

实验 2：配置 Web 服务器，分析 HTTP 交互过程

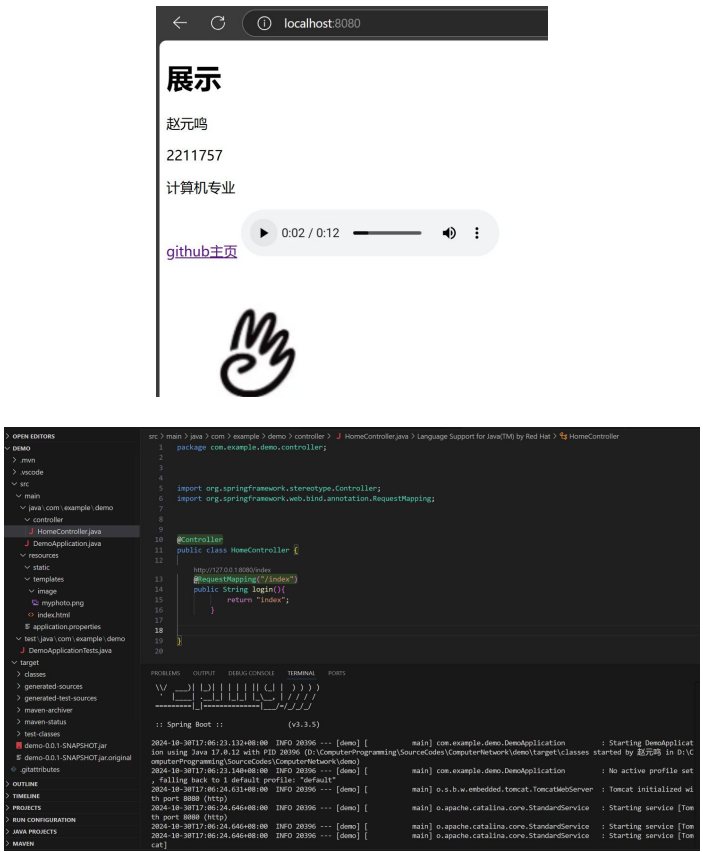
(赵元鸣 2211757 计算机科学与技术 0850 班)

一. 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，包含简单文本信息（至少包含专业、学号、姓名）、自己的 LOGO、自我介绍的音频信息。
- (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，使用 Wireshark 过滤器使其只显示 HTTP 协议。
- (4) 现场演示。
- (5) 提交 HTML 文档、Wireshark 捕获文件和实验报告，对 HTTP 交互过程进行详细说明。

二. 实验流程

1. 利用 springboot，同时使用 maven 框架搭建 web 服务器：



2. 使用 wireshark 抓包：

选择 Adapter for loopback traffic capture 接口，设定设置过滤器(ip.dst == 127.0.0.1 or ip.src == 127.0.0.1) and(tcp.srcport == 8080 or tcp.dstport == 8080)，得到抓包结果如下。

[illegible]

```

# Frame 1280: 56 bytes on wire (448 bits), 56 bytes captured (448 bits) on interface [Device]bpf_lo0, 0000000000000000
# MultiHopback
# Ethernet II, Version 2, Src: f22:0:0:0, Dst: f22:0:0:0
# Transmission Control Protocol, Src Port: 65535, Dst Port: 8089, Seq. #: 1, Len: 0
#   Window: 0
#   Destination Port: 8089
#   [Stream index: 85]
#   [Stream Packet Number: 1]
#   [Conversation Completeness: Complete, WITH_DATA [131]]
#   [TCP Segment Len: 0]
#   Sequence Number: 0 (relative sequence number)
#   Sequence Number: 0 (raw) [57939409]
#   [Next Sequence Number: 1 (relative sequence number)]
#   Acknowledgment Number: 0
#   Acknowledgment Number (raw): 0
#   [End Seq. = Window Length: 22 bytes (0)]
#   [Flags: none (0)]
#   RST: 0...0 = Reserved: Not set
#   ...0...0 = Accurate ECH, Not set
#   ...0...0 = Congestion Window Reduced: Not set
#   ...0...0 = ECHEcho, Not set
#   ...0...0 = Urgent: Not set
#   ...0...0 = Acknowledgment: Not set
#   ...0...0 = Push: Not set
#   ...0...0 = Reset: Not set
#   [...0...0 = Syn: Set]
#   [...0...0 = Fin: Not set]
#   [TCP flags: none (0)]
#   Window: 65535
#   [Calculated window size: 65535]
#   Checksum: 0x0000 [unverified]
#   [Checksum Status: Unverified]
#   Urgent pointer: 0
#   # Options: (12 bytes), Maximum segment size, No-Operation (NOP), Window scale, No-Operation (NOP),
#   # (Timestamps)

```

第二次握手：服务器端收到数据包后由标志位 SYN=1 得知客户端请求建立连接，然后便发送确认报文段（SYN+ACK 信令）到客户端，接着服务器进入 SYN_RECV 状态，握手成功说明服务端的数据可以被客户端收到，即说明服务端的发功能，客户端的收功能可用。同时客户端知道自己的数据已经正确到达服务端，自己的发功能正常。但是服务端自己不知道数据是否被接收。Seq = 0 表示初始建立值为 0，表示当前还没有发送数据；Ack = 1 表示当前端成功接收的数据位数，因为包含 SYN 或 FIN 标志位尽管客户端没有发送任何有效数据，确认号仍加 1。

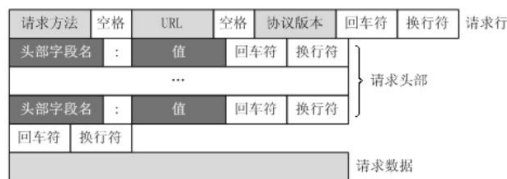
```
Frame 1204: 56 bytes on wire (448 bits), 56 bytes captured (448 bits) on interface \Device\NPF_{...}_id 0
Null/Loopback
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
0100 .... = Version: 4
.... 0101 = Header Length: 20 bytes (5)
Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
Total Length: 52
Identification: 0x4244 (16964)
010 .... = Flags: 0x2, Don't fragment
... 0 0000 0000 0000 = Fragment Offset: 0
Time to Live: 128
Protocol: TCP (6)
Header Checksum: 0x0000 [validation disabled]
[Header checksum status: Unverified]
Source Address: 127.0.0.1
Destination Address: 127.0.0.1
[Stream index: 0]
Transmission Control Protocol, Src Port: 8080, Dst Port: 65190, Seq: 0, Ack: 1, Len: 0
Source Port: 8080
Destination Port: 65190
[Stream index: 35]
[Stream Packet Number: 2]
[Conversation completeness: Complete, WITH_DATA (31)]
[TCP Segment Len: 0]
Sequence Number: 0 (relative sequence number)
Sequence Number (raw): 3643014113
[Next Sequence Number: 1 (relative sequence number)]
Acknowledgment Number: 1 (relative ack number)
Acknowledgment number (raw): 1673939410
1000 .... = Header Length: 32 bytes (8)
Flags: 0x02 (SYN, ACK)
Window: 65535
[Calculated window size: 65535]
Checksum: 0xd604 [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]
0000 02 00 00 00 45 00 00 34 42 44 40 00 00 06 00 00 ... E 800 ...
0010 7f 00 00 01 7f 00 00 01 1f 90 fe a6 d9 23 fb e1 ... # ...
0020 63 c6 49 d2 80 12 ff ff d6 04 00 00 02 04 ff d7 c I ...
0030 01 03 03 00 01 01 04 92
```

第三次握手：客户端收到服务器端确认后，检查 ack 是否为 x+1，ACK 是否为 1，如果正确则再发送确认报文段给服务器，客户端进入 ESTABLISHED 状态，服务器收到后也进入 ESTABLISHED 状态。

```
Frame 1205: 44 bytes on wire (352 bits), 44 bytes captured (352 bits) on interface \Device\NPF_{...}_id 0
Null/Loopback
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
0100 .... = Version: 4
.... 0101 = Header Length: 20 bytes (5)
Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
Total Length: 40
Identification: 0x4245 (16965)
010 .... = Flags: 0x2, Don't fragment
... 0 0000 0000 0000 = Fragment Offset: 0
Time to Live: 128
Protocol: TCP (6)
Header Checksum: 0x0000 [validation disabled]
[Header checksum status: Unverified]
Source Address: 127.0.0.1
Destination Address: 127.0.0.1
[Stream index: 0]
Transmission Control Protocol, Src Port: 65190, Dst Port: 8080, Seq: 1, Ack: 1, Len: 0
Source Port: 65190
Destination Port: 8080
[Stream index: 35]
[Stream Packet Number: 3]
[Conversation completeness: Complete, WITH_DATA (31)]
[TCP Segment Len: 0]
Sequence Number: 1 (relative sequence number)
Sequence Number (raw): 1673939410
[Next Sequence Number: 1 (relative sequence number)]
Acknowledgment Number: 1 (relative ack number)
Acknowledgment number (raw): 3643014114
0101 .... = Header Length: 20 bytes (5)
Flags: 0x010 (ACK)
Window: 8442
[Calculated window size: 216152]
[Window size scaling factor: 256]
Checksum: 0xf001 [unverified]
[Checksum status: Unverified]
Urgent Pointer: 0
Options: (0 bytes)
[Timestamps]
[SEQ/ACK analysis]
0000 02 00 00 00 45 00 00 28 42 45 40 00 00 06 00 00 ... E ( BE@ 15
0010 7f 00 00 01 7f 00 00 01 fe a6 1f 90 63 c6 49 d2 ... c I
0020 49 23 fb e2 50 10 20 fa f0 01 00 00
```

2. http 协议，发送和接受数据：

(1) 请求：GET / HTTP/1.1\r\n 称为请求行，由三个部分组成：请求方法、URI、HTTP 协议版本，他们之间用空格分隔。该部分位于数据首行。



基本格式为：该部分的请求方法字段（GET）给出了请求类型，URI 给出请求的资源位置），最后 HTTP 协议版本给出 HTTP 的版本号。接下来的部分称为请求头部，其紧跟着请求行，该部分主要是用于描述请求正文，主要是用于说明请求源、连接类型、以及一些 Cookie 信息等。下面时点开 http 的 get 中显示的协议的具体内容：

```

Hypertext Transfer Protocol
GET / HTTP/1.1\r\n
Host: 127.0.0.1:8080\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
\r\n

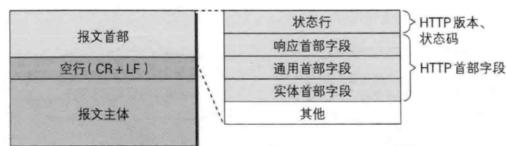
```

然后除了最常用的 get 和 post 请求方式，还有如下的请求方式：

- GET：用于从服务器获取资源，也是最常见请求方式。GET 请求将请求的参数附加在 URL 的末尾，发送给服务器。
- POST：用于向服务器提交数据，一般用于发送表单数据。POST 请求将请求的参数放在请求的主体中，而不是 URL 中。
- PUT：用于向服务器上传文件或更新资源。PUT 请求会将请求的数据存储在服务器上指定的位置。
- DELETE：用于删除服务器上的资源。
- HEAD：用于获取服务器对资源的头部信息，而不获取实际的资源内容。
- OPTIONS：用于获取服务器支持的请求方法。

GET	TRACE	UNLOCK
POST	CONNECT	MKCOL
PUT	COPY	MOVE
DELETE	LINK	PROPFIND
OPTIONS	UNLINK	REPORT
HEAD	PURGE	VIEW
PATCH	LOCK	

（2）响应： 响应报文由状态行、响应头部、空行和响应体 4 个部分构成。首先是状态行，主要给出响应 HTTP 协议的版本号、响应返回状态码、响应描述；响应头部主要是返回一些服务器的基本信息，以及一些 Cookie 值等；响应体为所请求的数据。




```

Frame 1215: 49 bytes on wire (392 bits), 49 bytes captured (392 bits) on interface \Device\NPF_{...}
Null/Loopback
  Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
    0000 ... = Version: 4
    ... 0101 = Header Length: 20 bytes (5)
    Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 45
    Identification: 0x424f (16975)
    010 ... = Flags: 0x2, Don't fragment
    ... 0000 0000 0000 = Fragment Offset: 0
    Time to Live: 128
    Protocol: TCP (6)
    Header Checksum: 0x0000 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 127.0.0.1
    Destination Address: 127.0.0.1
    [Stream index: 0]
  Transmission Control Protocol, Src Port: 8080, Dst Port: 65190, Seq: 827, Ack: 713, Len: 5
    [2 Reassembled TCP Segments (831 bytes): #1213(826), #1215(5)]
    Hypertext Transfer Protocol, has 2 chunks (including last chunk)
      HTTP/1.1 200 OK
      Content-Type: text/html;charset=UTF-8\r\n
      Content-Language: zh-CN\r\n
      Transfer-Encoding: chunked\r\n
      Date: Fri, 01 Nov 2024 07:58:50 GMT\r\n
      Keep-Alive: timeout=60\r\n
      Connection: keep-alive\r\n
      \r\n
      [Request in frame: 1211]
      [Time since request: 0.002161000 seconds]
      [Request URI: /]
      [Full request URI: http://127.0.0.1:8080/]
    HTTP chunked response
      File Data: 625 bytes
    Line-based text data: text/html (25 lines)
      <!DOCTYPE html>\r\n
      <html lang="en">\r\n
      <head>\r\n
      <meta charset="UTF-8">\r\n
      <title>计算机网络作业</title>\r\n
      <body>\r\n
      <h1>展示</h1>\r\n
      <div class="container">\r\n
      <div class="info">\r\n

```

(3) 在本次实验数据中, 可以看到在三次握手建立连接与四次挥手关闭连接中间有 3 次 HTTP 请求与响应。以传递 text/html 那一轮为例, 首先客户端向服务器端发送请求

```

1211 73.198962 127.0.0.1 127.0.0.1 HTTP 756 GET / HTTP/1.1

Frame 1211: 756 bytes on wire (6048 bits), 756 bytes captured (6048 bits) on interface \Device\NPF_{...}
Null/Loopback
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
0100 ....= Version: 4
0101 ....= Header Length: 20 bytes (5)
Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
Total Length: 752
Identification: 0x424b (16971)
010 ....= Flags: 0x02, Don't fragment
0000000000000000 = Fragment Offset: 0
Time to live: 128
Protocol: TCP (6)
Header Checksum: 0x0000 [validation disabled]
[Header checksum status: Unverified]
Source Address: 127.0.0.1
Destination Address: 127.0.0.1
[Stream index: 0]

Transmission Control Protocol, Src Port: 65190, Dst Port: 8880, Seq.: 1, Ack.: 1, Len: 712
Hypertext Transfer Protocol
GET / HTTP/1.1\r\n
Host: 127.0.0.1:8080\r\n
Connection: keep-alive\r\n
sec-ch-ua: "Chromium";v="130", "Microsoft Edge";v="130", "NotA_Brand";v="99"\r\n
sec-chua-mobile: ?0\r\n
sec-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 Accept: */*
Content-Type: application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/png,*/*;q=0.8
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
\r\n

```

然后服务器端回复 ACK 表示收到请求

```

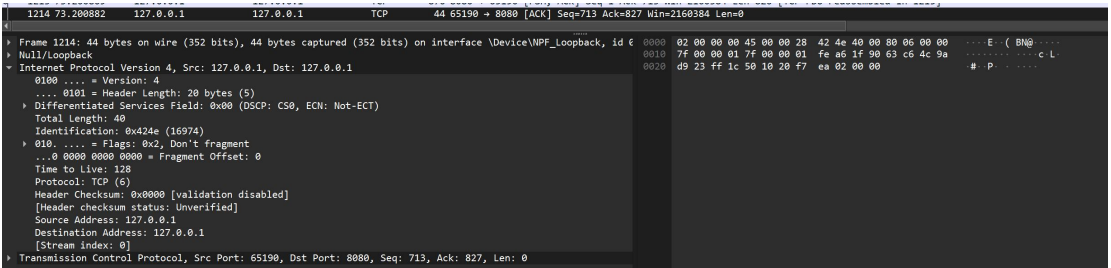
1212 73.198931 127.0.0.1 127.0.0.1 TCP 44 8080 → 65190 [ACK] Seq=1 Ack=713 Win=2160384 Len=0
  Frame 1212: 44 bytes on wire (352 bits), 44 bytes captured (352 bits) on interface \Device\NPF_{...} id 0000 02 00 00 00 45 00 00 28 42 4c 00 00 80 06 00 00 ...E ( BL_
  Null/Loopback 0010 7f 00 00 01 7f 00 00 01 1f 90 fe a6 d9 23 fb e2 ... ..
  Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1 0020 63 c6 c4 9a 50 10 20 f7 ed 3c 00 00 ...C L P ...
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
    Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 40
    Identification: 0x424c (16972)
    010. .... = Flags: 0x2, Don't Fragment
    ...0 0000 0000 0000 = Fragment Offset: 0
    Time to Live: 128
    Protocol: TCP (6)
    Header Checksum: 0x0000 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 127.0.0.1
    Destination Address: 127.0.0.1
    [Stream index: 0]
  Transmission Control Protocol, Src Port: 8080, Dst Port: 65190, Seq: 1, Ack: 713, Len: 0

```

之后服务器端向客户端发送响应报文

1213	73.200869	127.0.0.1	127.0.0.1	TCP	870	8080 → 65190	[PSH, ACK] Seq=1 Ack=713 Win=2160384 Len=826	[TCP PDU reassembled in 1215]	
<pre> > Frame 1213: 870 bytes on wire (6960 bits), 870 bytes captured (6960 bits) on interface \Device\NPF_{Loopback, Null\Loopback} Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1 0100 = Version: 4 0101 = Header Length: 20 bytes (5) > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT) Total Length: 866 Identification: 0x424d (16973) > 010... = Flags: 0x2, Don't fragment ...0 0000 0000 0000 = Fragment Offset: 0 Time to Live: 128 Protocol: TCP (6) Header Checksum: 0x0000 [validation disabled] [Header checksum status: Unverified] Source Address: 127.0.0.1 Destination Address: 127.0.0.1 [Stream index: 0] > Transmission Control Protocol, Src Port: 8080, Dst Port: 65190, Seq: 1, Ack: 713, Len: 826 </pre>									

最后客户端回复 ACK 表示收到请求



3. 下面对四次挥手进行分析：

第一次挥手，Client 发送一个 FIN，用来关闭 Client 到 Server 的数据传送，Client 进入 FIN_WAIT_1 状态。第二次挥手时，Server 收到 FIN 后，发送一个 ACK 给 Client，确认序号为收到序号+1（与 SYN 相同，一个 FIN 占用一个序号），Server 进入 CLOSE_WAIT 状态。第三次挥手时，Server 发送一个 FIN，用来关闭 Server 到 Client 的数据传送，Server 进入 LAST_ACK 状态。第四次挥手时，Client 收到 FIN 后，Client 进入 TIME_WAIT 状态，接着发送一个 ACK 给 Server，确认序号为收到序号+1，Server 进入 CLOSED 状态，完成四次挥手。

386	64.741138	127.0.0.1	127.0.0.1	TCP	44 8080 → 62942 [FIN, ACK] Seq=1 Ack=1 Win=2161152 Len=0
387	64.741162	127.0.0.1	127.0.0.1	TCP	44 62942 → 8080 [ACK] Seq=1 Ack=2 Win=2161152 Len=0
400	67.472836	127.0.0.1	127.0.0.1	TCP	44 62942 → 8080 [FIN, ACK] Seq=1 Ack=2 Win=2161152 Len=0
401	67.472888	127.0.0.1	127.0.0.1	TCP	44 8080 → 62942 [ACK] Seq=2 Ack=2 Win=2161152 Len=0

四. 总结

在本次实验中，我们搭建了一个简单的 Web 服务器，并制作了包含个人信息和音频自我介绍的 Web 页面。通过使用 Wireshark，我们捕获了浏览器与 Web 服务器之间的 HTTP 交互过程，并对其中的三次握手和四次挥手过程进行了详细分析。

通过实验，我们深入了解了 TCP 连接的建立和关闭机制，掌握了 HTTP 协议的基本请求和响应结构，学会了如何使用 Wireshark 过滤和分析网络数据包。这些实践不仅增强了我们对网络通信原理的理解，还提高了我们在实际应用中分析和解决问题的能力。