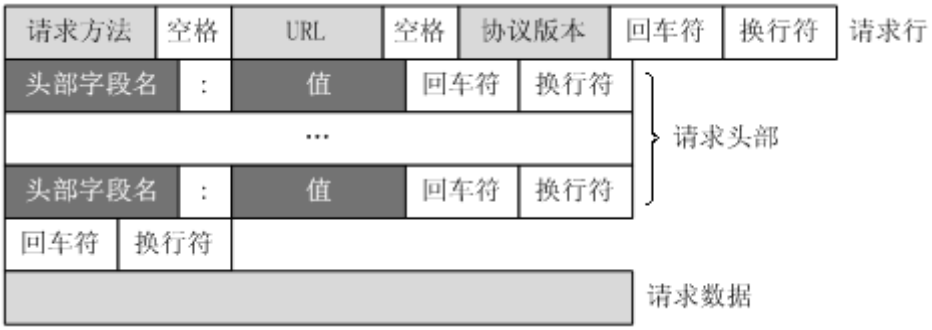
Http

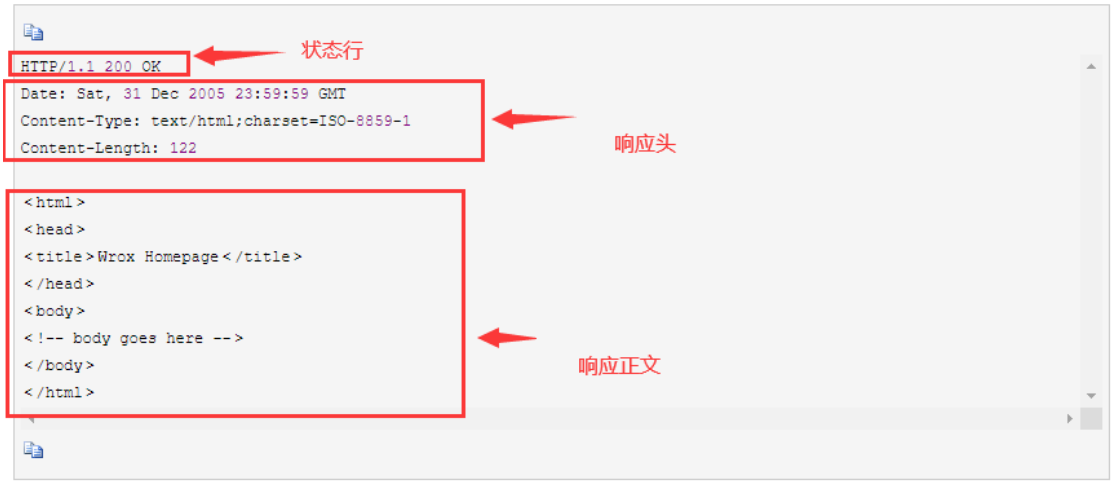
> HTTP(HyperText Transfer Protocol):超文本传输协议
>
> HTTP是位于TCP之上的协议，是应用层协议，规定了客户端和服务器之间的通信格式，并不负责数据包的传输，因为**负责数据包传输的是传输层协议TCP**。可以把HTTP理解为一种文本规范，客户端把自己想要传输的数据按照此种规范传给服务端，服务端收到之后同样按照规范来解析数据。
>
> HTTP 采用 BS 架构，也就是浏览器到服务器的架构，客户端通过浏览器发送 HTTP 请求给服务器，服务器经过解析响应客户端的请求。
>
> HTTP：80端口
>
> HTTPS：443端口

HTTP请求报文格式：

> 
>
>
> 一个HTTP请求报文由请求行（request line）、请求头部（header）、空行和请求数据4个部分组成
>
> 
>
>
> 请求行由请求方法字段、URL字段和HTTP协议版本字段3个字段组成，它们用空格分隔。例如，GET /index.html HTTP/1.1。HTTP协议的请求方法有GET、POST、HEAD、PUT、DELETE、OPTIONS、TRACE、CONNECT。
>
> 请求头部由关键字/值对组成，每行一对，关键字和值用英文冒号":"分隔。请求头部通知服务器有关于客户端请求的信息。
>
> 最后一个请求头之后是一个空行，发送回车符和换行符，通知服务器以下不再有请求头。
>
> 请求数据不在GET方法中使用，而是在POST方法中使用。POST方法适用于需要客户填写表单的场合。与请求数据相关的最常使用的请求头是Content-Type和Content-Length。

HTTP响应报文格式

HTTP响应报文主要由状态行、响应头部、空行和响应正文4部分组成



响应状态码：

- 1xx：指示信息--表示请求已接收，继续处理。
- 2xx：成功--表示请求已被成功接收、理解、接受。
- 3xx：重定向--要完成请求必须进行更进一步的操作。
- 4xx：客户端错误--请求有语法错误或请求无法实现。
- 5xx：服务器端错误--服务器未能实现合法的请求。

HTTP版本

HTTP/0.9

这个版本只有一个命令：GET。通过 GET 可以获取服务器的资源.服务器只能回应 HTML 格式的字符串，不能回应其它格式，也就是说**图像、视频等多媒体资源，在 HTTP/0.9 这个版本上是无法进行传输的。**

HTTP/1.0　核心丰富了传输的内容

增加了 POST 命令和 HEAD 命令，丰富了浏览器与服务器的互动手段。这个版本的 HTTP 协议可以发送任何格式的内容，包括传输文字、图像、视频、文件等

HTTP/1.1　核心加入了持久化连接

将持久化连接加入了 HTTP 标准，即 TCP 连接默认不关闭，可以被多个请求复用。此外，HTTP/1.1 版还新增了许多方法，例如：PUT、PATCH、HEAD、OPTIONS、DELETE。

一次完整的HTTP请求所经历的7个步骤

> **1. 建立TCP连接**
>
> 在HTTP工作开始之前，Web浏览器首先要通过网络与Web服务器建立连接，该连接是通过TCP来完成的
>
> **2. Web浏览器向Web服务器发送请求命令** (请求行)
>
> 一旦建立了TCP连接，Web浏览器就会向Web服务器发送请求命令。例如：GET/sample/hello.jsp HTTP/1.1。
>
> **3. Web浏览器发送请求头信息**
>
> 浏览器发送其请求命令之后，还要以头信息的形式向Web服务器发送一些别的信息，之后浏览器发送了一空白行来通知服务器，它已经结束了该头信息的发送。
>
> **4. Web服务器应答**
>
> 客户机向服务器发出请求后，服务器会客户机回送应答， HTTP/1.1 200 OK ，应答的第一部分是协议的版本号和应答状态码。
>
> **5. Web服务器发送应答头信息**
>
> 正如客户端会随同请求发送关于自身的信息一样，服务器也会随同应答向用户发送关于它自己的数据及被请求的文档。
>
> **6. Web服务器向浏览器发送数据**
>
> Web服务器向浏览器发送头信息后，它会发送一个空白行来表示头信息的发送到此为结束，接着，它就以Content-Type应答头信息所描述的格式发送用户所请求的实际数据。
>
> **7. Web服务器关闭TCP连接**
>
> 一般情况下，一旦Web服务器向浏览器发送了请求数据，它就要关闭TCP连接，然后如果浏览器或者服务器在其头信息加入了这行代码：
>
> Connection:keep-alive
>
> TCP连接在发送后将仍然保持打开状态，于是，浏览器可以继续通过相同的连接发送请求。保持连接节省了为每个请求建立新连接所需的时间，还节约了网络带宽。

HTTPS(Hyper Text Transfer Protocol over SecureSocket Layer)

> 当对于安全需求，首先想到的就是对信息进行加密。SSL ，安全套接层，顾名思义是在 TCP 上提供的安全套接字层。其位于应用层和传输层之间，应用层数据不再直接传递给传输层而是传递给 SSL 层，SSL 层对从应用层收到的数据进行加密，利用数据加密、身份验证和消息完整性验证机制，为网络上数据的传输提供安全性保证。HTTPS 便是指 Hyper Text Transfer Protocol over SecureSocket Layer。

队首阻塞

> 1. 队首阻塞
>
> 就是需要排队，队首的事情没有处理完的时候，后面的人都要等着。
>
> 2. http1.0的队首阻塞 客户端请求需要一次处理

对于同一个tcp连接，所有的http1.0请求放入队列中，只有前一个请求的响应收到了，然后才能发送下一个请求。

3. http1.1的队首阻塞 客户端的请求可以并行发送，但是服务端的响应需要依次返回

对于同一个tcp连接，http1.1允许一次发送多个http1.1请求，也就是说，不必等前一个响应收到，就可以发送下一个请求，这样就解决了http1.0的客户端的队首阻塞。但是，http1.1规定，服务器端的响应的发送要根据请求被接收的顺序排队，也就是说，先接收到的请求的响应也要先发送。这样造成的问题是，如果最先收到的请求的处理时间长的话，响应生成也慢，就会阻塞已经生成了的响应的发送。也会造成队首阻塞。

可见，http1.1的队首阻塞发生在服务器端。

4. **HTTP/2废弃了管道化的方式**，而是创新性的引入了帧、消息和数据流等概念。**客户端和服务器可以把 HTTP 消息分解为互不依赖的帧，然后乱序发送，最后再在另一端把它们重新组合起来。**因为没有顺序的概念了，所以就不需要阻塞了，这就解决了http1.1的服务端队首阻塞的问题。

HTTP/2(https://www.hollischuang.com/archives/2066)

1.二进制分帧

在HTTP/2中，在应用层（HTTP2.0）和传输层（TCP或者UDP）之间加了一层：二进制分帧层。这是HTTP2中最大的改变。HTTP2之所以性能会比HTTP1.1有那么大的提高，很大程度上正是由于这一层的引入。

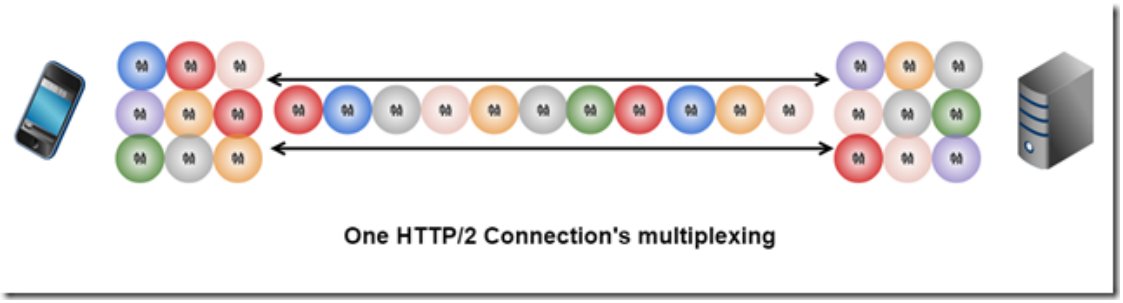
在二进制分帧层中， HTTP/2 会将所有传输的信息分割为更小的消息和帧（frame）,并对它们采用二进制格式的编码。

这种单连接多资源的方式，减少了服务端的压力，使得内存占用更少，连接吞吐量更大。而且，TCP连接数的减少使得网络拥塞状况得以改善，同时慢启动时间的减少，使拥塞和丢包恢复速度更快。

2.多路复用

多路复用允许同时通过单一的HTTP/2.0连接发起多重的请求-响应消息。在HTTP1.1协议中，浏览器客户端在同一时间，针对同一域名下的请求有一定数量的限制，超过了这个限制的请求就会被阻塞。

多路复用允许同时通过**单一的 HTTP2.0 连接发起多重的“请求-响应”消息。**



HTTP2的请求的TCP的connection一旦建立，后续请求以stream的方式发送。每个stream的基本组成单位是frame（二进制帧）。客户端和服务器可以把 HTTP 消息分解为互不依赖的帧，然后乱序发送，最后再在另一端把它们重新组合起来。也就是说， HTTP2.0 通信都在一个连接上完成，这个连接可以承载任意数量的双向数据流。就好比，我请求一个页面 http://www.hollischuang.com 。**页面上所有的资源请求都是客户端与服务器上的一条 TCP 上请求和响应的**！

3.header压缩

HTTP/1.1的header带有大量信息，而且每次都要重复发送。HTTP/2 为了减少这部分开销，采用了HPACK 头部压缩算法对Header进行压缩。

4.服务端推送

简单来讲就是当用户的浏览器和服务器在建立连接后，服务器主动将一些资源推送给浏览器并缓存起来的机制。有了缓存，当浏览器想要访问已缓存的资源的时候就可以直接从缓存中读取了。

TCP队首阻塞

HTTP/2虽然解决了http1.1.的队首阻塞问题，但是HTTP/2仍然会存在TCP队头阻塞的问题，那是因为HTTP/2其实还是依赖TCP协议实现的。

TCP传输过程中会把数据拆分为一个个**按照顺序**排列的数据包，这些数据包通过网络传输到了接收端，接收端再**按照顺序**将这些数据包组合成原始数据，这样就完成了数据传输。

但是如果其中的某一个数据包没有按照顺序到达，接收端会一直保持连接等待数据包返回，这时候就会阻塞后续请求。这就发生了**TCP队头阻塞**。

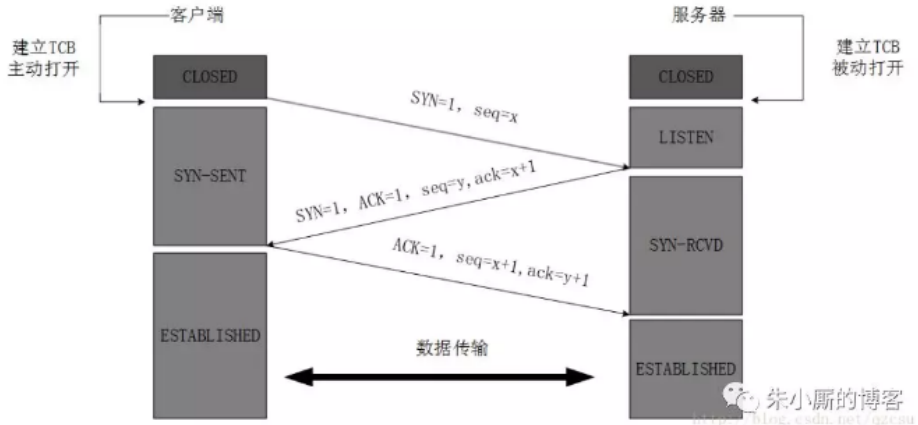
TCP

标志位

- URG：指示报文中有紧急数据，应尽快传送（相当于高优先级的数据）。
- PSH：为1表示是带有push标志的数据，指示接收方在接收到该报文段以后，应尽快将这个报文段交给应用程序，而不是在缓冲区排队。
- RST：TCP连接中出现严重差错（如主机崩溃），必须释放连接，在重新建立连接。
- FIN：发送端已完成数据传输，请求释放连接。
- SYN：处于TCP连接建立过程。 (Synchronize Sequence Numbers)
- ACK：确认序号标志，为1时表示确认号有效，为0表示报文中不含确认信息，忽略确认号字段。

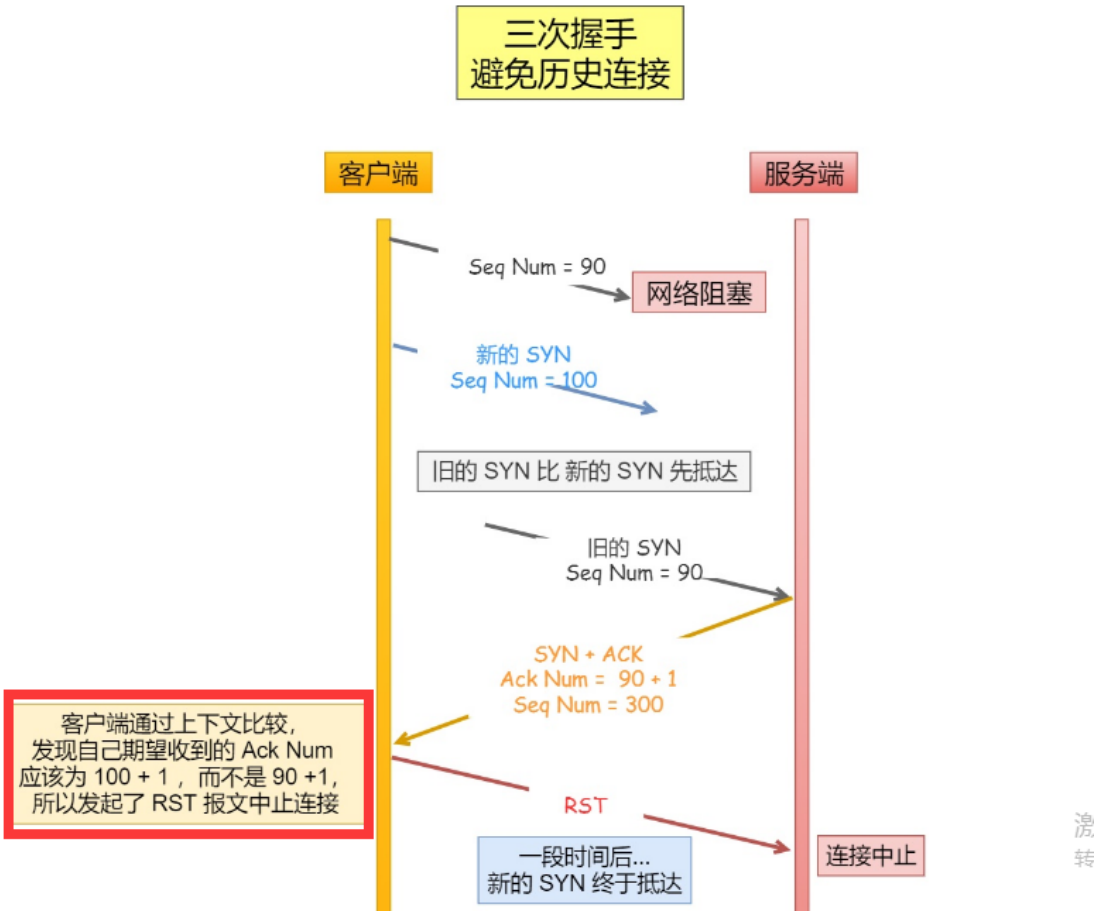
SYN标记用于建立连接的过程，表示希望建立TCP连接 FIN用于连接断开，表示数据已经传输完毕希望释放连接 ACK为确认序列号

三次握手：



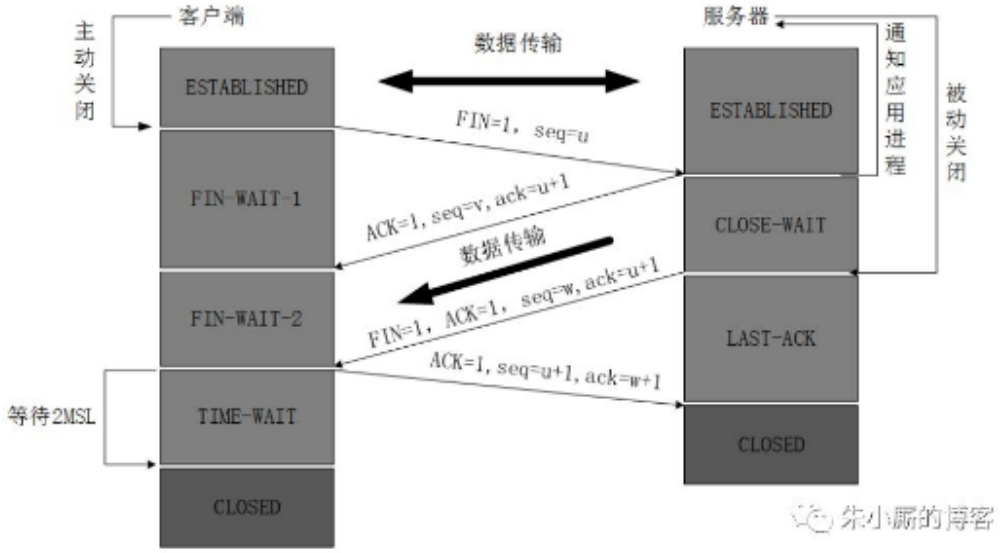
为何需要三次握手：

1、三次握手才可以阻止重复历史连接的初始化（主要原因）
2、三次握手才可以同步双方的初始序列号
3、三次握手才可以避免资源浪费



序列号在 TCP 连接中占据着非常重要的作用，所以当客户端发送携带「初始序列号」的 SYN 报文的时候，需要服务端回一个 ACK 应答报文，表示客户端的 SYN 报文已被服务端成功接收，那当服务端发送「初始序列号」给客户端的时候，依然也要得到客户端的应答回应，这样一来一回，才能确保双方的初始序列号能被可靠的同步。

四次断开



`MSL` 是 Maximum Segment Lifetime，**报文最大生存时间**，它是任何报文在网络上存在的最长时间，超过这个时间报文将被丢弃。`2MSL` 的时间是从**客户端接收到 FIN 后发送 ACK 开始计时的。**如果在 TIME-WAIT 时间内，因为客户端的 ACK 没有传输到服务端，客户端又接收到了服务端重发的 FIN 报文，那么 **2MSL 时间将重新计时。**TIME-WAIT保证「被动关闭连接」的一方能被正确的关闭，即保证最后的 ACK 能让被动关闭方接收，从而帮助其正常关闭；

TCP最大连接

> "TCP连接四元组是源IP地址、源端口、目的IP地址和目的端口。任意一个元素发生了改变，那么就代表的是一条完全不同的连接了。拿我的Nginx举例，它的端口是固定使用80。另外我的IP也是固定的，这样目的IP地址、目的端口都是固定的。剩下源IP地址、源端口是可变的。所以理论上我的Nginx上最多可以建立2的32次方（ip数）×2的16次方（port数）个连接。这是两百多万亿的一个大数

> "进程每打开一个文件（linux下一切皆文件，包括socket），都会消耗一定的内存资源。如果有不怀好心的人启动一个进程来无限的创建和打开新的文件，会让服务器崩溃。所以linux系统出于安全角度的考虑，在多个位置都限制了可打开的文件描述符的数量，包括系统级、用户级、进程级。这三个限制的含义和修改方式如下："

- 系统级：当前系统可打开的最大数量，通过fs.file-max参数可修改
- 用户级：指定用户可打开的最大数量，修改/etc/security/limits.conf
- 进程级：单个进程可打开的最大数量，通过fs.nr_open参数可修改