



南開大學
Nankai University

计算机学院
计算机网络实验报告

实验二

姓名：杨冰雪
学号：2110508
专业：计算机科学与技术

2023 年 11 月 1 日

目录

1 实验要求	2
2 实验过程	2
2.1 搭建 web 服务器	2
2.2 制作 web 页面	2
2.3 Wireshark 捕获浏览器	3
2.4 分析交互过程	4
2.4.1 TCP 三次握手	4
2.4.2 HTTP 请求数据包	5
2.4.3 HTTP 响应数据包	7
2.4.4 TCP 四次挥手	8
3 实验总结	10

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（至少包含专业、学号、姓名）、自己的 LOGO、自我介绍的音频信息。页面不要太复杂，包含要求的基本信息即可。
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. 使用 HTTP，不要使用 HTTPS。
4. 提交实验报告。

2 实验过程

2.1 搭建 web 服务器

实验平台：Windows

实验工具：Apache

1. 下载 Windows 下的最新 ZIP 压缩包，解压到自定义的文件夹。
2. 修改配置文件，用文本编辑器打开 Apache24\conf\httpd.conf 配置文件，修改此处 Apache 的地址。
3. 调用以下命令安装 Apache 服务并启动服务。

```
1 "httpd -k install -n Apache2.4"  
2 "httpd -k start"
```

4. 测试 Apache 服务器是否成功搭建，在浏览器访问 **http://localhost** 如显示 It works! 证明安装并启动成功。

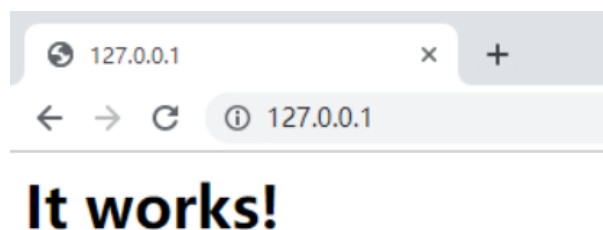


图 2.1: web 服务器搭建结果

2.2 制作 web 页面

实验工具：PyCharm

使用 PyCharm 建立一个简单的 web 网页，包含我的姓名、学号、专业、一张图片作为自己的 logo 和自我介绍的音频信息。代码实现如下：

web 网页代码

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <title>ComputerNet-lab2</title>
6     <link rel="icon" href="./data/logo.jpg">
7 </head>
8 <body background="./data/background.jpg" >
9 <center><h3>杨冰雪</h3></center>
10 <center><h3>2110508</h3></center>
11 <center><h3>计算机科学与技术</h3></center>
12 <center></center><br>
13 <center><audio controls height="100" width="100">
14     <source src="./data/self_introduction.mp3" type="audio/mpeg">
15     <source src="horse.ogg" type="audio/ogg">
16     <embed height="50" width="100" src="./data/self_introduction.mp3">
17 </audio></center>
18 </body>
19 </html>
```

其效果如下：



图 2.2: web 网页展示

2.3 Wireshark 捕获浏览器

实验工具：Wireshark

1. 开启 Wireshark 软件，由于此处为本机作为客户端去访问本机作为服务器搭建的网站，因此是一个 loopback，选择 Adapter for loopback traffic capture。

2. 进入后我们使用过滤器，筛选出 `tcp.port == 80`，以分析浏览器与 Web 服务器的交互过程。
3. 使用浏览器打开我们自建的 html 页面。
4. 可以看出使用的是 http1.1 协议，客户端首先发送一个访问页面的 get 请求，服务端收到 get 请求后将所请求内容发送给客户端，下图中 304 Not Modified 说明抓包成功。

No.	Time	Source	Destination	Protocol	Length	Info
418	7.495318	:::1	:::1	TCP	76	49190 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=65475 WS=256 SACK_PERM
419	7.495427	:::1	:::1	TCP	76	80 → 49190 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65475 WS=256 SACK_PERM
420	7.495480	:::1	:::1	TCP	64	49190 → 80 [ACK] Seq=1 Ack=1 Win=2618880 Len=0
421	7.496039	:::1	:::1	HTTP	829	GET / HTTP/1.1
422	7.496085	:::1	:::1	TCP	64	80 → 49190 [ACK] Seq=1 Ack=766 Win=2618880 Len=0
423	7.496901	:::1	:::1	HTTP	312	HTTP/1.1 304 Not Modified
424	7.496943	:::1	:::1	TCP	64	49190 → 80 [ACK] Seq=766 Ack=249 Win=2618880 Len=0
425	7.497216	:::1	:::1	TCP	76	49191 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=65475 WS=256 SACK_PERM
426	7.497327	:::1	:::1	TCP	76	80 → 49191 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65475 WS=256 SACK_PERM
427	7.497395	:::1	:::1	TCP	64	49191 → 80 [ACK] Seq=1 Ack=1 Win=2618880 Len=0

图 2.3: WireShark 抓包

2.4 分析交互过程

2.4.1 TCP 三次握手

客户端与服务端建立连接之前，TCP 协议会进行“三次握手”，观察上图中所得到的数据包，发现在 GET / HTTP/1.1 之前出现了【SYN】，【SYN,ACK】，【ACK】，这就是“三次握手”的过程。

第一次握手 第一次握手时，客户端将 TCP 报文标志位 SYN 置为 1，随机产生一个序号值 seq=0，保存在 TCP 首部的序列号 (Sequence Number) 字段里，指明客户端打算连接的服务器的端口，并将该数据包发送给服务器端，发送完毕后，客户端进入 SYN_SENT 状态，等待服务器端确认。

```
> Frame 418: 76 bytes on wire (608 bits), 76 bytes captured (608 bits) on interface \Device\NPF_{Loopback}, id 0
> Null/Loopback
> Internet Protocol Version 6, Src: ::1, Dst: ::1
▼ Transmission Control Protocol, Src Port: 49190, Dst Port: 80, Seq: 0, Len: 0
  Source Port: 49190
  Destination Port: 80
  [Stream index: 18]
  [Conversation completeness: Complete, WITH_DATA (31)]
  [TCP Segment Len: 0]
  Sequence Number: 0 (relative sequence number)
  Sequence Number (raw): 449145671
  [Next Sequence Number: 1 (relative sequence number)]
  Acknowledgment Number: 0
  Acknowledgment number (raw): 0
  1000 .... = Header Length: 32 bytes (8)
▼ Flags: 0x002 (SYN)
  000. .... = Reserved: Not set
  ...0 .... = Accurate ECN: Not set
  ....0... = Congestion Window Reduced: Not set
  ....0... = ECN-Echo: Not set
  ....0... = Urgent: Not set
  ....0... = Acknowledgment: Not set
  ....0... = Push: Not set
  ....0... = Reset: Not set
  ....0... = Syn: Set
  ....0... = Fin: Not set
  [TCP Flags: .....S.]
  Window: 65535
  [Calculated window size: 65535]
  Checksum: 0x2e7c [unverified]
  [Checksum Status: Unverified]
  Urgent Pointer: 0
> Options: (12 bytes), Maximum segment size, No-Operation (NOP), Window scale, No-Operation (NOP), No-Operation (NOP), SACK permitted
> [Timestamps]
```

图 2.4: 第一次握手

第二次握手 第二次握手实际上是服务器端收到数据包后由标志位 SYN=1 知道客户端请求建立连接，服务器端将 TCP 报文标志位 SYN 和 ACK 都置为 1，ack=1，随机产生一个序号值 seq=0，并将该数据包发送给客户端以确认连接请求，服务器端进入 SYN_RCVD 状态。

```

> Frame 419: 76 bytes on wire (608 bits), 76 bytes captured (608 bits) on interface \Device\NPF_{Loopback, id 0}
> Null/Loopback
> Internet Protocol Version 6, Src: ::1, Dst: ::1
> Transmission Control Protocol, Src Port: 80, Dst Port: 49190, Seq: 0, Ack: 1, Len: 0
  Source Port: 80
  Destination Port: 49190
  [Stream index: 18]
  [Conversation completeness: Complete, WITH_DATA (31)]
  [TCP Segment Len: 0]
  Sequence Number: 0 (relative sequence number)
  Sequence Number (raw): 3400116630
  [Next Sequence Number: 1 (relative sequence number)]
  Acknowledgment Number: 1 (relative ack number)
  Acknowledgment number (raw): 449145672
  1000 .... = Header Length: 32 bytes (8)
  Flags: 0x012 (SYN, ACK)
    000. .... = Reserved: Not set
    ...0 .... = Accurate ECN: Not set
    ....0... = Congestion Window Reduced: Not set
    ....0... = ECN-Echo: Not set
    ....0... = Urgent: Not set
    ....1... = Acknowledgment: Set
    ....0... = Push: Not set
    ....0... = Reset: Not set
  > ....1... = Syn: Set
    ....0... = Fin: Not set
    [TCP Flags: .....A..S.]
  Window: 65535
  [Calculated window size: 65535]
  Checksum: 0xba2a [unverified]
  [Checksum Status: Unverified]
  Urgent Pointer: 0
  > Options: (12 bytes), Maximum segment size, No-Operation (NOP), Window scale, No-Operation (NOP), No-Operation (NOP), SACK permitted
  > [Timestamps]
  > [SEQ/ACK analysis]

```

图 2.5: 第二次握手

第三次握手 第三次握手是客户端客户端收到确认后，检查 ack 是否为 1，ACK 是否为 1，如果正确则将标志位 ACK 置为 1，ack=1，再次发送确认包 (ACK) 向服务端，服务器端检查 ack 是否为 1，ACK 是否为 1，如果正确则连接建立成功，客户端和服务端进入 ESTABLISHED 状态，完成三次握手，随后客户端与服务端之间可以开始传输数据了。

```

> Frame 420: 64 bytes on wire (512 bits), 64 bytes captured (512 bits) on interface \Device\NPF_{Loopback, id 0}
> Null/Loopback
> Internet Protocol Version 6, Src: ::1, Dst: ::1
> Transmission Control Protocol, Src Port: 49190, Dst Port: 80, Seq: 1, Ack: 1, Len: 0
  Source Port: 49190
  Destination Port: 80
  [Stream index: 18]
  [Conversation completeness: Complete, WITH_DATA (31)]
  [TCP Segment Len: 0]
  Sequence Number: 1 (relative sequence number)
  Sequence Number (raw): 449145672
  [Next Sequence Number: 1 (relative sequence number)]
  Acknowledgment Number: 1 (relative ack number)
  Acknowledgment number (raw): 3400116631
  0101 .... = Header Length: 20 bytes (5)
  Flags: 0x010 (ACK)
    000. .... = Reserved: Not set
    ...0 .... = Accurate ECN: Not set
    ....0... = Congestion Window Reduced: Not set
    ....0... = ECN-Echo: Not set
    ....0... = Urgent: Not set
    ....1... = Acknowledgment: Set
    ....0... = Push: Not set
    ....0... = Reset: Not set
    ....0... = Syn: Not set
    ....0... = Fin: Not set
    [TCP Flags: .....A....]
  Window: 10230
  [Calculated window size: 2618880]
  [Window size scaling factor: 256]
  Checksum: 0xcd17 [unverified]
  [Checksum Status: Unverified]
  Urgent Pointer: 0
  > [Timestamps]
  > [SEQ/ACK analysis]

```

图 2.6: 第三次握手

2.4.2 HTTP 请求数据包

三次握手后，服务器和客户端建立了连接，才开始传输数据和进行 http 访问。首先浏览器向域名发出 HTTP 请求，使用的协议为 HTTP1.1，请求访问的界面是 index.html，请求访问方式为 GET。

- User-Agent: 标识客户端使用的浏览器和操作系统信息。
- Accept: 指定客户端能够处理的内容类型, 即可接受的媒体类型。
- Content-Type: 指定请求体中的数据格式类型。
- Cookie: 包含来自客户端的 Cookie 信息。
- Referer: 指示当前请求是从哪个 URL 页面发起的。
- Host: 指定服务器的域名或 IP 地址。

```
1 " Host: localhost\r\n"
```

- Content-Length: 指定请求体的长度。

3. 空行

发送回车符和换行符, 其意为告诉服务器请求头部到此截止。

4. 请求体

请求体是封装 POST 请求消息的请求参数的, 由于此处为 GET 请求, 故而没有数据, 若为 POST 请求, 此处要提交数据。

2.4.3 HTTP 响应数据包

在请求被目的主机通过后, 本来应该回应其封装好的 TEXT/HTML 形式数据, 但服务器判断出在本地已经缓存了要访问的资源, 所以这里没有响应体, 服务器会返回 HTTP/304 Not Modified 响应头, 客户端收到 304 响应后, 就会从本地缓存中读取对应的资源。

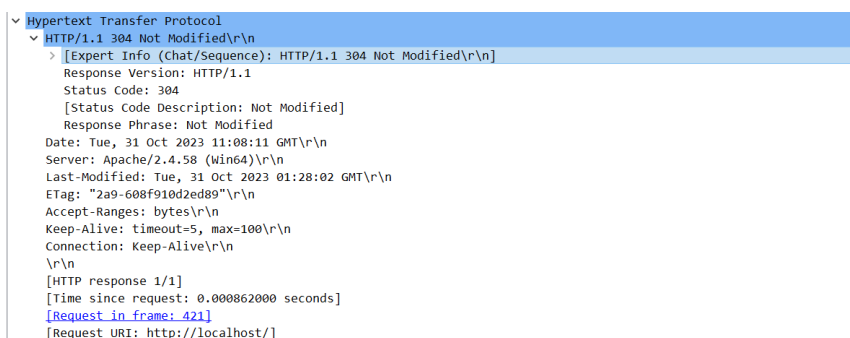


图 2.9: 响应报文 _index.html

而对于自我介绍的音频, 回应其封装好的 audio/mpeg 形式数据。206 Partial Content 表示该服务器已经成功处理了部分 GET 请求。此类响应使用断点续传或者将一个文档分解为多个下载段同时下载。Content-Range 用以指示本次响应中返回的内容的范围; Content-Length 表示响应返回的内容范围的真实字节数。


```

v Hypertext Transfer Protocol
  > HTTP/1.1 206 Partial Content\r\n
    Date: Tue, 31 Oct 2023 11:08:29 GMT\r\n
    Server: Apache/2.4.58 (win64)\r\n
    Last-Modified: Mon, 23 Oct 2023 12:14:59 GMT\r\n
    ETag: "813e1a9-608612bc8974a"\r\n
    Accept-Ranges: bytes\r\n
  > Content-Length: 6386022\r\n
    Content-Range: bytes 6229658-12615679/135520681\r\n
    Keep-Alive: timeout=5, max=100\r\n
    Connection: Keep-Alive\r\n
    Content-Type: audio/mpeg\r\n
  \r\n
  [HTTP response 1/1]
  [Time since request: 0.028742000 seconds]
  [Request in frame: 611]
  [Request URI: http://localhost/data/self_introduction.mp3]
  File Data: 6386022 bytes
v Media Type
  Media type: audio/mpeg (6386022 bytes)

```

图 2.10: 响应报文 _self_introduction.mp3

响应报文由响应行、响应头、空行和响应体四部分组成

1. 响应行

```
1 "HTTP/1.1 304 Not Modified\r\n"
```

- 响应行一般由协议版本、状态码及其描述组成。
- 在本次实验中，协议版本为 HTTP1.1，状态码为 304。

2. 响应头

响应头用于描述服务器的基本信息，以及数据的描述，服务器通过这些数据的描述信息，可以通知客户端如何处理等一会儿它回送的数据。Content-Length 指明了返回的内容长度。

3. 空行

与 HTTP 请求报文类似，此处空行的意义为响应头部结束。

4. 响应体

响应体就是响应的消息体，本次实验中访问了 html 页面和一个音频，故而返回所编写的 HTML 代码和一个音频部分内容。

2.4.4 TCP 四次挥手

当数据传送完成后，断开连接，TCP 进行了“四次挥手”。

761 30.570385	::1	::1	TCP	64 80 → 49191	[FIN, ACK] Seq=6386378 Ack=597 Win=2618880 Len=0
762 30.570427	::1	::1	TCP	64 49191 → 80	[ACK] Seq=597 Ack=6386379 Win=623616 Len=0
826 39.199423	::1	::1	TCP	64 49191 → 80	[FIN, ACK] Seq=597 Ack=6386379 Win=623616 Len=0
827 39.199519	::1	::1	TCP	64 80 → 49191	[ACK] Seq=6386379 Ack=598 Win=2618880 Len=0

图 2.11: TCP 四次挥手

第一次挥手 主动关闭方发送一个请求结束的报文【FIN,ACK】给被动关闭方，以关闭主动关闭方到被动关闭方的数据传送，并进入 FIN_WAIT1 状态。

```

Transmission Control Protocol, Src Port: 80, Dst Port: 49191, Seq: 6386378, Ack: 597, Len: 0
  Source Port: 80
  Destination Port: 49191
  [Stream index: 19]
  [Conversation completeness: Complete, WITH_DATA (31)]
  [TCP Segment Len: 0]
  Sequence Number: 6386378 (relative sequence number)
  Sequence Number (raw): 419837579
  [Next Sequence Number: 6386379 (relative sequence number)]
  Acknowledgment Number: 597 (relative ack number)
  Acknowledgment number (raw): 3611356689
  0101 .... = Header Length: 20 bytes (5)
  Flags: 0x011 (FIN, ACK)
    000. .... = Reserved: Not set
    ...0 .... = Accurate ECN: Not set
    ....0... = Congestion Window Reduced: Not set
    ....0... = ECN-Echo: Not set
    ....0... = Urgent: Not set
    ....1... = Acknowledgment: Set
    ....0... = Push: Not set
    ....0... = Reset: Not set
    ....0... = Syn: Not set
    ....1... = Fin: Set
  [TCP Flags: .....A...F]

```

图 2.12: 第一次挥手

第二次挥手 被动关闭方接收到主动关闭方发送的【FIN,ACK】并发送【ACK】，表示确认。确认序号 = 收到序号 + 1；此时被动关闭方进入 CLOSE_WAIT 状态；主动关闭方收到被动关闭方的【ACK】后，进入 FIN_WAIT2 状态。

```

Source Port: 49191
Destination Port: 80
[Stream index: 19]
[Conversation completeness: Complete, WITH_DATA (31)]
[TCP Segment Len: 0]
Sequence Number: 597 (relative sequence number)
Sequence Number (raw): 3611356689
[Next Sequence Number: 597 (relative sequence number)]
Acknowledgment Number: 6386379 (relative ack number)
Acknowledgment number (raw): 419837580
0101 .... = Header Length: 20 bytes (5)
Flags: 0x010 (ACK)
  000. .... = Reserved: Not set
  ...0 .... = Accurate ECN: Not set
  ....0... = Congestion Window Reduced: Not set
  ....0... = ECN-Echo: Not set
  ....0... = Urgent: Not set
  ....1... = Acknowledgment: Set
  ....0... = Push: Not set
  ....0... = Reset: Not set
  ....0... = Syn: Not set
  ....0... = Fin: Not set
[TCP Flags: .....A....]
Window: 2436

```

图 2.13: 第二次挥手

第三次挥手 被动关闭方发送一个【FIN,ACK】以关闭被动关闭方到主动关闭方的数据传送，并进入 LAST_ACK 状态。

```

Transmission Control Protocol, Src Port: 49191, Dst Port: 80, Seq: 597, Ack: 6386379, Len: 0
  Source Port: 49191
  Destination Port: 80
  [Stream index: 19]
  [Conversation completeness: Complete, WITH_DATA (31)]
  [TCP Segment Len: 0]
  Sequence Number: 597 (relative sequence number)
  Sequence Number (raw): 3611356689
  [Next Sequence Number: 598 (relative sequence number)]
  Acknowledgment Number: 6386379 (relative ack number)
  Acknowledgment number (raw): 419837580
  0101 .... = Header Length: 20 bytes (5)
  Flags: 0x011 (FIN, ACK)
    000. .... = Reserved: Not set
    ...0 .... = Accurate ECN: Not set
    ....0... = Congestion Window Reduced: Not set
    ....0... = ECN-Echo: Not set
    ....0... = Urgent: Not set
    ....1... = Acknowledgment: Set
    ....0... = Push: Not set
    ....0... = Reset: Not set
    ....0... = Syn: Not set
    ....1... = Fin: Set
  [TCP Flags: .....A...F]
Window: 2436

```

图 2.14: 第三次挥手

第四次挥手 主动关闭方收到被动关闭方发送的【FIN,ACK】并发送【ACK】，此时主动关闭方进入 TIME_WAIT 状态，经过 2MSL 时间后关闭连接；被动关闭方收到主动关闭方的【ACK】后，关闭连接。

```
Source Port: 80
Destination Port: 49191
[Stream index: 19]
[Conversation completeness: Complete, WITH_DATA (31)]
[TCP Segment Len: 0]
Sequence Number: 6386379 (relative sequence number)
Sequence Number (raw): 419837580
[Next Sequence Number: 6386379 (relative sequence number)]
Acknowledgment Number: 598 (relative ack number)
Acknowledgment number (raw): 3611356690
0101 .... = Header Length: 20 bytes (5)
▼ Flags: 0x010 (ACK)
  000. .... = Reserved: Not set
  ...0 .... = Accurate ECN: Not set
  .... 0... = Congestion Window Reduced: Not set
  .... .0.. = ECN-Echo: Not set
  .... ..0. = Urgent: Not set
  .... ...1 .... = Acknowledgment: Set
  .... .... 0... = Push: Not set
  .... .... .0.. = Reset: Not set
  .... .... ..0. = Syn: Not set
  .... .... ...0 = Fin: Not set
[TCP Flags: .....A....]
Window: 10230
```

图 2.15: 第四次挥手

3 实验总结

通过本次实验，学会了使用 Apache 来搭建 web 服务器，也对 html 有了更深入的了解，熟悉了 tcp 协议的三次握手和四次挥手，学会了通过使用 Wireshark 来捕获数据包并对其进行分析。