



南开大学
Nankai University

南 开 大 学

网 络 空 间 安 全 学 院

计算机网络实验报告

实验二

张政泽

年级：2022 级

专业：信息安全

学号：2213573

2024 年 11 月 1 日

目录

一、 Web 页面制作	1
二、 Web 服务器搭建	2
三、 Wireshark 捕获	5
四、 HTTP 交互过程	7
(一) 三次握手建立连接	7
(二) 客户端发送 HTTP 请求	8
(三) 服务端处理请求并发送响应	9
(四) 客户端接收响应	10
(五) 四次挥手关闭连接	10
五、 收获及问题总结	11

一、 Web 页面制作

实验中要求 Web 页面包含简单的文本信息（包括专业、学号、姓名）以及 Logo（png 格式）、音频信息（MP3 格式）。编写的 Web 页面代码如下：

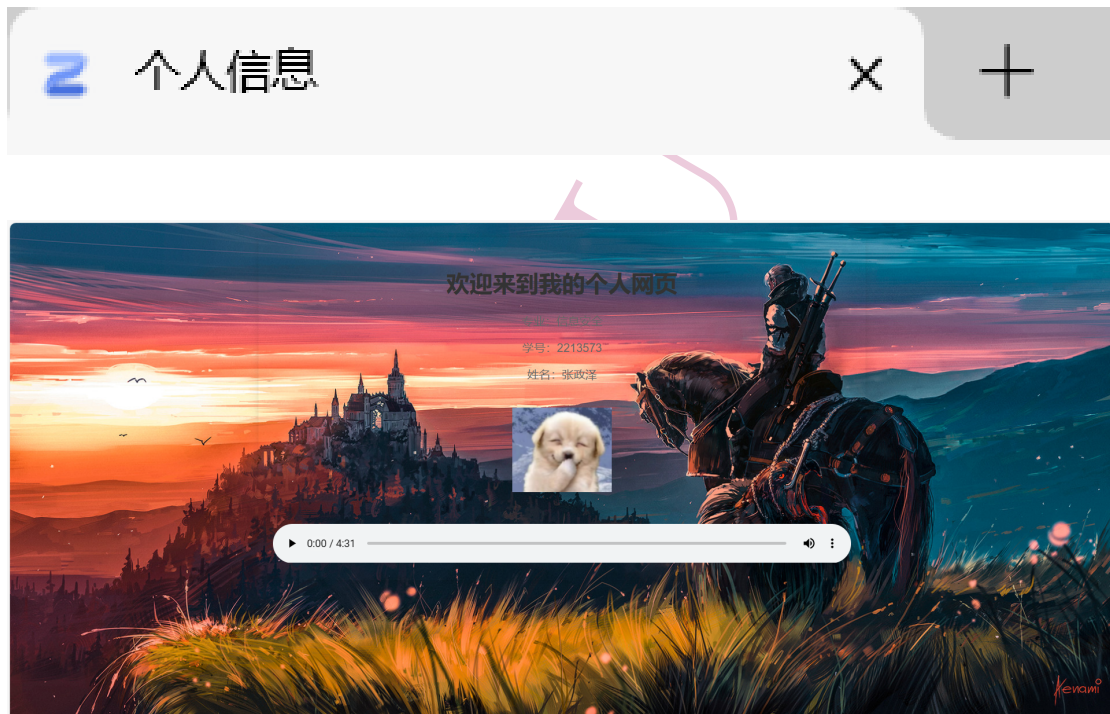
```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <title>个人信息</title>
6   <link rel="icon" type="image/x-icon" href="/z.png">
7
8   <style>
9     body {
10       font-family: Arial, sans-serif; /* 设置字体 */
11       margin: 0;
12       padding: 0;
13       background-color: #f4f4f4; /* 设置背景颜色 */
14       background-image: url('./bg.png'); /* 设置背景图片 */
15       background-size: cover; /* 覆盖整个背景 */
16       background-position: center; /* 图片居中显示 */
17       background-repeat: no-repeat; /* 不重复图片 */
18     }
19     .container {
20       max-width: 800px; /* 设置最大宽度 */
21       margin: 20px auto; /* 上下20px, 左右自动 */
22       padding: 20px;
23       background-color: transparent; /* 设置容器背景颜色 */
24       box-shadow: 0 0 10px rgba(0, 0, 0, 0.1); /* 添加阴影效果 */
25       text-align: center;
26     }
27     h1 {
28       color: #333; /* 设置标题颜色 */
29     }
30     p {
31       color: #666; /* 设置段落颜色 */
32     }
33     img {
34       max-width: 100%; /* 图片最大宽度为容器宽度 */
35       height: auto; /* 高度自适应 */
36       margin: 20px auto; /* 使图片居中 */
37     }
38     audio {
39       width: 100%; /* 音频播放器宽度 */
40       margin-top: 20px; /* 上边距 */
41     }
42   </style>
43 </head>
44 <body>
```

```

45 <div class="container">
46   <h1>欢迎来到我的个人网页</h1>
47   <p>专业：信息安全</p>
48   <p>学号：2213573</p>
49   <p>姓名：张政泽</p>
50   
51   <audio controls>
52     <source src="./music.mp3" type="audio/mpeg">
53   </audio>
54 </div>
55 </body>
56 </html>

```

页面效果如下图所示：



二、Web 服务器搭建

使用 Python 来搭建本地服务器，代码如下：

```

1 from http.server import SimpleHTTPRequestHandler, HTTPServer
2 from http.server import BaseHTTPRequestHandler, HTTPServer
3 import os
4
5 class CustomHandler(BaseHTTPRequestHandler):
6     def do_GET(self):
7         self.send_response(200)
8
9         # 根据路径返回不同的内容
10

```

```
11     if self.path == '/index.html':
12         index_file_path = 'Homework.html'
13         try:
14             with open(index_file_path, 'rb') as file:
15                 self.send_header('Content-type', 'text/html')
16                 self.end_headers()
17                 content = file.read()
18                 self.wfile.write(content)
19         except FileNotFoundError:
20             # 如果文件不存在, 返回404错误
21             self.send_response(404)
22             self.end_headers()
23             self.wfile.write(b'File_not_found.')
24         except Exception as e:
25             # 其他错误
26             self.send_response(500)
27             self.end_headers()
28             self.wfile.write(str(e).encode())
29     elif self.path == '/Logo.png':
30         try:
31             with open("./Logo.png", 'rb') as file:
32                 # 设置Content-Type
33                 self.send_header('Content-type', 'image/png')
34                 self.end_headers()
35                 content = file.read()
36                 self.wfile.write(content)
37         except FileNotFoundError:
38             # 如果文件不存在, 返回404错误
39             self.send_response(404)
40             self.end_headers()
41             self.wfile.write(b'File_not_found.')
42         except Exception as e:
43             # 其他错误
44             self.send_response(500)
45             self.end_headers()
46             self.wfile.write(str(e).encode())
47     elif self.path == '/z.png':
48         try:
49             with open("./z.png", 'rb') as file:
50                 # 设置Content-Type
51                 self.send_header('Content-type', 'image/png')
52                 self.end_headers()
53                 content = file.read()
54                 self.wfile.write(content)
55         except FileNotFoundError:
56             # 如果文件不存在, 返回404错误
57             self.send_response(404)
58             self.end_headers()
```

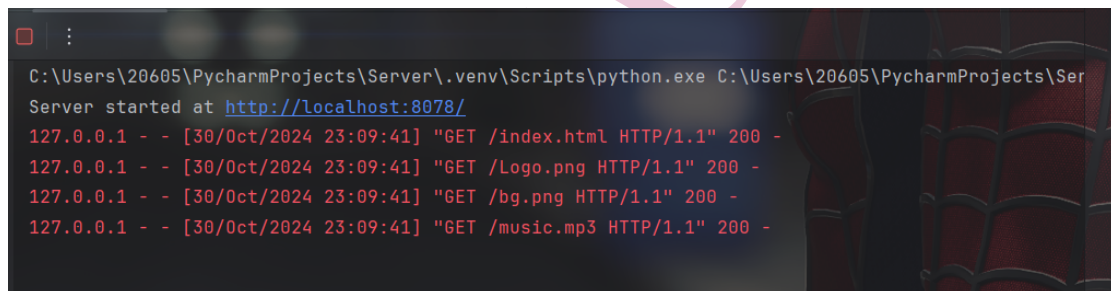
```
59         self.wfile.write(b'File_not_found.')
60     except Exception as e:
61         # 其他错误
62         self.send_response(500)
63         self.end_headers()
64         self.wfile.write(str(e).encode())
65     elif self.path == '/bg.png':
66         try:
67             with open("./bg.png", 'rb') as file:
68                 # 设置Content-Type
69                 self.send_header('Content-type', 'image/png')
70                 self.end_headers()
71                 content = file.read()
72                 self.wfile.write(content)
73         except FileNotFoundError:
74             # 如果文件不存在, 返回404错误
75             self.send_response(404)
76             self.end_headers()
77             self.wfile.write(b'File_not_found.')
78     except Exception as e:
79         # 其他错误
80         self.send_response(500)
81         self.end_headers()
82         self.wfile.write(str(e).encode())
83     elif self.path == '/music.mp3':
84         try:
85             with open("./music.mp3", 'rb') as file:
86                 # 设置Content-Type
87                 self.send_header('Content-type', 'audio/mpeg')
88                 self.end_headers()
89                 content = file.read()
90                 self.wfile.write(content)
91         except FileNotFoundError:
92             # 如果文件不存在, 返回404错误
93             self.send_response(404)
94             self.end_headers()
95             self.wfile.write(b'File_not_found.')
96     except Exception as e:
97         # 其他错误
98         self.send_response(500)
99         self.end_headers()
100        self.wfile.write(str(e).encode())
101    else:
102        # 对于其他路径, 返回404错误
103        self.send_response(404)
104        self.end_headers()
105        self.wfile.write(b'Not_Found.')
106
```

```
107 def run():
108     server_address = ('', 8078)
109     httpd = HTTPServer(server_address, CustomHandler)
110     print("Server started at http://localhost:8078/")
111     httpd.serve_forever()
112
113 if __name__ == '__main__':
114     run()
```

代码解析:

1. do_GET 方法会在服务器接收到 HTTP GET 请求时被调用。首先会发送一个 HTTP 响应状态码 200, 表示请求成功;紧接着会根据请求的路径尝试打开对应的文件, 若文件存在则发送其内容, 若文件不存在则返回 404 错误, 文件类型包括 html 文件、png 文件、MP3 文件等, 它们的处理逻辑类似, 但为了匹配不同的文件类型会设置不同的 Content-type
2. run 函数设置服务器的地址和端口 (本地 8078 端口), 创建一个 HTTPServer 实例, 并启动服务器, 使其无限循环等待和处理请求

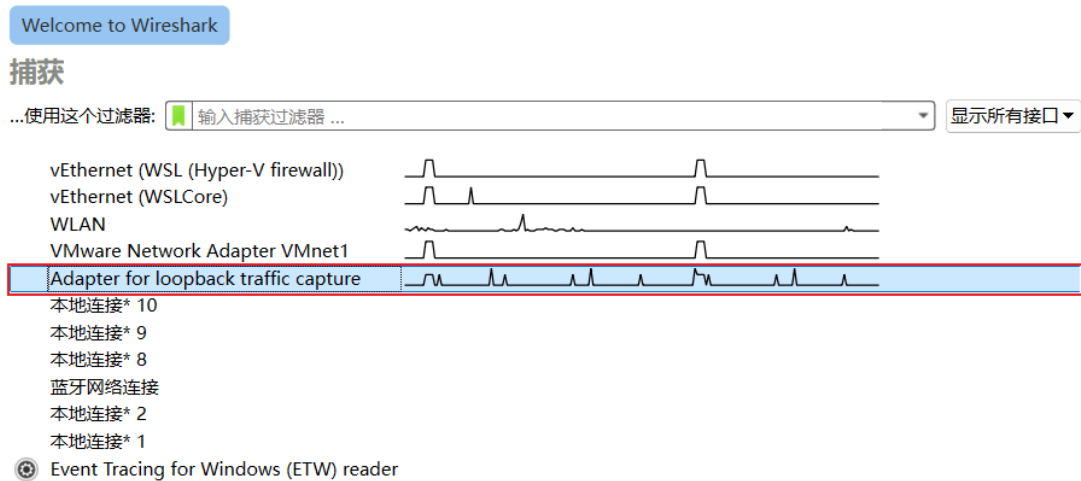
启动服务器, 并输入本地 IP 地址 + 端口号 + 索引进行访问, 服务器端输出如下部分响应信息:



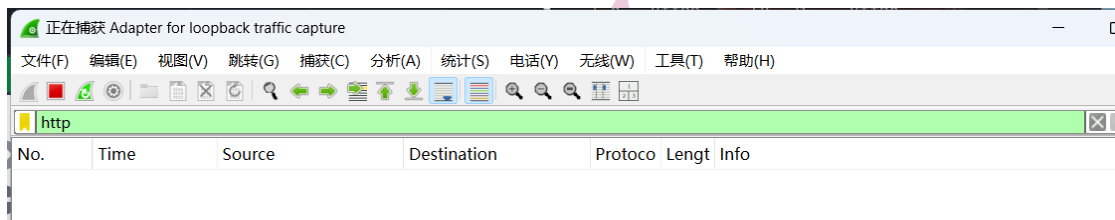
```
C:\Users\20605\PycharmProjects\Server\.venv\Scripts\python.exe C:\Users\20605\PycharmProjects\Server
Server started at http://localhost:8078/
127.0.0.1 - - [30/Oct/2024 23:09:41] "GET /index.html HTTP/1.1" 200 -
127.0.0.1 - - [30/Oct/2024 23:09:41] "GET /Logo.png HTTP/1.1" 200 -
127.0.0.1 - - [30/Oct/2024 23:09:41] "GET /bg.png HTTP/1.1" 200 -
127.0.0.1 - - [30/Oct/2024 23:09:41] "GET /music.mp3 HTTP/1.1" 200 -
```

三、Wireshark 捕获

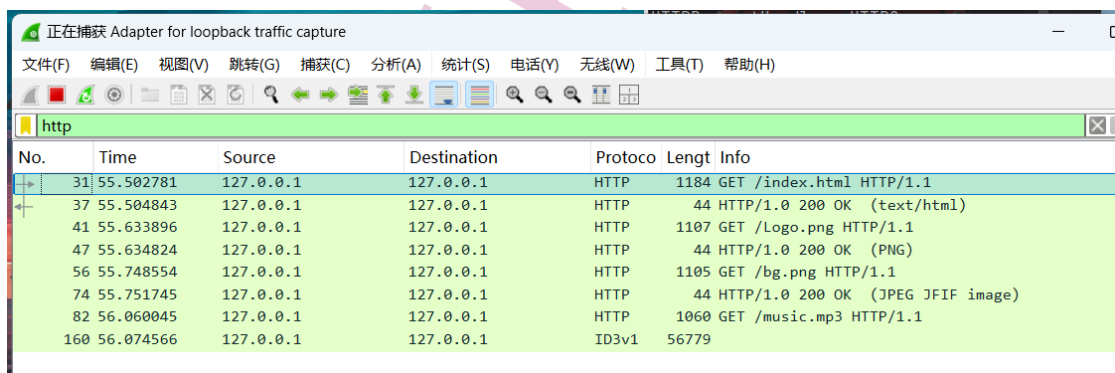
启动 wireshark, 由于服务器搭建在本机, 且访问服务器的也是本机, 因此选择 Adapter for loopback traffic capture 选项, 这一选项是 Windows 中使用的一个特殊网络适配器选项, 用于捕获发往和来自本机的网络流量, 即环回流量。



接下来设置过滤器为 http:



访问 Web 页面，并观察 Wireshark 捕获情况



同时为了了解 HTTP 的完整交互过程，再设置过滤器为 TCP && ip.addr=127.0.0.1, 捕获结果如下:

No.	Time	Source	Destination	Protocol	Length	Info
25	55.497531	127.0.0.1	127.0.0.1	TCP	56	56717 → 8078 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
26	55.497603	127.0.0.1	127.0.0.1	TCP	56	8078 → 56717 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
27	55.497628	127.0.0.1	127.0.0.1	TCP	44	56717 → 8078 [ACK] Seq=1 Ack=1 Win=327424 Len=0
28	55.497818	127.0.0.1	127.0.0.1	TCP	56	56718 → 8078 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
29	55.497851	127.0.0.1	127.0.0.1	TCP	56	8078 → 56718 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
30	55.497867	127.0.0.1	127.0.0.1	TCP	44	56718 → 8078 [ACK] Seq=1 Ack=1 Win=327424 Len=0
31	55.502781	127.0.0.1	127.0.0.1	HTTP	1184	GET /index.html HTTP/1.1
32	55.502826	127.0.0.1	127.0.0.1	TCP	44	8078 → 56717 [ACK] Seq=1 Ack=1141 Win=2160128 Len=0
33	55.503221	127.0.0.1	127.0.0.1	TCP	161	8078 → 56717 [PSH, ACK] Seq=1 Ack=1141 Win=2160128 Len=117 [TCP PDU reassembled in 37]
34	55.503954	127.0.0.1	127.0.0.1	TCP	44	56717 → 8078 [ACK] Seq=1141 Ack=118 Win=327168 Len=0
35	55.504725	127.0.0.1	127.0.0.1	TCP	1933	8078 → 56717 [PSH, ACK] Seq=118 Ack=1141 Win=2160128 Len=1889 [TCP PDU reassembled in 37]
36	55.504759	127.0.0.1	127.0.0.1	TCP	44	56717 → 8078 [ACK] Seq=1141 Ack=2007 Win=325376 Len=0
37	55.504843	127.0.0.1	127.0.0.1	HTTP	44	HTTP/1.0 200 OK (text/html)
38	55.508061	127.0.0.1	127.0.0.1	TCP	44	56717 → 8078 [ACK] Seq=1141 Ack=2008 Win=325376 Len=0
39	55.507230	127.0.0.1	127.0.0.1	TCP	44	56717 → 8078 [FIN, ACK] Seq=1141 Ack=2008 Win=325376 Len=0
40	55.507272	127.0.0.1	127.0.0.1	TCP	44	8078 → 56717 [ACK] Seq=2008 Ack=1142 Win=2160128 Len=0
41	55.633896	127.0.0.1	127.0.0.1	HTTP	1107	GET /Logo.png HTTP/1.1
42	55.633956	127.0.0.1	127.0.0.1	TCP	44	8078 → 56718 [ACK] Seq=1 Ack=1064 Win=2160128 Len=0
43	55.634475	127.0.0.1	127.0.0.1	TCP	161	8078 → 56718 [PSH, ACK] Seq=1 Ack=1064 Win=2160128 Len=117 [TCP PDU reassembled in 47]
44	55.635080	127.0.0.1	127.0.0.1	TCP	44	56718 → 8078 [ACK] Seq=1064 Ack=118 Win=327168 Len=0
45	55.634714	127.0.0.1	127.0.0.1	TCP	25277	8078 → 56718 [PSH, ACK] Seq=118 Ack=1064 Win=2160128 Len=25233 [TCP PDU reassembled in 47]
46	55.634759	127.0.0.1	127.0.0.1	TCP	44	56718 → 8078 [ACK] Seq=1064 Ack=25351 Win=302080 Len=0
47	55.634824	127.0.0.1	127.0.0.1	HTTP	44	HTTP/1.0 200 OK (PNG)
48	55.634838	127.0.0.1	127.0.0.1	TCP	44	56718 → 8078 [ACK] Seq=1064 Ack=25352 Win=302080 Len=0
49	55.645180	127.0.0.1	127.0.0.1	TCP	44	56718 → 8078 [FIN, ACK] Seq=1064 Ack=25352 Win=302080 Len=0
50	55.645165	127.0.0.1	127.0.0.1	TCP	44	8078 → 56718 [ACK] Seq=25352 Ack=1065 Win=2160128 Len=0
51	55.747569	127.0.0.1	127.0.0.1	TCP	56	56720 → 8078 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
52	55.747763	127.0.0.1	127.0.0.1	TCP	56	8078 → 56720 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
53	55.747783	127.0.0.1	127.0.0.1	TCP	44	56720 → 8078 [ACK] Seq=1 Ack=1 Win=2161152 Len=0
54	55.748554	127.0.0.1	127.0.0.1	HTTP	1105	GET /img HTTP/1.1
57	55.748585	127.0.0.1	127.0.0.1	TCP	44	8078 → 56720 [ACK] Seq=1 Ack=1062 Win=2160128 Len=0
58	55.749476	127.0.0.1	127.0.0.1	TCP	161	8078 → 56720 [PSH, ACK] Seq=1 Ack=1062 Win=2160128 Len=117 [TCP PDU reassembled in 74]
59	55.749511	127.0.0.1	127.0.0.1	TCP	44	56720 → 8078 [ACK] Seq=1062 Ack=118 Win=2161152 Len=0
60	55.750635	127.0.0.1	127.0.0.1	TCP	65539	8078 → 56720 [ACK] Seq=118 Ack=1062 Win=2160128 Len=65495 [TCP PDU reassembled in 74]

61	55.750664	127.0.0.1	127.0.0.1	TCP	65539	8078 → 56720 [ACK] Seq=65613 Ack=1062 Win=2160128 Len=65495 [TCP PDU reassembled in 74]
62	55.750675	127.0.0.1	127.0.0.1	TCP	65539	8078 → 56720 [ACK] Seq=131108 Ack=1062 Win=2160128 Len=65495 [TCP PDU reassembled in 74]
63	55.750688	127.0.0.1	127.0.0.1	TCP	65539	8078 → 56720 [ACK] Seq=196683 Ack=1062 Win=2160128 Len=65495 [TCP PDU reassembled in 74]
64	55.750710	127.0.0.1	127.0.0.1	TCP	65539	8078 → 56720 [ACK] Seq=260309 Ack=1062 Win=2160128 Len=65495 [TCP PDU reassembled in 74]
65	55.750740	127.0.0.1	127.0.0.1	TCP	65539	8078 → 56720 [ACK] Seq=327593 Ack=1062 Win=2160128 Len=65495 [TCP PDU reassembled in 74]
66	55.750776	127.0.0.1	127.0.0.1	TCP	65539	8078 → 56720 [ACK] Seq=393888 Ack=1062 Win=2160128 Len=65495 [TCP PDU reassembled in 74]
67	55.750804	127.0.0.1	127.0.0.1	TCP	65539	8078 → 56720 [ACK] Seq=458583 Ack=1062 Win=2160128 Len=65495 [TCP PDU reassembled in 74]
68	55.750830	127.0.0.1	127.0.0.1	TCP	65539	8078 → 56720 [ACK] Seq=524078 Ack=1062 Win=2160128 Len=65495 [TCP PDU reassembled in 74]
69	55.750871	127.0.0.1	127.0.0.1	TCP	65539	8078 → 56720 [ACK] Seq=589573 Ack=1062 Win=2160128 Len=65495 [TCP PDU reassembled in 74]
70	55.751339	127.0.0.1	127.0.0.1	TCP	44	56720 → 8078 [ACK] Seq=1062 Ack=655068 Win=2161152 Len=0
71	55.751401	127.0.0.1	127.0.0.1	TCP	65539	8078 → 56720 [ACK] Seq=655068 Ack=1062 Win=2160128 Len=65495 [TCP PDU reassembled in 74]
72	55.751418	127.0.0.1	127.0.0.1	TCP	5044	8078 → 56720 [PSH, ACK] Seq=720563 Ack=1062 Win=2160128 Len=5000 [TCP PDU reassembled in 74]
73	55.751469	127.0.0.1	127.0.0.1	TCP	44	56720 → 8078 [ACK] Seq=1062 Ack=725563 Win=2090752 Len=0
74	55.751745	127.0.0.1	127.0.0.1	HTTP	44	HTTP/1.0 200 OK (JPEG image)
75	55.751764	127.0.0.1	127.0.0.1	TCP	44	56720 → 8078 [ACK] Seq=1062 Ack=725564 Win=2090752 Len=0
76	55.759304	127.0.0.1	127.0.0.1	TCP	44	[TCP Window Update] 56720 → 8078 [ACK] Seq=1062 Ack=725564 Win=2161152 Len=0
77	55.760622	127.0.0.1	127.0.0.1	TCP	44	56720 → 8078 [FIN, ACK] Seq=1062 Ack=725564 Win=2161152 Len=0
78	55.760678	127.0.0.1	127.0.0.1	TCP	44	8078 → 56720 [ACK] Seq=725564 Ack=1063 Win=2160128 Len=0
79	56.059143	127.0.0.1	127.0.0.1	TCP	56	56724 → 8078 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
80	56.059190	127.0.0.1	127.0.0.1	TCP	56	8078 → 56724 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
81	56.059218	127.0.0.1	127.0.0.1	TCP	44	56724 → 8078 [ACK] Seq=1 Ack=1 Win=2161152 Len=0
82	56.060045	127.0.0.1	127.0.0.1	HTTP	1060	GET /music.mp3 HTTP/1.1
83	56.060066	127.0.0.1	127.0.0.1	TCP	44	8078 → 56724 [ACK] Seq=1 Ack=1017 Win=2160128 Len=0
84	56.060507	127.0.0.1	127.0.0.1	TCP	162	8078 → 56724 [PSH, ACK] Seq=1 Ack=1017 Win=2160128 Len=118 [TCP PDU reassembled in 160]
85	56.060524	127.0.0.1	127.0.0.1	TCP	44	56724 → 8078 [ACK] Seq=1017 Ack=119 Win=2151152 Len=0
86	56.063525	127.0.0.1	127.0.0.1	TCP	65539	8078 → 56724 [ACK] Seq=119 Ack=1017 Win=2160128 Len=65495 [TCP PDU reassembled in 160]
87	56.063545	127.0.0.1	127.0.0.1	TCP	65539	8078 → 56724 [ACK] Seq=56514 Ack=1017 Win=2160128 Len=65495 [TCP PDU reassembled in 160]
88	56.063553	127.0.0.1	127.0.0.1	TCP	65539	8078 → 56724 [ACK] Seq=131109 Ack=1017 Win=2160128 Len=65495 [TCP PDU reassembled in 160]
89	56.063560	127.0.0.1	127.0.0.1	TCP	65539	8078 → 56724 [ACK] Seq=196604 Ack=1017 Win=2160128 Len=65495 [TCP PDU reassembled in 160]
90	56.063583	127.0.0.1	127.0.0.1	TCP	65539	8078 → 56724 [ACK] Seq=260809 Ack=1017 Win=2160128 Len=65495 [TCP PDU reassembled in 160]
91	56.063594	127.0.0.1	127.0.0.1	TCP	65539	8078 → 56724 [ACK] Seq=327594 Ack=1017 Win=2160128 Len=65495 [TCP PDU reassembled in 160]
92	56.063602	127.0.0.1	127.0.0.1	TCP	65539	8078 → 56724 [ACK] Seq=393889 Ack=1017 Win=2160128 Len=65495 [TCP PDU reassembled in 160]
93	56.063610	127.0.0.1	127.0.0.1	TCP	65539	8078 → 56724 [ACK] Seq=458584 Ack=1017 Win=2160128 Len=65495 [TCP PDU reassembled in 160]

四、 HTTP 交互过程

HTTP 的基本交互流程为：建立连接-> 发送 HTTP 请求-> 服务器端发送 HTTP 响应-> 客户端接收响应-> 关闭连接,HTTP 协议具有以下特征：

1. 基于请求与响应：客户端发送请求，服务端响应数据
2. 无状态：当客户端向服务器发送请求，服务端响应完毕后，两者会断开连接并且不保存连接状态
3. 应用层协议：Http 属于应用层协议
4. 基于 TCP/IP 协议传输数据：客户端与服务器通过更底层的 TCP 建立连接

HTTP 协议中的数据传输基于传输层的 TCP 协议来完成，通过建立传输层的 TCP 连接来实现客户端与服务端之间的通信。

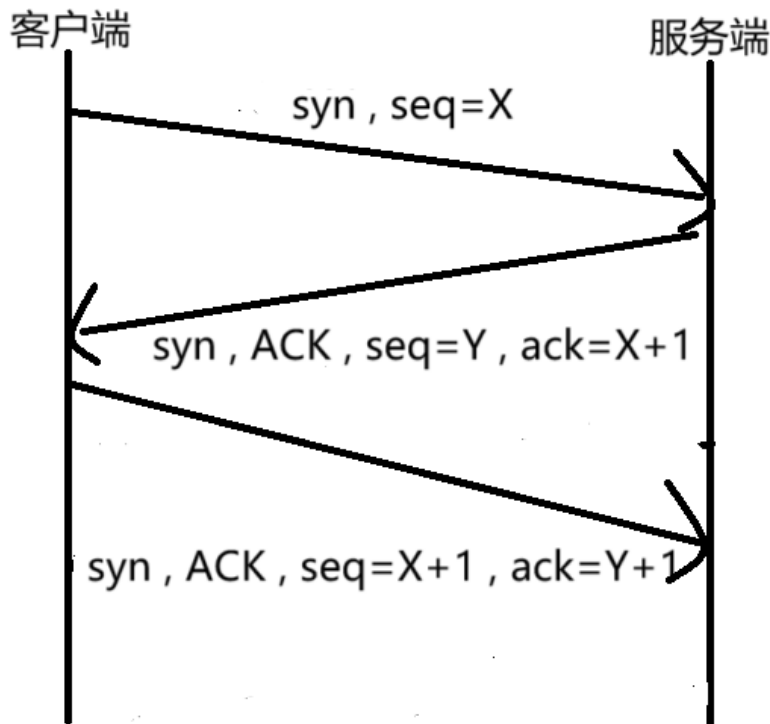
(一) 三次握手建立连接

TCP 通过三次握手建立连接，在这一过程中同步连接双方的序列号和确认号并交换 TCP 窗口大小信息。具体流程为：

1. 第一次握手-客户端发送 SYN 报文, 设置 SYN 字段为 1 及初始序列号 X，表示客户端希望建立一个连接并告知服务器客户端的初始序列号

2. 第二次握手-服务端在收到客户端的 SYN 请求后, 若同意建立连接则会响应一个 TCP 段: 服务端发送 SYN+ACK 报文, 设置 SYN 字段为 1, 设置初始序列号 Y 及 ACK 字段为 X+1
3. 第三次握手-客户端接收到服务器的 SYN-ACK 响应后, 发送一个带有 ACK 标志位的 TCP 段作为确认: 客户端发送 ACK 报文, 设置发送序列号为 X+1 及 ACK 字段为 Y+1

流程图如下所示:



Wireshark 捕获的结果中可以发现这一过程

25	9.678889	127.0.0.1	127.0.0.1	TCP	56 64657 → 8078 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
26	9.678169	127.0.0.1	127.0.0.1	TCP	56 8078 → 64657 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
27	9.678224	127.0.0.1	127.0.0.1	TCP	44 64657 → 8078 [ACK] Seq=1 Ack=1 Win=327424 Len=0

1. 首先, 64657 端口 (客户端) 向 8078 端口 (服务器) 发送 SYN 报文, 设置发送序列号 X=1
2. 接着, 8078 端口向 64657 端口回复 SYN+ACK 报文, 设置发送序列号为 0 及确认序列号为 1
3. 最后, 64657 端口再次向 8078 端口发送 ACK 报文, 设置发送序列号为 1, 确认序列号为 1

另外, 查阅资料发现, TCP 需要通过三次握手建立连接而不是两次的原因: 在网络传输过程中, 可能由于网络阻塞导致第一次握手请求信息未能发送到服务器, 这种情况下待超时后客户端会再次发送第一次握手请求。若旧的连接请求在之后的某个时刻到达服务器, 若此时是两次握手的方式, 那么就会错误的打开一个不必要的新连接; 而三次握手方式需要客户端的再次确认, 可以避免这一问题。

(二) 客户端发送 HTTP 请求

HTTP 请求包括以下几个部分:

1. 请求行: 包含请求方法 (GET/POST)、请求 URL 和 HTTP 的版本
2. 请求头: 包含以下字段: Host、Connection、Content-Type、User-Agent、Accept、Referer、Accept-Language、Cookie 等
3. 请求体 (可选): 在 POST 和 PUT 请求中包含要发送给服务器的数据, 如表单上的数据及文件上传内容等。

wireshark 捕获到的一个 HTTP 请求如下:

```
> GET /index.html HTTP/1.1\r\n
Host: localhost:8078\r\n
Connection: keep-alive\r\n
sec-ch-ua: "Chromium";v="130", "Microsoft Edge";v="130", "Not?A_Brand";v="99"\r\n
sec-ch-ua-mobile: ?0\r\n
sec-ch-ua-platform: "Windows"\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/130.0.0.0 Safari/537.36 Edg/130.0.0.0\r\n
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7\r\n
Sec-Fetch-Site: none\r\n
Sec-Fetch-Mode: navigate\r\n
Sec-Fetch-User: ?1\r\n
Sec-Fetch-Dest: document\r\n
Accept-Encoding: gzip, deflate, br, zstd\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
> [_]Cookie: username=localhost-8888=2|1:0|10:1730122448|23:username=localhost-8888|204:eyJ1c2VybmFtZSI6I1M2NiZTQ0YzUxN2M0YzFhYVNmMTBhYzJlKjEwMWE2ODM1MCIsICJuYW11\r\n
\r\n
[Response in frame: 37]
[Full request URI: http://localhost:8078/index.html]
```

这一 HTTP 请求的第一行为请求行, 剩余部分为请求头, 未包含请求体。

(三) 服务端处理请求并发送响应

服务端在接收到请求后, 会解析请求, 根据请求内容进行处理并生成 HTTP 响应。生成的 HTTP 响应包含以下几个部分:

1. 状态行: 包括 HTTP 版本、状态码和状态描述。不同状态码对应的信息如下:
 - (a) 100 Continue: 服务器已收到请求的初始部分, 客户端可以继续发送请求的其余部分。
 - (b) 101 Switching Protocols: 服务器正在切换协议, 例如从 HTTP/1.1 切换到 WebSocket。
 - (c) 200 OK: 请求成功, 且服务器返回了请求的资源。
 - (d) 201 Created: 请求成功并且服务器创建了新的资源, 常见于 POST 请求。
 - (e) 202 Accepted: 请求已接受, 但尚未处理完成。
 - (f) 204 No Content: 请求成功, 但没有返回任何内容。
 - (g) 301 Moved Permanently: 请求的资源已永久移动到新位置, 新的 URL 在响应中给出。
 - (h) 302 Found: 请求的资源暂时移动到另一个位置 (常见用于临时重定向)。
 - (i) 304 Not Modified: 请求的资源未修改, 自上次请求以来未更改, 客户端可以使用缓存的版本。
 - (j) 400 Bad Request: 请求无效, 通常由于语法错误。
 - (k) 401 Unauthorized: 请求未被授权, 需提供身份验证凭据。
 - (l) 403 Forbidden: 服务器拒绝请求, 用户没有访问权限。
 - (m) 404 Not Found: 请求的资源未找到, URL 无效或资源已删除。
 - (n) 408 Request Timeout: 服务器在等待请求时超时。
 - (o) 500 Internal Server Error: 服务器遇到意外情况, 导致无法完成请求。

- (p) 501 Not Implemented: 服务器不支持请求的方法
 - (q) 502 Bad Gateway: 服务器作为网关或代理时, 接收到上游服务器的无效响应
 - (r) 503 Service Unavailable: 服务器当前无法处理请求 (可能由于暂时过载或维护)。
 - (s) 504 Gateway Timeout: 服务器作为网关时, 未能及时从上游服务器获取请求。
2. 响应头: 包括关于响应的额外信息, 如 Server、Content-type 等
 3. 响应体: 包含实际需要返回给客户端的数据, 如 HTML 文档、JS 脚本、CSS 文件等

wireshark 捕获到的一个 HTTP 响应如下:

```
▼ Hypertext Transfer Protocol
  > HTTP/1.0 200 OK\r\n
    Server: BaseHTTP/0.6 Python/3.12.3\r\n
    Date: Fri, 01 Nov 2024 06:06:29 GMT\r\n
    Content-type: image/png\r\n
    \r\n
    [Request in frame: 45]
    [Time since request: 0.005069000 seconds]
    [Request URI: /Logo.png]
    [Full request URI: http://localhost:8078/Logo.png]
    File Data: 25233 bytes
```

HTTP 版本: 1.0, 状态码: 200, 状态描述: OK, 响应的类型:image/png

(四) 客户端接收响应

当客户端接收到服务器的响应信息后, 通过解析响应信息, 从响应体中获取到所请求的 HTML/CSS/JavaScript 信息, 然后将这些信息呈现给用户。

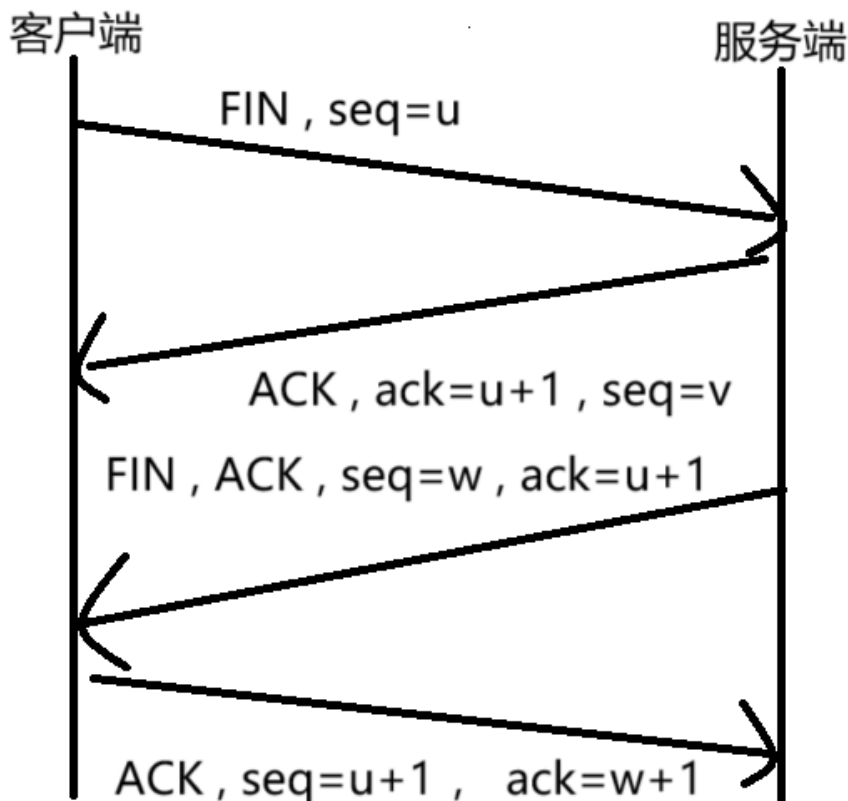
1. 首先根据响应中状态行获取 HTTP 版本、状态码和状态描述信息, 再从响应头中获取响应头各个字段的信息。
2. 进一步评估状态码, 根据解析得到的状态码来决定下一步动作。
3. 若状态码为 2XX 表示请求成功, 继续处理响应体内容。根据 Content-type 字段的值, 客户端对响应体进行相应的处理, 例如对于 text/html 类型的内容会将其作为网页进行渲染。
4. 最后, 将处理好的响应结果呈现给用户。

(五) 四次挥手关闭连接

终止一个 TCP 连接需要经过四次挥手的过程, 由于 TCP 连接的特性 (数据在两个方向上能同时传递), 因此每个方向需要单独关闭连接, 即客户端完成数据发送任务后需发送一个终止连接信息, 服务端需要对此报文进行响应; 同样地, 当服务器完成数据发送任务后需要发送一个终止连接信息, 客户端也需要对此报文进行响应。具体的流程如下:

1. 第一次挥手: 客户端向服务端发送报文, FIN 标志位被置位, 设置序列号 seq 为 u, 表示客户端完成了数据传输任务, 请求关闭连接。
2. 第二次挥手: 服务端向客户端发送报文, 设置序列号 seq=v, ack=u+1
3. 第三次挥手: 报文 FIN 标志位被置位, 序列号 seq=w, ack=u+1
4. 第四次挥手: 客户端向服务端发送报文, 序列号 seq=u+1, ack=w+1

流程图如下所示：



Wireshark 捕获的结果中可以发现这一过程

201.70.873617	127.0.0.1	127.0.0.1	TCP	44 64663 → 8078 [FIN, ACK] Seq=1 Ack=1 Win=2161152 Len=0
202.70.873678	127.0.0.1	127.0.0.1	TCP	44 8078 → 64663 [ACK] Seq=1 Ack=2 Win=2161152 Len=0
203.70.873936	127.0.0.1	127.0.0.1	TCP	44 8078 → 64663 [FIN, ACK] Seq=1 Ack=2 Win=2161152 Len=0
204.70.874028	127.0.0.1	127.0.0.1	TCP	44 64663 → 8078 [ACK] Seq=2 Ack=2 Win=2161152 Len=0

注：64663 端口代表客户端，8078 端口代表服务端

1. 客户端主动发起关闭连接请求，客户端向服务端发送 FIN 包：客户端向服务端发送报文，其 FIN 标志位被置位，序列号 $seq=1$ ，表示客户端完成了数据传输任务，请求关闭连接。
2. 服务端收到 FIN 包后发送 ACK 包，确认收到 FIN 包：服务端向客户端发送报文，序列号 $seq=1, ack=2(1+1)$
3. 服务器在发送完所有数据后，发起关闭连接请求，服务器向客户端发送 FIN 包：其 FIN 标志位被置位，序列号 $seq=1, ack=2(1+1)$
4. 客户端收到 FIN 包后发送 ACK 包，确认收到 FIN 包：客户端向服务端发送报文，序列号 $seq=2, ack=2$

五、收获及问题总结

此次实验相对来说比较简单，主要就是 Web 网页编写 + Wireshark 流量捕获，实验过程中主要遇到了两个问题：第一个就是对 html 语言不太熟悉，因此在网页编写上遇到一定困难，但

好在并没有要求太复杂的网页；第二个就是在 wireshark 流量捕获时，由于在本地搭建服务器并本地访问，即使用本地回环地址，因此捕获时应该选择 Adapter for loopback traffic capture 选项，而我一开始选择的是 WLAN，导致未能捕获到预期的结果。

此次实验过程中特别是在撰写实验报告时，我详细地学习了解了 HTTP 的完整交互过程，同时也进一步地了解了传输层的 TCP 协议，尤其是其中的三次握手建立连接和四次挥手关闭连接的操作；另外在编写 Web 网页时学习了一些 html 语言相关的知识；最后就是熟悉了 Wireshark 工具的基本使用。

NIJUB