**TCP三次握手**

TCP A TCP B

​

1. CLOSED LISTEN

​

2. SYN-SENT --> <SEQ=100><CTL=SYN> ...

​

3. (duplicate) ... <SEQ=90><CTL=SYN> --> SYN-RECEIVED

​

4. SYN-SENT <-- <SEQ=300><ACK=91><CTL=SYN,ACK> <-- SYN-RECEIVED

​

5. SYN-SENT --> <SEQ=91><CTL=RST> --> LISTEN

​

6. ... <SEQ=100><CTL=SYN> --> SYN-RECEIVED

​

7. SYN-SENT <-- <SEQ=400><ACK=101><CTL=SYN,ACK> <-- SYN-RECEIVED

​

8. ESTABLISHED --> <SEQ=101><ACK=401><CTL=ACK> --> ESTABLISHED

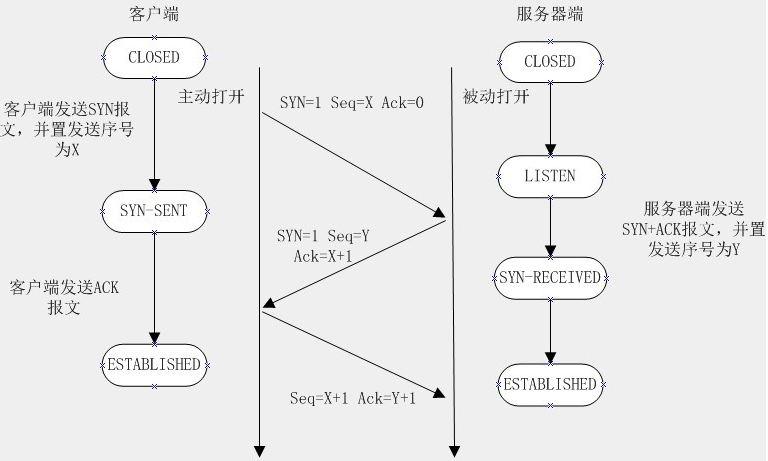
​

Recovery from Old Duplicate SYN

这是 复用连接时，旧在途包发往新连接中的例子。

* 3中，一个旧的重复的 SYN到达 B。
* 4中， B分别不出是否旧的，照样子正常回包。
* 5中，A检测到 B 返回的ACK不正确，所以返回 RST(reset)
* 6中，B接收到 RST(reset)信号，于是变成 LISTEN 状态。
* 7中，新连接正常的 SYN终于到达了，三次握手正常进行。

这种是简化的情况，但是可以看出 TCP 是如何处理复用旧链接的包到达的



TCP四次挥手

 第一次挥手(FIN=1，seq=x)

假设客户端想要关闭连接，客户端发送一个 FIN 标志位置为1的包，表示自己已经没有数据可以发送了，但是仍然可以接受数据。

发送完毕后，客户端进入 FIN\_WAIT\_1 状态。

 第二次挥手(ACK=1，ACKnum=x+1)

服务器端确认客户端的 FIN 包，发送一个确认包，表明自己接受到了客户端关闭连接的请求，但还没有准备好关闭连接。

发送完毕后，服务器端进入 CLOSE\_WAIT 状态，客户端接收到这个确认包之后，进入 FIN\_WAIT\_2 状态，等待服务器端关闭连接。

 第三次挥手(FIN=1，seq=y)

服务器端准备好关闭连接时，向客户端发送结束连接请求，FIN 置为1。

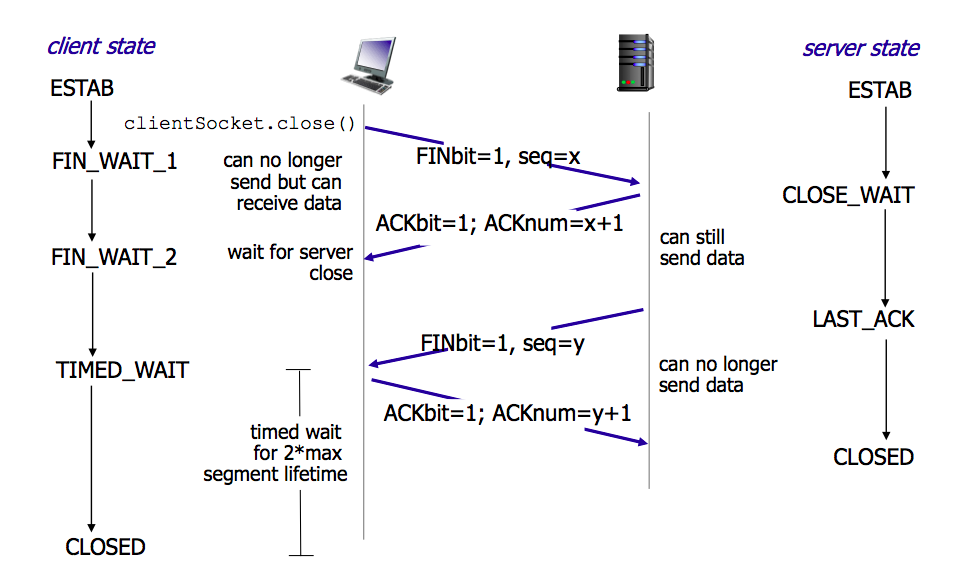
发送完毕后，服务器端进入 LAST\_ACK 状态，等待来自客户端的最后一个ACK。

 第四次挥手(ACK=1，ACKnum=y+1)

客户端接收到来自服务器端的关闭请求，发送一个确认包，**并进入 TIME\_WAIT状态，等待可能出现的要求重传的 ACK 包。**

服务器端接收到这个确认包之后，关闭连接，进入 CLOSED 状态。

客户端等待了某个固定时间（两个最大段生命周期，2MSL，2 Maximum Segment Lifetime）之后，没有收到服务器端的 ACK ，认为服务器端已经正常关闭连接，于是自己也关闭连接，进入 CLOSED 状态



**为什么TCP链接需要三次握手，两次不可以么，为什么？**

**为了防止 已失效的链接请求报文突然又传送到了服务端，因而产生错误。**

　　客户端发出的连接请求报文并未丢失，而是在某个网络节点长时间滞留了，以致延误到链接释放以后的某个时间才到达Server。这是，Server误以为这是Client发出的一个新的链接请求，于是就向客户端发送确认数据包，同意建立链接。若不采用“三次握手”，那么只要Server发出确认数据包，新的链接就建立了。由于client此时并未发出建立链接的请求，所以其不会理睬Server的确认，也不与Server通信；而这时Server一直在等待Client的请求，这样Server就白白浪费了一定的资源。若采用“三次握手”，在这种情况下，由于Server端没有收到来自客户端的确认，则就会知道Client并没有要求建立请求，就不会建立链接。

**TCP和HTTP**

TCP是底层通讯协议，定义的是数据传输和连接方式的规范；  
HTTP是应用层协议，定义的是传输数据的内容的规范；  
HTTP协议中的数据是利用TCP协议传输的，所以支持HTTP也就一定支持TCP ；

**socket**

Socket是应用层与TCP/IP协议族通信的中间软件抽象层，它是一组接口。在设计模式中，Socket其实就是一个门面模式，它把复杂的TCP/IP协议族隐藏在Socket接口后面，对用户来说，一组简单的接口就是全部，让Socket去组织数据，以符合指定的协议。

**http https区别和联系**

Http协议运行在TCP之上，明文传输，客户端与服务器端都无法验证对方的身份；Https是身披SSL(Secure Socket Layer)外壳的Http，运行于SSL上，SSL运行于TCP之上，是添加了加密和认证机制的HTTP。二者之间存在如下不同：

* 端口不同：Http与Http使用不同的连接方式，用的端口也不一样，前者是80，后者是443；
* 资源消耗：和HTTP通信相比，Https通信会由于加减密处理消耗更多的CPU和内存资源；
* 开销：Https通信需要证书，而证书一般需要向认证机构购买；

Https的加密机制是一种共享密钥加密和公开密钥加密并用的混合加密机制。

**对称加密与非对称加密**

　　对称密钥加密是指加密和解密使用同一个密钥的方式，这种方式存在的最大问题就是密钥发送问题，即如何安全地将密钥发给对方；而非对称加密是指使用一对非对称密钥，即公钥和私钥，公钥可以随意发布，但私钥只有自己知道。发送密文的一方使用对方的公钥进行加密处理，对方接收到加密信息后，使用自己的私钥进行解密。

　　由于非对称加密的方式不需要发送用来解密的私钥，所以可以保证安全性；但是和对称加密比起来，它非常的慢，所以我们还是要用对称加密来传送消息，但对称加密所使用的密钥我们可以通过非对称加密的方式发送出去。

**HTTPS验证流程**

1）客户端发起一个http请求，连接到服务器的443端口。

2）服务端把自己的信息以数字证书的形式返回给客户端（证书内容有密钥公钥，网站地址，证书颁发机构，失效日期等）。证书中有一个公钥来加密信息，私钥由服务器持有。

3）验证证书的合法性

客户端收到服务器的响应后会先验证证书的合法性（证书中包含的地址与正在访问的地址是否一致，证书是否过期）。

4）生成随机密码（RSA签名）

如果验证通过，或用户接受了不受信任的证书，浏览器就会生成一个随机的对称密钥（session key）并用公钥加密，让服务端用私钥解密，解密后就用这个对称密钥进行传输了，并且能够说明服务端确实是私钥的持有者。

7 应用层 6 表示层 5 会话层 4 传输层 3 网络层 2 数据链路层 1 物理层

**TCP UDP区别**

1. TCP提供面向连接的传输，通信前要先建立连接（三次握手机制）； UDP提供无连接的传输，通信前不需要建立连接。
2. TCP只支持点对点通信，UDP支持一对一、一对多、多对一、多对多的通信模式；

3） TCP提供可靠的传输（有序，无差错，不丢失，不重复）； UDP提供不可靠的传输。

4） TCP面向字节流的传输，因此它能将信息分割成组，并在接收端将其重组； UDP是面向报文的，没有分组开销。UDP长度限制

5） TCP提供拥塞控制和流量控制机制； UDP不提供拥塞控制和流量控制机制。

**TCP和UDP分别对应的常见应用层协议**

1). TCP对应的应用层协议

FTP：定义了文件传输协议，使用21端口。常说某某计算机开了FTP服务便是启动了文件传输服务。下载文件，上传主页，都要用到FTP服务。

Telnet：它是一种用于远程登陆的端口，用户可以以自己的身份远程连接到计算机上，通过这种端口可以提供一种基于DOS模式下的通信服务。如以前的BBS是-纯字符界面的，支持BBS的服务器将23端口打开，对外提供服务。

SMTP：定义了简单邮件传送协议，现在很多邮件服务器都用的是这个协议，用于发送邮件。如常见的免费邮件服务中用的就是这个邮件服务端口，所以在电子邮件设置-中常看到有这么SMTP端口设置这个栏，服务器开放的是25号端口。

POP3：它是和SMTP对应，POP3用于接收邮件。通常情况下，POP3协议所用的是110端口。也是说，只要你有相应的使用POP3协议的程序（例如Fo-xmail或Outlook），就可以不以Web方式登陆进邮箱界面，直接用邮件程序就可以收到邮件（如是163邮箱就没有必要先进入网易网站，再进入自己的邮-箱来收信）。

HTTP：从Web服务器传输超文本到本地浏览器的传送协议。

2). UDP对应的应用层协议

DNS：用于域名解析服务，将域名地址转换为IP地址。DNS用的是53号端口。

SNMP：简单网络管理协议，使用161号端口，是用来管理网络设备的。由于网络设备很多，无连接的服务就体现出其优势。

TFTP(Trival File Transfer Protocal)：简单文件传输协议，该协议在熟知端口69上使用UDP服务

**TCP协议如何来保证传输的可靠性**

TCP提供一种面向连接的、可靠的字节流服务。其中，面向连接意味着两个使用TCP的应用（通常是一个客户和一个服务器）在彼此交换数据之前必须先建立一个TCP连接。在一个TCP连接中，仅有两方进行彼此通信；而字节流服务意味着两个应用程序通过TCP链接交换8bit字节构成的字节流，TCP不在字节流中插入记录标识符。

**对于可靠性，TCP通过以下方式进行保证：**

* **数据包校验**：目的是检测数据在传输过程中的任何变化，若校验出包有错，则丢弃报文段并且不给出响应，这时TCP发送数据端超时后会重发数据；
* **对失序数据包重排序**：既然TCP报文段作为IP数据报来传输，而IP数据报的到达可能会失序，因此TCP报文段的到达也可能会失序。TCP将对失序数据进行重新排序，然后才交给应用层；
* **丢弃重复数据**：对于重复数据，能够丢弃重复数据；
* **应答机制**：当TCP收到发自TCP连接另一端的数据，它将发送一个确认。这个确认不是立即发送，通常将推迟几分之一秒；
* **超时重发**：当TCP发出一个段后，它启动一个定时器，等待目的端确认收到这个报文段。如果不能及时收到一个确认，将重发这个报文段；
* **流量控制**：TCP连接的每一方都有固定大小的缓冲空间。TCP的接收端只允许另一端发送接收端缓冲区所能接纳的数据，这可以防止较快主机致使较慢主机的缓冲区溢出，这就是流量控制。TCP使用的流量控制协议是可变大小的滑动窗口协议。

**滑动窗口TCP:**

对ACK的再认识，ack通常被理解为收到数据后给出的一个确认ACK，ACK包含两个非常重要的信息：  
**一是期望接收到的下一字节的序号n，该n代表接收方已经接收到了前n-1字节数据，此时如果接收方收到第n+1字节数据而不是第n字节数据，接收方是不会发送序号为n+2的ACK的。**举个例子，假如接收端收到1-1024字节，它会发送一个确认号为1025的ACK,但是接下来收到的是2049-3072，它是不会发送确认号为3072的ACK,而依旧发送1025的ACK。  
  
**二是当前的窗口大小m，如此发送方在接收到ACK包含的这两个数据后就可以计算出还可以发送多少字节的数据给对方，假定当前发送方已发送到第x字节，则可以发送的字节数就是y=m-(x-n).这就是滑动窗口控制流量的基本原理**

**TCP慢启动：**来避免发送过多数据到网络中而导致网络拥塞

慢启动初始启动时设置拥塞窗口值（cwnd）为1、2、4或10个MSS。[[7]](https://zh.wikipedia.org/wiki/TCP%E6%8B%A5%E5%A1%9E%E6%8E%A7%E5%88%B6" \l "cite_note-9)[[8]](https://zh.wikipedia.org/wiki/TCP%E6%8B%A5%E5%A1%9E%E6%8E%A7%E5%88%B6#cite_note-RFC_3390-10):1拥塞窗口在每接收到一个确认包时增加，每个RTT内成倍增加，当然实际上并不完全是指数增长，因为接收方会延迟发送确认，通常是每接收两个分段则发送一次确认包。[[9]](https://zh.wikipedia.org/wiki/TCP%E6%8B%A5%E5%A1%9E%E6%8E%A7%E5%88%B6#cite_note-11)发送速率随着慢启动的进行而增加，直到遇到出现丢失、达到慢启动阈值（ssthresh）、或者接收方的接收窗口进行限制。

当达到慢启动阈值（ssthresh）时，慢启动算法就会转换为线性增长的阶段，算法控制每个RTT内拥塞窗口只增加1个分段量。虽然称为“慢启动”，但实际上比拥塞控制阶段的窗口增加更为激进

**当发生超时的时候，慢启动阈值降为超时前拥塞窗口的一半大小、拥塞窗口会降为1个MSS（数据最大字节数），并且重新回到慢启动阶段。**

**TCP快速重传**

是对TCP发送方降低等待重发丢失分段用时的一种改进。

**超时重传：**TCP发送方每发送一个分段都会启动一个超时计时器，如果没能在特定时间内接收到相应分段的确认，发送方就假设这个分段在网络上丢失了，需要重发。这也是 TCP 用来估计 RTT 的测量方法。

重复确认（duplicate cumulative acknowledgements，DupAcks）就是这个阶段的基础，其基于以下过程：如果接收方接收到一个数据分段，就会将该分段的序列号加上数据字节长的值，作为分段确认的确认号，发送回发送方，表示期望发送方发送下一个序列号的分段。但是如果接收方提前收到更下一个序列号的分段——或者说接收到无序到达的分段，即之前期望确认号对应的分段出现接收丢失——接收方需要立即使用之前的确认号发送分段确认。此时如果发送方收到**接收方相同确认号的分段**确认超过1次，并且该对应序列号的分段超时计时器仍没超时的话，则这就是出现重复确认，需要进入快速重传。

快送重传就是基于以下机制：如果假设重复阈值为3，当发送方收到4次相同确认号的分段确认（第1次收到确认期望序列号，加3次重复的期望序列号确认）时，则可以认为继续发送更高序列号的分段将会被接受方丢弃，而且会无法有序送达。发送方应该忽略超时计时器的等待重发，立即重发重复分段确认中确认号对应序列号的分段。

**TCP介绍：**

TCP 负责应用软件（比如你的浏览器）和网络软件之间的通信。IP 负责计算机之间的通信。**TCP 负责将数据分割并装入 IP 包，IP 负责将包发送至接受者**，传输过程要经IP路由器负责根据通信量、网络中的错误或者其他参数来进行正确地寻址，然后在它们到达的时候重新组合它们。

**客户端不断进行请求链接会怎样？**

**DDos(Distributed Denial of Service)攻击**

服务器端会为每个请求创建一个链接，并向其发送确认报文，然后等待客户端进行确认

DDos 攻击

* 客户端向服务端发送请求链接数据包
* 服务端向客户端发送确认数据包
* 客户端不向服务端发送确认数据包，服务器一直等待来自客户端的确认

DDos 预防 **( 没有彻底根治的办法，除非不使用TCP )**

* 限制同时打开SYN半链接的数目
* 缩短SYN半链接的Time out 时间
* 关闭不必要的服务

**IP协议（网际协议）:计算机之间的通信**

IP协议是计算机用来相互识别的通信的一种机制，每台计算机都有一个IP.用来在internet上标识这台计算机。 **IP 负责在因特网上发送和接收数据包。** 通过 IP，消息（或者其他数据）被分割为小的独立的包，并通过因特网在计算机之间传送。**IP 负责将每个包路由至它的目的地**

**DNS:** 域名解析

**浏览器输入URL到返回页面全过程**

* 1.根据域名，进行DNS域名解析；
* 2.拿到解析的IP地址，建立TCP连接；
* 3.向IP地址，发送HTTP请求；
* 4.服务器处理请求；
* 5.返回响应结果；
* 6.关闭TCP连接；
* 7.浏览器解析HTML；
* 8.浏览器布局渲染；

大致就是:浏览器输入地址，然后浏览器这个进程去调操作系统某个库里的gethostbyname函数(例如，Linux GNU glibc标准库的gethostbyname函数)，然后呢这个函数通过网卡给DNS服务器发UDP请求，接收结果，然后将结果给返回给浏览器。

这张图其实已经讲明白大致的流程，但是细节上可能有些差异。 例如

* (1)我们在用chrome浏览器的时候，其实会先去浏览器的dns缓存里头查询，dns缓存中没有，再去调用gethostbyname函数
* (2)gethostbyname函数在试图进行DNS解析之前首先检查域名是否在本地 Hosts 里，如果没找到再去DNS服务器上查

**DNS怎么做域名解析**

查询www.163.com的DNS请求到达本地DNS服务器之后，本地DNS服务器会首先查询它的**缓存记录**，如果缓存中有此条记录，就可以直接返回结果。如果没有，本地DNS服务器还要向DNS根服务器进行查询。

**根DNS服务器**没有记录具体的域名和IP地址的对应关系，而是告诉本地DNS服务器，你可以到**域服务器**上去继续查询，并给出域服务器的地址。

本地DNS服务器继续向**域服务器**发出请求，在这个例子中，请求的对象是.com域服务器。.com域服务器收到请求之后，也不会直接返回域名和IP地址的对应关系，而是告诉本地DNS服务器，你的域名的**解析服务器**的地址。

最后，本地DNS服务器向域名的**解析服务器**发出请求，这时就能收到一个**域名和IP地址对应关系**，本地DNS服务器不仅要把IP地址返回给用户电脑，还要把这个对应关系保存在缓存中，以备下次别的用户查询时，可以直接返回结果，加快网络访问。

**HTTP响应状态**

* 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。

常见状态代码、状态描述的说明如下。

* 200 OK：客户端请求成功。
* 400 Bad Request：客户端请求有语法错误，不能被服务器所理解。
* 401 Unauthorized：请求未经授权，这个状态代码必须和WWW-Authenticate报头域一起使用。
* 403 Forbidden：服务器收到请求，但是拒绝提供服务。
* 404 Not Found：请求资源不存在，举个例子：输入了错误的URL。
* 500 Internal Server Error：服务器发生不可预期的错误。
* 503 Server Unavailable：服务器当前不能处理客户端的请求，一段时间后可能恢复正常，举个例子：HTTP/1.1 200 OK（CRLF）。
* 301 : 永久性转移   
  302 ：暂时性转移   
  304 ： 已缓存

**HTTP1.0,1.1,2.0**