WEB服务器的搭建与页面的编写

使用python，在对应项目文件夹下打开命令行，在终端中输入命令：

```
python -m http.server 8081
```

表示在8081端口上启动一个http服务器，此时若创建成功，终端中会显示

```
Serving HTTP on :: port 8081 (http://[::]:8081/) ...
```

此时在浏览器中输入`localhost:8081`即可访问该项目文件夹下的内容。

接下来需要在项目文件夹下编写html文件，命名为test，然后在浏览器中输入`localhost:8081/test.html`即可访问。

```
<html>
<head>
    <meta charset="utf-8">
    <title>Test</title>
</head>
<body>
    <h1>我是牢大</h1>
    <p>我将在复活节归来</p>
    <p>牢大 logo:</p>
    <img src="kobe.jpg" alt="kobe" width="300" height="300">
    <audio controls>
        <source src="kobe.mp3" type="audio/mpeg">
    </audio>

    <p>作者：  计算机科学与技术  2114036  曹骜天</p>
</body>
</html>
```

此html的文件结构如下：由和标签包裹，和标签中包含了该html文件的元信息，和标签中包含了该html文件的主要内容，其中包含了一个标题，两个段落，一张图片，一段音频，以及一段文字。

在网页中打开该html文件，显示如下：

我是牢大

我将在复活节归来

牢大 logo:

0:00 / 1:29

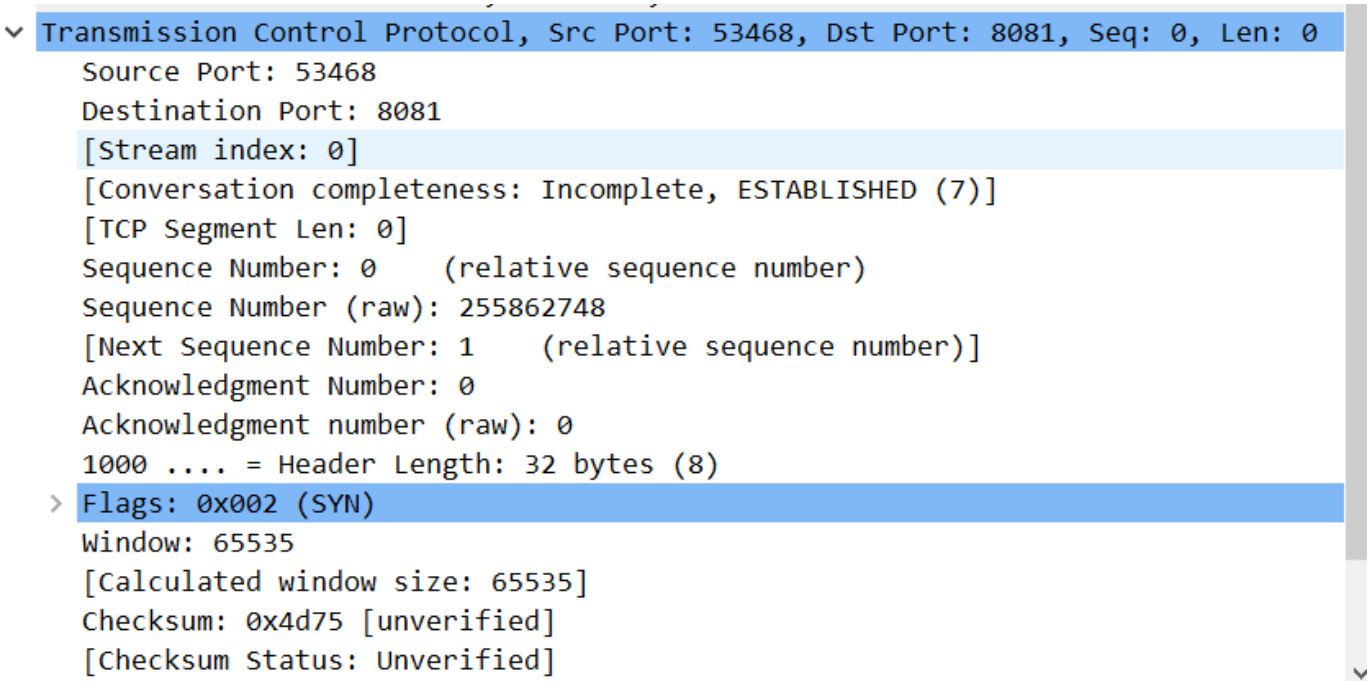
采用wireshark抓包分析http协议

三次握手的过程

我们假定服务器端的地址为8081（如上所述），可以看到在有http请求之前发生了三次握手：

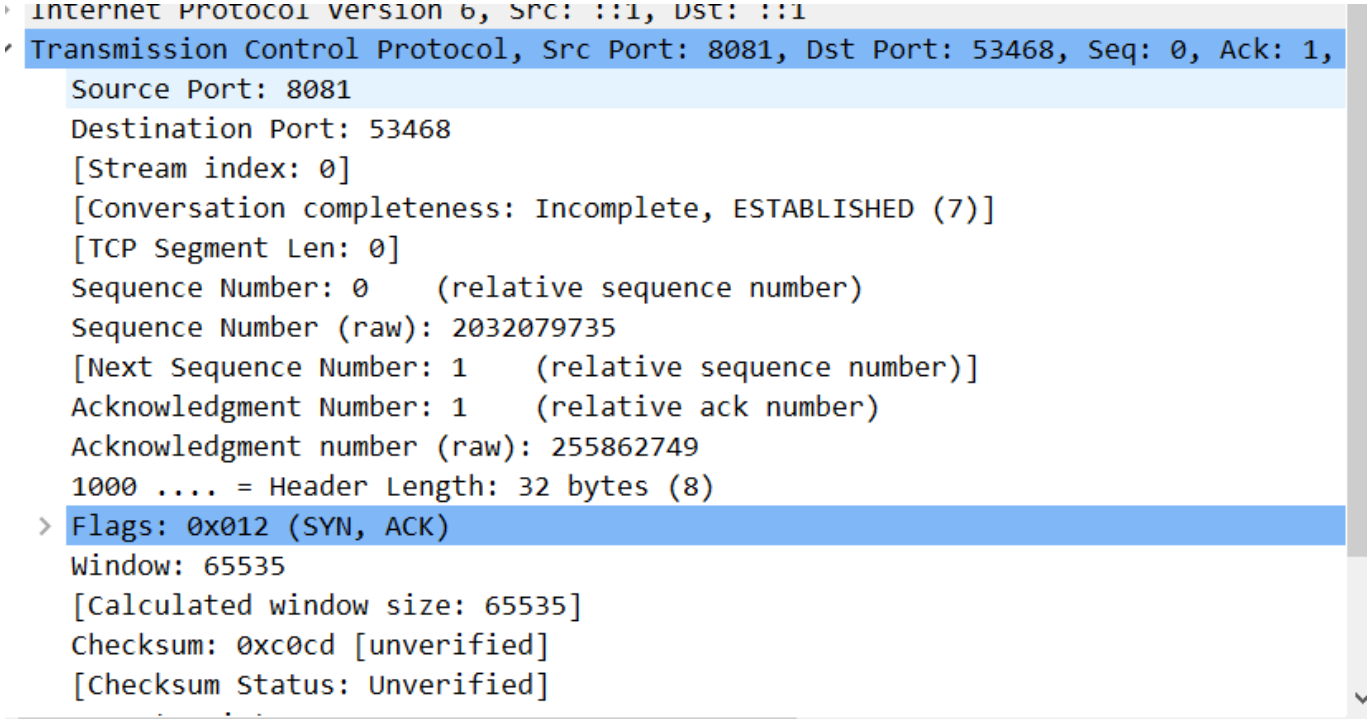
1	0.000000	::1	::1	TCP	76 53468 → 8081 [SYN]
2	0.000077	::1	::1	TCP	76 8081 → 53468 [SYN,
3	0.000145	::1	::1	TCP	64 53468 → 8081 [ACK]
4	0.041360	::1	::1	HTTP	664 GET /kobe.mp3 HTTP

在第一次握手的过程中，捕获到的信息如下：



本次握手是由客户端向服务器端发送建立连接的请求。其中，Sequence Number = 0说明用户选择的初始化序列号(client_isn)为0，Window Size = 65535说明用户选择的窗口大小为65535，TCP Flags中的SYN标志位为1，其余标志位均为0。我们将此报文段称为TCP SYN报文段。

第二次握手捕获的信息如下：



服务器收到客户端发过来报文，由SYN=1知道客户端要求建立连接。在此时将SYN和ACK的值置为1，表示同意建立连接。同时将acknowledgement number设置为client_isn+1，表示服务器端期望收到的下一个报文段的序列号。同时将sequence number置为0，表示(server_isn)。我们将此报文段称为TCP SYN+ACK报文段。

第三次握手的捕获信息如下：

```
> Internet Protocol Version 6, Src: ::1, Dst: ::1
v Transmission Control Protocol, Src Port: 53468, Dst Port: 8081, Seq: 1, Ack: 1,
    Source Port: 53468
    Destination Port: 8081
    [Stream index: 0]
    [Conversation completeness: Incomplete, ESTABLISHED (7)]
    [TCP Segment Len: 0]
    Sequence Number: 1    (relative sequence number)
    Sequence Number (raw): 255862749
    [Next Sequence Number: 1    (relative sequence number)]
    Acknowledgment Number: 1    (relative ack number)
    Acknowledgment number (raw): 2032079736
    0101 .... = Header Length: 20 bytes (5)
  > Flags: 0x010 (ACK)
    Window: 10230
    [Calculated window size: 2618880]
    [Window size scaling factor: 256]
    Checksum: 0xd3ba [unverified]
```

客户端收到 SYNACK的报文段信息后，将SYN置为0，ACK置为1，表示收到了服务器端的确认信息。同时将acknowledgement number设置为server_isn+1，表示客户端期望收到的下一个报文段的序列号。同时将sequence number置为client_isn+1，可进行发送数据。 接下来，可对服务器端发送http请求。

http请求的过程

对于http的报文格式，我们抓取了请求报文和响应报文两种，分别如下： 请求报文：

```
v Hypertext Transfer Protocol
  > GET /kobe.mp3 HTTP/1.1\r\n
    Host: localhost:8081\r\n
    Connection: keep-alive\r\n
    sec-ch-ua: "Chromium";v="118", "Microsoft Edge";v="118", "Not=A?Brand";v="99"
    Accept-Encoding: identity;q=1, *;q=0\r\n
    sec-ch-ua-mobile: ?0\r\n
    User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHT
    sec-ch-ua-platform: "Windows"\r\n
    Accept: */*\r\n
    Sec-Fetch-Site: same-origin\r\n
    Sec-Fetch-Mode: no-cors\r\n
    Sec-Fetch-Dest: audio\r\n
    Referer: http://localhost:8081/test.html\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
    Range: bytes=0-\r\n
    \r\n
    [Full request URI: http://localhost:8081/kobe.mp3]
    [HTTP request 1/1]
    [Response in frame: 77]
```

可以看到，get为请求的方式，后接请求的资源以及http版本号，后面为请求头，包含了请求的主机，连接方式，用户代理，以及其他信息。可以看到：host主机名为8081，连接方式为keep-alive，用户代理为Mozilla/5.0，表示使用的是Mozilla浏览器。

响应报文：

```
Hypertext Transfer Protocol
  HTTP/1.0 200 OK\r\n
  Server: SimpleHTTP/0.6 Python/3.10.10\r\n
  Date: Wed, 01 Nov 2023 02:40:06 GMT\r\n
  Content-type: audio/mpeg\r\n
  Content-Length: 3678680\r\n
  Last-Modified: Tue, 31 Oct 2023 08:57:44 GMT\r\n
  \r\n
  [HTTP response 1/1]
  [Time since request: 0.007677000 seconds]
  [Request in frame: 4]
  [Request URI: http://localhost:8081/kobe.mp3]
  File Data: 3678680 bytes
```

响应报文的第一行为状态行，200表示请求成功，后面为响应头，包含了服务器的信息，以及其他信息。可以看到服务器为SimpleHTTP/0.6 Python/3.10.10，表示使用的是python的http服务器。http版本为http1.0版本，注意，利用python http.server创建的服务器默认为http1.1版本。**但在实验中发现响应为http1.0版本，因此还是遵循http1.0的特性，即每一次请求完毕后立刻断开连接**

四次挥手的过程

在完成了http请求之后，客户端和服务器端需要断开连接，此时发生了四次挥手（中间的TCP window update为窗口的更新变化，可以忽略）：

```
::1    ::1    HTTP      9216 HTTP/1.0 200 OK  (audio/mpeg)
::1    ::1    TCP         64 53458 → 8081 [ACK] Seq=601 Ack=3678873 Win=4564 Len=0
::1    ::1    TCP         64 8081 → 53458 [FIN, ACK] Seq=3678873 Ack=601 Win=10230 Len=0
::1    ::1    TCP         64 53458 → 8081 [ACK] Seq=601 Ack=3678874 Win=4564 Len=0
::1    ::1    TCP         64 [TCP Window Update] 53458 → 8081 [ACK] Seq=601 Ack=3678874 Win=4819 Len=0
::1    ::1    TCP         64 [TCP Window Update] 53458 → 8081 [ACK] Seq=601 Ack=3678874 Win=10230 Len=0
::1    ::1    TCP         64 53458 → 8081 [FIN, ACK] Seq=601 Ack=3678874 Win=10230 Len=0
::1    ::1    TCP         64 8081 → 53458 [ACK] Seq=3678874 Ack=602 Win=10230 Len=0
```

```
Transmission Control Protocol, Src Port: 8081, Dst Port: 53458, Seq: 3678873, Ack: 601, Len: 0
  Source Port: 8081
  Destination Port: 53458
  [Stream index: 1]
  [Conversation completeness: Incomplete (28)]
  [TCP Segment Len: 0]
  Sequence Number: 3678873    (relative sequence number)
  Sequence Number (raw): 2846757395
  [Next Sequence Number: 3678874    (relative sequence number)]
  Acknowledgment Number: 601    (relative ack number)
  Acknowledgment number (raw): 2318484468
  0101 .... = Header Length: 20 bytes (5)
  Flags: 0x011 (FIN, ACK)
  Window: 10230
  [Calculated window size: 10230]
  [Window size scaling factor: -1 (unknown)]
  Checksum: 0x0d91 [unverified]
  [Checksum Status: Unverified]
```

第一次挥手时，由服务器端发送断开连接的请求，其中FIN=1表示想要断开连接

```
> Frame 80: 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
v Transmission Control Protocol, Src Port: 53458, Dst Port: 8081, Seq: 601, Ack: 3678874, Len: 0
    Source Port: 53458
    Destination Port: 8081
    [Stream index: 1]
    [Conversation completeness: Incomplete (28)]
    [TCP Segment Len: 0]
    Sequence Number: 601    (relative sequence number)
    Sequence Number (raw): 2318484468
    [Next Sequence Number: 601    (relative sequence number)]
    Acknowledgment Number: 3678874    (relative ack number)
    Acknowledgment number (raw): 2846757396
    0101 .... = Header Length: 20 bytes (5)
  > Flags: 0x010 (ACK)
    Window: 4564
    [Calculated window size: 4564]
    [Window size scaling factor: -1 (unknown)]
    Checksum: 0x23b3 [unverified]
    [Checksum Status: Unverified]
```

第二次挥手表示客户端收到服务器端的确认，由于不传输数据，Sequence Number为上一次客户端的值，Ack则是服务端的Seq+1，表示收到了服务器的数据。标志位ACK为1。

```
v Transmission Control Protocol, Src Port: 53458, Dst Port: 8081, Seq: 601, Ack: 3678874, Len: 0
    Source Port: 53458
    Destination Port: 8081
    [Stream index: 1]
    [Conversation completeness: Incomplete (28)]
    [TCP Segment Len: 0]
    Sequence Number: 601    (relative sequence number)
    Sequence Number (raw): 2318484468
    [Next Sequence Number: 602    (relative sequence number)]
    Acknowledgment Number: 3678874    (relative ack number)
    Acknowledgment number (raw): 2846757396
    0101 .... = Header Length: 20 bytes (5)
  > Flags: 0x011 (FIN, ACK)
    Window: 10230
    [Calculated window size: 10230]
    [Window size scaling factor: -1 (unknown)]
    Checksum: 0x0d90 [unverified]
    [Checksum Status: Unverified]
```

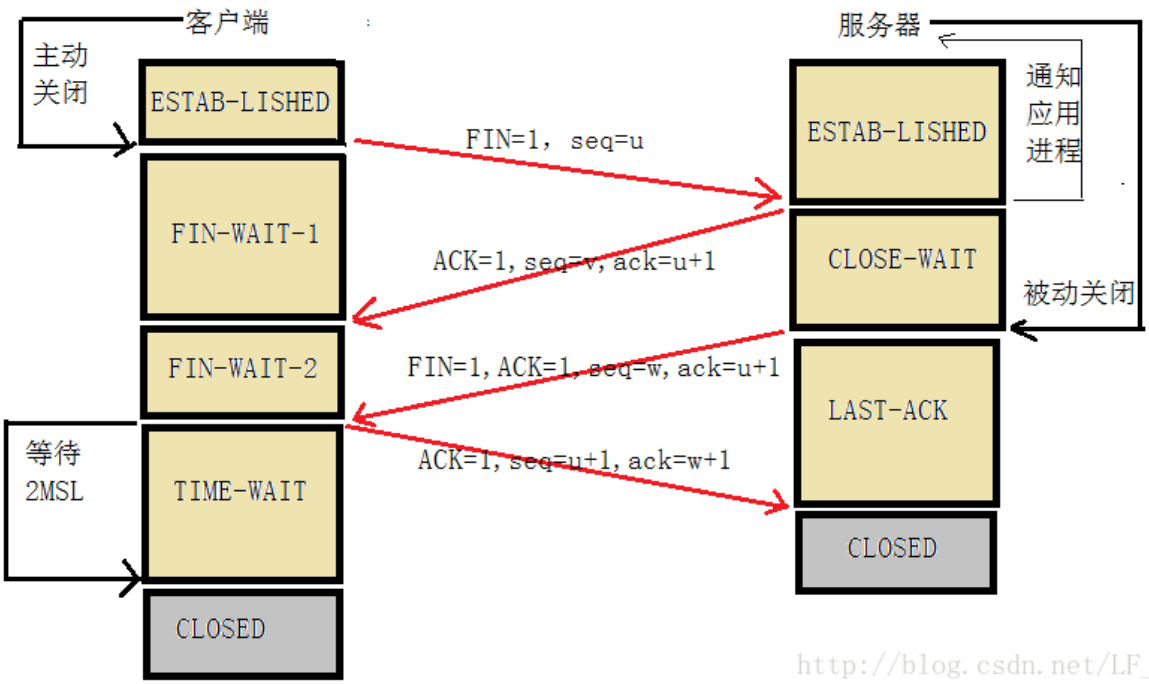
接下来， 随后客户端又发送标志位FIN,ACK，Seq=上一次的客户端Seq，不加1因为没有传输有效的数据，Ack=上一次客户端的确认码，不加1，原因是也没有有效数据，代表客户端也可以关闭此连接了，不传输数据了。

```
    Source Port: 8081
    Destination Port: 53458
    [Stream index: 1]
    [Conversation completeness: Incomplete (28)]
    [TCP Segment Len: 0]
    Sequence Number: 3678874    (relative sequence number)
    Sequence Number (raw): 2846757396
    [Next Sequence Number: 3678874    (relative sequence number)]
    Acknowledgment Number: 602    (relative ack number)
    Acknowledgment number (raw): 2318484469
    0101 .... = Header Length: 20 bytes (5)
  > Flags: 0x010 (ACK)
    Window: 10230
    [Calculated window size: 10230]
    [Window size scaling factor: -1 (unknown)]
    Checksum: 0x0d90 [unverified]
    [Checksum Status: Unverified]
```

最后，服务器端收到客户端的FIN,ACK。Ack为发送端的Seq+1,表示收到了通知消息，Seq不变，表示服务器端也可以关闭此连接了，不传输数据了。

一些问题解答

1. 为什么在最后发送完ACK后需要等待一会儿才关闭?



这最主要是因为两个理由：

1. 为了保证客户端发送的最后一个ACK报文段能够到达服务器。因为这个ACK有可能丢失，从而导致处在LAST-ACK状态的服务器收不到对FIN-ACK的确认报文。服务器会超时重传这个FIN-ACK，接着客户端再重传一次确认，重新启动时间等待计时器。最后客户端和服务器都能正常的关闭。假设客户端不等待2MSL，而是在发送完ACK之后直接释放关闭，一但这个ACK丢失的话，服务器就无法正常的进入关闭连接状态。

2. 他还可以防止已失效的报文段。客户端在发送最后一个ACK之后，再经过经过2MSL，就可以使本链接持续时间内所产生的所有报文段都从网络中消失。从保证在关闭连接后不会有还在网络中滞留的报文段去骚扰服务器。

注意：在服务器发送了FIN-ACK之后，会立即启动超时重传计时器。客户端在发送最后一个ACK之后会立即启动时间等待计时器。

2. 在采用此方式时，会出现http请求和相应版本不一致的情况，如前所示，请求版本为http1.1，响应版本却为http1.0。发现在windows10系统下才存在此bug,将操作系统升级到windows11则解决问题。