计算机网络课程实验报告

郭裕彬 2114052 物联网工程

实验2：配置Web服务器，编写简单页面，分析交互过程

实验要求：

- 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。
- 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。
- 使用HTTP，不要使用HTTPS。
- 提交实验报告。

Web服务器搭建、页面制作

- 服务器租用阿里云2核2GiB云服务器ECS，使用1Mbps专用网络，公网IP为8.130.116.33。装载系统为CentOS 7.4 64位，借助宝塔面板进行网站的管理。Web服务端使用Nginx 1.22.1。
- 编写简单的html网页，包含姓名、学号、专业信息，一张个人Logo图片`logo.png`和自我介绍的m4a音频文件`introduction.m4a`。

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
    <div>
        <h1>基本信息</h1>
```

```
            姓名：郭裕彬
            <br><br>
            专业：物联网工程
            <br><br>
            学号：2114052
            <br><br>
            <img src="LOGO.png" alt="LOGO" title="LOGO"
width="250" height="250"/>
            <br><br>
            <audio src="introduction.m4a" controls></audio>
        </div>
</body>
</html>
```

- 将所有文件挂在到服务器硬盘对应的文件夹中

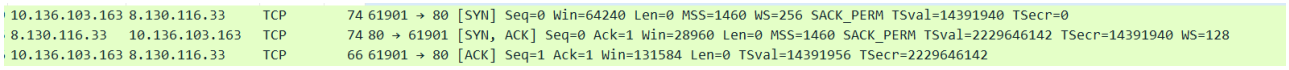
- 关闭对该网站自动的HTTP转HTTPS访问规则

```
9   #SSL-START SSL相关配置，请勿删除或修改下一行带注释的404规则
10  #error_page 404/404.html;
11  #HTTP_TO_HTTPS_START
12  #if ($server_port !~ 443){
13  #   rewrite ^(/.*)$ https://$host$1 permanent;
14  #}
15  #HTTP_TO_HTTPS_END
```

- 使用http访问，得到页面如下，能够正常访问

Wireshark捕获交互过程

三次握手



- 客户端首先向服务器发送TCP连接请求报文，进入同步已发送状态，该报文设置seq初始值为x=0，作为初始选择序号；SYN位被置为1，标识该报文类型。

- 服务器收到报文后，向客户端发送TCP连接请求确认报文，进入同步已接收状态。该报文首部中同步位SYN和确认位ACK都设置为1，表明是一个TCP连接请求；seq设置初始值为y=0，作为TCP服务器进程选择的初始序号；ack的值被设置为1（x+1），作为对客户端选择的初始序号seq的确认。
- 客户端收到TCP连接请求确认报文后，向服务器发送一个普通的TCP确认报文，进入连接已建立状态。该报文首部中确认位ACK被设置为1，表明这是一个普通的TCP确认报文段；seq设置为1，表明是客户端发送的第二个报文；ack的值被设置为1（y+1），作为对服务器选择的初始序号seq的确认。

HTTP的请求和响应

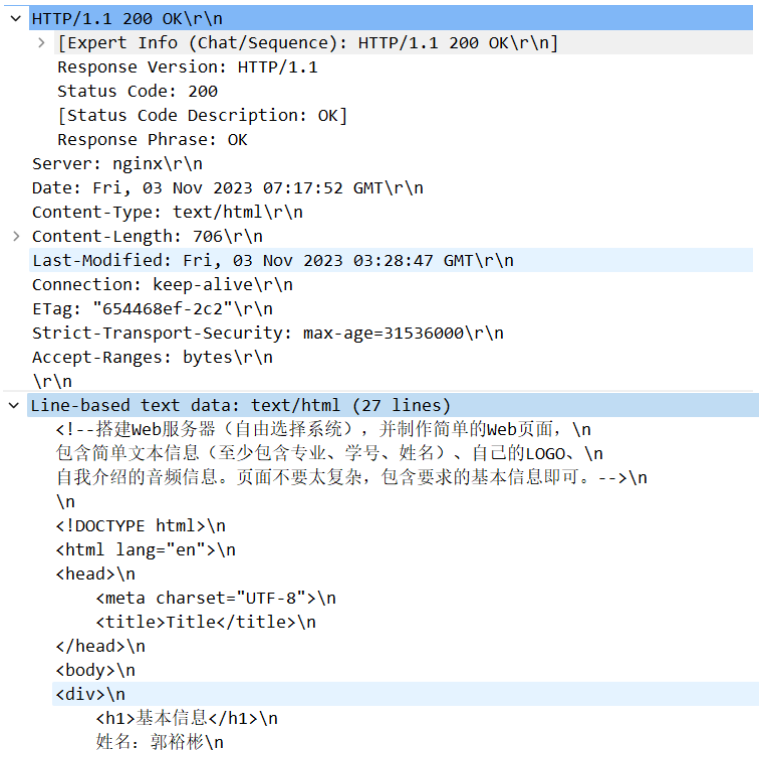
获取页面布局

```
10.136.103.163 8.130.116.33    HTTP      500 GET / HTTP/1.1
8.130.116.33   10.136.103.163  TCP        66 80 → 61901 [ACK] Seq=1 Ack=435 Win=30080 Len=0 TSval=2229646158 TSecr=14391956
8.130.116.33   10.136.103.163  HTTP     1048 HTTP/1.1 200 OK  (text/html)
10.136.103.163 8.130.116.33    TCP        66 61901 → 80 [ACK] Seq=435 Ack=983 Win=130560 Len=0 TSval=14392018 TSecr=2229646158
```

- 以获取页面布局为例：首先客户端向服务端发送一个不带请求体的GET请求行，指明使用的协议为HTTP1.1，资源uri为/，表明获取的是基本的页面文本。

```
GET / HTTP/1.1\r\n
  [Expert Info (Chat/Sequence): GET / HTTP/1.1\r\n]
  Request Method: GET
  Request URI: /
  Request Version: HTTP/1.1
```

- 服务器收到该报文后，向客户端发回一个ACK确认报文，确认该HTTP请求已收到。
- 服务端向客户端发送响应报文，包括响应头和响应体。响应头中响应行包含了使用的协议HTTP1.1、返回的状态码200和该状态码对应的文本“OK”；此外响应头中还有各种基本的信息，例如Content-Type标识了响应体的数据类型为text/html，Connection标识了该链接为持久型连接，对应HTTP1.1；响应体包含了该响应的数据信息，本次请求的为HTML页面文本，数据量较小，直接包含在该条数据报中。



- 客户端收到上述响应报文后，向服务端发送一个ACK确认报文，确认该HTTP响应已收到。

获取logo图片

- 与上述过程类似，不同点在于图片数据文件较大，超过了客户端初始定义的MSS（实践中往往使用MTU值减去IP数据报包头和TCP数据段的包头共40Bytes得到的1460Bytes作为MSS）1460Bytes，故数据在TCP层进行了分段，从下图可以看出，图片问及哎你33745Bytes，共分成了24个段来传输，其中最大的有效数据长度为1448Bytes(1460Bytes减去12Bytes的TCP Timestamp Option)。

```
[24 Reassembled TCP Segments (33745 bytes): #1400(1448), #1401(1448), #1402(1448), #1403(1448),
  [Frame: 1400, payload: 0-1447 (1448 bytes)]
  [Frame: 1401, payload: 1448-2895 (1448 bytes)]
  [Frame: 1402, payload: 2896-4343 (1448 bytes)]
  [Frame: 1403, payload: 4344-5791 (1448 bytes)]
  [Frame: 1404, payload: 5792-7239 (1448 bytes)]
  [Frame: 1405, payload: 7240-8687 (1448 bytes)]
  [Frame: 1406, payload: 8688-10135 (1448 bytes)]
  [Frame: 1407, payload: 10136-11583 (1448 bytes)]
  [Frame: 1408, payload: 11584-13031 (1448 bytes)]
  [Frame: 1409, payload: 13032-14479 (1448 bytes)]
  [Frame: 1414, payload: 14480-15927 (1448 bytes)]
  [Frame: 1415, payload: 15928-17375 (1448 bytes)]
  [Frame: 1416, payload: 17376-18823 (1448 bytes)]
  [Frame: 1417, payload: 18824-20271 (1448 bytes)]
  [Frame: 1418, payload: 20272-21719 (1448 bytes)]
  [Frame: 1419, payload: 21720-23167 (1448 bytes)]
  [Frame: 1420, payload: 23168-24615 (1448 bytes)]
  [Frame: 1421, payload: 24616-26063 (1448 bytes)]
  [Frame: 1422, payload: 26064-27511 (1448 bytes)]
  [Frame: 1423, payload: 27512-28959 (1448 bytes)]
  [Frame: 1428, payload: 28960-30407 (1448 bytes)]
  [Frame: 1429, payload: 30408-31855 (1448 bytes)]
  [Frame: 1430, payload: 31856-33303 (1448 bytes)]
  [Frame: 1431, payload: 33304-33744 (441 bytes)]
  [Segment count: 24]
  [Reassembled TCP length: 33745]
  [Reassembled TCP Data: 485454502f312e3120323030204f4b0d0a5365727665723a206e67696e780d0a446174
```

获取自我介绍音频

- 与上述过程类似，不同 点在于音频文件在分段的基础上使用了HTTP1.1的断点续传方式，虽然只是启用了该方式、并没有实际上获取文件部分内容。请求报文中出现了字段`Range:bytes=0-`，表明直接拉取整个文件内容。
- 对应地，响应头中的状态码也从200变为了206 Partial Content，尽管传送的仍然是整个文件。响应报文中的Content-Range字段标识了当前发送的范围和文件的总大小。从下图可以看出，音频文件的大小为143100Bytes。

```
1562 8.130.116.33   10.136.103.163   HTTP      126 HTTP/1.1 206 Partial Content  (audio/x-m4a)
1563 10.136.103.163 8.130.116.33     TCP        66 61902 → 80 [ACK] Seq=354 Ack=143413 Win=131584 L
    Status Code: 206
    [Status Code Description: Partial Content]
    Response Phrase: Partial Content
  Server: nginx\r\n
  Date: Fri, 03 Nov 2023 07:17:52 GMT\r\n
  Content-Type: audio/x-m4a\r\n
> Content-Length: 143100\r\n
  Last-Modified: Thu, 02 Nov 2023 04:38:06 GMT\r\n
  Connection: keep-alive\r\n
  ETag: "654327ae-22efc"\r\n
  Strict-Transport-Security: max-age=31536000\r\n
  Content-Range: bytes 0-143099/143100\r\n
  \r\n
```

四次挥手

```
10.136.103.163 8.130.116.33     TCP        66 61901 → 80 [FIN, ACK] Seq=807 Ack=34728 Win=131584 Len=0 TSval=14400304 TSecr=2229646259
10.136.103.163 8.130.116.33     TCP        66 61902 → 80 [FIN, ACK] Seq=354 Ack=143413 Win=131584 Len=0 TSval=14400304 TSecr=2229646422
8.130.116.33   10.136.103.163   TCP        66 80 → 61901 [FIN, ACK] Seq=34728 Ack=808 Win=31104 Len=0 TSval=2229654510 TSecr=14400304
8.130.116.33   10.136.103.163   TCP        66 80 → 61902 [FIN, ACK] Seq=143413 Ack=355 Win=30080 Len=0 TSval=2229654511 TSecr=14400304
10.136.103.163 8.130.116.33     TCP        66 61901 → 80 [ACK] Seq=808 Ack=34729 Win=131584 Len=0 TSval=14400328 TSecr=2229654510
10.136.103.163 8.130.116.33     TCP        66 61902 → 80 [ACK] Seq=355 Ack=143414 Win=131584 Len=0 TSval=14400328 TSecr=2229654511
```

- 客户端主动发送一个FIN包进入FIN_WATI1状态
- 服务器接收到主动方发送的FIN，发送ACK包，同时由于服务器方已经没有要发送的数据了，并且TCP延迟发送机制开启，服务器向客户端发送一个FIN包，这两个包即第二三次挥手合并发送，服务器方跳过CLOSE_WAIT进入LAST_ACK状态
- 客户端接收到服务器发送的ACK&FIN，发送ACK包。客户端越过FIN_WAIT2状态，进入TIME_WAIT状态，经过2MSL的时间后关闭连接；服务器收到客户端的ACK后，关闭连接。等待2MSL的原因是防止这个ACK包在传输过程中丢失，2MSL的时间能够满足服务器在未收到ACK后重传到达客户端的情况，从而保证了连接的正常关闭。

思考及分析

- HTTP1.1不能像HTTP2那样实现多路复用，所以当浏览器检测到需要传输多个比较大的数据时，就会开启多个端口建立多个TCP连接以进行数据的传输，例如在提交的捕获文件中，图片接近传输完成时，浏览器开启了81902第二个端口与服务器三次握手、准备开始音频文件的传输。但整体情况而言，除了在此处81901、81902两个端口有交叉出现表示同时使用的情况外，两个端口传输数据的时间基本上是不重叠的，因此两个端口的使用是在本特殊情况下没有很好的效果、还是说目的就是避免前一个数据文件传输时出现丢包等待等情况浪费时间才开的第二个端口，需要查阅更多的资料来确定。

```
TCP      1514 80 → 61901 [ACK] Seq=27047 Ack=807 W
TCP      1514 80 → 61901 [ACK] Seq=28495 Ack=807 W
TCP        66 61901 → 80 [ACK] Seq=807 Ack=29943 W
TCP        74 80 → 61902 [SYN, ACK] Seq=0 Ack=1 Wi
TCP        66 61902 → 80 [ACK] Seq=1 Ack=1 Win=131
HTTP      419 GET /introduction.m4a HTTP/1.1
TCP      1514 80 → 61901 [ACK] Seq=29943 Ack=807 W
TCP      1514 80 → 61901 [ACK] Seq=31391 Ack=807 W
TCP      1514 80 → 61901 [ACK] Seq=32839 Ack=807 W
HTTP      507 HTTP/1.1 200 OK  (PNG)
TCP        66 80 → 61902 [ACK] Seq=1 Ack=354 Win=3
TCP        66 61901 → 80 [ACK] Seq=807 Ack=34728 W
TCP      1514 80 → 61902 [ACK] Seq=1 Ack=354 Win=3
```

- 观察到，在数据分段传输的时候，每隔一定的包数量，客户端就会回送对于这些包里最后一个包的ACK确认报文，判断这是用于向服务端确认在此之前的报文都成功接收；但整体情况观察下来，这些ACK确认包发送的间隔并不统一，与时间段也没有明显的关联。多次测试时这些ACK包也不总是在同一个位置发送，查阅资料发现，ACK确认包发送的参考因素有很多，浏览器也会通过流量控制和拥塞控制根据网络条件和双端规格来调整ACK发送间隔，故这种确认ACK的包的发送情况是不可规定和预测的。

```
      1514 80 → 61902 [ACK]
        66 61902 → 80 [ACK]
      1514 80 → 61902 [ACK]
      1514 80 → 61902 [ACK]
        66 61902 → 80 [ACK]
      1514 80 → 61902 [ACK]
      1514 80 → 61902 [ACK]
      1514 80 → 61902 [ACK]
      1514 80 → 61902 [ACK]
      1514 80 → 61902 [ACK]
      1514 80 → 61902 [ACK]
      1514 80 → 61902 [ACK]
      1514 80 → 61902 [ACK]
        66 61902 → 80 [ACK]
```