 65 63 6b 6f HTML, li ke Gecko
00f0 29 20 43 68 72 6f 6d 65 2f 31 32 30 2e 30 2e 30) Chrome /120.0.0
0100 2e 30 20 53 61 66 61 72 69 2f 35 33 37 2e 33 36 .0 Safar i/537.36
0110 20 45 64 67 2f 31 32 30 2e 30 2e 30 2e 30 0d 0a Edg/120 .0.0.0··
0120 41 63 63 65 70 74 3a 20 74 65 78 74 2f 68 74 6d Accept: text/htm
0130 6c 2c 61 70 70 6c 69 63 61 74 69 6f 6e 2f 78 68 l,applic ation/xh
0140 74 6d 6c 2b 78 6d 6c 2c 61 70 70 6c 69 63 61 74 tml+xml, applicat
0150 69 6f 6e 2f 78 6d 6c 3b 71 3d 30 2e 39 2c 69 6d ion/xml; q=0.9,im
0160 61 67 65 2f 77 65 62 70 2c 69 6d 61 67 65 2f 61 age/webp ,image/a
Show packet bytes
Close
Help

响应：

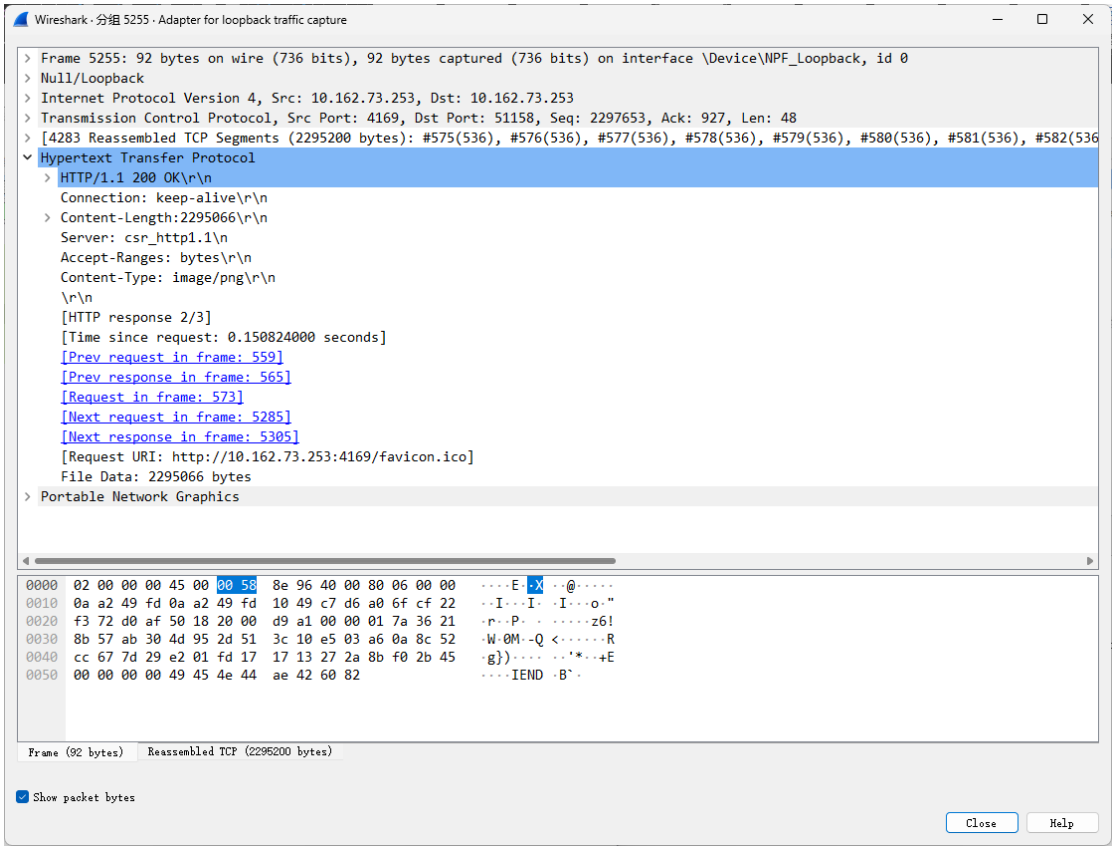

Wireshark · 分组 1022 · Adapter for loopback traffic capture
[5 Reassembled TCP Segments (2500 bytes): #1018(536), #1019(536), #1020(536), #1021(536), #1022(356)]
Hypertext Transfer Protocol
HTTP/1.1 200 OK\r\n
Connection: keep-alive\r\n
Content-Length:2391\r\n
Server: csr_http1.1\n
Content-Type: text/html\r\n
\r\n
[HTTP response 1/2]
[Time since request: 0.015839000 seconds]
[Request in frame: 1016]
[Next request in frame: 1328]
[Next response in frame: 5823]
[Request URI: http://10.162.73.253:4169/test.html]
File Data: 2391 bytes
Line-based text data: text/html (93 lines)
0000 48 54 54 50 2f 31 2e 31 20 32 30 30 20 4f 4b 0d HTTP/1.1 200 OK·
0010 0a 43 6f 6e 6e 65 63 74 69 6f 6e 3a 20 6b 65 65 ·Connect ion: kee
0020 70 2d 61 6c 69 76 65 0d 0a 43 6f 6e 74 65 6e 74 p-alive· ·Content
0030 2d 4c 65 6e 67 74 68 3a 20 32 33 39 31 0d 0a 53 -Length: 2391··S
0040 72 76 65 72 3a 20 63 73 72 5f 68 74 74 70 31 2e rver: cs r_http1.
0050 31 0a 43 6f 6e 74 65 6e 74 2d 54 79 70 65 3a 20 1·Conten t-Type:
0060 74 65 78 74 2f 68 74 6d 6c 0d 0a 0d 0a 3c 21 44 text/htm l····<!D
0070 4f 43 54 59 50 45 20 68 74 6d 6c 3e 0d 0a 3c 68 OCTYPE h tml>··<h
0080 74 6d 6c 20 6c 61 6e 67 3d 22 7a 68 22 3e 0d 0a tml lang ="zh">··
0090 3c 68 65 61 64 3e 0d 0a 20 20 20 20 3c 6d 65 74 <head>·· <met
00a0 61 20 63 68 61 72 73 65 74 3d 22 55 54 46 2d 38 a charse t="UTF-8
00b0 22 3e 0d 0a 20 20 20 20 3c 6d 65 74 61 20 6e 61 ">·· <meta na
00c0 6d 65 3d 22 76 69 65 77 70 6f 72 74 22 20 63 6f me="view port" co
00d0 6e 74 65 6e 74 3d 22 77 69 64 74 68 3d 64 65 76 ntent="w idth=dev
00e0 69 63 65 2d 77 69 64 74 68 2c 20 69 6e 69 74 69 ice-widt h, initi
00f0 61 6c 2d 73 63 61 6c 65 3d 31 2e 30 22 3e 0d 0a al-scale =1.0">··
0100 20 20 20 20 3c 74 69 74 6c 65 3e 4c 61 62 36 5f <tit le>Lab6_
0110 57 65 62 53 65 72 76 65 72 5f 33 32 31 30 31 30 WebServe r_321010
0120 34 31 36 39 5f e9 83 91 e4 b9 94 e5 b0 b9 3c 2f 4169_··· ·····</
Frame (400 bytes)
Reassembled TCP (2500 bytes)
Show packet bytes
Close
Help

图片文件（PNG）：

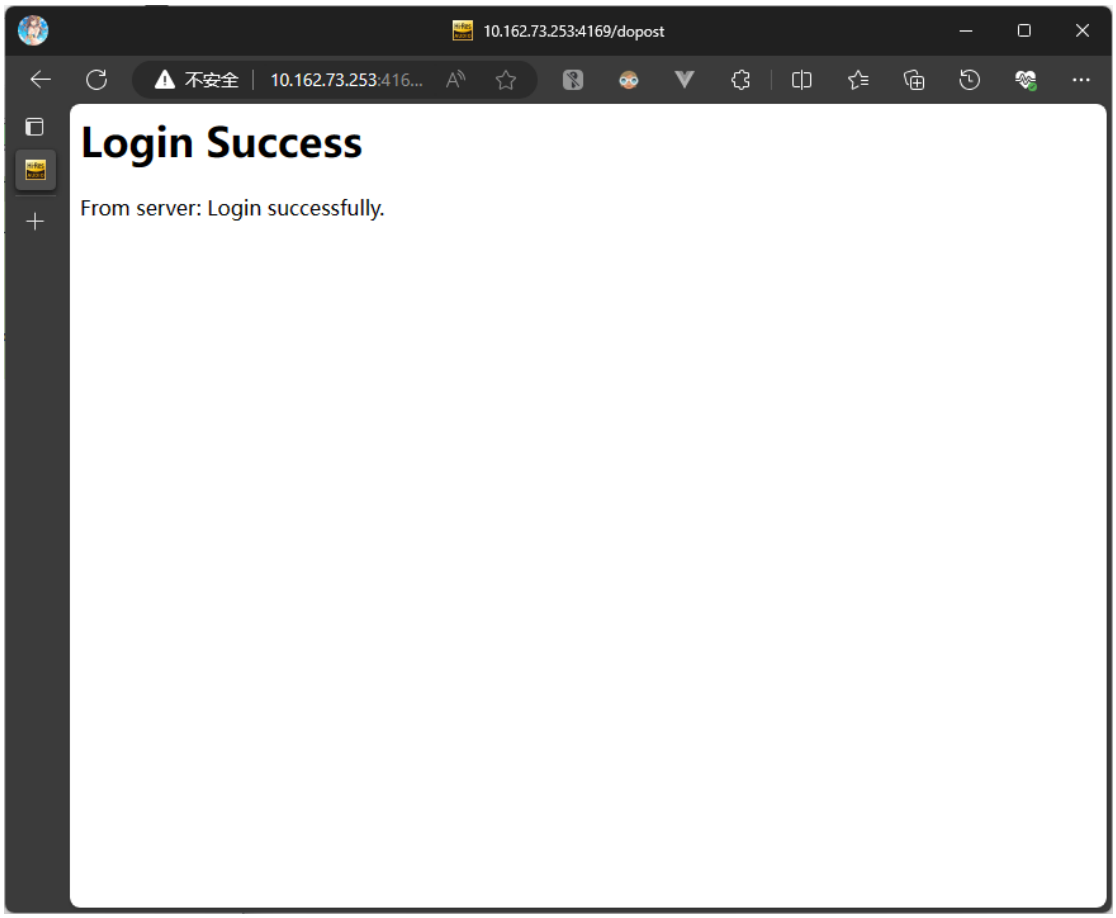
请求：


Wireshark · 分组 1328 · Adapter for loopback traffic capture
Transmission Control Protocol, Src Port: 51158, Dst Port: 4169, Seq: 479, Ack: 2501, Len: 422
Hypertext Transfer Protocol
GET /logo.png HTTP/1.1\r\n
Host: 10.162.73.253:4169\r\n
Connection: keep-alive\r\n
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/120.0.0.0 Safari/537.36 Edg/
Accept: image/webp,image/apng,image/svg+xml,image/*,*/*;q=0.8\r\n
Referer: http://10.162.73.253:4169/test.html\r\n
Accept-Encoding: gzip, deflate\r\n
Accept-Language: zh-CN,zh;q=0.9,en;q=0.8,en-GB;q=0.7,en-US;q=0.6\r\n
\r\n
[Full request URI: http://10.162.73.253:4169/logo.png]
[HTTP request 2/2]
[Prev request in frame: 1016]
[Response in frame: 5823]
0000 02 00 00 00 45 00 01 ce 6b 02 40 00 80 06 00 00 ····E··· k·@·····
0010 0a a2 49 fd 0a a2 49 fd c7 d6 10 49 f3 72 cb 6b ··I···I· ···I·r·k
0020 a0 29 ba 4e 50 18 04 02 d7 f3 00 00 47 45 54 20 ·)·NP··· ····GET
0030 2f 6c 6f 67 6f 2e 70 6e 67 20 48 54 54 50 2f 31 /logo.pn g HTTP/1
0040 2e 31 0d 0a 48 6f 73 74 3a 20 31 30 2e 31 36 32 .1··Host : 10.162
0050 2e 37 33 2e 32 35 33 3a 34 31 36 39 0d 0a 43 6f .73.253: 4169··Co
0060 6e 6e 65 63 74 69 6f 6e 3a 20 6b 65 65 70 2d 61 nnection : keep-a
0070 6c 69 76 65 0d 0a 55 73 65 72 2d 41 67 65 6e 74 live··Us er-Agent
0080 3a 20 4d 6f 7a 69 6c 6c 61 2f 35 2e 30 20 28 57 : Mozill a/5.0 (W
0090 69 6e 64 6f 77 73 20 4e 54 20 31 30 2e 30 3b 20 indows N T 10.0;
00a0 57 69 6e 36 34 3b 20 78 36 34 29 20 41 70 70 6c Win64; x 64) Appl
00b0 65 57 65 62 4b 69 74 2f 35 33 37 2e 33 36 20 28 eWebKit/ 537.36 (
00c0 4b 48 54 4d 4c 2c 20 6c 69 6b 65 20 47 65 63 6b KHTML, l ike Geck
00d0 6f 29 20 43 68 72 6f 6d 65 2f 31 32 30 2e 30 2e o) Chrom e/120.0.
00e0 30 2e 30 20 53 61 66 61 72 69 2f 35 33 37 2e 33 0.0 Safa ri/537.3
00f0 36 20 45 64 67 2f 31 32 30 2e 30 2e 30 2e 30 0d 6 Edg/12 0.0.0.0·
0100 0a 41 63 63 65 70 74 3a 20 69 6d 61 67 65 2f 77 ·Accept: image/w
0110 65 62 70 2c 69 6d 61 67 65 2f 61 70 6e 67 2c 69 ebp,imag e/apng,i
0120 6d 61 67 65 2f 73 76 67 2b 78 6d 6c 2c 69 6d 61 mage/svg +xml,ima
0130 67 65 2f 2a 2c 2a 2f 2a 3b 71 3d 30 2e 38 0d 0a ge/*,*/* ;q=0.8··
0140 52 65 66 65 72 65 72 3a 20 68 74 74 70 3a 2f 2f Referer: http://
No.: 1328 · Time: 23.859557 · Source: 10.162.73.253 · Destination: 10.162.73.253 · Protocol: HTTP · Length: 466 · Info: GET /logo.png HTTP/1.1
Show packet bytes
Close
Help

响应：


Wireshark · 分组 5255 · Adapter for loopback traffic capture
Frame 5255: 92 bytes on wire (736 bits), 92 bytes captured (736 bits) on interface \Device\NPF_Loopback, id 0
Null/Loopback
Internet Protocol Version 4, Src: 10.162.73.253, Dst: 10.162.73.253
Transmission Control Protocol, Src Port: 4169, Dst Port: 51158, Seq: 2297653, Ack: 927, Len: 48
[4283 Reassembled TCP Segments (2295200 bytes): #575(536), #576(536), #577(536), #578(536), #579(536), #580(536), #581(536), #582(536
Hypertext Transfer Protocol
HTTP/1.1 200 OK\r\n
Connection: keep-alive\r\n
Content-Length:2295066\r\n
Server: csr_http1.1\n
Accept-Ranges: bytes\r\n
Content-Type: image/png\r\n
\r\n
[HTTP response 2/3]
[Time since request: 0.150824000 seconds]
[Prev request in frame: 559]
[Prev response in frame: 565]
[Request in frame: 573]
[Next request in frame: 5285]
[Next response in frame: 5305]
[Request URI: http://10.162.73.253:4169/favicon.ico]
File Data: 2295066 bytes
Portable Network Graphics
0000 02 00 00 00 45 00 00 58 8e 96 40 00 80 06 00 00 ····E··X ··@·····
0010 0a a2 49 fd 0a a2 49 fd 10 49 c7 d6 a0 6f cf 22 ··I···I· ·I···o·"
0020 f3 72 d0 af 50 18 20 00 d9 a1 00 00 01 7a 36 21 ·r··P· · ·····z6!
0030 8b 57 ab 30 4d 95 2d 51 3c 10 e5 03 a6 0a 8c 52 ·W·0M·-Q <·····R
0040 cc 67 7d 29 e2 01 fd 17 17 13 27 2a 8b f0 2b 45 ·g})···· ··'*··+E
0050 00 00 00 00 49 45 4e 44 ae 42 60 82 ····IEND ·B`·
Frame (92 bytes)
Reassembled TCP (2295200 bytes)
Show packet bytes
Close
Help

- 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。



服务器相关处理代码片段：

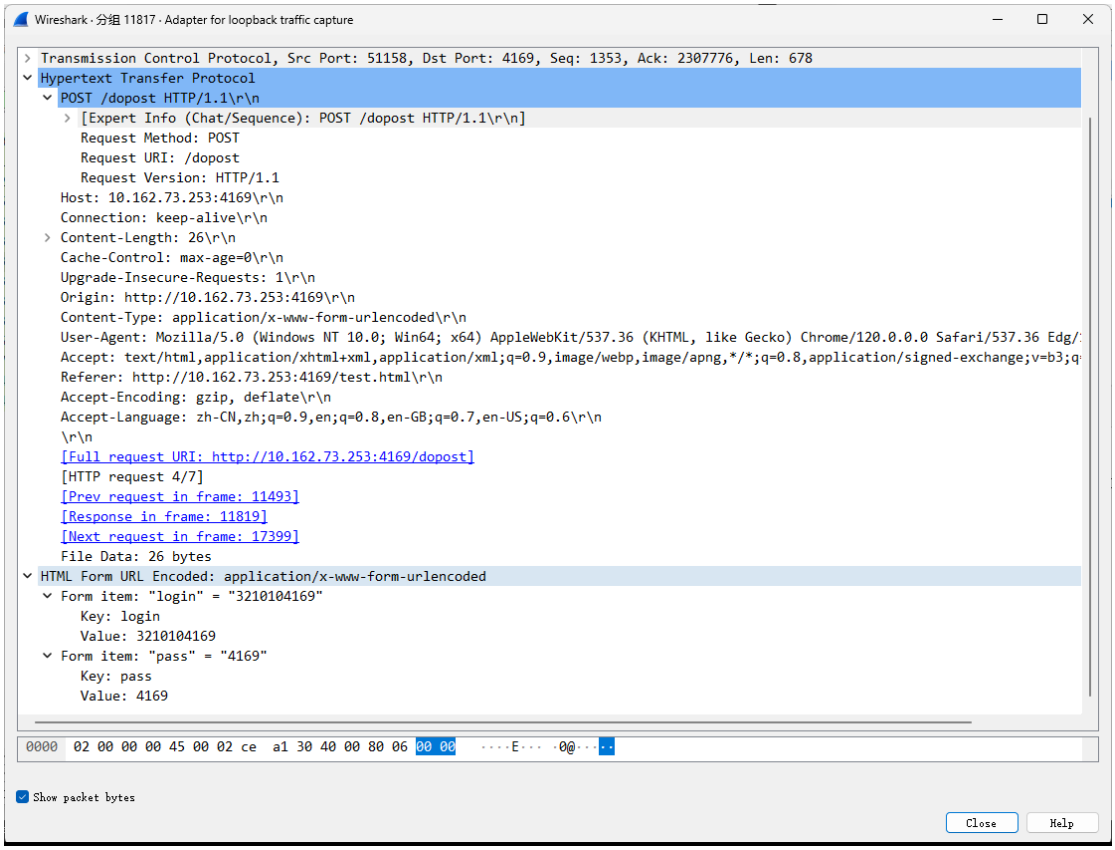
```
void HandlePost(SOCKET clientSocket, const std::string& file, const std::string& request) {
    std::cout << "Handling POST request" << std::endl;
    std::string successMsg = "<html><body><h1>Login Success</h1><p>From server: Login successfully.</p></body></html>\r\n";
    std::string failedMsg = "<html><body><h1>Login Failed</h1><p>From server: Login failed.</p></body></html>\r\n";
    // parse "login=xxx&pass=xxx"
    std::string login = request.substr(request.find("login=") + strlen("login="));
    login = login.substr(0, login.find("&"));
    std::string password = request.substr(request.find("pass=") + strlen("pass="));
    password = password.substr(0, password.find("&"));

    std::string response = "HTTP/1.1 200 OK\r\n";
    response += "Content-Type: text/html\r\n";
    response += "Content-Length: ";

    if (login == "3210104169" && password == "4169") {
        response += std::to_string(successMsg.size());
        response += "\r\n\r\n";
        response += successMsg;
        std::cout << "Login successfully" << std::endl;
    } else {
        response += std::to_string(failedMsg.size());
        response += "\r\n\r\n";
        response += failedMsg;
        std::cout << "Login failed" << std::endl;
    }

    SendHttpResponse(clientSocket, response);
}
```

在请求中解析出输入的账号密码，判断账号是否为 3210104169，密码是否为 4169，如果是，则登录成功，服务器发回登录成功的响应信息，否则发挥登录失败的响应信息。

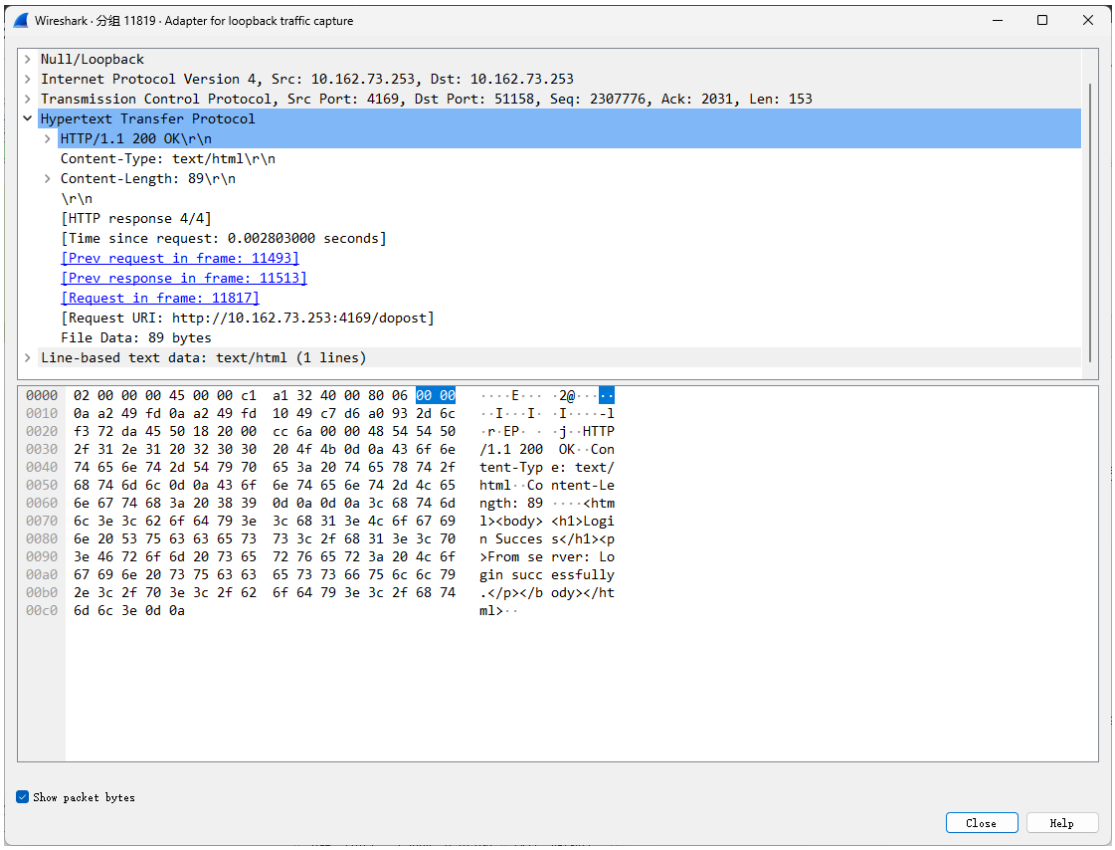
Wireshark 抓取的数据包截图（HTTP 协议部分）

```
11817 137.134573    10.162.73.253        10.162.73.253        HTTP      722 POST /dopost HTTP/1.1  (application/x-w
11819 137.137376    10.162.73.253        10.162.73.253        HTTP      197 HTTP/1.1 200 OK  (text/html)
```

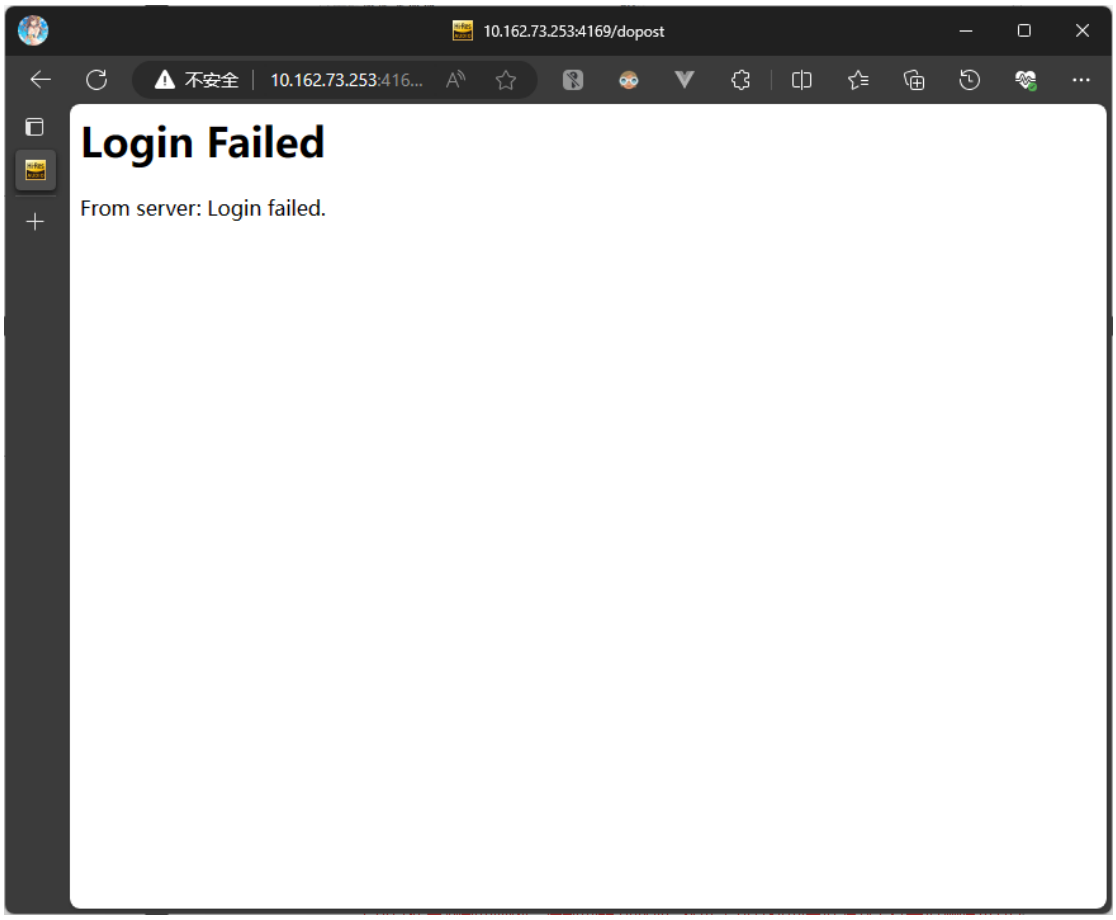
POST 请求：



响应：

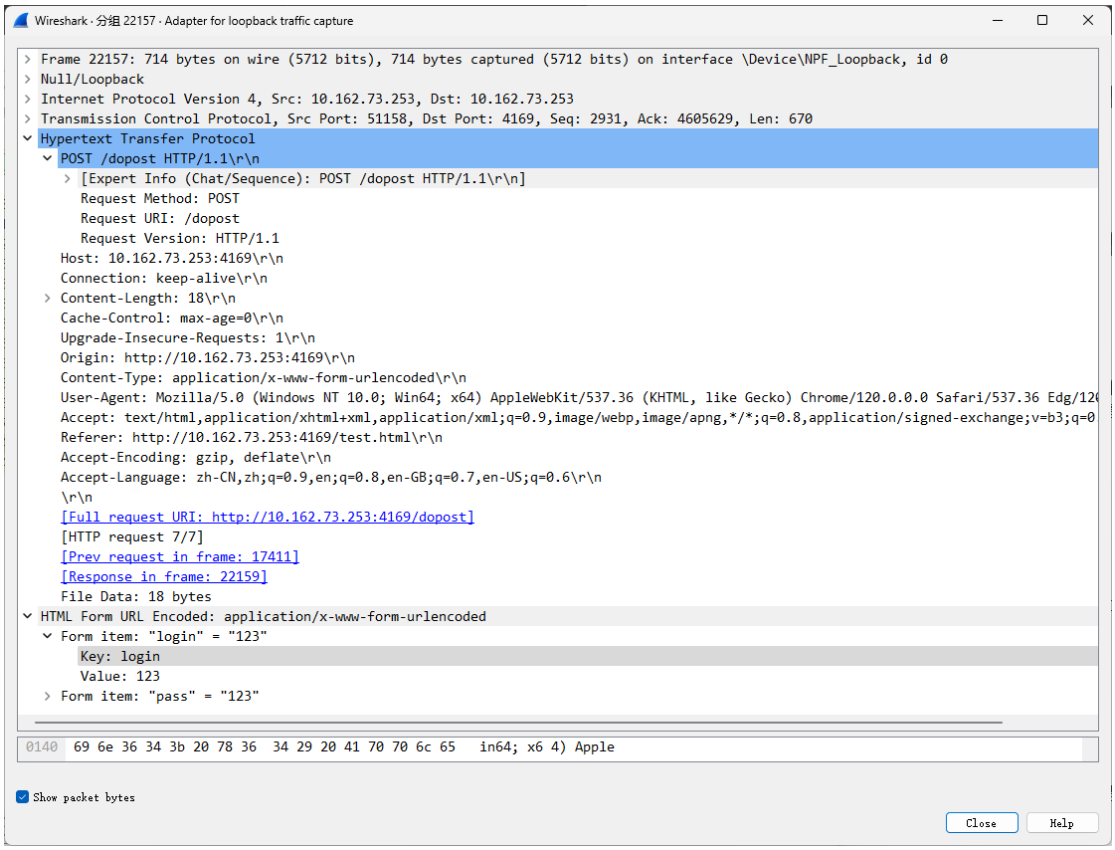


- 浏览器输入错误的登录名或密码，点击登录按钮（login）后的显示截图。

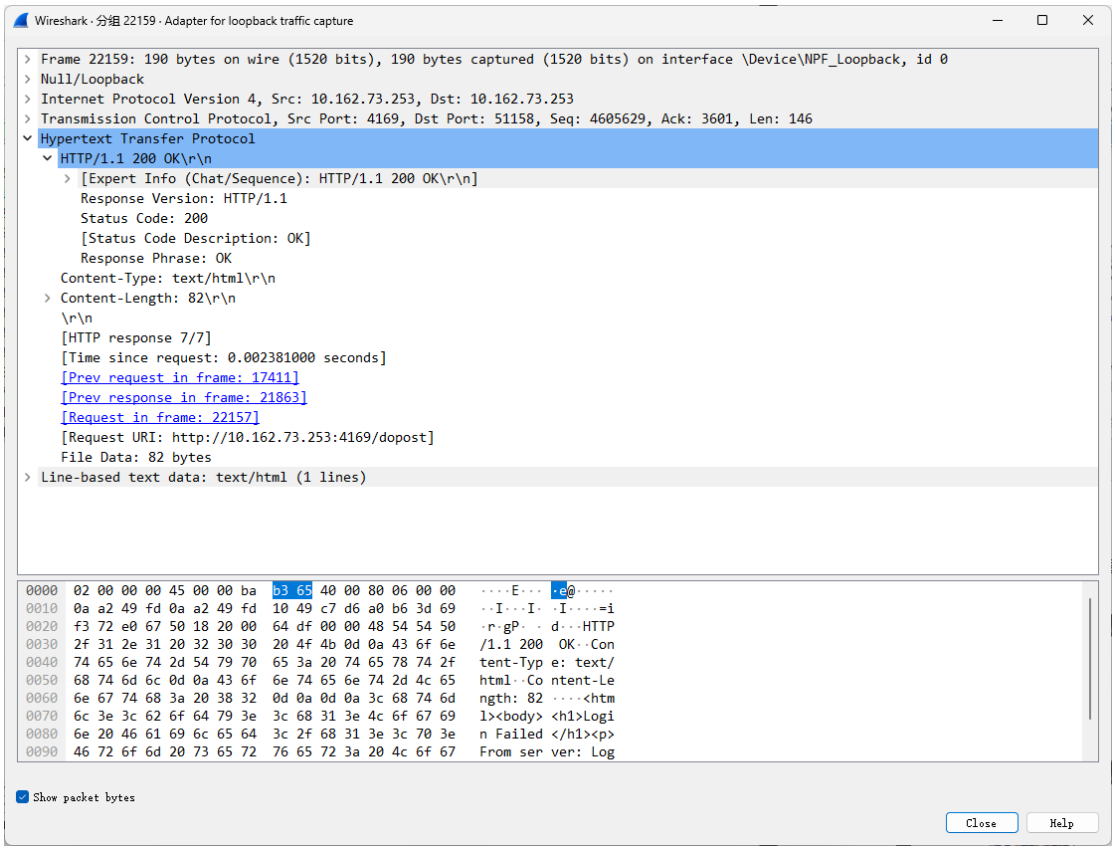


- Wireshark 抓取的数据包截图（HTTP 协议部分）

POST 请求：

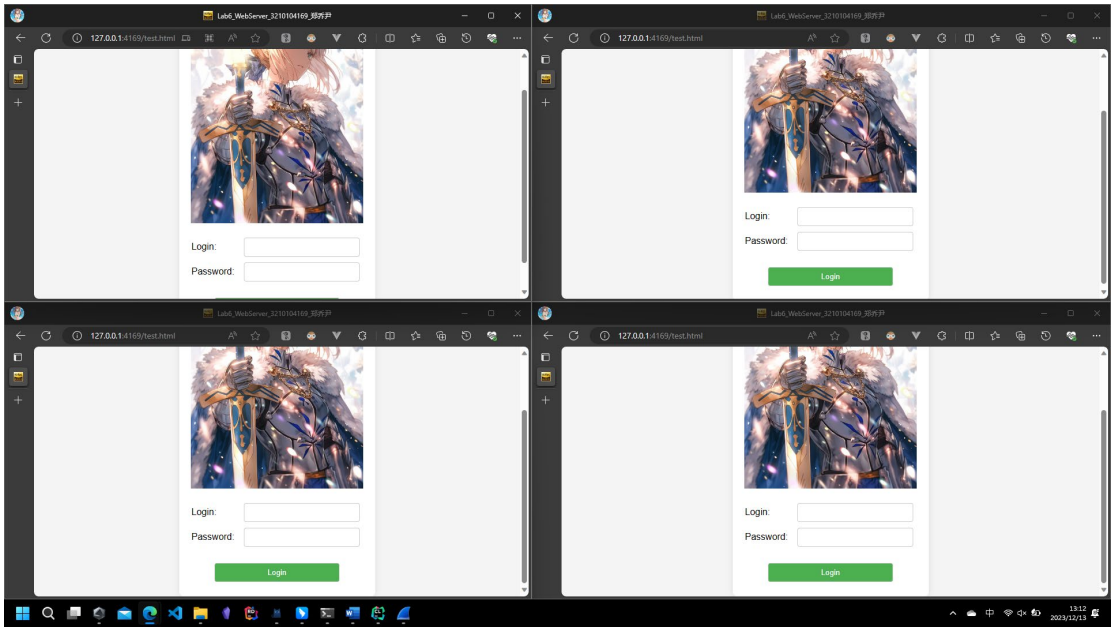


响应：

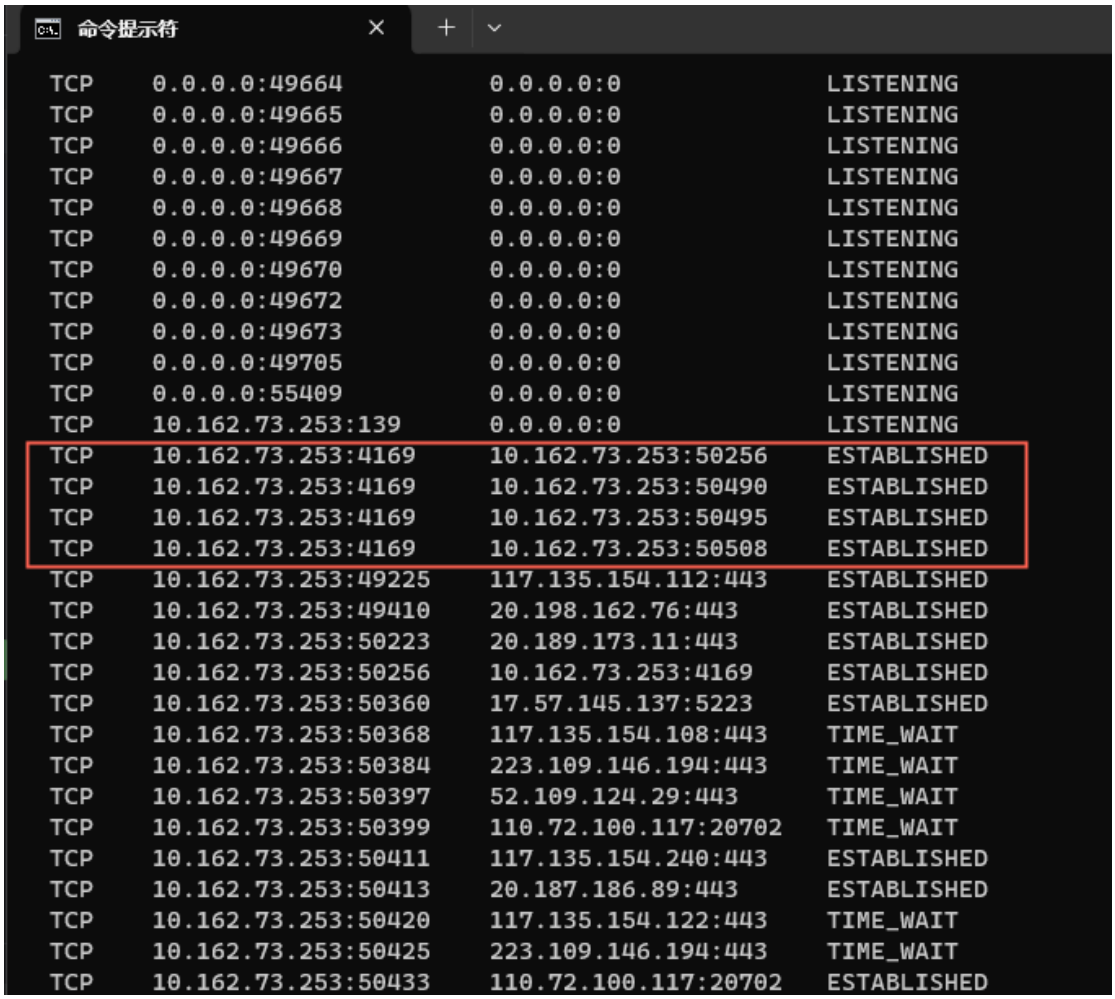


- 多个浏览器同时访问包含图片的 HTML 文件时，浏览器的显示内容截图（将浏览

器窗口缩小并列）



- 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）



六、 实验结果与分析

- HTTP 协议是怎样对头部和体部进行分隔的？

  HTTP 消息由头部和体部组成。头部和体部之间通常由一个空行(CRLF, \r\n）分隔。一旦遇到空行，解析器就知道头部结束了，接下来的是体部。

- 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？

  浏览器通常是根据 HTTP 响应头部中的 Content-Type 字段来确定文件的类型。如果 Content-Type 被设置为 text/html，浏览器会将其作为 HTML 文档处理。

- HTTP 协议的头部是不是一定是文本格式？体部呢？

  头部：HTTP 头部几乎总是以文本格式出现，遵循特定的键值对结构。

  体部：HTTP 体部的格式比较多样。它可以是文本，比如 HTML 或 JSON 数据，也可以是二进制数据，比如图像或文件下载。体部的数据类型通常由头部中的 Content-Type 字段指定。

- 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？

  在 HTTP 的 POST 请求中，传递的数据通常放在请求的体部。头部和体部之间由一个空行(\r\n)分隔。在 POST 请求中，头部可能包含一些描述体部内容的信息，如 Content-Type（指示数据的类型）和 Content-Length（指示数据的长度）。

七、 讨论、心得

实验过程中遇到的困难，得到的经验教训，对本实验安排的更好建议（看完请删除本句）