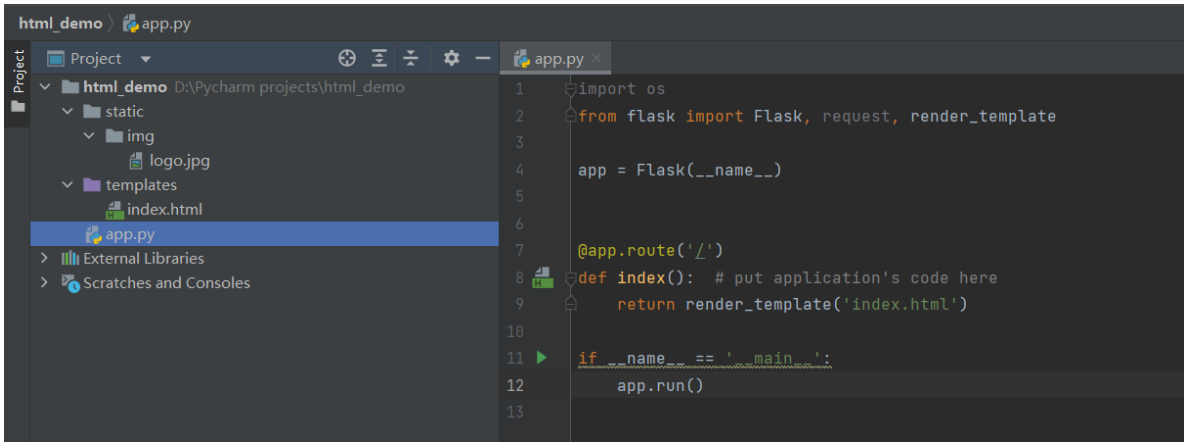
计算机网络第二次实验

学号：2010764 姓名：高森森 班号：0993

一、Web服务器搭建

平台：基于Python的Flask

使用Flask框架搭建一个简单的Web服务器，框架如下：



启动该服务器后，在浏览器输入127.0.0.1:5000即可进入该网站。

二、简单的web页面

我们设计如下简单的web页面，它包含了姓名、学号、专业，以及自己设计的一张logo图片。

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>hello</title>
</head>
<body>
<h2>专业：计算机科学与技术</h2>
<h2>学号：2010764</h2>
<h2>姓名：高森森</h2>
<h2>下面是我的LOGO:</h2>
<img src="../static/img/logo.jpg" alt="图片加载失败">
</body>
</html>
```

web页面展示：

专业：计算机科学与技术

学号：2010764

姓名：高森森

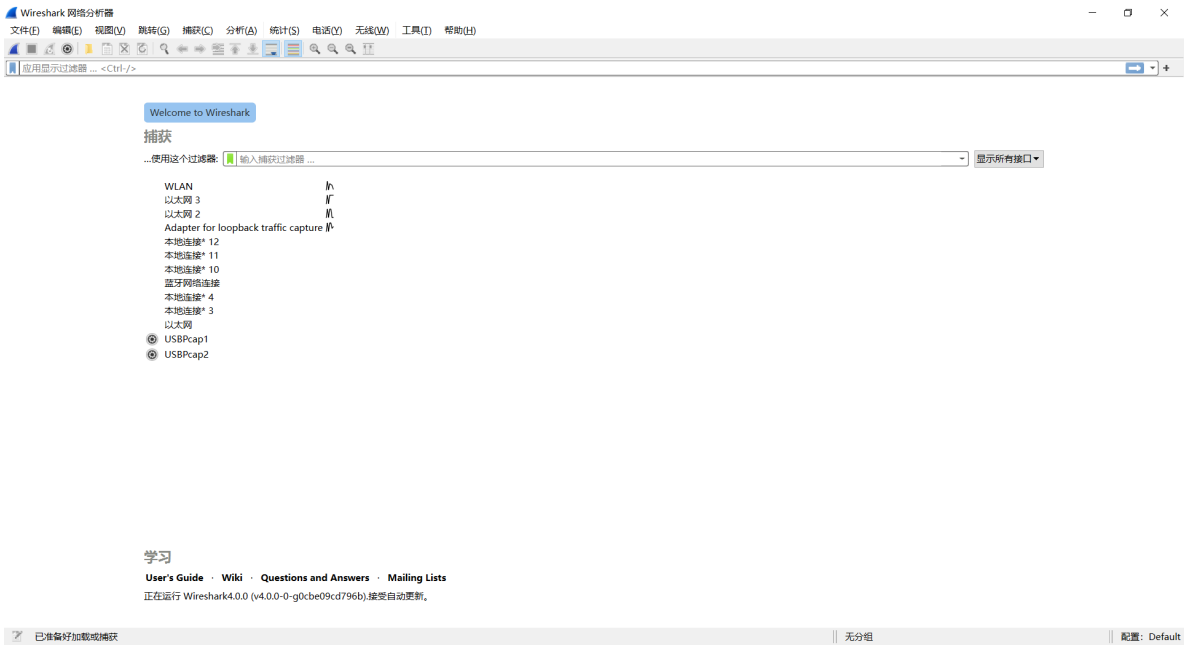
下面是我的LOGO:

NEVER GIVE UP

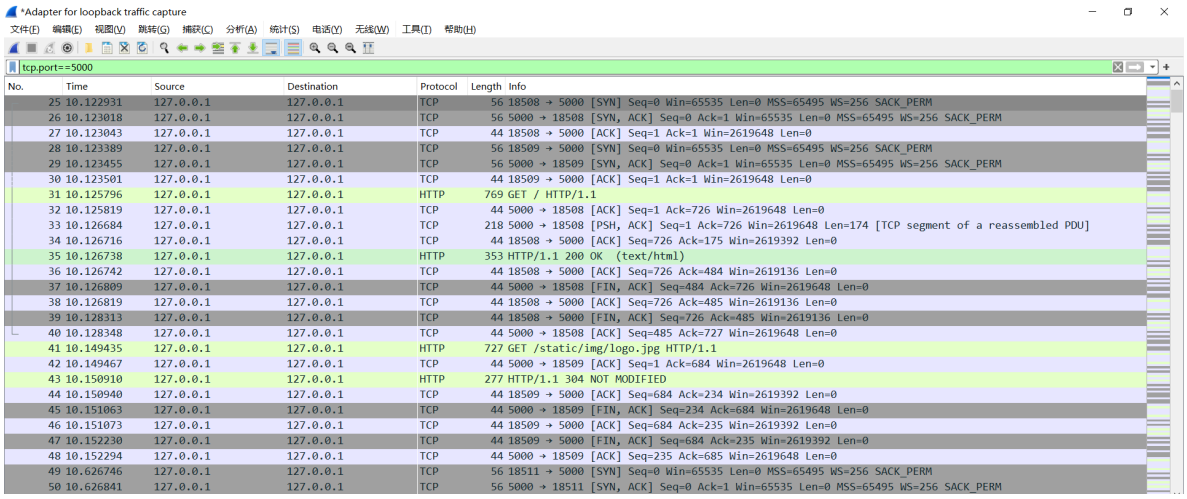
三、wireshark捕获交互过程

1. 抓包

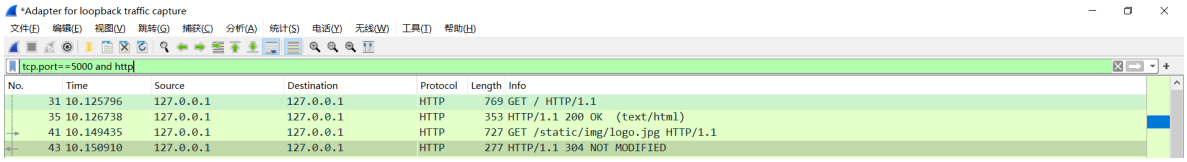
我们先开启Wireshark，并在浏览器中输入自己设计网页的网址，观察Wireshark界面如下所示：



由于是抓取通过127.0.0.1本地回环地址的包，所以选择Adapter for loopback traffic capture。进入后我们使用过滤器，设置tcp.port==5000查看端口为5000的所有包。下图是抓取5000端口的包。

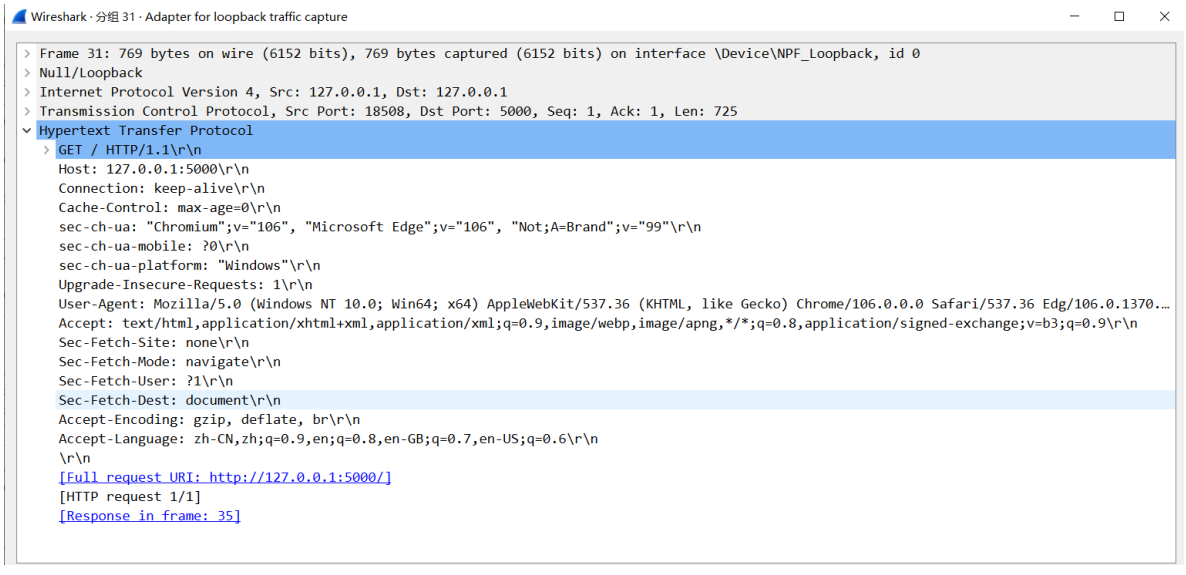


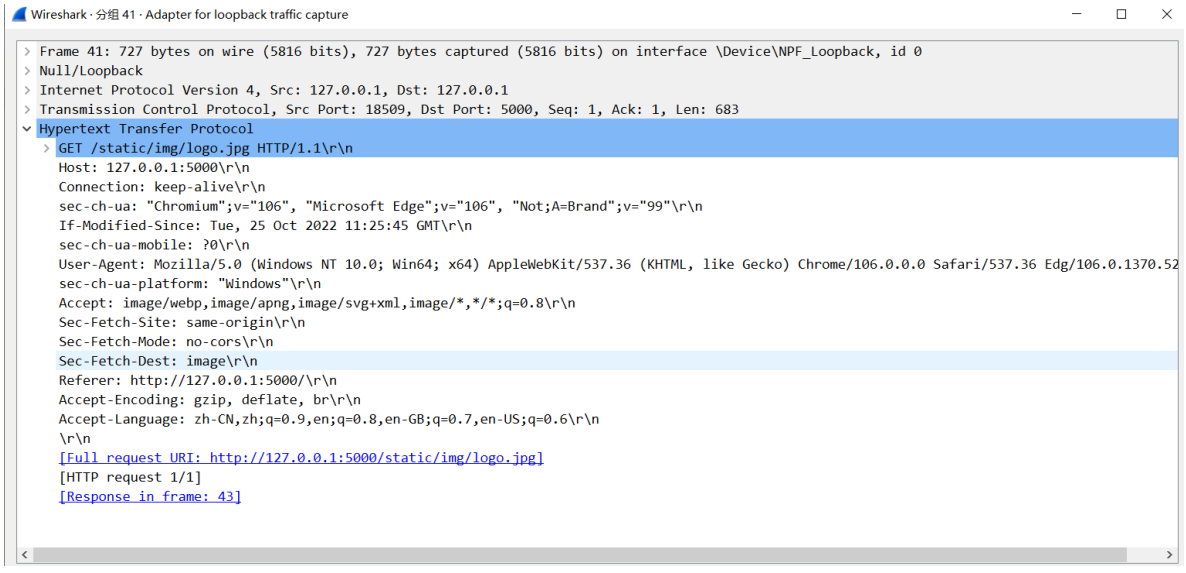
2. HTTP报文



首先我们可以看出，这里的HTTP报文采用的是http1.1协议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。

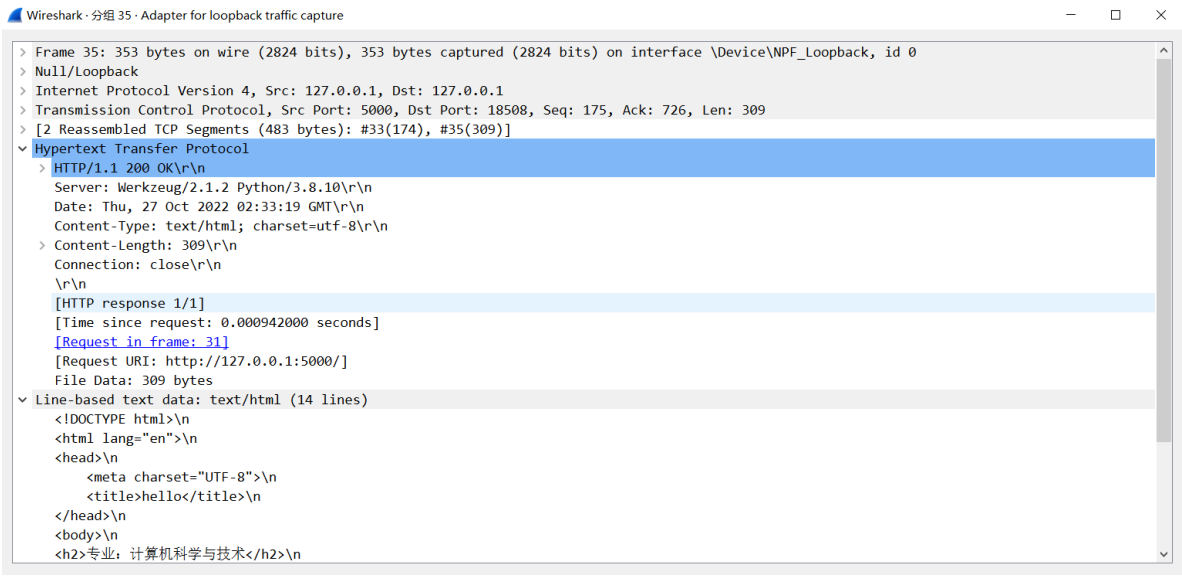
- HTTP请求报文

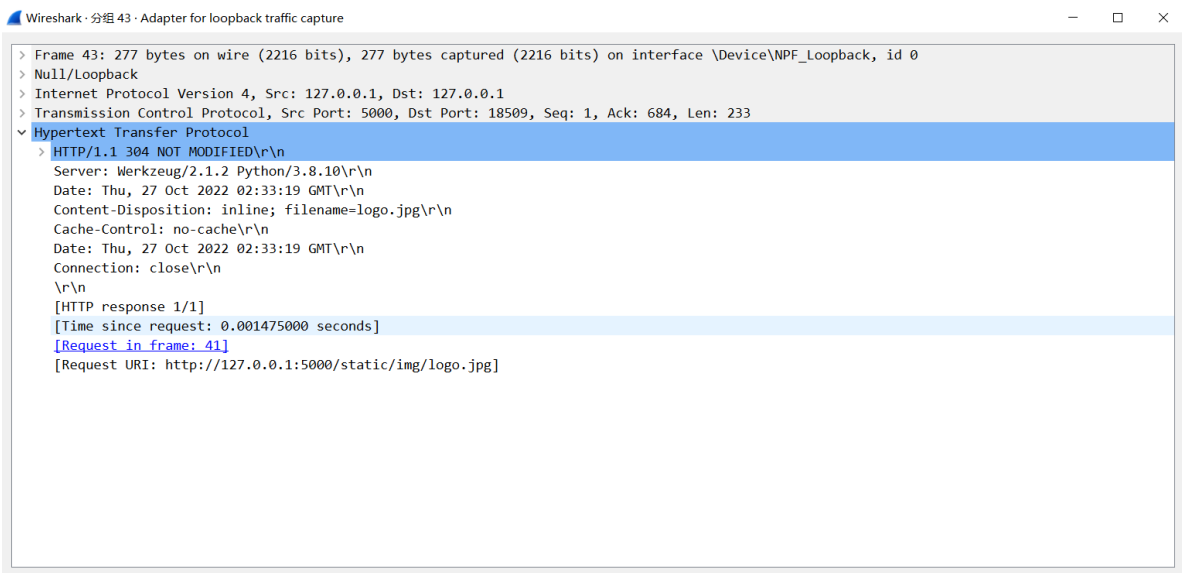




GET /HTTP/1.1 \r\n称为请求行，由三部分组成：请求方法、URL、HTTP协议版本。其中URL给出了请求资源的位置。接下来的部分称为请求头部，其紧跟着请求行，该部分主要是用于描述请求正文，说明请求源、连接类型、以及一些Cookie信息等。

- HTTP响应报文





响应报文由状态行、响应头部、空行和响应体4个部分组成。其中状态行主要给出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（可以是html文档内容或图片等多种格式数据）。

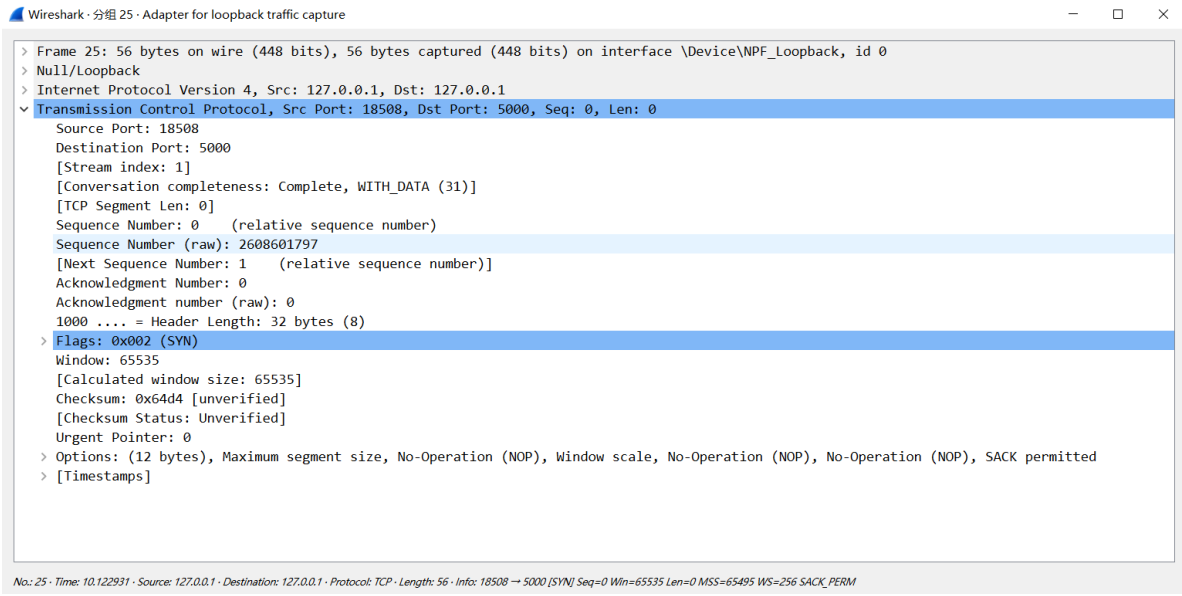
响应状态码和响应描述

我们会发现HTTP响应的状态码和响应描述有200OK和304 NOT MODIFIED两种情况，这是因为客户端已经有缓存的文档，发出请求时，服务器判断客户端缓存是否是最新的，如果是，返回304 NOT MODIFIED，告诉客户端可以继续使用。

持久连接和非持久连接

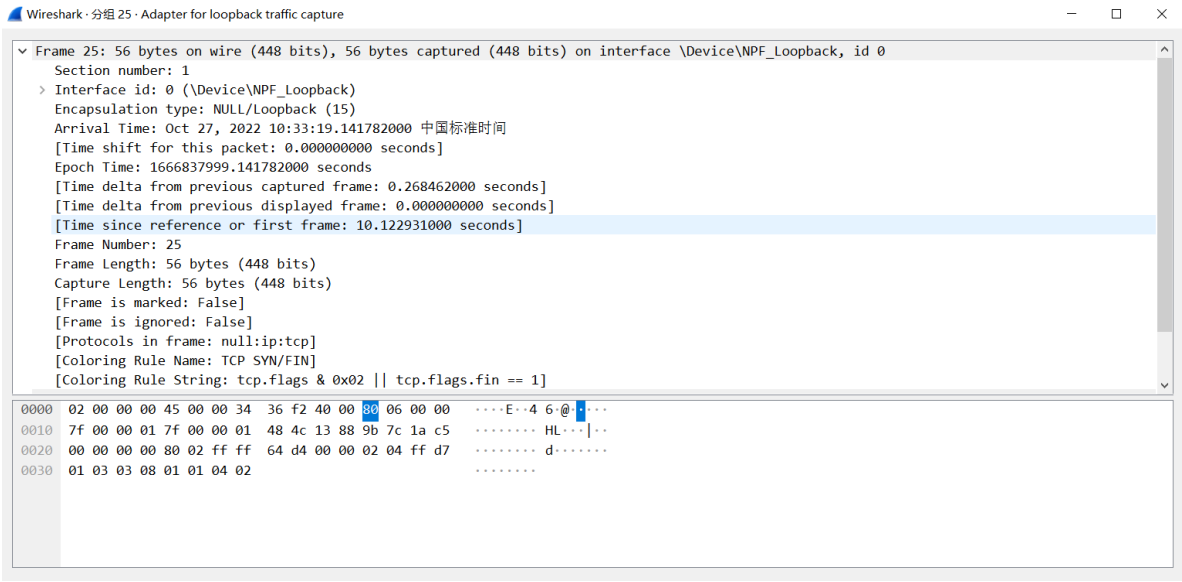
HTTP报文里有描述连接类型的connection，如果为keep-alive则为持久连接，如果为close则为非持久连接，当客户端和服务器均是keep-alive是持久连接，任一方为close则不是持久连接。非持久的TCP连接仅能进行一次请求和响应。

3. TCP报文

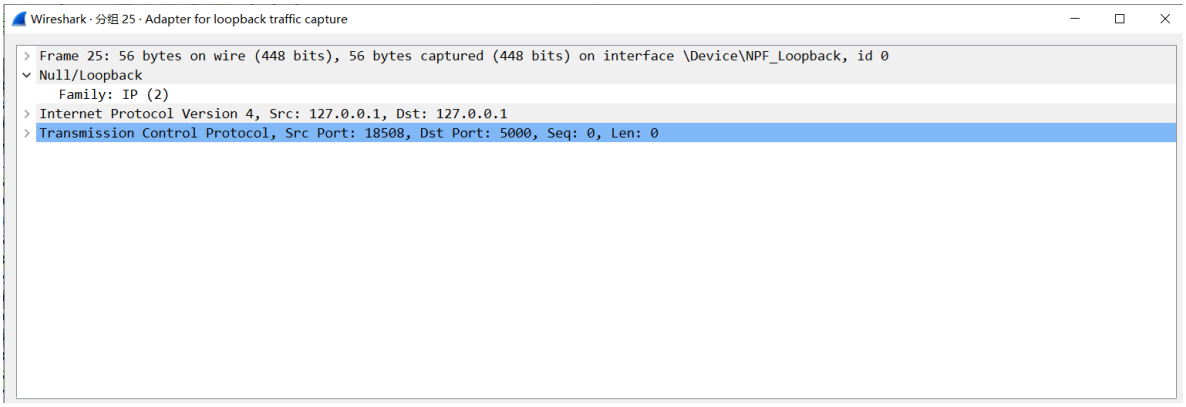


TCP报文分为如下四部分：

- 物理层数据帧概况

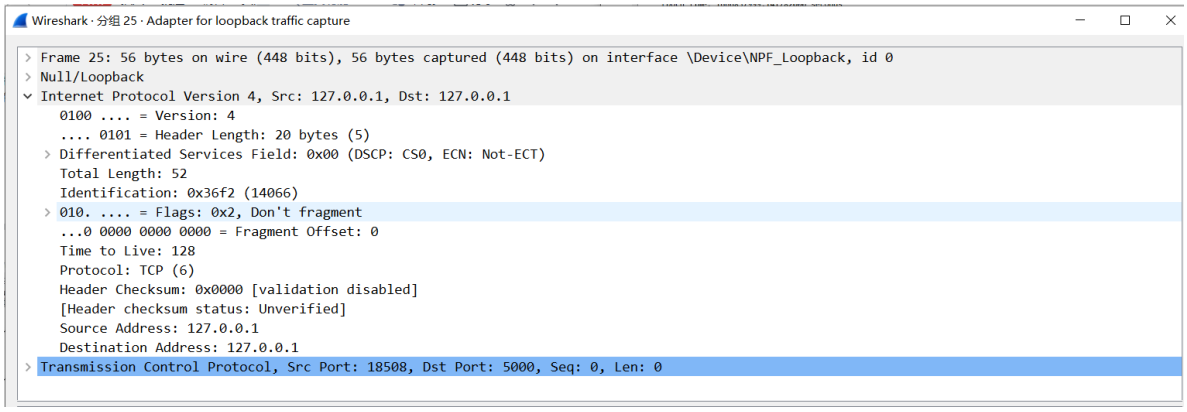


- 数据层头部信息



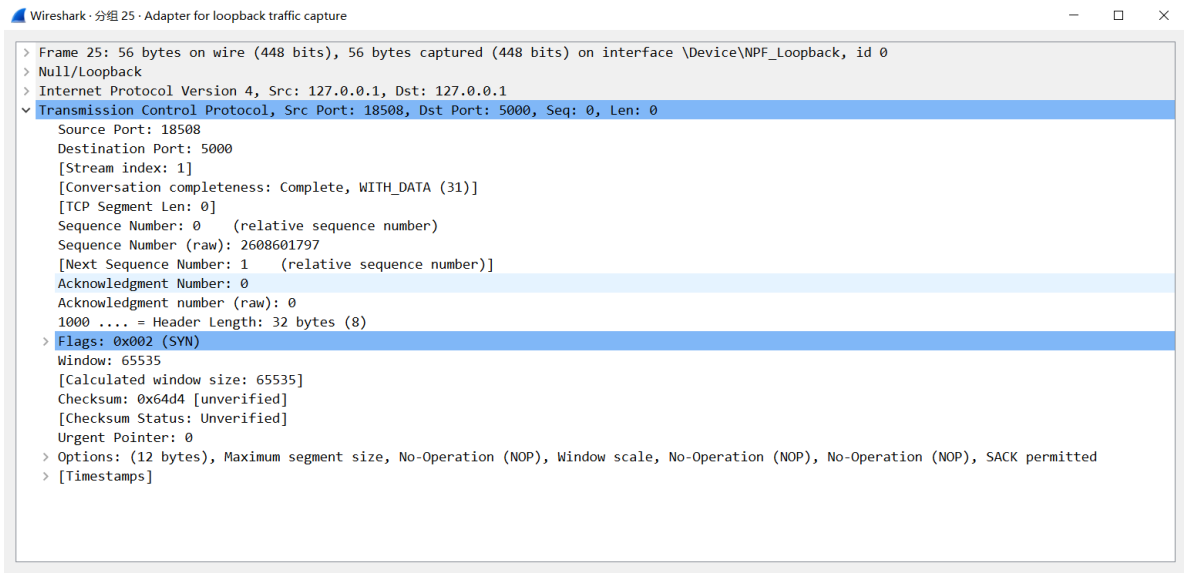
由于是本地地址回环，所以数据层头部并没有什么信息。

- 互联网层IP包头部信息



该头部信息指明使用IPV4协议，包头长度为20字节，IP包总长度为52字节，标记字段为0x2，生存期为128，包内封装的上层协议为TCP。该头部信息还包含了头部校验和，源IP地址（127.0.0.1），目的IP地址（127.0.0.1）等。

- 传输层数据段头部信息



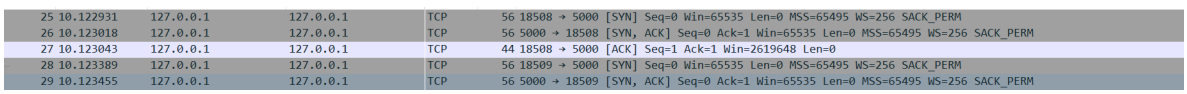
上述信息包含：源端口、目的端口，标志为SYN（发送SYN报文到服务器），seq=0（表明为第一次握手）。

4. TCP的三次握手

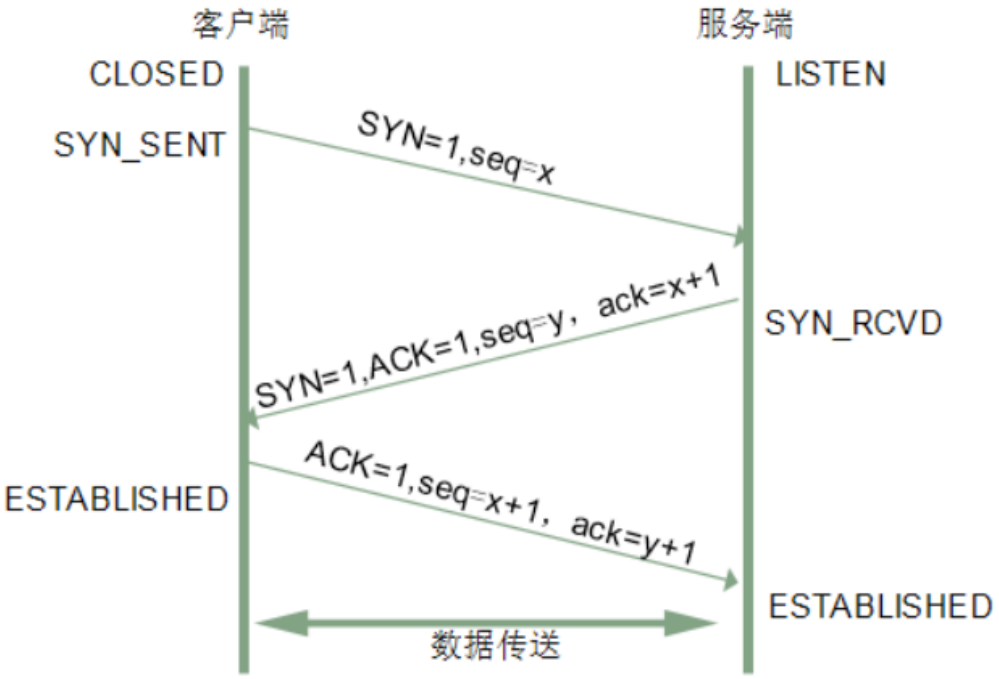
在明确TCP的三次握手之前，我们需要明确TCP报文的 6 个标志位：

- SYN(synchronous)： 发送/同步标志，用来建立连接，和下面的第二个标志位ACK搭配使用。连接开始时，SYN=1，ACK=0，代表连接开始但是未获得响应。当连接被响应的时候，标志位会发生变化，其中ACK会置为1，代表确认收到连接请求，此时的标志位变成了 SYN=1，ACK=1。
- ACK(acknowledgement)：确认标志，表示确认收到请求。
- PSH(push)：表示推送操作，就是指数据包到达接收端以后，不对其进行队列处理，而是尽可能的将数据交给应用程序处理；
- FIN(finish)：结束标志，用于结束一个TCP会话；
- RST(reset)：重置复位标志，用于复位对应的TCP连接。
- URG(urgent)：紧急标志，用于保证TCP连接不被中断，并且督促中间层设备尽快处理。

由下面的报文，我们可以看出浏览器在请求时开启了多线程，同时使用10508和10509两个端口与服务器进行交互。



三次握手其实就是指建立一个TCP连接时，需要客户端和服务器总共发送3个包。进行三次握手的主要作用就是为了确认双方的接收能力和发送能力是否正常、指定自己的初始化序列号为后面的可靠性传送做准备。实质上就是连接服务器指定端口，建立TCP连接，并同步连接双方的序列号和确认号，交换TCP窗口大小信息。



- 第一次握手

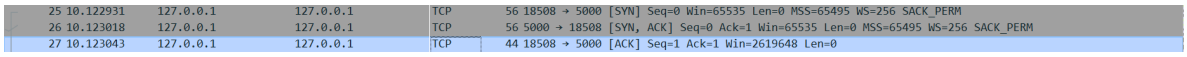
  客户端给服务端发一个 SYN 报文，并指明客户端的初始化序列号 ISN。此时客户端处于 `SYN_SENT` 状态。首部的同步位SYN=1，初始序号seq=x，SYN=1的报文段不能携带数据，但要消耗掉一个序号。

- 第二次握手

  服务器收到客户端的 SYN 报文之后，会以自己的 SYN 报文作为应答，并且也是指定了自己的初始化序列号 ISN(s)。同时会把客户端的 ISN + 1 作为ACK 的值，表示自己已经收到了客户端的 SYN，此时服务器处于 `SYN_RCVD` 的状态。

- 第三次握手

  客户端收到 SYN 报文之后，会发送一个 ACK 报文，当然，也是一样把服务器的 ISN + 1 作为 ACK 的值，表示已经收到了服务端的 SYN 报文，此时客户端处于 ESTABLISHED 状态。服务器收到 ACK 报文之后，也处于 ESTABLISHED 状态，此时，双方已建立起了连接。



如上图所示，即为10508端口tcp三次握手过程。

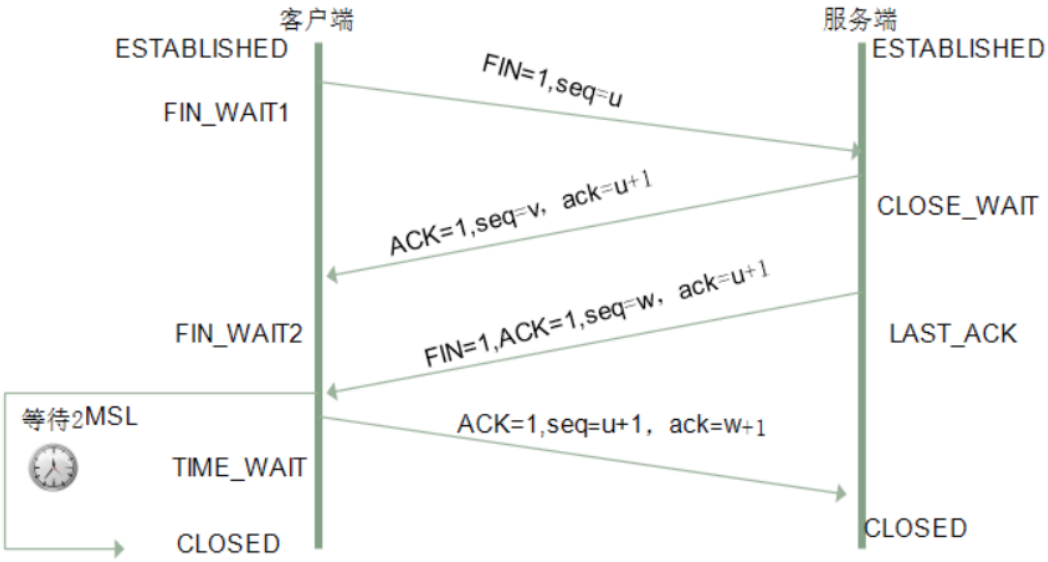
必须进行三次握手的原因：

只有经历三次握手，客户端和服务器才都确认双方的发送能力和接收能力正常

- 第一次握手：服务端得出结论：客户端的发送能力、服务端的接收能力是正常的。
- 第二次握手：客户端得出结论：服务端的接收、发送能力，客户端的接收、发送能力是正常的。不过此时服务器并不能确认客户端的接收能力是否正常。
- 第三次握手：服务端得出结论：客户端的接收、发送能力正常，服务器自己的发送、接收能力也正常。

5. TCP的四次挥手

建立一个连接需要三次握手，而终止一个连接要经过四次挥手。这由TCP的**半关闭**造成的。所谓的半关闭，其实就是TCP提供了连接的一端在结束它的发送后还能接收来自另一端数据的能力。



- 第一次挥手

  客户端发送一个 FIN 报文，报文中会指定一个序列号。此时客户端处于 `FIN_WAIT1` 状态。即发出**连接释放报文段**（FIN=1，序号seq=u），并停止再发送数据，主动关闭TCP连接，进入FIN_WAIT1（终止等待1）状态，等待服务端的确认。

- 第二次挥手

  服务端收到 FIN 之后，会发送 ACK 报文，且把客户端的序列号值 +1 作为 ACK 报文的序列号值，表明已经收到客户端的报文了，此时服务端处于 CLOSE_WAIT 状态。

  即服务端收到连接释放报文段后即发出确认报文段（ACK=1，确认号ack=u+1，序号seq=v），服务端进入CLOSE_WAIT（关闭等待）状态，此时的TCP处于半关闭状态，客户端到服务端的连接释放。客户端收到服务端的确认后，进入FIN_WAIT2（终止等待2）状态，等待服务端发出的连接释放报文段。

- 第三次挥手

  如果服务端也想断开连接了，和客户端的第一次挥手一样，发给 FIN 报文，且指定一个序列号。此时服务端处于 LAST_ACK 的状态。

  即服务端没有要向客户端发出的数据，服务端发出连接释放报文段（FIN=1，ACK=1，序号seq=w，确认号ack=u+1），服务端进入LAST_ACK（最后确认）状态，等待客户端的确认。

- 第四次挥手

  客户端收到 FIN 之后，一样发送一个 ACK 报文作为应答，且把服务端的序列号值 +1 作为自己 ACK 报文的序列号值，此时客户端处于 TIME_WAIT 状态。需要过一阵子以确保服务端收到自己的 ACK 报文之后才会进入 CLOSED 状态，服务端收到 ACK 报文之后，就处于关闭连接了，处于 CLOSED 状态。

  即客户端收到服务端的连接释放报文段后，对此发出确认报文段（ACK=1，seq=u+1，ack=w+1），客户端进入TIME_WAIT（时间等待）状态。此时TCP未释放掉，需要经过时间等待计时器设置的时间2MSL后，客户端才进入CLOSED状态。

四次挥手的原因：

当服务端收到FIN报文时，很可能并不会立即关闭SOCKET，所以只能先回复一个ACK报文，告诉客户端，“你发的FIN报文我收到了”。只有等到我服务端所有的报文都发送完了，我才能发送FIN报文，因此不能一起发送。故需要四次挥手。

客户端为什么需要等待2MSL再关闭连接？

MSL为报文段最大生存时间。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。