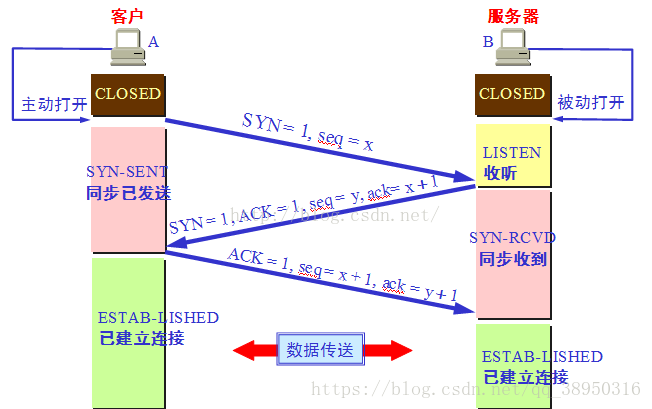
1. **什么是三次握手？**



**同步序列编号（Synchronize Sequence Numbers）**

第一次握手：client将SYN置为1，随机产生一个初始序列号seq发给server，进入SYN\_SENT状态；

第二次握手：server收到client的SYN=1之后，知道客户端请求建立连接，将自己的SYN置1，ACK置1，产生一个acknowledge number=sequence number+1,并随机产生一个自己的初始序列号，发送给客户端，进入SYN\_RCVD状态。

第三次握手：客户端检查ack是否为序列号+1，ACK是否为1，检查正确后将自己的ACK置为1，产生一个acknowledge number=服务器发的序列号+1，发给服务器；进入ESTABLISHED状态；服务器检查ACK为1和acknowledge number为序列号+1之后，也进入ESTABLISHED状态，完成三次握手，连接建立。

1. **TCP建立连接可以两次握手吗？为什么？**

不可以。原因有二：

1. 如果client发送的syn包在网络中长时间传输，但是并没有丢失，导致延误到连接释放之后的某个时间才到达，服务端做出响应之后，同意建立连接，但其实客户端并没有数据想要发送给server，因此不会理睬server的确认，这会导致server在空等。
2. 两次握手也无法保证client正确接收到第二次握手的报文。

**3.可以采用四次握手吗？**

可以，但是传输效率会降低。

**4．第三次握手中，如果客户端的ACK未送达服务器，会怎样？**

Server端：

因为没有收到ACK确认，因此会重发之前的SYN+ACK(默认发5次，之后自动关闭连接进入CLOSED状态)，client收到后会重新传ACK给server

Client端：两种情况

在server端进行超时重传的过程中，如果client向服务器发送数据，数据头部的ACK是为1的，所以服务器收到数据之后会读取ACK number，进入establish状态。

在Server进入CLOSED状态之后，如果client向服务器发送数据，服务器会以RST包应答。

RST包：reset报文是指TCP报头的标志字段中的reset位置一的报文

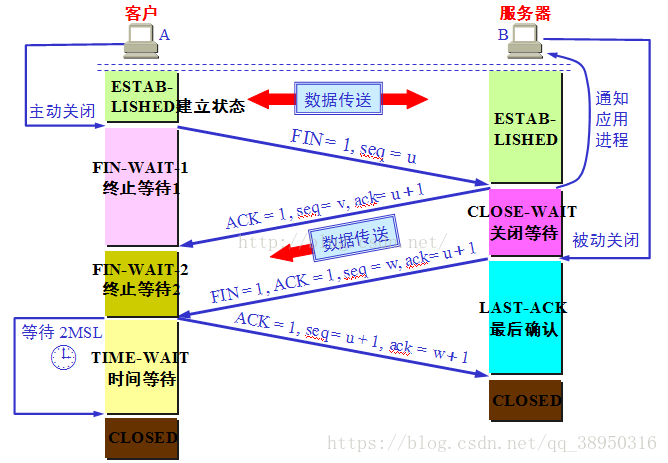
**5.如果连接已经建立，但是客户端出现了故障怎么办？**

服务器每次收到一次客户端的请求之后都会重新复位一个计时器，时间通常设置成2小时，若2小时还没有收到客户端的任何数据，服务器就会发送一个探测报文段，以后每隔75秒发送一次。若连续发送10个探测报文段仍然没有反应，服务器就会认为客户端出现了故障，接着就会关闭连接。

**6.初始序列号是什么？**

TCP连接的一方A,随机选择一个32位的序列号作为发送数据的初始序列号（假设为1000），把这个序列号发送给另一方B，以便在传输数据时，B确认什么样的数据编号是合法的，A如果收到了确认号为2001，就说明编号为1001-2000的数据已经被B成功接受。

**7.什么是四次挥手？**



第一次挥手：client将FIN置为1，发送一个序列号seq给server；进入FIN\_WAIT\_1状态。

第二次挥手：server收到FIN后，发送一个ACK=1，acknowledge number=序列号+1；进入CLOSE\_WAIT状态。此时客户端已经没有要发送的数据了，但是仍然可以接受来自服务器的数据。

第三次挥手：server将FIN置1，发送一个序列号给client;进入LAST\_ACK状态；

第四次挥手：client收到服务器的FIN后，进入TIME\_WAIT状态；接着将ACK置为1，发送一个acknowledge number=序列号+1给服务器；服务器收到后，确认acknowledge number后，变为CLOSED状态，不再向客户端发送数据。客户端等待2\*MSL的时间，也进入CLOSED状态，完成四次挥手。

**8.为什么不把服务器的ACK和FIN合并起来，变成三次挥手？**

因为服务器收到客户端断开连接的请求时，可能还有数据没有发送完毕，这时，先回复ACK，表示接收到了断开连接的请求。等到数据发送完毕之后在发送FIN,断开服务器到客户端的数据传送。

**9.如果第二次挥手时服务器的ACK没有送达客户端，会怎么样？**

客户端没有收到ACK确认，会重新发送FIN请求。

**10.客户端TIME\_WAIT状态的意义是什么？**

第四次挥手时，客户端发送给服务器的ACK可能丢失，TIME\_WAIT状态就是用来重发可能丢失的ACK报文的，如果Server没有收到ACK,就会重发FIN,如果client在2\*MSL的时间内收到了FIN，就会重新发送ACK并再次等待2MSL，防止server没有收到ACK而不断重发FIN

**11.TCP如何实现流量控制？**

使用滑动窗口协议实现流量控制，防止发送方发送速率太快，接收方缓存区不够导致溢出。接收方会维护一个接收窗口，接收窗口根据自己的资源情况动态调整。在返回ACK时将接收窗口大小放在TCP报文中的窗口字段中告知发送方。发送窗口的大小不能超过接受窗口的大小，只有当发送方发送并收到确认之后，才能将发送窗口右移。

发送窗口的上限为接受窗口和拥塞窗口的较小值。接受窗口表明了接收方的接收能力，拥塞窗口表明了网络的传送能力。

**12.什么是零窗口？**

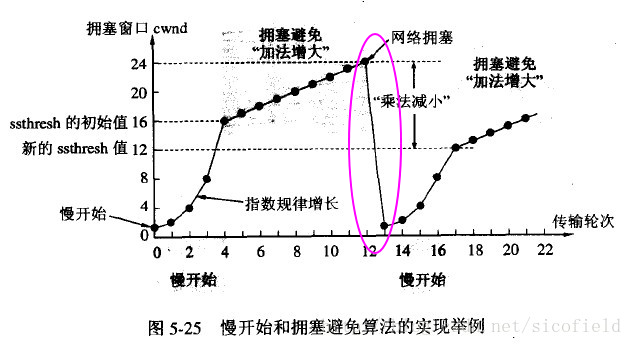
如果接收方没有能力接受数据，就会将接受窗口设置为0，这时发送方必须暂停发送数据，但是启动一个持续计时器，到期后发送一个大小为1字节的探测数据包，以查看接受窗口状态。如果接收方能够接受数据，就会在返回的报文中更新接受窗口的大小，恢复数据传送。

**13.TCP的拥塞控制是怎么实现的？**

拥塞控制主要是由四个算法组成：慢启动，拥塞避免，快重传，快恢复

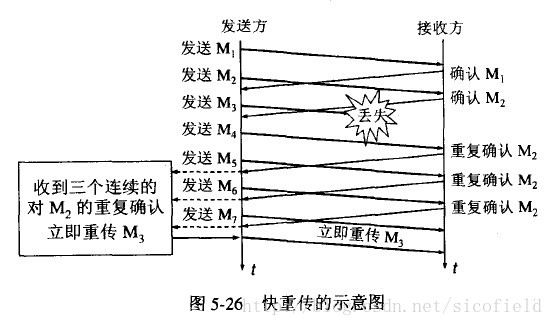
慢启动：刚开始发送数据时，先把拥塞窗口设置为一个最大报文段MSS的数值，每收到一个新的确认报文之后，就把拥塞窗口加一个MSS,这样每经过一个传输轮次，拥塞窗口都会加倍。

拥塞避免：当拥塞窗口的大小达到慢开始门限，开始执行拥塞避免算法，拥塞窗口的大小不再指数增加，而是线性增加。

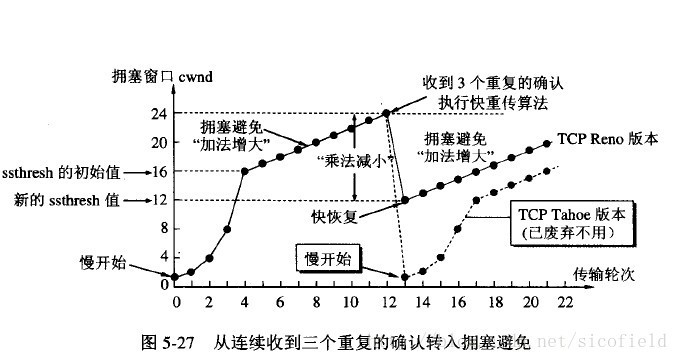


无论在慢开始阶段还是在拥塞避免阶段，只要发送方判断网络出现拥塞（其根据就是没有收到确认），就要把慢开始门限ssthresh设置为出现拥塞时的发送方窗口值的一半（但不能小于2）。然后把拥塞窗口cwnd重新设置为1，执行慢开始算法。**（这是不使用快重传的情况）**

快重传：快重传要求接收方在收到一个失序的报文段后就立即发出**重复确认**（为的是使发送方及早知道有报文段没有到达对方）而不要等到自己发送数据时捎带确认。快重传算法规定，发送方只要一连收到三个重复确认就应当立即重传对方尚未收到的报文段，而不必继续等待设置的重传计时器时间到期。



快恢复：当发送方连续收到三个重复确认时，就把慢开始门限减半，然后执行拥塞避免算法。不执行慢开始算法的原因：因为如果网络出现拥塞的话就不会收到好几个重复的确认，所以发送方认为现在网络可能没有出现拥塞。  
也有的快重传是把开始时的拥塞窗口cwnd值再增大一点，即等于 ssthresh + 3\*MSS 。这样做的理由是：既然发送方收到三个重复的确认，就表明有三个分组已经离开了网络。这三个分组不再消耗网络的资源而是停留在接收方的缓存中。可见现在网络中减少了三个分组。因此可以适当把拥塞窗口扩大些。



**14.TCP与UDP的区别**

1.tcp是面向连接的，udp是无连接的（无连接的意思是udp发送数据之前不需要建立连接）

2.tcp是可靠的，udp不可靠(udp接收方收到报文后，不需要给出任何确认)

3.tcp只支持点对点通信，udp支持一对一，一对多，多对一，多对多

4.tcp是面向字节流的，udp是面向报文的；面向字节流指的是发送数据时以字节为单位，一个数据包可以拆分成若干组进行发送，而udp一个报文只能一次发完。

5.tcp有拥塞控制，Udp没有。

6.tcp首部20字节，udp首部8字节

7.udp的主机不需要维护复杂的连接状态表

**15.tcp如何保证传输的可靠性？**

1.数据包校验

2.对失序的数据包重新排序（tcp报文具有序列号）

3.丢弃重复数据

4.应答机制：接收方收到数据后，会发送一个确认

5.超时重发：发送方发出数据后，启动一个定时器，超时未收到接收方的确认，则重发这个数据

6.流量控制：确保接收端能够接收发送方的数据而不会缓冲区溢出

**16.http与https有什么区别？**

1.端口不同：http使用的是80端口https使用443端口

2.http（超文本传输协议）是明文传输,https运行在SSL之上，添加了加密和认证机制，更加安全；

3.https由于加密解密会带来更大的CPU和内存开销

4.https通信需要证书，一般需要向证书颁发机构购买

**17.https的连接过程**

1. 客户端向服务器发送请求，同时发送客户端支持的一套加密规则（包括对称加密、非对称加密、摘要算法）；

2. 服务器从中选出一组加密算法与HASH算法，并将自己的身份信息以证书的形式发回给浏览器。证书里面包含了网站地址，**加密公钥**（用于非对称加密），以及证书的颁发机构等信息（证书中的私钥只能用于服务器端进行解密）；

3. 客户端验证服务器的合法性，包括：证书是否过期，CA 是否可靠，发行者证书的公钥能否正确解开服务器证书的“发行者的数字签名”，服务器证书上的域名是否和服务器的实际域名相匹配；

4. 如果证书受信任，或者用户接收了不受信任的证书，浏览器会生成一个**随机密钥**（用于对称算法），并用服务器提供的公钥加密（采用非对称算法对密钥加密）；使用Hash算法对握手消息进行**摘要**计算，并对摘要使用之前产生的密钥加密（对称算法）；将加密后的随机密钥和摘要一起发送给服务器；

5.服务器使用自己的私钥解密，得到对称加密的密钥，用这个密钥解密出Hash摘要值，并验证握手消息是否一致；如果一致，服务器使用对称加密的密钥加密握手消息发给浏览器；

6. 浏览器解密并验证摘要，若一致，则握手结束。之后的数据传送都使用对称加密的密钥进行加密

总结：非对称加密算法用于在握手过程中加密生成的密码；对称加密算法用于对真正传输的数据进行加密；HASH算法用于验证数据的完整性。

**18.什么是对称加密,非对称加密?**

对称加密：加密和解密采用相同的密钥。如：DES、RC2、RC4

非对称加密：需要两个密钥：公钥和私钥。如果用公钥加密，需要用私钥才能解密。如：RSA

区别：对称加密速度更快，通常用于大量数据的加密；非对称加密安全性更高（不需要传送私钥）

**19.GET和POST的区别？**

1.get在url后面带数据，其因为浏览器的限制（最多传送2K的url）和服务器限制（最多处理64K的url）导致传输的数据量小，并且由于操作会被第三方看到，并不是安全的。而post的数据是在http请求体里面的。

2.get限制form表单的数据集的值只能是ASCLL字符，而post支持更多更多的字符集

3.get用于请求数据 post可用于修改数据

4.get产生一个tcp数据包，浏览器把http header和data一并发过去，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。并不是所有浏览器都会在POST中发送两次包，Firefox就只发送一次

5.get默认会被浏览器缓存，而Post除非做出设置，否则不会

**20.从输入网址到获得页面的过程**

1. 浏览器查询 DNS，获取域名对应的IP地址:具体过程包括浏览器搜索自身的DNS缓存、搜索操作系统的DNS缓存、读取本地的Host文件和向本地DNS服务器进行查询等。对于向本地DNS服务器进行查询，如果要查询的域名包含在本地配置区域资源中，则返回解析结果给客户机，完成域名解析(此解析具有权威性)；如果要查询的域名不由本地DNS服务器区域解析，但该服务器已缓存了此网址映射关系，则调用这个IP地址映射，完成域名解析（此解析不具有权威性）。如果本地域名服务器并未缓存该网址映射关系，那么将根据其设置发起递归查询或者迭代查询；

2. 浏览器获得域名对应的IP地址以后，浏览器向服务器请求建立链接，发起三次握手；

3. TCP/IP链接建立起来后，浏览器向服务器发送HTTP请求；

4. 服务器接收到这个请求，并根据路径参数映射到特定的请求处理器进行处理，并将处理结果及相应的视图返回给浏览器；

5. 浏览器解析并渲染视图，若遇到对js文件、css文件及图片等静态资源的引用，则重复上述步骤并向服务器请求这些资源；

6. 浏览器根据其请求到的资源、数据渲染页面，最终向用户呈现一个完整的页面。

**21.http状态码**

200("OK")  
一切正常。实体主体中的文档（若存在的话）是某资源的表示。

400("Bad Request")  
客户端方面的问题。实体主体中的文档（若存在的话）是一个错误消息。希望客户端能够理解此错误消息，并改正问题。

服务器收到客户端通过PUT或者POST请求提交的表示，表示的格式正确，但服务器不懂它什么意思”的情况。

500("Internal Server Error")  
服务期方面的问题。实体主体中的文档（如果存在的话）是一个错误消息。该错误消息通常无济于事，因为客户端无法修复服务器方面的问题。

404("Not Found") 和410("Gone")  
当客户端所请求的URI不对应于任何资源时，发送此响应代码。404用于服务器端不知道客户端要请求哪个资源的情况；410用于服务器端知道客户端所请求的资源曾经存在，但现在已经不存在了的情况。

301("Moved Permanently")  
重要程度：中等。

服务器知道客户端试图访问的是哪个资源，但它不喜欢客户端用当前URI来请求该资源。它希望客户端记住另一个URI，并在今后的请求中使用那个新的URI。你可以通过这个响应代码来防止由于URI变更而导致老URI失效。

响应报头：服务器应当把规范URI放在Location响应报头里。  
实体主体：服务器可以发送一个包含新URI的信息，不过这不是必需的。

302("Found")  
重要程度：应该了解，特别市编写客户端时。但我不推荐使用它。

这个响应代码市造成大多数重定向方面的混乱的最根本原因。它应该是像307那样被处理。实际上，在HTTP 1.0中，响应代码302的名称是”Moved Temporarily”，不幸的是，在实际生活中，绝大多数客户端拿它像303一样处理。它的不同之处在于当服务器为客户端的PUT，POST或者DELETE请求返回302响应代码时，客户端要怎么做。  
为了消除这一混淆，在HTTP 1.1中，该响应代码被重命名为"Found"，并新加了一个响应代码307。这个响应代码目前仍在广泛使用，但它的含义市混淆的，所以我建议你的服务发送307或者303，而不要发送302.除非你知道正在与一个不能理解303或307的HTTP 1.0客户端交互。

响应报头：把客户端应重新请求的那个URI放在Location报头里。  
实体主体：一个包含指向新URI的链接的超文本文档（就像301一样）。

**22.udp如何实现可靠传输？**

传输层无法保证数据的可靠传输，只能通过应用层来实现。

1.添加seq/ack机制，确保数据发送到对端

2.添加发送接受缓冲区，主要用于超时重传

3.添加超时重传机制

目前有开源程序实现了可靠的数据传输,例如RUDP

**23.http如何保持登陆状态？**

**24.tcp中调用close()函数后发生的事情**

**close()把描述符计数减一，等于0时，内核自动发送fin。**

**Shutdow**

**25.大量的TIME\_WAIT会有什么问题？TIME\_WAIT能不能有别的客户端来请求服务器服务？**

**大量的TIME\_WAIT常常发生于短链接**

26.如何避免产生大量的CLOSE\_WAIT？什么时候产生大量的CLOSE\_WAIT状态？

Close\_wait状态产生于当客户端调用close()函数时，客户端主动发送fin报文请求断开连接。客户端异常断开时，比如直接Kill掉，那么如果服务器端对于这种异常并没有相应的处理，其实此时，内核在异常断开时依旧会发送fin报文给服务端，服务端进入close\_wait状态，并且会持续下去，这时，全连接的套接字fd会一直持有，而文件描述符个数有限。

如何避免？

比如使用了epoll,那么可以监听客户端连接套接字上面是否有EPOLLRDHUP事件发生。当客户端意外断开时，这个事件就会被触发，我们针对性地执行close就行。此时，服务端在接收到fin报文后并不会在CLOSE\_WAIT状态停留，而是直接进入LAST\_ACK。LAST\_ACK状态会超时自动消失。

如果服务端直接kill掉，这时客户端直接进入TIME\_WAIT状态，并会等待2MSL，2MSL大约是两分钟的时间，此时如果再启动server，会报端口占用的。这个等待就是服务器端接收ACK报文的。如果2MSL内没有收到ACK，会重发FIN。可以在socket中添加SO\_REUSEADDR来避免2MSL。

TIME\_WAIT状态的作用：

1.客户端发送的ACK报文可能并不能准确到达，服务端没收到ACK后会重发FIN报文

2.客户端发送的ACK报文可能重发，如果此时服务端在断开连接后立马启用同ip,端口的新服务，则会出现问题。2MSL保证了迷途的ACK在网络中消逝。

对于高并发短链接的业务场景，服务器端响应完请求后会立即主动关闭连接。而正常关闭连接都会出现time\_wait。time\_wait会导致tcp四元组保持2MSL，导致端口不会释放掉。

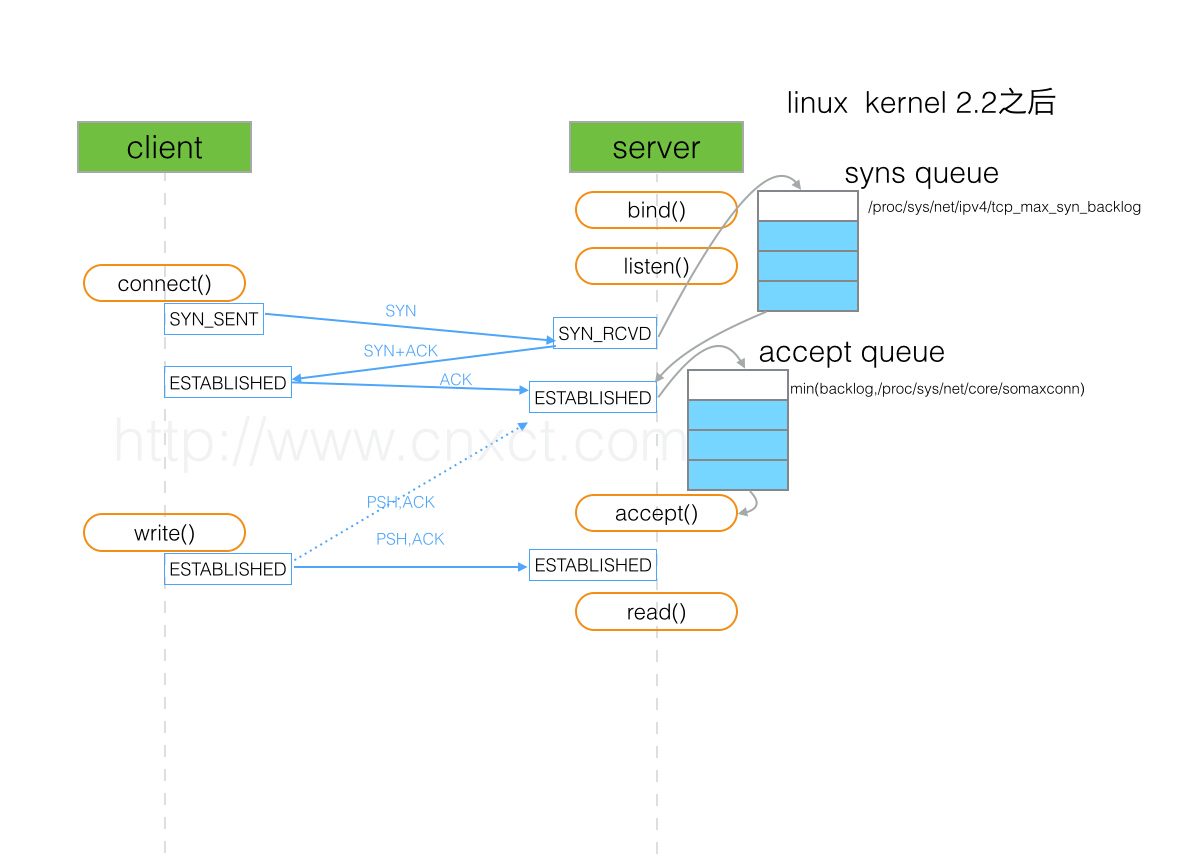
通过修改sysctl.conf文件可以修改相关的设定

27.SYN洪泛攻击

攻击者发送大量SYN数据包，但是当服务器返回ACK时，攻击者并不进行确认，那么这个连接就处于挂起状态，也就是半连接。服务器收不到第三次的ACK包会重复发送ACK给攻击者。

解决方法：降低SYN timeout的时间，使得主机尽快释放半连接的占用或者采用SYN cookie设置，当短时间内收到某个IP的重复SYN请求，就认为受到了攻击，可以采用防火墙设置针对某个ip进行阻挡。

28.tcp全连接半连接队列



在三次握手协议中，服务器维护一个半连接队列，该队列为每一个客户端的SYN包开设一个条目，服务端在接收到SYN包的时候，就已经创建了request\_sock结构，存储在半连接队列中，该条目表明服务器已经收到SYN包，并向客户发出确认，正在等待客户的确认包。

该队列就是SYN队列，长度是记录在/proc/net/ipv4/tcp\_max\_syn\_backlog中的，可以配置。

当第三次握手时，当server接收到ACK报文后，会进入一个accept队列，该队列的长度也是可调整的值。

Linux会默认发送5次SYN-ACK包，重试的间隔时间从1s开始，下一次的重传时间是前一次的双倍。

Linux利用SYNcookie机制来预防SYN洪泛攻击，就是将连接信息编码到ISN中返回给客户端。这时就不需要将半连接保存在队列中，而是利用客户端随后发来的ACK带回的ISN还原连接信息，以完成连接的建立，避免了半连接队列被SYN攻击包填满。

29.tcp初始化序列号ISN

Tcp初始化序列号不能设置成一个固定值，这样容易被攻击者猜测出后续序列号，从而遭受到攻击。

可以通过ISN随机生成算法来生成序列号，其生成规则为：

ISN=M+F(localhost,localport,remotehost,remoteport)

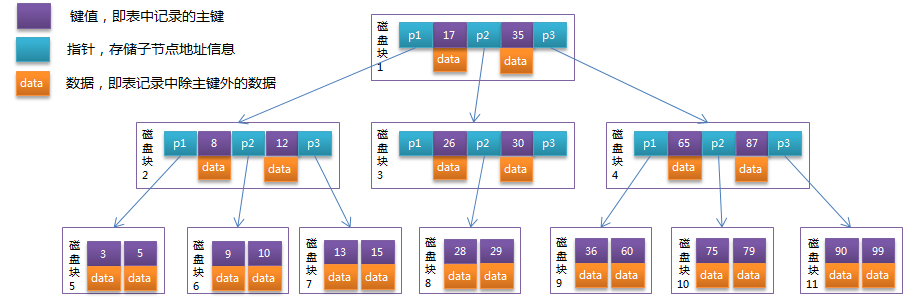
M是一个计时器，计时器每隔4ms加1，F是一个hash算法，根据tcp四元组生成一个随机数值。

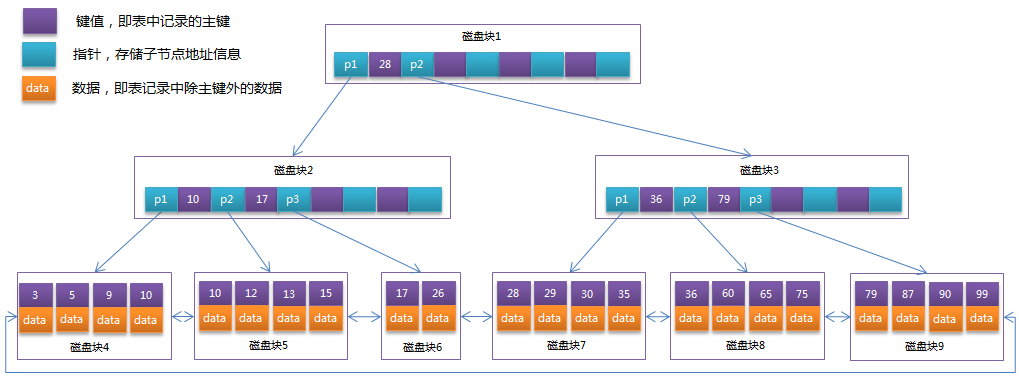
30数据库索引采用B+树的原因？

数据库文件保存在硬盘中，每次读取数据库都是在磁盘上搜索，一次磁盘IO的开销是比较大的。红黑树或者平衡二叉树虽然时间复杂度也不错，但是树的深度偏深，每次需要重复进行磁盘io的次数较多，，不适合。

为什么不采用B树呢？

B+树相比较于B树的优点在于非叶子节点只存储键值信息，所有叶子节点之间都有一个链表指针。数据记录在叶子节点上。B树的每一个节点不只保存了数据的key值，还包含了data值。每一个页的存储空间是有限的，如果data会导致单个节点的key的数量减小，当存储的数据量很大时同样会导致B-Tree的深度较大，增加了查询时的磁盘IO次数，进而影响查询效率。

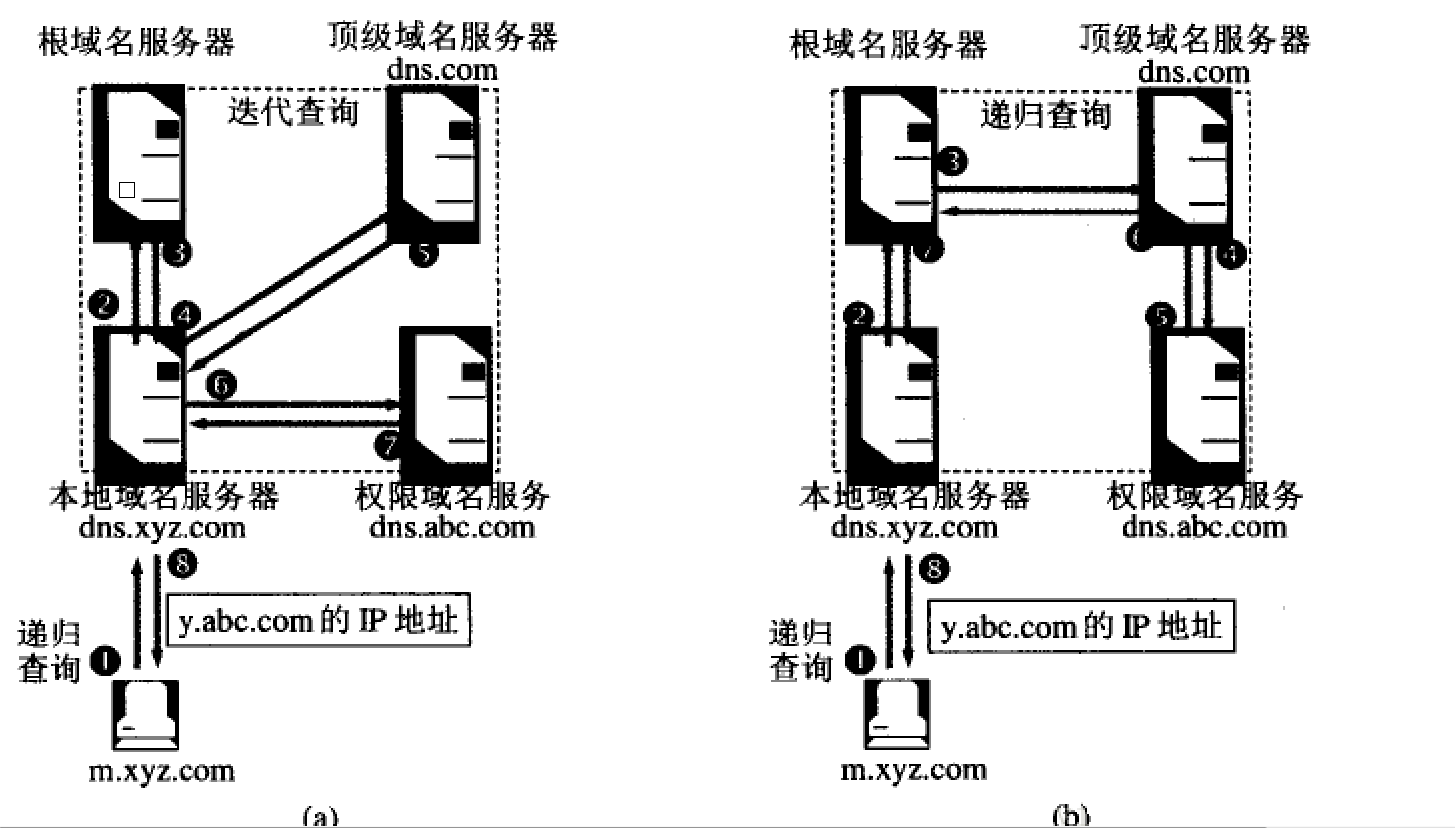




为什么不用hash？

Hash虽然可以快速定位，但是没有顺序，IO复杂度较高。

域名解析的过程：



迭代查询：

1、主机m.abc.com先向本地服务器dns.xyz.com进行递归查询。

2、本地服务器采用迭代查询。它先向一个根域名服务器查询。

3、根域名服务器告诉本地服务器，下一次应查询的顶级域名服务器dns.com的IP地址。

4、本地域名服务器向顶级域名服务器dns.com进行查询。

5、顶级域名服务器dns.com告诉本地域名服务器，下一步应查询的权限服务器dns.abc.com的IP地址。

6、本地域名服务器向权限域名服务器dns.abc.com进行查询。

7、权限域名服务器dns.abc.com告诉本地域名服务器，所查询的主机的IP地址。

8、本地域名服务器最后把查询结果告诉m.xyz.com。

http协议的演变以及逻辑：

http0.9只有一个get命令，没有http header等描述数据的信息，服务器只能返回html格式的字符串数据，服务器发送完毕后就立即关闭tcp连接。

http1.0新增了post命令和head命令，新增了status code和http header。但是每一个tcp连接还是和之前的一样，每一次只能发送一个请求。数据发送完毕后，连接就关闭，如果还要请求其它资源，就必须新建一个连接。

http1.1支持长久连接。持久连接的意思就是说tcp连接在服务器发送完数据默认不关闭，可以被多个请求复用。

http1.1引入管道的机制，即在同一个tcp连接中，客户端可以同时发送多个请求。

在一个tcp连接中可以传输多个Http请求和应答，减少了建立和关闭连接的消耗和延迟。一个包含许多图像的网页文件的多个请求和应答可以在一个连接中传输，但是每一个单独的网页文件的请求依旧要使用各自的连接。http1.1还允许客户端不用等待上一次请求结果返回，就可以发送下一次请求，但是服务器端在响应时确要按照先后顺序依次回送响应结果。

http请求的格式：

request报文的格式：

请求方法URL协议/版本

请求头

请求正文

**GET /barite/account/stock/groups HTTP/1.1**

**QUARTZ-SESSION: MC4xMDQ0NjA3NTI0Mzc0MjAyNg.VPXuA8rxTghcZlRCfiAwZlAIdCA**

**DEVICE-TYPE: ANDROID**

**API-VERSION: 15**

**Host: shitouji.bluestonehk.com**

**Connection: Keep-Alive**

**Accept-Encoding: gzip**

**User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/68.0.3440.106 Safari/537.36**

**Response报文的格式：**

**http的响应由三部分组成：**

**协议/版本 状态码 描述**

**响应头**

**响应正文**

**HTTP/1.1 200 OK**

**Server: nginx/1.6.3**

**Date: Mon, 15 Oct 2018 03:30:28 GMT**

**Content-Type: application/json;charset=UTF-8**

**Pragma: no-cache**

**Cache-Control: no-cache**

**Expires: Thu, 01 Jan 1970 00:00:00 GMT**

**Content-Encoding: gzip**

**Transfer-Encoding: chunked**

**Proxy-Connection: Keep-alive**

**{"errCode":0，"message":"success"**

**几个关键的字段：**

**Host:指定想要访问的服务器的域名，可以用来区分同一台服务器上面的不同服务**

**Connection：keep-alive表示要求服务器不要关闭tcp连接，close表示明确要求关闭连接，默认值为keep-alive**

**https的过程：**

**首先客户端提出请求，然后服务端会返回一个数字证书，里面有一个公钥，私钥保存在服务器自己身边，然后客户端得到这个公钥后，自己通过计算产生一个私钥，这个私钥的产生过程是通过随机算法产生的。然后浏览器通过公钥对这个产生的对话密钥进行加密，发送给服务器，服务器获得这个报文后用公钥解密之后获得了对话密钥。然后大家用这个对话密钥进行后面传输报文的加密。**

**非对称加密算法：**

**RSA算法工作原理：基于大数的因数分解**

**1.随机找两个质数a和b,并计算它们的乘积n,比如5\*11=55，二进制为110111，该加密算法则是六位**

**2.计算n的欧拉函数m=f(n)**

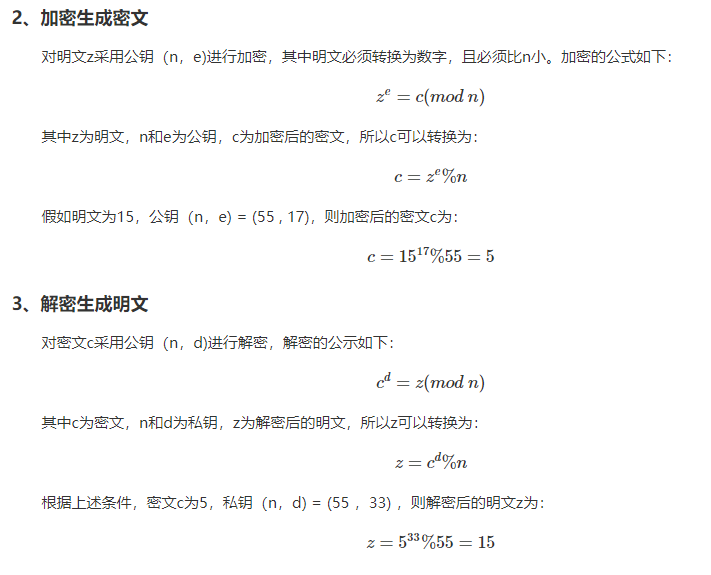
**计算的方式就是m=f(55)=f(5)f(11)=(5-1)(11-1)=40**

**3.随机选择一个整数e,条件是1<e<m,且e与m互质**

**假设我们随机选择17**

**4.计算e对于f(n)的模反元素d，即计算一个整数d,使得（e\*d）%m=1**

**则生成的公钥就是(n,e)，私钥就是(n,d)**



 首先选取两个互质的数相乘获得一个乘积n，对这个乘积n使用欧拉函数获得一个值m，再随机选择一个与m互质的数e，计算其对于欧拉函数的模反元素d

公钥：（n,e）私钥：（n,d）

**对称加密算法：AES DES等不了解**

**哈希算法：SHA1算法**

**原理：接收一段明文，然后以一种不可逆的方式将其转换为一段密文，也可以理解为输入一段二进制码，将他们转换为长度较短，位数固定的输出序列，这就是信息摘要的过程。**

**SHA1算法产生的是一个20个字节的报文摘要。**

**粘包：出现粘包的原因是多种的，包括：**

**1.发送方造成的粘包原因是因为TCP为了传输的效率，如果每次传输的包都很小，TCP会根据Nagle算法直到缓冲区满才发送，这样就会造成粘包。**

**从接收端来考虑，发生粘包问题往往是因为接收方用户进程不及时接收数据，从而导致粘包问题，如果接收方先把收到的数据放在系统的接收缓冲区，用户进程从该缓冲区取数据，若下一包数据到达时，前一包数据尚未被用户进程取走，则下一包数据放到接收缓冲区时就会接在前一包数据之后，而用户进程根据预先设定的缓冲区大小从系统接收缓冲区取数据，就会造成一次取到多个包。**

**什么时候需要考虑粘包问题：**

**1.当连接是短链接时，不需要考虑粘包问题**

**2.如果发送数据无结构，如文件传输，这样发送方只管发送，接收方只管接收存储就好，也不用考虑粘包**

**3.长连接时需要考虑粘包的问题，因为长连接会发送不同结构的数据。这时候需要做封包拆包的操作。**

**如何封包拆包？**

**发送端给每一个数据包添加包首部，首部中至少应该包含数据包的长度，这样接收方在接收到数据后，通过读取包首部的长度字段，就知道了接下来接收的数据包的实际长度。**

**发送方将每一个数据包封装成固定长度，不够的可以通过补0填充，这样接收端从接收缓冲区读取固定长度的数据就自然会把数据包拆分开来。**

**可以在数据包之间设置边界，如添加特殊字符。**

**当发送端不断调用send函数将数据拷贝到发送缓冲区内，经过网络，到达接收端的接收缓冲区，如果接收端一直不调用recv函数的话，最终两个缓冲区都会满。这时虽然可以通过tcp的滑动窗口来实现发送方不再发送报文给接收端。但是如果发送端一直调用send函数，这时候会出现两种情况，**

**如果send设置了阻塞模式，则会发生阻塞。**

**如果send是非阻塞模式，则会返回一个错误码，linux平台下是EAGAIN**

**Http Get的body能不能带数据？**

**Get通常是通过将数据拼接到URL后面传送的，而又因为URL的长度是有一定的限制的，所以导致说get能够传输的数据量有限。但是get的body中其实也可以带数据。之所以说get的数据跟在了url后面，是因为我们在前端通常使用form表单提交时，method属性只有get和post两种方法，method为get时，就会把内容拼接到字符串后面。但其实是可以通过get 中的body来发送数据的。**

**Get方法中URL的限制并不是HTTP协议规范中设定的，而是由浏览器和服务器来给定的，过长的URL不会被一些浏览器允许，同时服务器处理时也会造成一定的负担。**

**同样的post的body同样没有长度的限制，都是因为比如tomcat的文件中有一个文件中有一个postsize字段，设置为0则表示取消对于post的限制。**

**Epoll即使使用ET模式（边缘触发模式），在一个线程读完某socket上的数据后开始处理这些数据，此时该socket上面又有新的数据可以读（即EPOLLIN再次被触发），另一个线程被唤醒读新的数据，这会造成两个线程同时操作同一个socket的局面。这时通过注册EPOLLONESHOT事件。**

**EPOLLONESSHOT事件：只监听一次，当监听完这次事件后，如果还需要监听这个socket的话，需要再次把这个socket加入到EPOLL队列中。**

**Http是一个无状态的协议，就是说这一次请求和上一次请求没有任何关系，互相不认识，没有关联。**

**1.客户端发送http请求到服务器端**

**2.服务器端接收到请求后，建立一个session,并发送一个http响应到客户端，这个响应头，其中就包括set-cookie头部。该头部包含了sessionid**

**3.在客户端发起第二次请求，假如服务器给了set-cookie,浏览器会自动在请求头中添加cookie**

**4.服务器接收请求，分解cookie,验证信息，核对成功后返回response给客户端。**

**Cookie的内容包括：名字，值，过期时间，路径和域。路径和域构成cookie的作用范围**

**Name和Value属性由程序设定，默认值都是空引用**

**Domain属性的默认值为当前URL的域名部分，不管发出这个cookie的页面在哪个目录下**

**Path属性的默认值为根目录**

**Expires属性，这个属性设置此Cookie的过期时间**

**会话cookie：如果不设置cookie的过期时间，cookie的过期时间就是浏览器的会话期间，关闭浏览器窗口，则cookie消失，这种cookie保存在内存中。**

**持久cookie：如果设置了cookie的过期时间，则浏览器就会把cookie保存在硬盘上，关闭后再次打开浏览器，这些cookie仍然有效直到超过设定的过期时间。**

**Session是另一种记录客户状态的机制，不同的cookie保存在客户端浏览器，而session是保存在服务器上的。**

**客户端浏览器再次访问时，只需要访问该session，查找客户状态就好了。每个用户访问服务器都会建立一个session，服务器自动分配一个sessionid作为该用户的唯一标识。**

**在http应答报文中，服务器把seeionid自动附加在http set-cookie字段中，当客户端提交表单时，浏览器会将用户的sessionid自动附加在Http头信息中，当服务器处理完这个表单后，将结果返回给sessionid所对应的用户。**

**区别：**

**Session在服务器端，cookie在客户端**

**Session默认存在于服务器上面的一个文件里面**

**Session的运行依赖于session id，而sessionid是存在于cookie中的**

**网络分析的步骤：**

**1.使用ipconfig/all观察本地网络设置是否正确**

**2.ping 127.0.0.1，检查本地的TCP/IP协议有没有设置好**

**3.ping本机ip地址，检查本机ip地址是否有误**

**4.ping 本网网关或本网ip地址，检查硬件是否有问题**

**5.ping 本地DNS地址，检查本地DNS服务器是否工作正常**

**6.ping 远程的ip地址，检查本机与外部的连接是否正常**