计算机网络面试题手册

说一下七层网络模型

- 物理层
- 数据链路层
- 网络层
- 传输层
- 会话层
- 表示层
- 应用层

简述一下http和https

https 的 SSL加密是在传输层实现的

（1）http 和 https 的基本概念：

- http 是一种超文本传输协议，是互联网上应用最为广泛的一种网络协议，是一个客户端和服务器端请求和应答的标准（TCP），用于从WWW 服务器传输超文本到本地浏览器的传输协议，它可以使浏览器更加高效，是网络传输减少。
- https 是以安全为目标的 HTTP 通道，简单讲是 HTTP 的安全版，即 HTTP 下加入 SSL 层，HTTPS 的安全基础是 SSL，因此加密的详细内容就需要 SSL。http 协议的主要作用是建立一个信息安全通道，来确保数组的传输，确保网站的真实性

（2）http 和https 的区别：

http 传输的数据都是未加密的，也就是明文的，网景公司设置了 SSL 协议来对 http 协议传输的数据进行加密处理，简单来说 https 协议是由 http 和 ssl 协议构建的可进行加密传输和身份认证的网络协议，比 http 协议的安全性更高

主要的区别如下：

- https 协议需要 ca 证书，费用较高；而http协议不需要
- http 是超文本传输协议，信息是明文传输，https 则是具有安全性的 ssl 加密传输协议。
- 使用不同的链接方式，端口也不同，一般而言，http 协议的端口为 80，https 的端口为443
- http 的连接很简单，是无状态的；https 协议是由 SSL + HTTP 协议构建的可进行加密传输、身份认证的网络协议，比 http 协议安全

（3）https 协议的工作原理

客户端在使用 https 方式与 Web 服务器通信时有以下几个步骤：

- 客户使用 https url 访问服务器，则要求 web 服务器建立 ssl 链接
- web 服务器接收到客户端的请求之后，会将网站的证书（证书包含了公钥），返回或者说传输给客

户端

- 客户端和 web 服务器端开始协商 SSL 连接的安全等级，也就是加密等级
- 客户端浏览器通过双方协商一致的安全等级，建立会话密钥，然后通过网站的公钥来加密会话密钥，并传送给网站
- web 服务器通过自己的私钥解密出会话密钥
- web 服务器通过会话密钥加密与客户端之间的通信

(4) https 协议的优点：

- 使用 https 协议可认证用户和服务器，确保数据发送到正确的客户机和服务器
- https 协议是由 ssl + http 协议构建的可进行加密传输、身份认证的网络协议，要比 http 协议安全，可防止数据在传输过程中不被窃取、改变，确保数据的完整性
- https 是现行架构下最安全的解决方案，虽然不是绝对安全，但它大幅增加了中间人攻击的成本
- 谷歌曾在 2014 年 8 月份调整搜索引擎算法，并称“比起同等HTTP网站，采用HTTPS加密的网站在搜索结果中的排名将会更高”

(5) https 协议的缺点：

- https 握手阶段比较费时，会使页面加载时间延长 50%，增加 10%~20%的耗电
- https 缓存不如 http 高效，会增加数据开销
- SSL 证书也需要钱，功能越大的证书费用越高
- SSL 证书需要绑定 IP，不能在同一个 ip 上绑定多个域名，ipv4 资源支持不了这种消耗

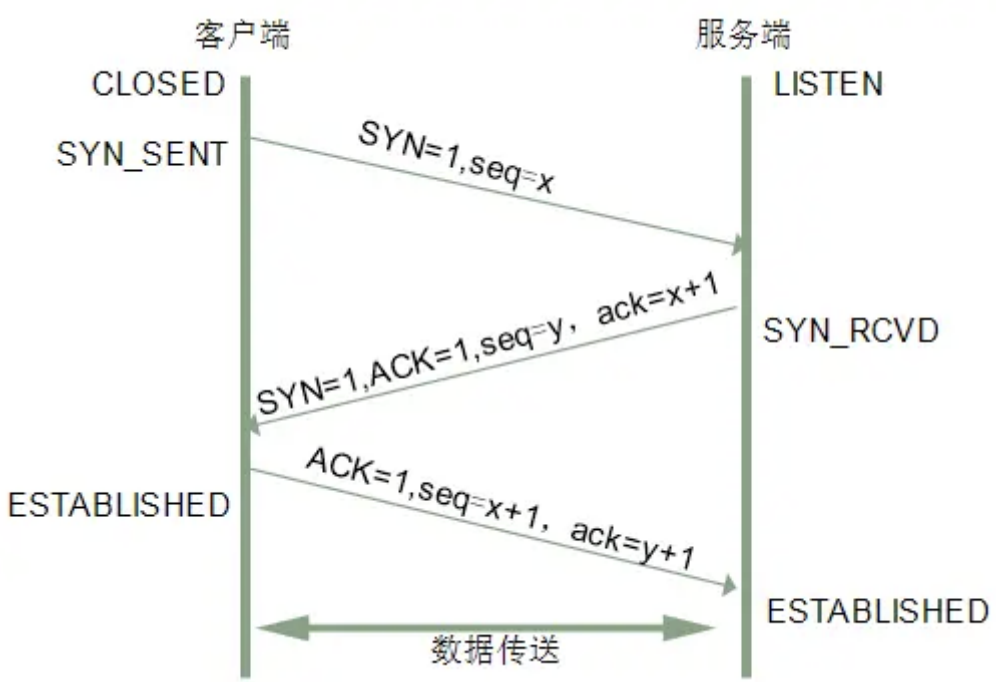
说一下三次握手

三次握手（Three-way Handshake）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。实质上其实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窗口大小的信息

刚开始客户端处于 Closed 的状态，服务端处于 Listen 状态。进行三次握手：

- 第一次握手：客户端给服务端发送一个 SYN 报文，并指明客户端的初始化序列号ISN（c），此时客户端处于 SYN_SEND 状态（首部的同步为SYN=1，初始序号seq=x，SYN= 1的报文不能携带数据，但要消耗一个序号）
- 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（c）。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 状态（在确认报文段中，SYN=1，ACK=1，确认号ack=x+1，初始序号seq=y）
- 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，双方已建立了连接（确认报文段ACK = 1，确认号ack = y + 1，序列号seq=x+1，ACK报文段可以携带数据，不携带数据则不消耗序号）



为什么需要三次握手？两次不行吗?

弄清这个问题，我们需要先弄明白三次握手的目的是什么，能不能只用两次握手来达到同样的目的。

- 第一次握手：客户端发送网络包，服务端收到了。这样服务端就能得出结论：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：服务端发包，客户端收到了。 这样客户端就能得出结论：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：客户端发包，服务端收到了。 这样服务端就能得出结论：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。

因此，需要三次握手才能确认双方的接收与发送能力是否正常

什么是半连接队列?

服务器第一次收到客户端的SYN之后，就会处于SYN_RCVD状态，此时双方还没有完全建立连接，服务器会把此种状态下请求连接放在一个队列里，我们把这种队列称之为半连接队列

当然还有一个全连接队列，就是已经完成三次握手，建立起连接的就会放在全连接队列中。如果队列满里就有可能出现丢包现象

这里补充一点关于SYN_ACK重传次数的问题：服务器发送完 SYN-ACK包，如果未收到客户确认包，服务器进行首次重传，等待一段时间仍未收到客户确认包，进行第二次重传。如果重传次数超过系统规定的最大重传次数，系统将该连接信息从半连接队列中删除。注意，每次重传等待的时间不一定相同，一般会是指数增长，例如间隔时间为1s、2s、4s

ISN（Initial Sequence Number）是固定的吗?

当一端为建立连接而发送它的SYN时，它为连接选择一个初始序号。ISN随时间而变化，因此每个连接都将具有不同的ISN。ISN可以看作是一个32比特的计数器，每4ms加1 。这样选择序号的目的在于防止在网络中被延迟的分组在以后又被传送，而导致某个连接的一方对它做错误的解释

三次握手的其中一个重要功能是客户端和服务端交换 ISN(Initial Sequence Number)，以便让对方知道接下来接收数据的时候如何按序列号组装数据。如果 ISN 是固定的，攻击者很容易猜出后续的确认号，因此 ISN 是动态生成的。

SYN攻击是什么?

服务器端的资源分配是在二次握手时分配的，而客户端的资源是在完成三次握手时分配的，所以服务器容易受到SYN洪泛攻击。SYN攻击就是Client在短时间内伪造大量不存在的IP地址，并向Server不断地发送SYN包，Server则回复确认包，并等待Client确认，由于源地址不存在，因此Server需要不断重发直至超时，这些伪造的SYN包将长时间占用未连接队列，导致正常的SYN请求因为队列满而被丢弃，从而引起网络拥塞甚至系统瘫痪。SYN 攻击是一种典型的 DoS/DDoS 攻击。

检测 SYN 攻击非常的方便，当你在服务器上看到大量的半连接状态时，特别是源IP地址是随机的，基本上可以断定这是一次SYN攻击。在 Linux/Unix 上可以使用系统自带的 netstats 命令来检测 SYN 攻击。

```
netstat -n -p TCP | grep SYN_RECV
```

常见的防御 SYN 攻击的方法有如下几种：

- 缩短超时（SYN Timeout）时间
- 增加最大半连接数
- 过滤网关防护
- SYN cookies 技术

什么是四次挥手?

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（也有将四次挥手叫做四次握手的）。这由TCP的半关闭（half-close）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受来自另一端数据的能力

TCP 的链接的拆除需要发送四个包，因此称为四次挥手（Four-way handsake），客户端或服务器均可主动发起挥手动作

刚开始双方都处于 ESTABLISHED 状态，假如是客户端先发起关闭请求。四次挥手的过程如下：

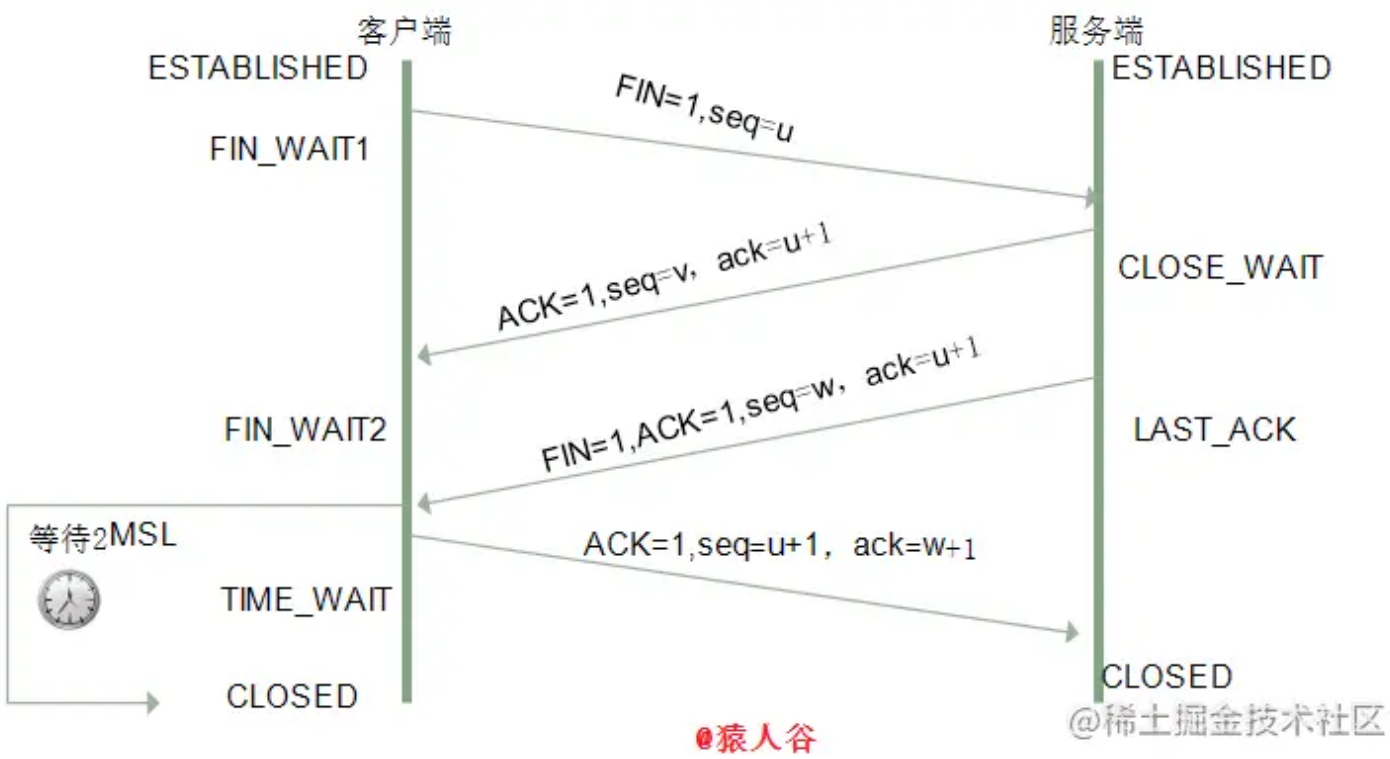
- 第一次挥手：客户端发送一个 FIN 报文，报文中会指定一个序列号。此时客户端处理 FIN_WAIT1

状态。即发出链接释放报文段（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，表明已经收到了客户端的报文了，此时服务端处于 CLOSE_WAIT 状态。即服务端收到了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 报文段作为应答，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链接释放报文段后，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

收到一个 FIN 只意味着在这一方向上没有数据流动。客户端执行主动关闭并进入TIME_WAIT 是正常的，服务端通常执行被动关闭，不会进入 TIME_WAIT 状态

在 socket 编程中，任何一方执行 close() 操作即可产生挥手操作



挥手为什么需要四次?

因为当服务端收到客户端的SYN连接请求报文后，可以直接发送SYN+ACK报文。其中ACK报文是用来应答的，SYN报文是用来同步的。但是关闭连接时，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

2MSL 等待状态

TIME_WAIT 状态也称为2MSL等待状态。每个具体TCP实现必须选择一个报文段最大生存时间MSL（Maximum Segment Lifetime），它是任何报文段被丢弃在网络内的最长时间。这个时间是有限的，因为TCP报文段以IP数据报在网络内传输，而IP数据报则有限制其生存时间的TTL字段。

对一个具体实现所给定的MSL值，处理的原则是：当TCP执行一个主动关闭，并发回最后一个ACK，该连接必须在TIME_WAIT 状态停留的时间为2倍的MSL。这样可让TCP再次发送最后得ACK以防这个ACK丢失（另一端超时并重发最后的FIN）

这种2MSL等待的另一个结果是这个TCP连接在2MSL等待期间，定义这个连接的插口（客户的IP地址和端口号，服务器的IP地址和端口号）不能再被使用。这个连接只能在2MSL结束后才能再被使用

四次挥手释放连接时，等待 2MSL 的意义?

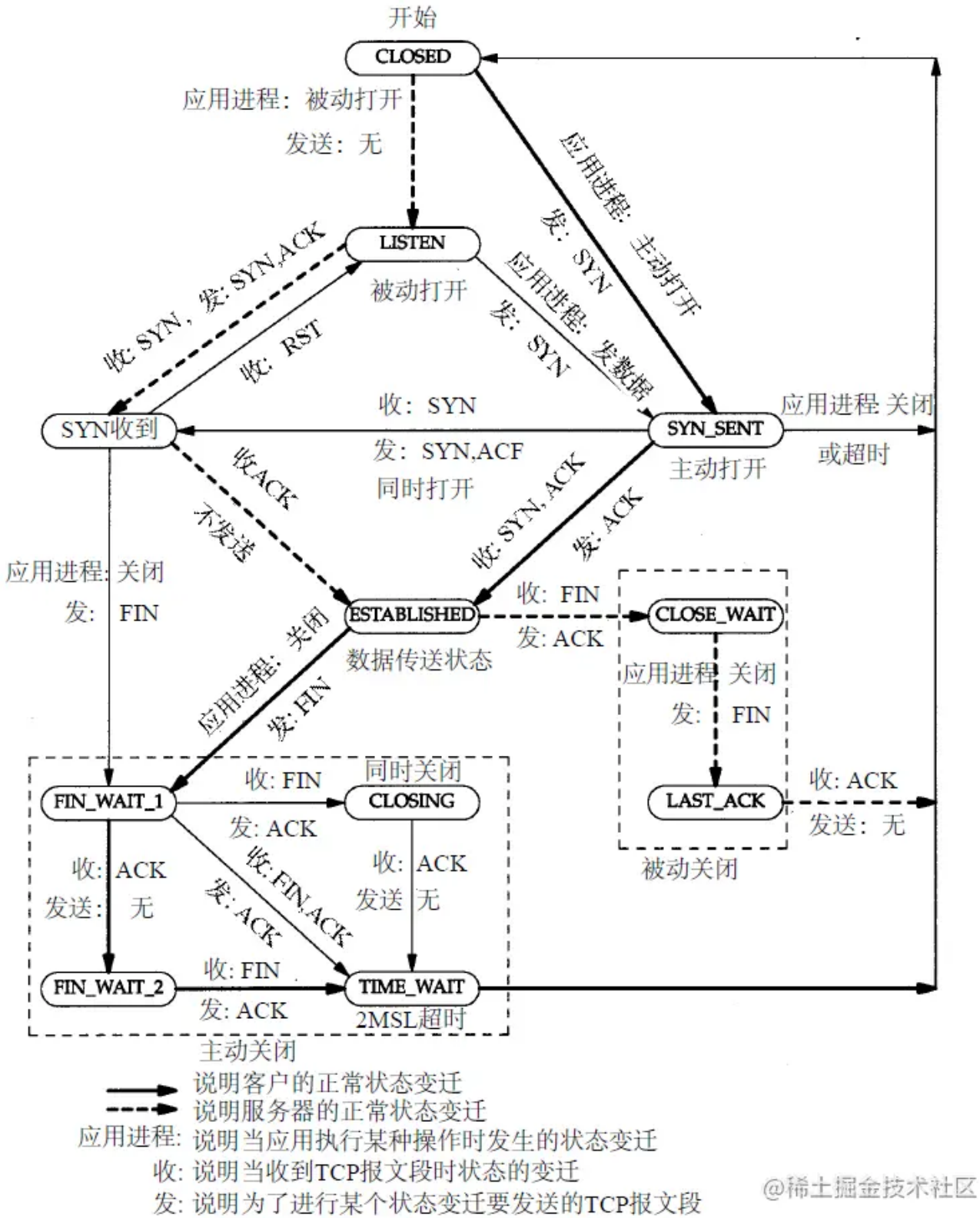
> MSL 是Maximum Segment Lifetime的英文缩写，可译为“最长报文段寿命”，它是任何报文在网络上存在的最长时间，超过这个时间报文将被丢弃

- 保证客户端发送的最后一个ACK报文段能够到达服务端。 这个ACK报文段有可能丢失，使得处于LAST-ACK状态的B收不到对已发送的FIN+ACK报文段的确认，服务端超时重传FIN+ACK报文段，而客户端能在2MSL时间内收到这个重传的FIN+ACK报文段，接着客户端重传一次确认，重新启动2MSL计时器，最后客户端和服务端都进入到CLOSED状态，若客户端在TIME-WAIT状态不等待一段时间，而是发送完ACK报文段后立即释放连接，则无法收到服务端重传的FIN+ACK报文段，所以不会再发送一次确认报文段，则服务端无法正常进入到CLOSED状态。
- 防止“已失效的连接请求报文段”出现在本连接中。客户端在发送完最后一个ACK报文段后，再经过2MSL，就可以使本连接持续的时间内所产生的所有报文段都从网络中消失，使下一个新的连接中不会出现这种旧的连接请求报文段。

为什么TIME_WAIT状态需要经过2MSL才能返回到CLOSE状态?

理论上，四个报文都发送完毕，就可以直接进入CLOSE状态了，但是可能网络是不可靠的，有可能最后一个ACK丢失。所以**TIME_WAIT状态就是用来重发可能丢失的ACK报文。**

TCP状态变迁图



TCP和UDP的区别

（1）TCP是面向链接的；UDP是无连接的，即发送数据前不需要先建立连接

（2）TCP提供可靠的服务，也就是说，通过TCP连接传送的数据，无差错，不丢失，不重复，且按序到达；UDP尽最大努力交付，即不保证可靠交付。因为TCP提供可靠的服务，面向连接，不会丢失数

据，因此适合大数据量的交换

（3）TCP是面向字节流；UDP是面向报文，并且网络不会出现拥塞，不会使得发送速率降低（因此可能出现丢包，对实时的应用比如IP电话和视频会议等）

（4）TCP只能是1对1；UDP支持1对1，1对多

（5）TCP的首部较大为20字节，而UDP只有8字节

（6）TCP是面向连接的可靠性传输，而UDP是不可靠的

	UDP	TCP
是否连接	无连接	面向连接
是否可靠	不可靠传输、不使用流量控制和拥塞控制	可靠传输（数据顺序和正确性），使用流量控制和拥塞控制
连接对象个数	支持一对一、一对多、多对一和多对多交互通信	只能是一对一通信
传输方式	面向报文	面向字节流
首部开销	首部开销小，仅8字节	首部最小20字节，最大60字节
适用场景	使用于实时应用，例如视频会议、直播	适用于要求可靠传输的应用，例如文件传输

TCP有哪些手段保证可靠交付?

参考 TCP协议如何保证可靠传输

- 应用数据被分割成TCP认为最适合发送的数据块
- TCP 给发送的每一个包进行编号，接收方对数据包进行排序，把有序数据传送到应用层
- 检验和：TCP 将保持它首部和数据的校验和。这是一个端到端的检验和，目的是检查数据在传输过程中的任何变化。如果收到段的校验和有差错，TCP将丢弃这个报文段和不确认收到次报文段
- TCP的接收端会丢弃重复的数据
- 流量控制：TCP 链接的每一方都有固定大小的缓冲空间，TCP 的接收端只允许发送端发送接收端缓冲区能接纳的数据。当接收方来不及处理发送方的数据，能提示发送方降低发送的速率，防止包丢失。TCP 使用的流量控制协议是可变大小的滑动窗口协议（TCP 利用滑动窗口实现流量控制）
- 拥塞控制：当网络拥塞时，减少数据的发送
- ARQ协议：也是为了实现可靠传输，它的基本原理就是每发完一个分组就停止发送，等待对方确认。在收到确认后再发下一个分组
- 超时重传：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

Http缓存

参考 [HTTP强缓存和协商缓存]

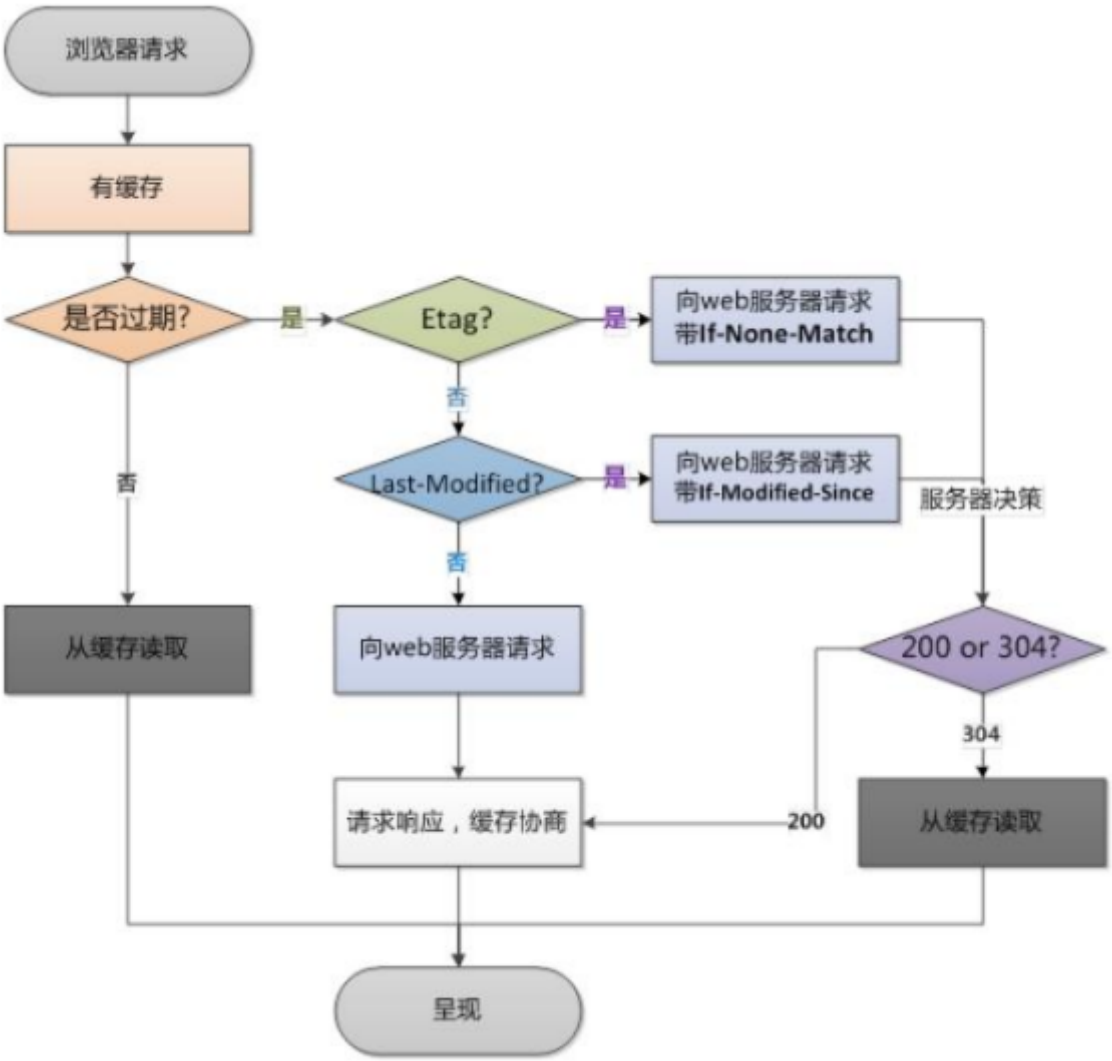
缓存分为两种：强缓存和协商缓存，根据响应的 header 内容来决定

普通刷新会启用协商缓存，忽略强缓存。只有在地址栏或收藏夹输入网址、通过链接引用资源等情况下，浏览器才会启用强缓存

缓存方式	获取资源方式	状态码	发送请求到服务器
强缓存	从缓存取	200（from cache）	否，直接从缓存取
协商缓存	从缓存取	304（not modified）	是，通过服务器来告知缓存是否可用

强缓存相关字段有 expires、cache-control。如果 cache-control 与 expires 同时存在的话，cache-control 的优先级高于 expires

协商缓存的相关字段有 Last-Modified/If-Modified-Since，Etag/If-None-Match，其中 Etag/If-None-match 的优先级比 Last-Modified/If-Mdified-Since 高



缓存位置：

缓存位置一般分为：Memory Cache（内存缓存）和 Disk Cache（硬盘缓存）

内存缓存：读取快、持续时间段、容量小

硬盘缓存：读取慢、持续时间长、容量大

缓存优先级：

Service Worker -> Memory Cache -> Disk Cache -> Push Cache

什么文件用强缓存？什么文件用协商缓存？

index.html 文件采用协商缓存，用户每次请求 index.html 不拿浏览器缓存，直接请求服务器，这样就保证资源更新了，用户能马上访问到新资源，如果服务端返回304，这时候再拿浏览器的缓存的index.htm，切记不要设置强缓存

其他资源采用强缓存+协商缓存

HTTP2 有什么新特性？

- **内容安全。**因为 HTTP2 是基于 https 的，天然具有安全特性，通过 http2.0 的特性可以避免单纯使用 https 的性能下降
- **多路复用**，无需多个TCP连接，因为其允许在单一的HTTP2连接上发起多重请求，因此可以不用依赖建立多个TCP连接。这个功能相当于是长链接的增强，每个 request 请求可以随机地混杂在一起，接收方可以根据 request 的 id 将 request 再归属到各自不同的服务端请求里面，另外多路复用中也支持了流的优先级，允许客户端告诉服务器哪些内容是优先级更高的资源，可以优先传输
- **二进制分帧**，将所有要传输的消息采用二进制编码，并且会将信息分割为更小的消息块。http 1.x 的解析是基于文本的，http2.0 将所有的传输信息分割为更小的消息和帧，并对他们采用二进制格式编码，基于二进制可以让协议有更多的扩展性，比如引入了帧来传输数据和指令
- **头部压缩**，用HPACK技术压缩头部，减小报文大小
- **服务端推送**，服务端可以在客户端发起请求前发送数据，换句话说，服务端可以对客户端的一个请求发送多个响应，并且资源可以正常缓存

注意：使用 http2 的前提必须是 https

http中的 keep-alive 有什么作用？

响应头中设置 keep-alive 可以在一个 TCP 链接上发送多个 http 请求

GET 和 POST 的区别？

- GET参数通过url传递；POST的参数是放在request body中
- GET 请求在url中传递的参数是有长度限制的；而POST没有
- GET 比 POST 更不安全，因为参数直接暴露在 url 中，所以不能用来传递敏感信息

- GET请求只能进行 url 编码，而 POST 支持多种编码方式
- GET 请求参数会被完整地保留在浏览历史记录里；而POST中的参数不会被保留
- GET 一般都用于从指定的资源请求数据（可以不用每次都与数据库连接，所以可以使用缓存）；POST 一般用于向指定的资源提交要被处理的数据
- GET 产生了一个 TCP 数据包；POST 产生两个 TCP 数据包
- 对于 GET 请求，浏览器会把 http header 和 data 一并发送出去，服务器响应 200；而对于 POST，浏览器先发送 header，服务器响应 100 continue，浏览器再发送 data，服务器响应 200 ok（返回数据）

GET 和 POST 本质上就是 TCP 链接，并无差别。但是由于 HTTP 的规定和浏览器/服务器的限制，导致他们在应用过程中体现出一些不同

备注：

我们经常说GET请求参数的大小是有限制的，而POST请求的参数大小是没有限制的。实际上HTTP协议从未规定 GET/POST 的请求长度限制是多少，对 GET 请求参数的限制是来源于浏览器或Web服务，浏览器或Web服务器限制了url 的长度，不同的浏览器和 WEB 服务器，限制的最大长度不一样，要支持 IE，则最大长度为 2083byte，若只支持 Chrome，则最大长度 8181byte

HTTP的状态码

状态码	英文名称	作用
100	Continue	继续，客户端应继续其请求
101	Switching Protocols	切换协议，服务器根据客户端的请求切换协议，只能切换到更高级的协议，例如：切换到 HTTP 的新版本协议
200	OK	请求成功，一般用于 GET 和 POST 请求
201	Created	已创建，成功请求并创建了新的资源
202	Accepted	已接受，已经接受请求，但未处理完成
203	Non-Authoritative Information	非授权信息，请求成功，但返回的meta信息不在原始的服务器，而是一个副本
204	No Content	无内容，服务器成功处理，但未返回内容，在未更新网页的情况下，可确保浏览器继续显示当前文档
205	Reset Content	重置内容。服务器处理成功，用户终端（例如：浏览器）应重置文档视

		图。可通过此返回码清楚浏览器的表单域
206	Patial Content	部分内容，服务器成功处理了部分GET请求
300	Multiple Choices	多种选择，请求的资源可包括多个位置，相应可返回一个资源特征与地址的列表用于用户终端（例如：浏览器）选择
301	Found	永久移动。请求的资源已被永久的移动到新URI，返回信息会包括新的URI，浏览器会自动定向到新 URI。今后任何新的请求都应使用新的 URI 代替
302	Found	临时移动。与301类似，但资源只是临时被移动。客户端应继续使用原有URI
303	See Other	查看其它地址。与301类似，使用GET 和POST请求查看
304	Not Modified	未修改。所请求的资源未修改，服务器返回此状态码时，不会返回任何资源。客户端通常会缓存访问过的资源，通过提供给一个头信息指出客户端希望只返回在指定日期之后修改的资源
305	Use Proxy	使用代理，所请求的资源必须通过代理访问
306	Unused	已经被废弃的HTTP状态码
307	Temporary Redirect	临时重定向。与302类似。使用GET请求重定向
400	Bad Request	客户端请求的语法错误，服务器无法理解
401	Unauthorized	请求要求用户的身份认证
402	Payment Required	保留，将来使用
403	Forbidden	服务器理解请求客户端的请求，但是拒绝执行此请求
404	Not Found	服务器无法根据客户端的请求找到资源（网页）。通过此代码，网站设计人员可设置“您所请求的资源无法找到”的个性页面
405	Method Not Allowed	客户端请求中的方法被禁止
406	Not Acceptable	服务器无法根据客户端请求的内容特性完成请求
407	Proxy Authentication Required	请求要求代理的身份认证，与401类

		似，但请求者应当使用代理进行授权
408	Request Time-out	服务器等待客户端发送的请求时间过长，超时
409	Conflict	服务器完成客户端的PUT请求时可能返回此代码，服务器处理请求时发生了冲突
401	Gone	客户端请求的资源已经不存在。410 不同于 404，如果资源以前有现在被永久删除了可使用 410 代码，网站设计人员可通过 301 代码指定资源的新位置
411	Length Required	服务器无法处理客户端发送的不带 Content-Length 的请求信息
412	Precondition Failed	客户端请求信息的先决条件错误
413	Request Entity Too Large	由于请求的实体过大，服务器无法处理，因此拒绝请求。为防止客户端的连续请求，服务器可能会关闭连接。如果只是服务器暂时无法处理，则会包含一个 Retry-After 的响应信息
414	Request-URI Too Large	请求的 URI 过长(URI 通常为网址)，服务器无法处理
415	Unsupported Media Type	服务器无法处理请求附带的媒体格式
416	Requested range not satisfiable	客户端请求的范围无效
417	Expectation Failed	服务器无法满足 Expect 的请求头信息
500	Internal Server Error	服务器内部错误，无法完成请求
501	Not Implemented	服务器不支持请求的功能，无法完成请求
502	Bad Gateway	作为网关或者代理工作的服务器尝试执行请求时，从远程服务器接收到了一个无效的响应

HTTP 常用请求头

协议头	说明
Accept	可接受的相应内容类型（Content-Types）
Accept-Charset	可接受的字符集
Accept-Encoding	可接受的响应内容的编码方式

Accept-Language	可接受的响应内容语言列表
Accept-Datetime	可接受的按照时间来表示的响应内容版本
Authorization	用于表示 HTTP 协议中需要认证资源的认证信息
Cache-Control	用来指定当前的请求/回复中的是否使用缓存机制
Connection	客户端（浏览器）想要优先使用的连接类型
Cookie	由之前服务器通过 Set-Cookie 设置的一个 HTTP 协议 Cookie
Content-Length	以 8 进制表示的请求体的长度
Content-MD5	请求体的内容的二进制 MD5 散列值（数字签名），以 Base64 编码的结果
Content-Type	请求体的 MIME 类型（用于 POST 和 PUT 请求中）
Date	发送该消息的日期和时间
Expect	表示客户端要求服务器作出特定的行为
From	发起此请求的用户的邮件地址
Host	表示服务器的域名以及服务器所监听的端口号。如果所请求的端口是对应的服务的标准端口（80），则端口号可以省略
If-Match	仅当客户端提供的实体与服务器上对应的实体相匹配时，才进行对应的操作，主要用于像 PUT 这样的方法中，仅当从用户上次更新某个资源后，该资源未被修改的情况下，才更新该资源
If-Modified-Since	允许在对应的资源未被修改的情况下返回304未修改
If-None-Match	允许在对应的内容未被修改的情况下返回304未修改（304 Not Modified）
If-Range	如果该实体未被修改过，则向返回所缺少的那一个或多个部分，否则，返回整个新的实体
If-Unmodified-Since	仅当该实体自某个特定时间以来未被修改的情况下，才发送回应
Max-Forwards	限制该消息可被代理及网关转发的次数
Origin	发起一个针对跨域资源共享的请求（该请求要求服务器在响应中加入一个 Access-Control-Allow-Origin 的消息头，表示访问控制所允许的来源）

Pragma	与具体的实现有关，这些字段可能在请求/回应链中的任何时候产生
Proxy-Authorization	用于向代理进行认证的认证信息
Range	表示请求某个实体的一部分，字节偏移以0开始
Referer	表示浏览器所访问的前一个页面，可以认为是之前访问页面的链接，将浏览器带到了当前页面
TE	浏览器预期接受的传输时的编码方式：可使用回应协议头 Transfer-Encoding 中的值（还可以使用“trailers”表示数据传输时的分块方式）用来表示浏览器希望在最后一个大小为 0 的块之后还接收到一些额外的字段
User-Agent	浏览器的身份标识字符串
Upgrade	要求服务器升级到一个高版本协议
Via	告诉服务器，这个请求是由哪些代理发出的
Warning	一个一般性的警告，表示在实体内容体中可能存在错误

什么是WebSocket?

WebSocket 是一个持久化的网络通信协议，可以在单个TCP连接上进行全双工通讯，没有了 Request 和 Response 的概念，两者地位完全平等，连接一旦建立，客户端和服务端之间实时可以进行双向数据传输

Websocket 和 HTTP 的关联和区别?

HTTP：

- HTTP 是非持久的协议，客户端想知道服务端的处理进度只能通过不停地使用 Ajax 进行轮询或者采用 long poll 的方式，但是前者对服务器的压力大，后者则会因为一直等待 Response 造成阻塞
- 虽然http1.1默认开启 Keep-alive 长连接保持了这个TCP通道使得在一个HTTP连接中，可以发送多个Request，接收多个Response，但是一个request只能有一个response。而且这个response也是被动的，不能主动发起
- websocket 虽然是一个独立于 HTTP 的一种协议，但是 websocket 必须依赖于 HTTP 协议进行一次握手（在握手阶段是一样的），握手成功后，数据就直接从TCP通道传输，与HTTP无关了。

WebSocket 握手

- WebSocket 握手请求报文：

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgradeg
Sec-WebSocket-Key: x3JJHMbDL1EzLkh9GBhXDw==
Sec-WebSocket-Protocol: chat, superchat
Sec-WebSocket-Version: 13
Origin: http://example.com
```

下面是与传统 HTTP 报文不同的地方：

```
Upgrade: websocket
Connection: Upgrade
```

表示发起的是WebSocket 协议

```
Sec-WebSocket-key: x3JJHMbDL1EzLkh9GBhXDw==  // 由浏览器随机生成的，验证是否可以进行 WebSocket 通信，防止恶意或者无意的连接
Sec-WebSocket-Protocol: Chat, superchat  // 用户自定义的字符串，用来标识服务所需要的协议
Sec-WebSocket-Version: 13  // 表示支持的 WebSocket 版本
```

- 服务器响应：

```
HTTP/1.1 101 Switching Protocols  // 101 表示要转换协议
Upgrade: websocket
Connection: Upgrade  // 表示升级新协议请求
Sec-WebSocket-Accept: HSmrc0sMlYUkAGmm5OPpG2HaGWk= // 是经过服务器确认，并且加密过后的 Sec-WebSocket-key。用来证明客户端和服务器之间能进行通信
Sec-WebSocket-Protocol: chat  // 表示最终使用的协议
```

WebSocket 状态?

WebSocket 对象中的 readyState 属性有四种状态：

- 0：表示正在连接
- 1：表示连接成功，可以通信了
- 2：表示连接正在关闭
- 3：表示连接已经关闭，或者打开连接失败

DNS 解析会出错吗？为什么？