

# 计算机网络实验报告

## 实验二：配置Web服务器，编写简单页面，分析交互过程

姓名：谢雯菲 学号：2110803

### 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。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。

### 实验工具

- 搭建服务器：XAMPP
- 捕获交互：Wireshark

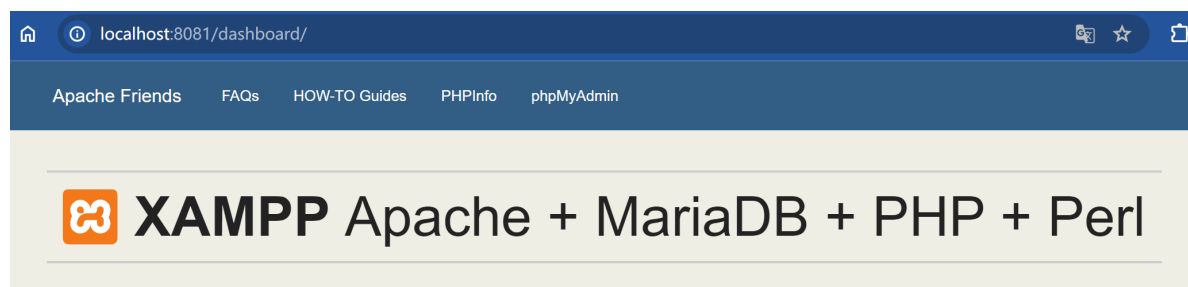
### 实验过程

#### 搭建Web服务器

在windows 11系统中利用XAMPP的Apache搭建服务器。




安装XAMPP并修改端口防止冲突。（因使用电脑已经安装过XAMPP且将Apache的端口设置为了8081，在此就不过多赘述）

访问网站 `http://localhost:8081`，跳出以下页面，说明本机服务器搭建成功。



#### 创建HTML文件

XAMPP服务器使用的Apache文件路径为：`C:\xampp\htdocs\dashboard`，在其中创建一个文件夹 `Fi` 存放编写网站需要的文件：

« xampp > htdocs > dashboard > Fi		在 F
名称		修改日期
 music.mp3		2023/3/12 1
 picture.jpg		2023/10/28
 test.html		2023/10/31

编写简单的HTML代码，实现展示姓名、学号、专业的功能，并包含一个图片文件和一个音乐文件。音乐文件可以通过 播放 按钮实现播放和暂停。

HTML文件代码如下：

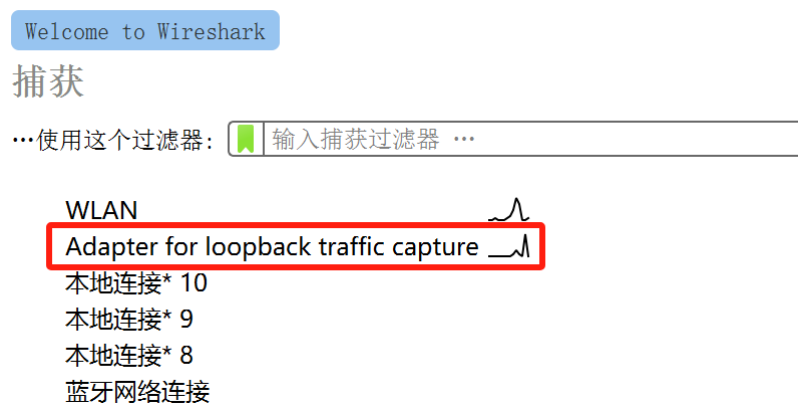
```
1  <!doctype html>
2  <head>
3      <meta charset="UTF-8">
4  </head>
5  <html>
6      <title>这是一个网页</title>
7      <body>
8          <p style="text-align:center;">姓名：谢雯菲</p>
9          <p style="text-align:center;">学号：2110803</p>
10         <p style="text-align:center;">专业：信息安全</p>
11     </body>
12     <p style="text-align:center;">
13         
14     </p>
15     <audio id="myAudio" src="music.mp3" type="audio/mpeg"></audio>
16     <p style="text-align:center;">
17         <button id="playButton" onclick="togglePlay()">播放</button>
18     </p>
19     <script>
20         var audio = document.getElementById("myAudio");
21         var playButton = document.getElementById("playButton");
22
23         function togglePlay() {
24             if (audio.paused) {
25                 audio.play();
26                 playButton.innerHTML = "暂停";
27             } else {
28                 audio.pause();
29                 playButton.innerHTML = "播放";
30             }
31         }
32     </script>
33 </html>
```

通过浏览器访问该文件，显示界面如下：



## 使用wireshark捕获数据包

打开wireshark, 由于客户端和服务端都是本机, 选择 `Adapter for loopback traffic capture` (用于捕获回环接口的数据) 进行捕获:



使用浏览器打开之前构建的服务器中的网页, 过滤只查看采用 `http` 交互的数据包, 发现捕获到了一连串显示源ip地址和目的ip地址为 `:::1` 的数据包:

1400	23.584054	:::1	:::1	HTTP	744	GET /dashboard/Fi/test.html HTTP/1.1
1402	23.584868	:::1	:::1	HTTP	1479	HTTP/1.1 200 OK (text/html)
1407	23.600152	:::1	:::1	HTTP	683	GET /dashboard/Fi/picture.jpg HTTP/1.1
1410	23.601369	:::1	:::1	HTTP	18129	HTTP/1.1 200 OK (JPEG JFIF image)
1412	23.966975	:::1	:::1	HTTP	639	GET /dashboard/Fi/music.mp3 HTTP/1.1
1466	23.974421	:::1	:::1	HTTP	27709	HTTP/1.1 206 Partial Content (audio/mpeg)
1774	29.118899	:::1	:::1	HTTP	670	GET /favicon.ico HTTP/1.1
1776	29.119823	:::1	:::1	HTTP	31273	HTTP/1.1 200 OK (image/x-icon)
1778	29.143185	:::1	:::1	HTTP	686	GET /dashboard/Fi/music.mp3 HTTP/1.1
1799	29.144527	:::1	:::1	HTTP	22219	HTTP/1.1 206 Partial Content (audio/mpeg)

查询得知, `:::1` 表示 IPv6 的回环地址, 接受来自本地计算机的连接。由于在访问服务器端时选用的是 `localhost` 进行的连接, 所以ip地址会在捕获数据包时显示为回环。

如果访问时采用ip地址进行访问, 那么就会显示正常的ip地址。但由于使用的电脑采用的是自动分配ip地址进行上网, 每次进行实验分析的ip地址都可能不同, 因此还是选用 `localhost` 进行访问。

## HTTP数据包分析

从上面的图片可以看到, 服务器端使用的是 `HTTP1.1` 协议。

## HTTP1.0与HTTP1.1的区别

- HTTP 1.0 规定浏览器与服务器只保持短暂的连接，浏览器的每次请求都需要与服务器建立一个TCP连接，服务器完成请求处理后立即断开TCP连接，服务器不跟踪每个客户也不记录过去的请求。
- HTTP 1.1 则支持持久连接 `Persistent Connection`，并且默认使用 `Persistent Connection`。在同一个TCP的连接中可以传送多个HTTP请求和响应。多个请求和响应可以重叠，多个请求和响应可以同时进行。

使用HTTP协议进行数据传输时，客户端会先向服务器端发送一个 `GET` 的HTTP请求，然后服务器端会做出响应并传输数据。

## 请求报文

查看上述捕获到的报文中的请求报文：

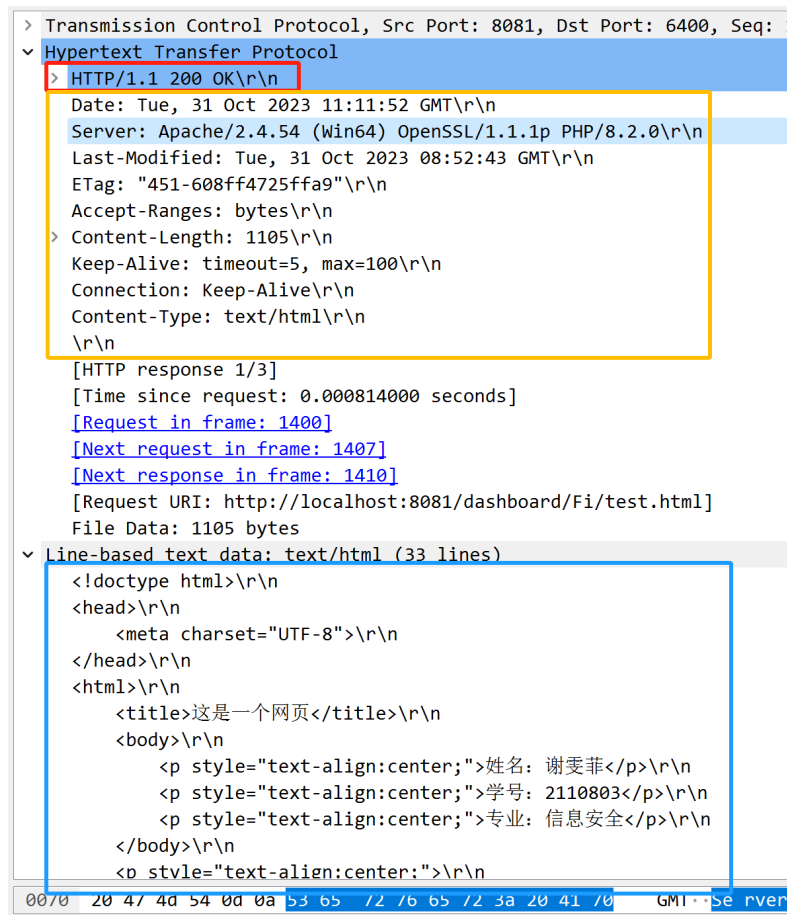
```
> Transmission Control Protocol, Src Port: 6400, Dst Port: 8081, Seq: 1, Ack: 1, Len: 680
  v Hypertext Transfer Protocol
    > GET /dashboard/Fi/test.html HTTP/1.1\r\n
      Host: localhost:8081\r\n
      Connection: keep-alive\r\n
      sec-ch-ua: "Chromium";v="118", "Google Chrome";v="118", "Not=A?Brand";v="99"\r\n
      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/118.0.0.0 Sa...
      Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,applica...
      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\r\n
      Accept-Language: zh-CN,zh;q=0.9\r\n
      \r\n
      [Full request URI: http://localhost:8081/dashboard/Fi/test.html]
      [HTTP request 1/3]
      [Response in frame: 1402]
      [Next request in frame: 1407]
```

- 红框部分：请求行。GET 表示请求方法，`dashboard/Fi/test.html` 是请求的URL，`HTTP/1.1` 是使用的HTTP版本，`\r\n` 是回车符和换行符。
- 黄框部分：请求头。每行包含 键-值 对，主要用于说明请求来意、连接类型、以及一些Cookie信息等，每一行都以 `\r\n` 结尾。最后有一个独立的 `\r\n` 回车换行指明请求头。

其中有一个表示TCP连接类型的行：`Connection: keep-alive\r\n`，表示开启keep-alive。如果开启keep-alive，则服务端在返回响应后不关闭TCP连接；同样的，在接收完响应报文后，客户端也不关闭连接，发送下一个HTTP请求时会重用该连接。在HTTP 1.0中，如果请求头中包含 `Connection: keep-alive\r\n`，则代表开启keep-alive，而服务端的返回报文头中，也会包含相同的内容。在HTTP 1.1协议中默认开启keep-alive，除非显式地关闭它。

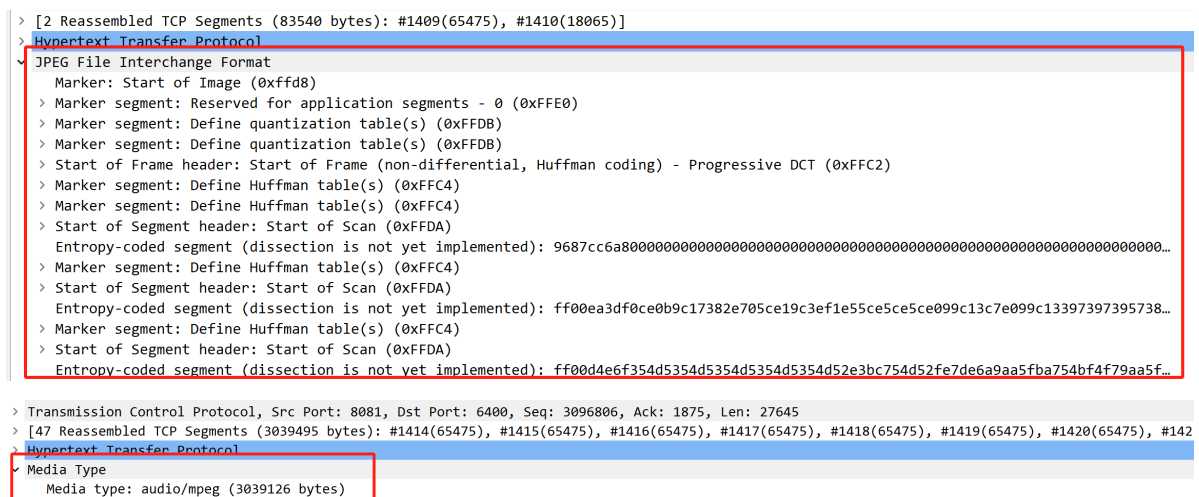
## 响应报文

查看响应报文：



- 红框部分：响应行。包含了状态码和解释。此处的 200 就是状态码，表示请求成功，OK 是可读形式的状态。
- 黄框部分：响应头。主要是返回一些服务器的基本信息，以及一些Cookie值等。
- 蓝框部分：响应体。可以是html文档内容或图片等多种格式的数据。在这里我们可以看到是一个html文档。

查看另外几个捕获到的响应数据包，发现有图片和音乐类型的数据。（图示如下）



## TCP传输控制协议

TCP协议是一种面向连接的、可靠的、基于字节流的传输层通信协议。TCP是全双工模式，为了建立可靠的连接需要三次握手，四次挥手使得TCP的全双工连接能够可靠的终止。

## TCP三次握手

TCP三次握手过程示意图如下所示：

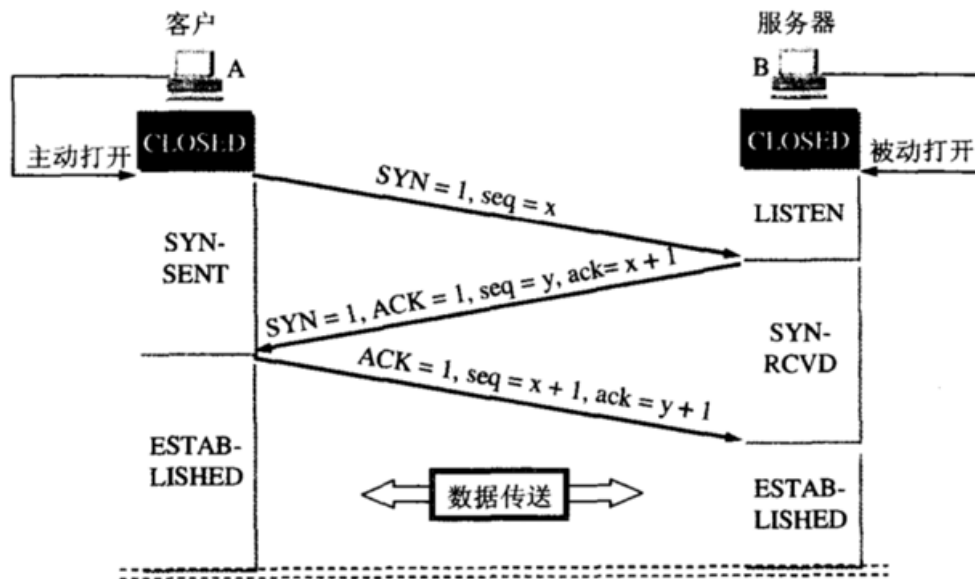


图 5-31 用三次握手建立 TCP 连接 [csdn.net/qg\\_28000789](https://www.csdn.net/qg_28000789)

最初两端的TCP进程都处于CLOSED关闭状态，在服务器端变成LISTEN状态后，客户端可以向服务器端发送请求。

- 第一次握手：客户端发送SYN包（首部的同步位SYN=1，初始序号seq=x）到服务器，转变为SYN-SENT状态，代表客户端请求建立连接。
- 第二次握手：服务器收到SYN包，转变为SYN-RCVD状态，同时自己也发送一个SYN包，即SYN+ACK包（SYN=1，ACK=1，确认序号=x+1，发送序号=y），表示服务器可以正常接收客户端数据包。
- 第三次握手：客户端收到服务器的SYN+ACK包，转变为ESTAB-LISHED状态，然后向服务器发送确认包ACK（ACK=1，确认序号=y+1，发送序号=x+1），客户端表示可以正常接收服务器数据包，这是为了保证可以全双工通讯，接下来就可以正常发送数据。此包发送完毕，服务器端收到后转换为ESTAB-LISHED状态，完成三次握手。

其中，确认号的数值等于发送方的发送序号+1，也就是接受方期望接收到的下一个序列号。

进入上述wireshark捕获到的数据包追踪的HTTP流，查看三次握手过程：

Source	Destination	Protocol	Length	Info
:::1	:::1	TCP	76	11003 → 8081 [SYN] Seq=0 Win=65535 Len=0 MSS=65475 WS=256 SACK_PERM
:::1	:::1	TCP	76	8081 → 11003 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65475 WS=256 ...
:::1	:::1	TCP	64	11003 → 8081 [ACK] Seq=1 Ack=1 Win=327168 Len=0
:::1	:::1	HTTP	737	GET /dashboard/Fi/music.mp3 HTTP/1.1

查看第一次握手的标志位信息：

```
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
 > .... ..1. = Syn: Set
 .... .... 0 = Fin: Not set
 [TCP Flags: .....S.]
```

由图可知，客户端发送的包中将SYN位置为1。

第一次握手的序号如下所示：

```
Sequence Number: 0      (relative sequence number)
Sequence Number (raw): 1780587096
[Next Sequence Number: 1      (relative sequence number)]
Acknowledgment Number: 0
Acknowledgment number (raw): 0
```

由图可知，发送的SYN包的初始序号为1780587096。

查看服务器发送的第二次握手的标志位信息：

```
✓ 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.]
```

由图可知，服务器发送的包中SYN和ACK位都置为1。

第二次握手的序号如下：

```
Sequence Number: 0      (relative sequence number)
Sequence Number (raw): 676573669
[Next Sequence Number: 1      (relative sequence number)]
Acknowledgment Number: 1      (relative ack number)
Acknowledgment number (raw): 1780587097
```

由图可知，其确认序号为第一次握手客户端发送序号+1（1780587096+1=1780587097），发送序号为676573669。

查看客户端发送的第三次握手的标志位信息：

```
✓ 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....]
```

由图可知，客户端发送的包中ACK位置为1。

第三次握手序号如下所示：

```
Sequence Number: 1      (relative sequence number)
Sequence Number (raw): 1780587097
[Next Sequence Number: 1      (relative sequence number)]
Acknowledgment Number: 1      (relative ack number)
Acknowledgment number (raw): 676573670
```

由图可知，发送序号为上一次客户端发送数据包序号+1（1780587096+1=1780587097），确认序号为接收到的服务器端的序号+1（676573669+1=676573670）。

综上，三次握手过程验证成功。



## TCP四次挥手

TCP四次挥手过程示意图如下所示：

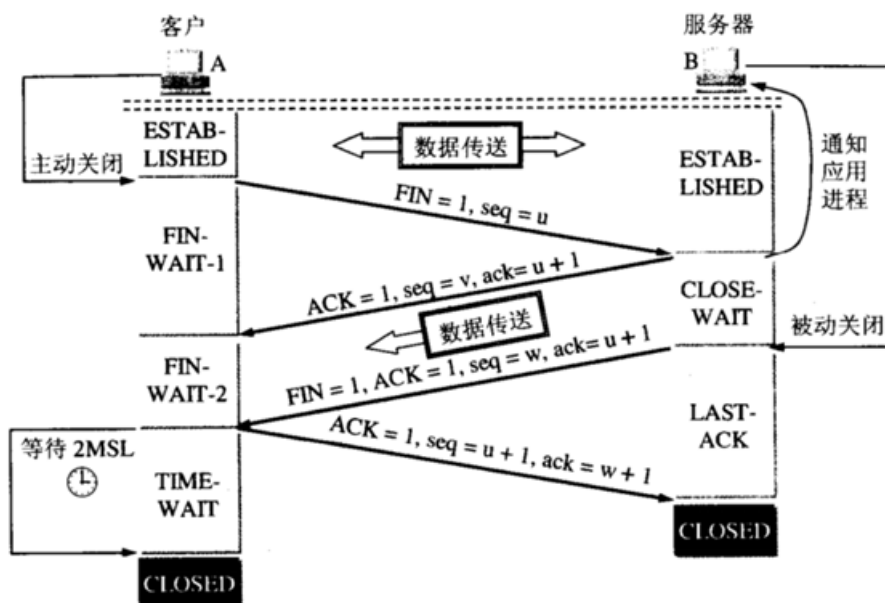


图 5-32 TCP 连接释放的过程

由于客户端与服务器端传输时是全双工的，因此断开请求既可以由客户端发起，也可以由服务器端发起。

数据传输结束后，A和B都处于ESTABLISHED状态。以客户A端主动释放连接为例进行分析：

- 第一次挥手：A发送一个FIN (FIN=1, 发送序号=u)，表示自己没有数据要发送了，想断开连接，来关闭A到B的数据传送，进入FIN-WAIT-1（停止等待1）状态。
- 第二次挥手：B收到FIN，向A发回一个ACK (ACK=1, 确认序号=u+1, 发送序号=v)，进入CLOSE-WAIT（关闭等待）状态，此时B已经停止接收A发来的消息了，但是B仍然可以向A发送消息。A收到后进入FIN-WAIT-2（停止等待2）状态。
- 第三次挥手：B关闭与A的连接，发送一个释放报文段 (FIN=1, ACK=1, 发送序号=w, 确认序号=u+1) 给A，表示自己没有数据要发送了，进入LAST-ACK（最后确认）状态。
- 第四次挥手：A收到B的FIN，发回ACK报文 (ACK=1, 发送序号=u+1, 确认序号=w+1) 确认，进入TIME-WAIT（时间等待）状态，等待时间2MSL后关闭连接。B在收到A发送的ACK后立即关闭连接。

查看wireshark中四次挥手过程：

::1	::1	TCP	64 8081 → 11003 [FIN, ACK] Seq=2581546 Ack=674 Win=2159872 Len=0
::1	::1	TCP	64 11003 → 8081 [ACK] Seq=674 Ack=2581547 Win=327168 Len=0
::1	::1	TCP	64 11003 → 8081 [FIN, ACK] Seq=674 Ack=2581547 Win=2160640 Len=0
::1	::1	TCP	64 8081 → 11003 [ACK] Seq=2581547 Ack=675 Win=2159872 Len=0

在本次实验中，可以看到四次挥手的发起者是服务器端。

第一次挥手：

```
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
```



服务器端发送的FIN位置为1。

发送序号如下：

```
Sequence Number: 2581546 (relative sequence number)
Sequence Number (raw): 679155215
[Next Sequence Number: 2581547 (relative sequence number)]
Acknowledgment Number: 674 (relative ack number)
Acknowledgment number (raw): 1780587770
```

发送序号为679155215。

第二次挥手：

```

  ▾ 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....]
```

客户端发送的ACK位置为1。

发送序号如下：

```
Sequence Number: 674 (relative sequence number)
Sequence Number (raw): 1780587770
[Next Sequence Number: 674 (relative sequence number)]
Acknowledgment Number: 2581547 (relative ack number)
Acknowledgment number (raw): 679155216
0101 .... = Header Length: 20 bytes (5)
```

由图可知，确认序号为接收到的服务器端的发送序号+1（679155215+1=679155216），发送序号为1780587770。

第三次挥手：

```

  ▾ 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]
```

客户端发送FIN位置为1的数据包。

发送序号如下：

```
Sequence Number: 674 (relative sequence number)
Sequence Number (raw): 1780587770
[Next Sequence Number: 675 (relative sequence number)]
Acknowledgment Number: 2581547 (relative ack number)
Acknowledgment number (raw): 679155216
0101 .... = Header Length: 20 bytes (5)
```

发送序号为1780587770 (w)。由于没有新的需要确认的数据包，因此确认序号（679155216）不变。

第四次挥手：

```

v 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.....]

```

服务器端发送ACK位置为1的数据。

发送序号如下：

```

Sequence Number: 2581547 (relative sequence number)
Sequence Number (raw): 679155216
[Next Sequence Number: 2581547 (relative sequence number)]
Acknowledgment Number: 675 (relative ack number)
Acknowledgment number (raw): 1780587771
0101 .... = Header Length: 20 bytes (5)

```

因为确认了客户端的FIN，所以确认序号为收到的客户端发送序号+1（ $1780587770+1=1780587771$ ），发送序号是上一次发出数据包的序号+1（ $679155215+1=679155216$ ）。

综上，TCP四次挥手过程验证成功，客户端和服务端断开连接。