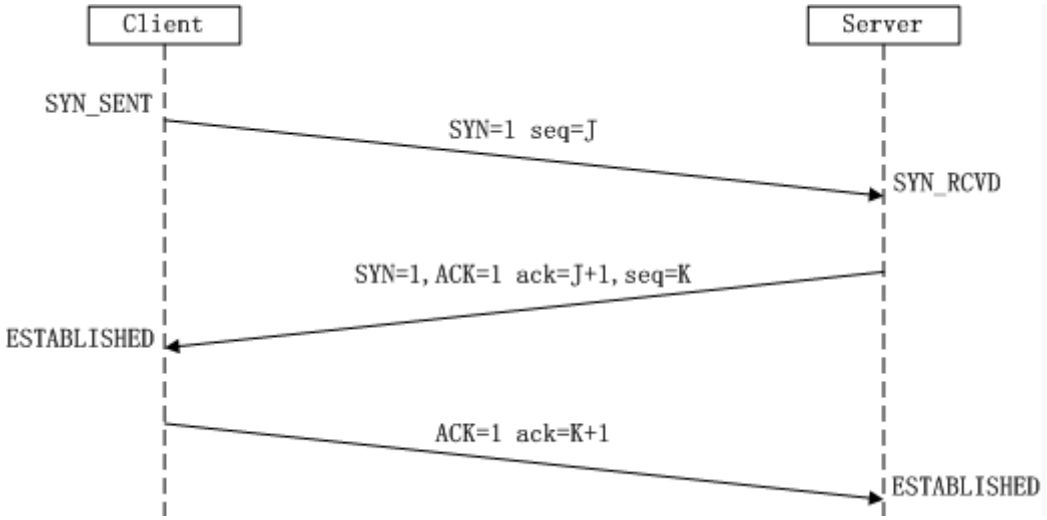
计算机网络

1、什么是三次握手 (three-way handshake)?



- 第一次握手：Client将SYN置1，随机产生一个初始序列号seq发送给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acknowledge number是否为序列号+1，ACK是否为1，检查正确之后将自己的ACK置为1，产生一个acknowledge number=服务器发的序列号+1，发送给服务器；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。

1.1、TCP建立连接可以两次握手吗？为什么?

不可以。有两个原因：

1. 可能会出现**已失效的连接请求报文段又传到了服务器端。**

> client 发出的第一个连接请求报文段并没有丢失，而是在某个网络结点长时间的滞留了，以致延误到连接释放以后的某个时间才到达 server。本来这是一个早已失效的报文段。但 server 收到此失效的连接请求报文段后，就误认为是 client 再次发出的一个新的连接请求。于是就向 client 发出确认报文段，同意建立连接。假设不采用 “三次握手”，那么只要 server 发出确认，新的连接就建立了。由于现在 client 并没有发出建立连接的请求，因此不会理睬 server 的确认，也不会向 server 发送数据。但 server 却以为新的运输连接已经建立，并一直等待 client 发来数据。这样，server 的很多资源就白白浪费掉了。采用 “三次握手” 的办法可以防止上述现象发生。例如刚才那种情况，client 不会向 server 的确认发出确认。server 由于收不到确认，就知道 client 并没有要求建立连接。

2. 其次，两次握手无法保证Client正确接收第二次握手的报文（Server无法确认Client是否收到），也无法保证Client和Server之间成功互换初始序列号。

1.2、可以采用四次握手吗？为什么?

可以。但是会降低传输的效率。

四次握手是指：第二次握手：Server只发送ACK和acknowledge number；而Server的SYN和初始序列号在第三次握手时发送；原来协议中的第三次握手变为第四次握手。出于优化目的，四次握手中的二、三可以合并。

1.3、第三次握手中，如果客户端的ACK未送达服务器，会怎样?

- Server端： 由于Server没有收到ACK确认，因此会重发之前的SYN+ACK（默认重发五次，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：
    1. 在Server进行超时重发的过程中，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
    2. 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。

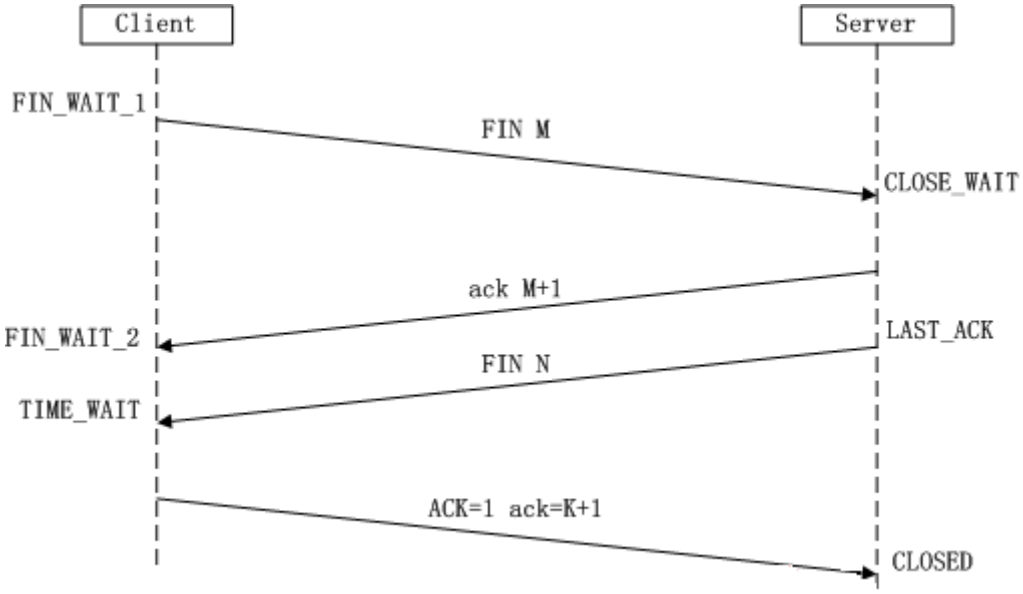
1.4、如果已经建立了连接，但客户端出现了故障怎么办?

服务器每收到一次客户端的请求后都会重新复位一个计时器，时间通常是设置为2小时，若两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秒钟发送一次。若一连发送10个探测报文仍然没反应，服务器就认为客户端出了故障，接着就关闭连接。

1.5、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（Sequence Number）作为发送数据的初始序列号（Initial Sequence Number，ISN），比如为1000，以该序列号为原点，对要传送的数据进行编号：1001、1002...三次握手时，把这个初始序列号传送给另一方B，以便在传输数据时，B可以确认什么样的数据编号是合法的；同时在进行数据传输时，A还可以确认B收到的每一个字节，如果A收到了B的确认编号（acknowledge number）是2001，就说明编号为1001-2000的数据已经被B成功接受。

2、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之后，发送一个ACK=1，acknowledge number=收到的序列号+1；进入CLOSE_WAIT状态。此时客户端已经没有要发送的数据了，但仍可以接受服务器发来的数据。
- 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（报文段最长寿命）时间后，也进入CLOSED状态。完成四次挥手。

2.1、为什么不能把服务器发送的ACK和FIN合并起来，变成三次挥手（CLOSE_WAIT状态意义是什么）?

因为服务器收到客户端断开连接的请求时，可能还有一些数据没有发完，这时先回复ACK，表示接收到了断开连接的请求。等到数据发完之后再发FIN，断开服务器到客户端的数据传送。

2.2、如果第二次挥手时服务器的ACK没有送达客户端，会怎样?

客户端没有收到ACK确认，会重新发送FIN请求。

2.3、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有可能丢失，TIME_WAIT状态就是用来重发可能丢失的ACK报文。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。

MSL(Maximum Segment Lifetime)，指一个片段在网络中最大的存活时间，2MSL就是一个发送和一个回复所需的最大时间。如果直到2MSL，Client都没有再次收到FIN，那么Client推断ACK已经被成功接收，则结束TCP连接。

3、TCP如何实现流量控制?

窗口字段 —— 占 2 字节。窗口字段用来控制对方发送的数据量，单位为字节。TCP 连接的一端根据设置的缓存空间大小确定自己的接收窗口大小，然后通知对方以确定对方的发送窗口的上限。

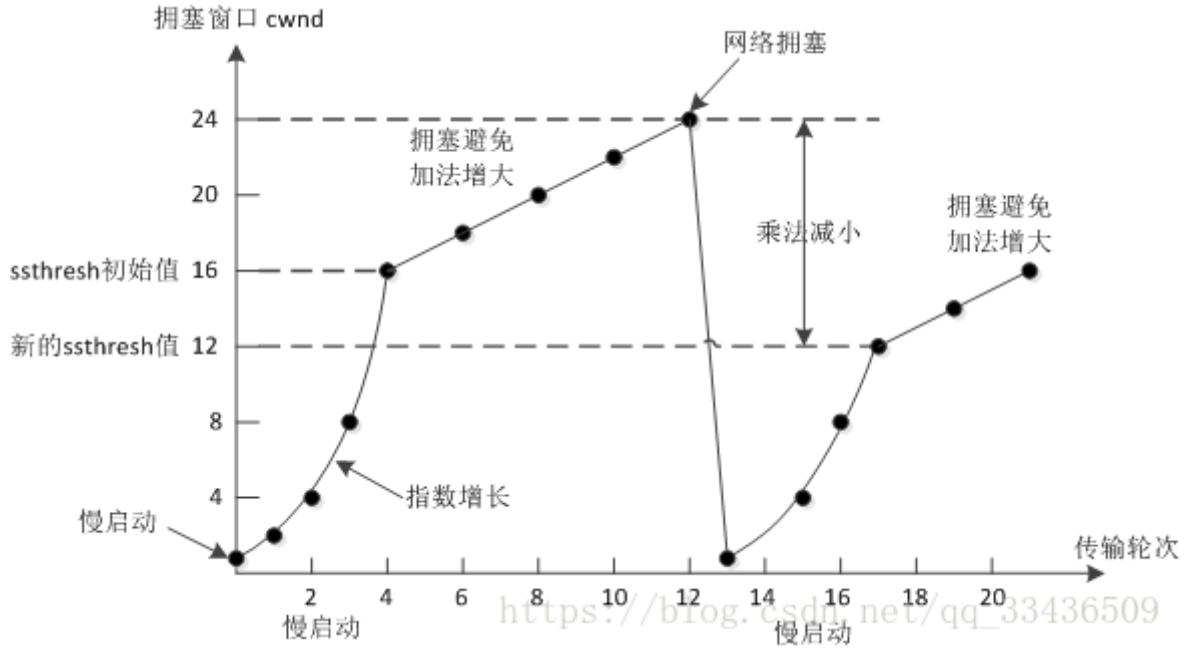
使用**滑动窗口协议实现流量控制**。防止发送方发送速率太快，接收方缓存区不够导致溢出。接收方会维护一个接收窗口 receiver window（窗口大小单位是字节），接受窗口的大小是根据自己的资源情况动态调整的，在返回ACK时将接受窗口大小放在TCP报文中的窗口字段告知发送方。发送窗口的大小不能超过接受窗口的大小，只有当发送方发送并收到确认之后，才能将发送窗口右移。

发送窗口的上限为接受窗口和拥塞窗口中的较小值。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。

3.1、什么是零窗口（接收窗口为0时会怎样）？

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

4、TCP的拥塞控制是怎么实现的?



拥塞控制主要由四个算法组成：**慢启动（Slow Start）、拥塞避免（Congestion voidance）、快重传 （Fast Retransmit）、快恢复（Fast Recovery）**

1. 慢启动：刚开始发送数据时，先把拥塞窗口（congestion window）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1个MSS。这样每经过一个传输轮次（或者说是每经过一个往返时间RTT），拥塞窗口的大小就会加倍

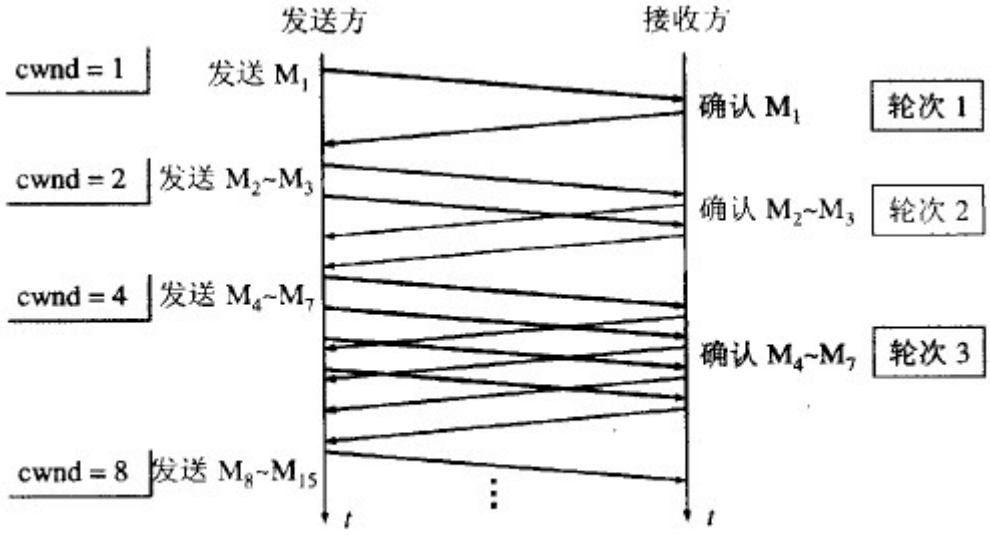


图 5-24　发送方每收到一个确认就把窗口 cwnd 加 1

2. 拥塞避免：当拥塞窗口的大小达到慢开始门限(slow start threshold)时，开始执行拥塞避免算法，拥塞窗口大小不再指数增加，而是线性增加，即每经过一个传输轮次只增加1MSS.

> 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）**

3. 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。

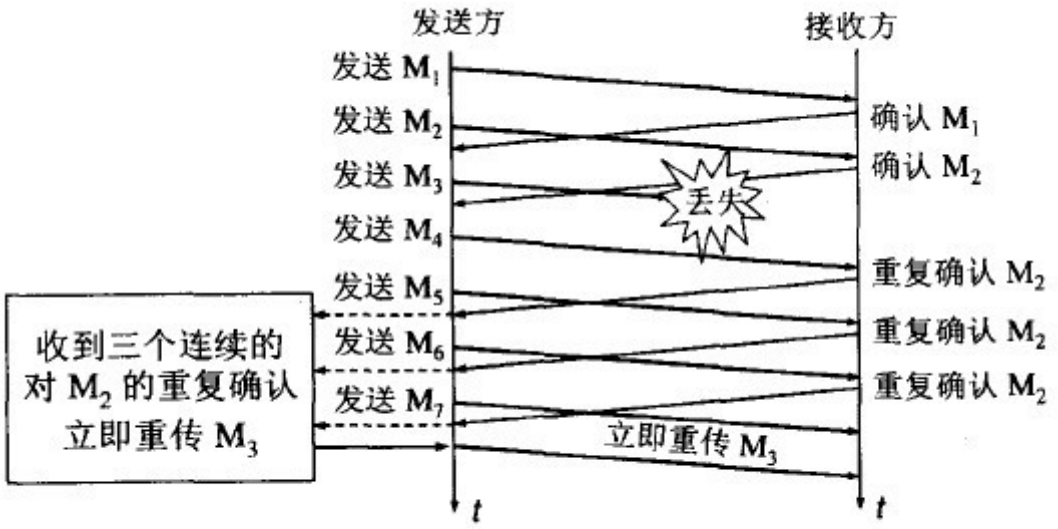


图 5-26 快重传的示意图

4. 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。

5、TCP与UDP的区别

1. TCP是面向连接的，UDP是无连接的；
2. TCP是可靠的，UDP不可靠；
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；
5. TCP有拥塞控制机制，UDP没有。网络出现的拥塞不会使源主机的发送速率降低，这对某些实时应用是很重要的，比如媒体通信，游戏；
6. TCP首部开销（20字节）比UDP首部开销（8字节）要大
7. UDP 的主机不需要维持复杂的连接状态表

5.1、什么时候选择TCP，什么时候选UDP?

对某些实时性要求比较高的情况，选择UDP，比如游戏，媒体通信，实时视频流（直播），即使出现传输错误也可以容忍；其它大部分情况下，HTTP都是用TCP，因为要求传输的内容可靠，不出现丢失

5.2、HTTP可以使用UDP吗?

HTTP不可以使用UDP，HTTP需要基于可靠的传输协议，而UDP不可靠

5.3、面向连接和无连接的区别

无连接的网络服务（数据报服务）-- 面向连接的网络服务（虚电路服务）

虚电路服务：首先建立连接，所有的数据包经过相同的路径，服务质量有较好的保证；

数据报服务：每个数据包含目的地址，数据路由相互独立（路径可能变化）；网络尽最大努力交付数据，但不保证不丢失、不保证先后顺序、不保证在时限内交付；网络发生拥塞时，可能会将一些分组丢弃；

数据报和虚电路的服务比较

	虚电路	数据报
是否需要建立连接	需要	不需要
分组中的目的地址	vc标识	完整地址
选路	在vc建立时选路，所有分组路由相同	每个分组独立选路，路由可能不同
路由器故障的影响	所有经过该路由器的vc都将终止	几乎不受影响
拥塞控制	易于实现	很难实现
差错控制和流量控制	由子网负责	由主机负责

6、TCP如何保证传输的可靠性

1. 数据包校验
2. 对失序数据包重新排序（TCP报文具有序列号）
3. 丢弃重复数据
4. 应答机制：接收方收到数据之后，会发送一个确认（通常延迟几分之一秒）；
5. 超时重发：发送方发出数据之后，启动一个定时器，超时未收到接收方的确认，则重新发送这个数据；
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

7、HTTP和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）信息是明文传输，HTTPS运行在SSL(Secure Socket Layer)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（CA）购买

7.1、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钥；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。

7.2、输入 www.baidu.com，怎么变成 https://www.baidu.com 的，怎么确定用HTTP还是HTTPS?

你访问的网站是如何自动切换到 HTTPS 的?

一种是原始的302跳转，服务器把所有的HTTp流量跳转到HTTPS。但这样有一个漏洞，就是中间人可能在第一次访问站点的时候就劫持。 解决方法是引入HSTS机制，用户浏览器在访问站点的时候强制使用HTTPS。

7.3、什么是对称加密、非对称加密？区别是什么?

- 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）

7.4、数字签名、报文摘要的原理

- 发送者A用私钥进行签名，接收者B用公钥验证签名。因为除A外没有人有私钥，所以B相信签名是来自A。A不可抵赖，B也不能伪造报文。
- 摘要算法:MD5、SHA

8、GET与POST的区别?

1. GET是幂等的，即读取同一个资源，总是得到相同的数据，POST不是幂等的；
2. GET一般用于从服务器获取资源，而POST有可能改变服务器上的资源；
3. 请求形式上：GET请求的数据附在URL之后，在HTTP请求头中；POST请求的数据在请求体中；
4. 安全性：GET请求可被缓存、收藏、保留到历史记录，且其请求数据明文出现在URL中。POST的参数不会被保存，安全性相对较高；
5. GET只允许ASCII字符，POST对数据类型没有要求，也允许二进制数据；
6. GET的长度有限制（操作系统或者浏览器），而POST数据大小无限制

9、Session与Cookie的区别?

- Session是服务器端保持状态的方案，Cookie是客户端保持状态的方案
- Cookie保存在客户端本地，客户端请求服务器时会将Cookie一起提交；Session保存在服务端，通过检索Sessionid查看状态。保存Sessionid的方式可以采用Cookie，如果禁用了Cookie，可以使用URL重写机制（把会话ID保存在URL中）。

10、从输入网址到获得页面的过程 (越详细越好)?

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。

11、HTTP请求有哪些常见状态码?

1. 2xx状态码：操作成功。200 OK
2. 3xx状态码：重定向。301 永久重定向；302暂时重定向
3. 4xx状态码：客户端错误。400 Bad Request；401 Unauthorized；403 Forbidden；404 Not Found；
4. 5xx状态码：服务端错误。500服务器内部错误；501服务不可用

12、什么是RIP (Routing Information Protocol, 距离矢量路由协议)? 算法是什么?

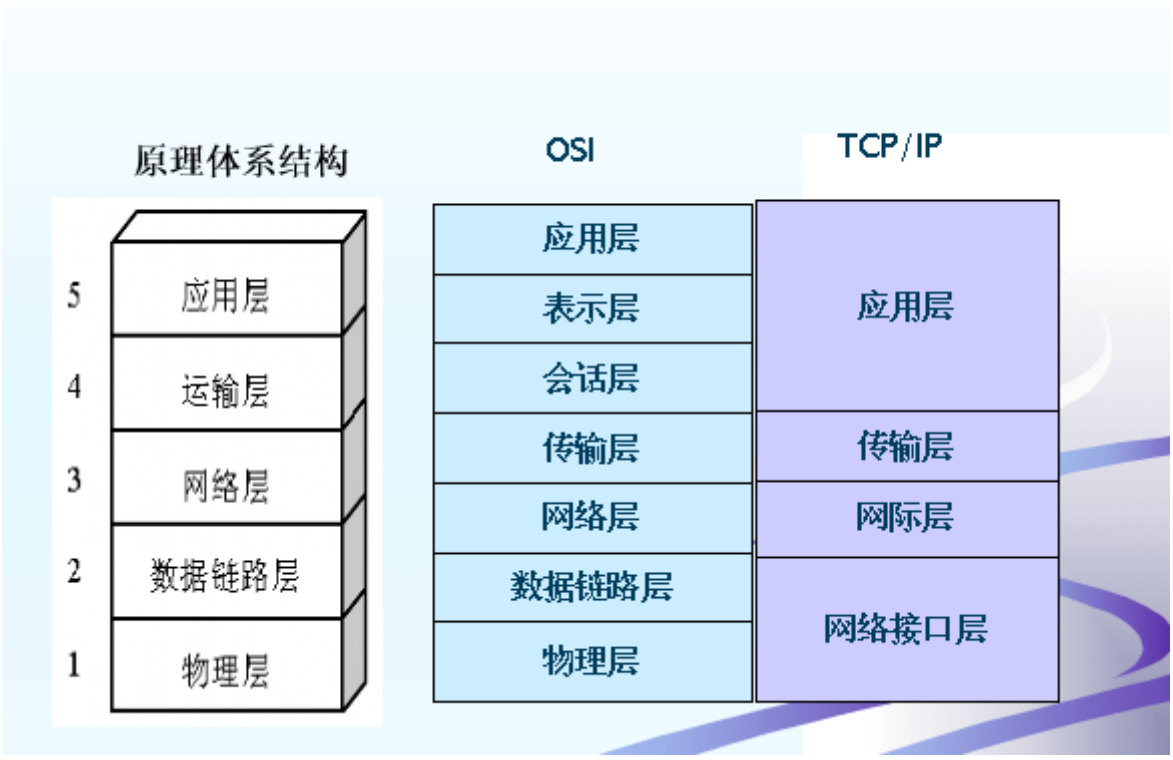
每个路由器维护一张表，记录该路由器到其它网络的"跳数"，路由器到与其直接连接的网络的跳数是1，每多经过一个路由器跳数就加1；更新该表时和相邻路由器交换路由信息；路由器允许一个路径最多包含15个路由器，如果跳数为16，则不可达。交付数据报时优先选取距离最短的路径。

（PS：RIP是应用层协议：https://www.zhihu.com/question/19645407）

优缺点

- 实现简单，开销小
- 随着网络规模扩大开销也会增大；
- 最大距离为15，限制了网络的规模；
- 当网络出现故障时，要经过较长的时间才能将此信息传递到所有路由器

13、计算机网络体系结构

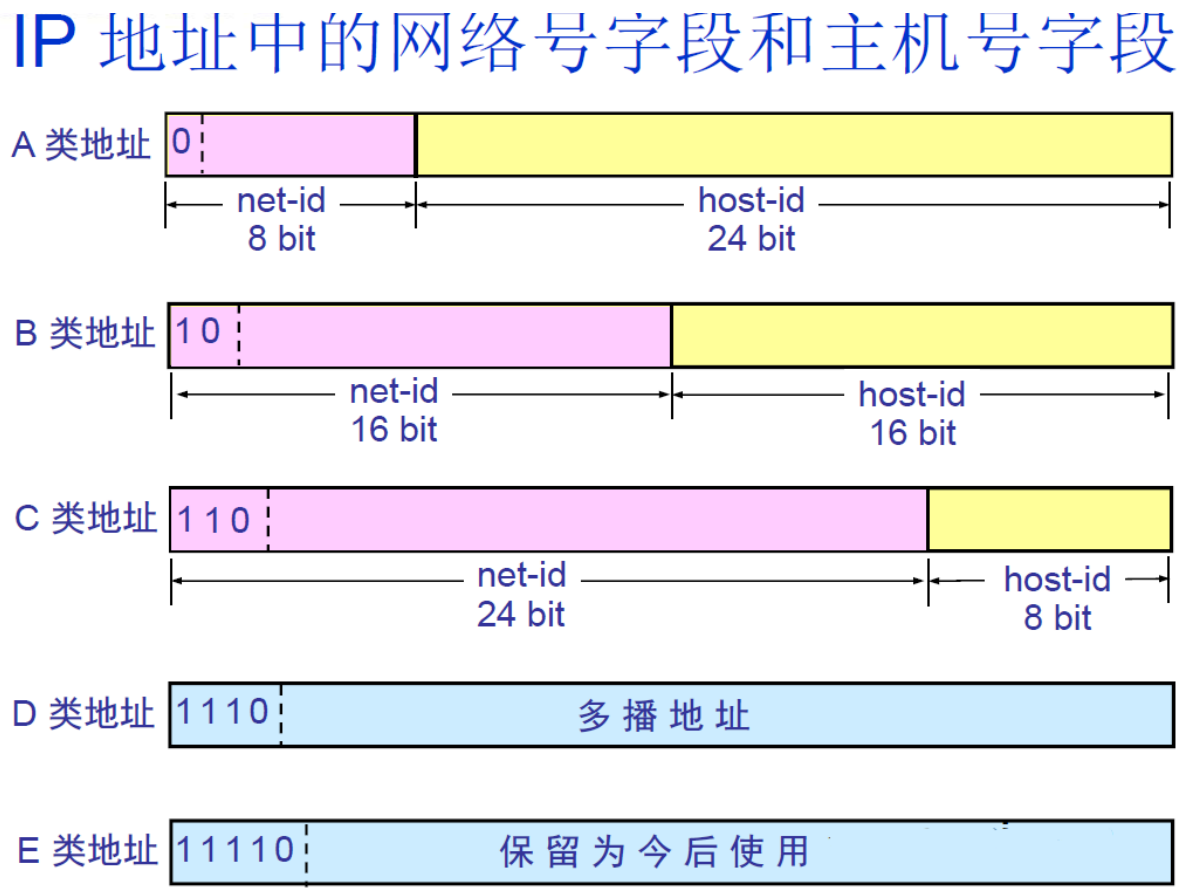


- Physical, Data Link, Network, Transport, Application
- 应用层：常见协议：
    - FTP(21端口)：文件传输协议
    - SSH(22端口)：远程登陆
    - TELNET(23端口)：远程登录
    - SMTP(25端口)：发送邮件
    - POP3(110端口)：接收邮件
    - HTTP(80端口)：超文本传输协议
    - DNS(53端口)：运行在UDP上，域名解析服务
- 传输层：TCP/UDP
- 网络层：IP、ARP、NAT、RIP...

13.1、路由器、交换机位于哪一层?

- 路由器网络层，根据IP地址进行寻址；
- 交换机数据链路层，根据MAC地址进行寻址

14、IP地址的分类?



路由器仅根据网络号net-id来转发分组，当分组到达目的网络的路由器之后，再按照主机号host-id将分组交付给主机；同一网络上的所有主机的网络号相同。

15、什么叫划分子网?

从主机号host-id借用若干个比特作为子网号subnet-id；子网掩码：网络号和子网号都为1，主机号为0；数据报仍然先按照网络号找到目的网络，发送到路由器，路由器再按照网络号和子网号找到目的子网：将子网掩码与目标地址逐比特与操作，若结果为某个子网的网络地址，则送到该子网。

16、什么是ARP协议 (Address Resolution Protocol)?

ARP协议完成了IP地址与物理地址的映射。每一个主机都设有一个 ARP 高速缓存，里面有**所在的局域网**上的各主机和路由器的 IP 地址到硬件地址的映射表。当源主机要发送数据包到目的主机时，会先检查自己的ARP高速缓存中有没有目的主机的MAC地址，如果有，就直接将数据包发到这个MAC地址，如果没有，就向**所在的局域网**发起一个ARP请求的广播包（在发送自己的 ARP 请求时，同时会带上自己的 IP 地址到硬件地址的映射），收到请求的主机检查自己的IP地址和目的主机的IP地址是否一致，如果一致，则先保存源主机的映射到自己的ARP缓存，然后给源主机发送一个ARP响应数据包。源主机收到响应数据包之后，先添加目的主机的IP地址与MAC地址的映射，再进行数据传送。如果源主机一直没有收到响应，表示ARP查询失败。

如果所要找的主机和源主机不在同一个局域网上，那么就要通过 ARP 找到一个位于本局域网上的某个路由器的硬件地址，然后把分组发送给这个路由器，让这个路由器把分组转发给下一个网络。剩下的工作就由下一个网络来做。

17、什么是NAT (Network Address Translation, 网络地址转换)?

用于解决内网中的主机要和因特网上的主机通信。由NAT路由器将主机的本地IP地址转换为全球IP地址，分为静态转换（转换得到的全球IP地址固定不变）和动态NAT转换。

操作系统

1、进程和线程有什么区别?

- 进程（Process）是系统进行资源分配和调度的基本单位，线程（Thread）是CPU调度和分派的基本单位；
- 线程依赖于进程而存在，一个进程至少有一个线程；
- 进程有自己的独立地址空间，线程共享所属进程的地址空间；
- 进程是拥有系统资源的一个独立单位，而线程自己基本上不拥有系统资源，只拥有一点在运行中必不可少的资源(如程序计数器,一组寄存器和栈)，和其他线程共享本进程的相关资源如内存、I/O、CPU等；
- 在进程切换时，涉及到整个当前进程CPU环境的保存和设置以及新被调度运行的CPU环境的设置，而线程切换只需保存和设置少量的寄存器的内容，并不涉及存储器管理方面的操作，可见，进程切换的开销远大于线程切换的开销；
- 线程之间的通信更方便，同一进程下的线程共享全局变量等数据，而进程之间的通信需要以进程间通信(IPC)的方式进行；
- 多线程程序只要有一个线程崩溃，整个程序就崩溃了，但多进程程序中一个进程崩溃并不会对其它进程造成影响，因为进程有自己的独立地址空间，因此多进程更加健壮

1.1、同一进程中的线程可以共享哪些数据?

- 进程代码段
- 进程的公有数据（全局变量、静态变量...）
- 进程打开的文件描述符
- 进程的当前目录
- 信号处理器/信号处理函数：对收到的信号的处理方式
- 进程ID与进程组ID

1.2、线程独占哪些资源?

- 线程ID
- 一组寄存器的值
- 线程自身的栈（堆是共享的）
- 错误返回码：线程可能会产生不同的错误返回码，一个线程的错误返回码不应该被其它线程修改；
- 信号掩码/信号屏蔽字(Signal mask)：表示是否屏蔽/阻塞相应的信号（SIGKILL,SIGSTOP除外）

2、进程间通信有哪些方式?

1. 管道(Pipe)
    - 管道是半双工的，数据只能向一个方向流动；需要双方通信时，需要建立起两个管道；
    - 一个进程向管道中写的内容被管道另一端的进程读出。写入的内容每次都添加在管道缓冲区的末尾，并且每次都是从缓冲区的头部读出数据；
    - 只能用于父子进程或者兄弟进程之间(具有亲缘关系的进程)
2. 命名管道
3. 消息队列
4. 信号(Signal)
5. 共享内存
6. 信号量(Semaphore)
    - 初始化操作、P操作、V操作；
    - P操作：信号量-1，检测是否小于0，小于则进程进入阻塞状态；
    - V操作：信号量+1，若小于等于0，则从队列中唤醒一个等待的进程进入就绪态
7. 套接字(Socket)

- https://imageslr.github.io/2020/02/26/ipc.html
- https://www.jianshu.com/p/c1015f5ffa74

3、进程同步问题

> 进程的同步是目的，而进程间通信是实现进程同步的手段

- 管程将共享变量以及对这些共享变量的操作封装起来，形成一个具有一定接口的功能模块，这样只能通过管程提供的某个过程才能访问管程中的资源。进程只能互斥地使用管程，使用完之后必须释放管程并唤醒入口等待队列中的进程。
- 当一个进程试图进入管程时，在**入口等待队列**等待。若P进程唤醒了Q进程，则Q进程先执行，P在**紧急等待队列**中等待。（**HOARE管程**）
- wait操作：执行wait操作的进程进入条件变量链末尾，唤醒紧急等待队列或者入口队列中的进程；
- signal操作：唤醒条件变量链中的进程，自己进入紧急等待队列，若条件变量链为空，则继续执行。（**HOARE管程**）
- **MESA管程**：将HOARE中的signal换成了notify（或者broadcast通知所有满足条件的），进行通知而不是立马交换管程的使用权，在合适的时候，条件队列首位的进程可以进入，进入之前必须用while检查条件是否合适。优点：没有额外的进程切换

3.1、临界区的概念?

各个进程中对临界资源（互斥资源/共享变量，一次只能给一个进程使用）进行操作的程序片段

3.2、同步与互斥的概念?

- 同步：多个进程因为合作而使得进程的执行有一定的先后顺序。比如某个进程需要另一个进程提供的消息，获得消息之前进入阻塞态；
- 互斥：多个进程在同一时刻只有一个进程能进入临界区

3.3、并发、并行、异步的区别?

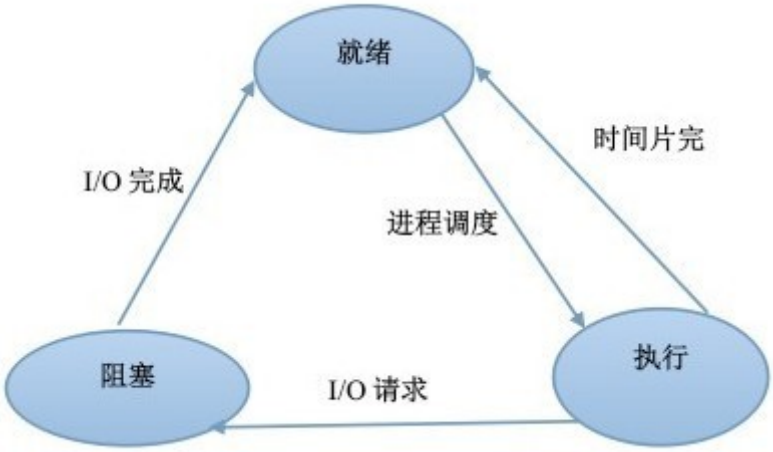
并发：在一个时间段中同时有多个程序在运行，但其实任一时刻，只有一个程序在CPU上运行，宏观上的并发是通过不断的切换实现的；

多线程：并发运行的一段代码。是实现异步的手段

并行（和串行相比）：在多CPU系统中，多个程序无论宏观还是微观上都是同时执行的

异步（和同步相比）：同步是顺序执行，异步是在等待某个资源的时候继续做自己的事

4、进程有哪几种状态?



- 就绪状态：进程已获得除处理机以外的所需资源，等待分配处理机资源
- 运行状态：占用处理机资源运行，处于此状态的进程数小于等于CPU数
- 阻塞状态： 进程等待某种条件，在条件满足之前无法执行

5、进程调度策略有哪些?

5.1、批处理系统:

5.1.1 先来先服务 first-come first-serverd（FCFS)

- 按照请求的顺序进行调度。非抢占式，开销小，无饥饿问题，响应时间不确定（可能很慢）；
- 对短进程不利，对IO密集型进程不利。

5.1.2 最短作业优先 shortest job first（SJF)

- 按估计运行时间最短的顺序进行调度。非抢占式，吞吐量高，开销可能较大，可能导致饥饿问题；
- 对短进程提供好的响应时间，对长进程不利。

5.1.3 最短剩余时间优先 shortest remaining time next（SRTN)

- 按剩余运行时间的顺序进行调度。(最短作业优先的抢占式版本)。吞吐量高，开销可能较大，提供好的响应时间；
- 可能导致饥饿问题，对长进程不利。

5.1.4 最高响应比优先 Highest Response Ratio Next（HRRN）

- 响应比 = 1+ 等待时间/处理时间。同时考虑了等待时间的长短和估计需要的执行时间长短，很好的平衡了长短进程。
- 非抢占，吞吐量高，开销可能较大，提供好的响应时间，无饥饿问题。

5.2、交互式系统

> 交互式系统有大量的用户交互操作，在该系统中调度算法的目标是快速地进行响应。

5.2.1 时间片轮转 Round Robin

- 将所有就绪进程按 FCFS 的原则排成一个队列，用完时间片的进程排到队列最后。
- 抢占式（时间片用完时），开销小，无饥饿问题，为短进程提供好的响应时间；
- 若时间片小，进程切换频繁，吞吐量低；若时间片太长，实时性得不到保证。

5.2.2 优先级调度算法

- 为每个进程分配一个优先级，按优先级进行调度。
- 为了防止低优先级的进程永远等不到调度，可以随着时间的推移增加等待进程的优先级。

5.2.3 多级反馈队列调度算法 Multilevel Feedback Queue

- 设置多个就绪队列1、2、3...，优先级递减，时间片递增。只有等到优先级更高的队列为空时才会调度当前队列中的进程。如果进程用完了当前队列的时间片还未执行完，则会被移到下一队列。
- 抢占式（时间片用完时），开销可能较大，对IO型进程有利，可能会出现饥饿问题。

5.3、什么叫优先级反转？如何解决？

高优先级的进程等待被一个低优先级进程占用的资源时，就会出现优先级反转，即优先级较低的进程比优先级较高的进程先执行。

解决方法：

- 优先级天花板(priority ceiling)：当任务申请某资源时，把该任务的优先级提升到可访问这个资源的所有任务中的最高优先级，这个优先级称为该资源的优先级天花板。简单易行。
- 优先级继承(priority inheritance)：当任务A申请共享资源S时，如果S正在被任务C使用，通过比较任务C与自身的优先级，如发现任务C的优先级小于自身的优先级，则将任务C的优先级提升到自身的优先级，任务C释放资源S后，再恢复任务C的原优先级。

6、什么是僵尸进程?

一个子进程结束后，它的父进程并没有等待它（调用wait或者waitpid），那么这个子进程将成为一个僵尸进程。僵尸进程是一个已经死亡的进程，但是并没有真正被销毁。它已经放弃了几乎所有内存空间，没有任何可执行代码，也不能被调度，仅仅在进程表中保留一个位置，记载该进程的进程ID、终止状态以及资源利用信息(CPU时间，内存使用量等等)供父进程收集，除此之外，僵尸进程不再占有任何内存空间。这个僵尸进程可能会一直留在系统中直到系统重启。

危害：占用进程号，而系统所能使用的进程号是有限的；占用内存。

以下情况不会产生僵尸进程：

- 该进程的父进程先结束了。每个进程结束的时候，系统都会扫描是否存在子进程，如果有则用Init进程接管，成为该进程的父进程，并且会调用wait等待其结束。
- 父进程调用wait或者waitpid等待子进程结束（需要每隔一段时间查询子进程是否结束）。wait系统调用会使父进程暂停执行，直到它的一个子进程结束为止。waitpid则可以加入`WNOHANG`(wait-no-hang)选项，如果没有发现结束的子进程，就会立即返回，不会将调用waitpid的进程阻塞。同时，waitpid还可以选择是等待任一子进程（同wait），还是等待指定pid的子进程，还是等待同一进程组下的任一子进程，还是等待组ID等于pid的任一子进程；
- 子进程结束时，系统会产生`SIGCHLD`(signal-child)信号，可以注册一个信号处理函数，在该函数中调用waitpid，等待所有结束的子进程（注意：一般都需要循环调用waitpid，因为在信号处理函数开始执行之前，可能已经有多个子进程结束了，而信号处理函数只执行一次，所以要循环调用将所有结束的子进程回收）；
- 也可以用`signal(SIGCLD, SIG_IGN)`(signal-ignore)通知内核，表示忽略`SIGCHLD`信号，那么子进程结束后，内核会进行回收。

什么是孤儿进程?

一个父进程已经结束了，但是它的子进程还在运行，那么这些子进程将成为孤儿进程。孤儿进程会被Init（进程ID为1）接管，当这些孤儿进程结束时由Init完成状态收集工作。

7、线程同步有哪些方式?

> 为什么需要线程同步：线程有时候会和其他线程共享一些资源，比如内存、数据库等。当多个线程同时读写同一份共享资源的时候，可能会发生冲突。因此需要线程的同步，多个线程按顺序访问资源。

- **互斥量** Mutex：互斥量是内核对象，只有拥有互斥对象的线程才有访问互斥资源的权限。因为互斥对象只有一个，所以可以保证互斥资源不会被多个线程同时访问；当前拥有互斥对象的线程处理完任务后必须将互斥对象交出，以便其他线程访问该资源；
- **信号量** Semaphore：信号量是内核对象，它允许同一时刻多个线程访问同一资源，但是需要控制同一时刻访问此资源的最大线程数量。信号量对象保存了**最大资源计数**和**当前可用资源计数**，每增加一个线程对共享资源的访问，当前可用资源计数就减1，只要当前可用资源计数大于0，就可以发出信号量信号，如果为0，则将线程放入一个队列中等待。线程处理完共享资源后，应在离开的同时通过`ReleaseSemaphore`函数将当前可用资源数加1。如果信号量的取值只能为0或1，那么信号量就成为了互斥量；
- **事件** Event：允许一个线程在处理完一个任务后，主动唤醒另外一个线程执行任务。事件分为手动重置事件和自动重置事件。手动重置事件被设置为激发状态后，会唤醒所有等待的线程，而且一直保持为激发状态，直到程序重新把它设置为未激发状态。自动重置事件被设置为激发状态后，会唤醒**一个**等待中的线程，然后自动恢复为未激发状态。
- **临界区** Critical Section：任意时刻只允许一个线程对临界资源进行访问。拥有临界区对象的线程可以访问该临界资源，其它试图访问该资源的线程将被挂起，直到临界区对象被释放。

互斥量和临界区有什么区别?

互斥量是可以命名的，可以用于不同进程之间的同步；而临界区只能用于同一进程中线程的同步。创建互斥量需要的资源更多，因此临界区的优势是速度快，节省资源。

8、什么是协程?

协程是一种用户态的轻量级线程，协程的调度完全由用户控制。协程拥有自己的寄存器上下文和栈。协程调度切换时，将寄存器上下文和栈保存到其他地方，在切回来的时候，恢复先前保存的寄存器上下文和栈，直接操作栈则基本没有内核切换的开销，可以不加锁的访问全局变量，所以上下文的切换非常快。

8.1、协程多与线程进行比较?

- 一个线程可以拥有多个协程，一个进程也可以单独拥有多个协程，这样python中则能使用多核CPU。
- 线程进程都是同步机制，而协程则是异步
- 协程能保留上一次调用时的状态，每次过程重入时，就相当于进入上一次调用的状态

9、什么是IO多路复用？怎么实现？

IO多路复用（IO Multiplexing）是指单个进程/线程就可以同时处理多个IO请求。

实现原理：用户将想要监视的文件描述符（File Descriptor）添加到select/poll/epoll函数中，由内核监视，函数阻塞。一旦有文件描述符就绪（读就绪或写就绪），或者超时（设置timeout），函数就会返回，然后该进程可以进行相应的读/写操作。

- `select`：将文件描述符放入一个集合中，调用select时，将这个集合从用户空间拷贝到内核空间（缺点1：每次都要复制，**开销大**），由内核根据就绪状态修改该集合的内容。（缺点2）**集合大小有限制**，32位机默认是1024（64位：2048）；采用水平触发机制。select函数返回后，需要通过遍历这个集合，找到就绪的文件描述符（缺点3：**轮询的方式效率较低**），当文件描述符的数量增加时，效率会线性下降；
- `poll`：和select几乎没有区别，区别在于文件描述符的存储方式不同，poll采用链表的方式存储，没有最大存储数量的限制；
- `epoll`：通过内核和用户空间共享内存，避免了不断复制的问题；支持的同时连接数上限很高（1G左右的内存支持10W左右的连接数）；文件描述符就绪时，采用回调机制，避免了轮询（回调函数将就绪的描述符添加到一个链表中，执行epoll_wait时，返回这个链表）；支持水平触发和边缘触发，采用边缘触发机制时，只有活跃的描述符才会触发回调函数。

总结，区别主要在于：

- 一个线程/进程所能打开的最大连接数
- 文件描述符传递方式（是否复制）
- 水平触发 or 边缘触发
- 查询就绪的描述符时的效率（是否轮询）

9.1、有哪些常见的IO模型？

- 同步阻塞IO（Blocking IO）：用户线程发起IO读/写操作之后，线程阻塞，直到可以开始处理数据；对CPU资源的利用率不够；
- 同步非阻塞IO（Non-blocking IO）：发起IO请求之后可以立即返回，如果没有就绪的数据，需要不断地发起IO请求直到数据就绪；不断重复请求消耗了大量的CPU资源；
- IO多路复用
- 异步IO（Asynchronous IO）：用户线程发出IO请求之后，继续执行，由内核进行数据的读取并放在用户指定的缓冲区内，在IO完成之后通知用户线程直接使用。

9.2、什么是文件描述符？

文件描述符在形式上是一个非负整数。实际上，它是一个索引值，指向内核为每一个进程所维护的该进程打开文件的记录表。当程序打开一个现有文件或者创建一个新文件时，内核向进程返回一个文件描述符。

内核通过文件描述符来访问文件。文件描述符指向一个文件。

9.3、什么时候使用select/poll，什么时候使用epoll？

当连接数较多并且有很多的不活跃连接时，epoll的效率比其它两者高很多；但是当连接数较少并且都十分活跃的情况下，由于epoll需要很多回调，因此性能可能低于其它两者。

9.4、什么是水平触发？什么是边缘触发？

- 水平触发（LT，Level Trigger）模式下，只要一个文件描述符就绪，就会触发通知，如果用户程序没有一次性把数据读写完，下次还会通知；
- 边缘触发（ET，Edge Trigger）模式下，当描述符从未就绪变为就绪时通知一次，之后不会再通知，直到再次从未就绪变为就绪（缓冲区从不可读/写变为可读/写）。
- 区别：边缘触发效率更高，减少了被重复触发的次数，函数不会返回大量用户程序可能不需要的文件描述符。
- 为什么边缘触发一定要用非阻塞（non-block）IO：避免由于一个描述符的阻塞读/阻塞写操作让处理其它描述符的任务出现饥饿状态。

10、什么是用户态和内核态？

为了限制不同程序的访问能力，防止一些程序访问其它程序的内存数据，CPU划分了用户态和内核态两个权限等级。

- 用户态只能受限地访问内存，且不允许访问外围设备，没有占用CPU的能力，CPU资源可以被其它程序获取；
- 内核态可以访问内存所有数据以及外围设备，也可以进行程序的切换。

所有用户程序都运行在用户态，但有时需要进行一些内核态的操作，比如从硬盘或者键盘读数据，这时就需要进行系统调用，使用**陷阱指令**，CPU切换到内核态，执行相应的服务，再切换为用户态并返回系统调用的结果。

10.1、为什么要分用户态和内核态？

- 安全性：防止用户程序恶意或者不小心破坏系统/内存/硬件资源；
- 封装性：用户程序不需要实现更加底层的代码；
- 利于调度：如果多个用户程序都在等待键盘输入，这时就需要进行调度；统一交给操作系统调度更加方便。

10.2、如何从用户态切换到内核态？

- 系统调用：比如读取命令行输入。本质上还是通过中断实现
- 用户程序发生异常时：比如缺页异常
- 外围设备的中断：外围设备完成用户请求的操作之后，会向CPU发出中断信号，这时CPU会转去处理对应的中断处理程序

11、什么是死锁?

在两个或者多个并发进程中，每个进程持有某种资源而又等待其它进程释放它们现在保持着的资源，在未改变这种状态之前都不能向前推进，称这一组进程产生了死锁(deadlock)。

12、死锁产生的必要条件?

- **互斥**：一个资源一次只能被一个进程使用；
- **占有并等待**：一个进程至少占有一个资源，并在等待另一个被其它进程占用的资源；
- **非抢占**：已经分配给一个进程的资源不能被强制性抢占，只能由进程完成任务之后自愿释放；
- **循环等待**：若干进程之间形成一种头尾相接的环形等待资源关系，该环路中的每个进程都在等待下一个进程所占有的资源。

13、死锁有哪些处理方法?

13.1、 鸵鸟策略

直接忽略死锁。因为解决死锁问题的代价很高，因此鸵鸟策略这种不采取任务措施的方案会获得更高的性能。当发生死锁时不会对用户造成多大影响，或发生死锁的概率很低，可以采用鸵鸟策略。

13.2、死锁预防

基本思想是破坏形成死锁的四个必要条件：

- 破坏互斥条件：允许某些资源同时被多个进程访问。但是有些资源本身并不具有这种属性，因此这种方案实用性有限；
- 破坏占有并等待条件：
    - 实行资源预先分配策略（当一个进程开始运行之前，必须一次性向系统申请它所需要的全部资源，否则不运行）；
    - 或者只允许进程在没有占用资源的时候才能申请资源（申请资源前先释放占有的资源）；
    - 缺点：很多时候无法预知一个进程所需的全部资源；同时，会降低资源利用率，降低系统的并发性；
- 破坏非抢占条件：允许进程强行抢占被其它进程占有的资源。会降低系统性能；
- 破坏循环等待条件：对所有资源统一编号，所有进程对资源的请求必须按照序号递增的顺序提出，即只有占有了编号较小的资源才能申请编号较大的资源。这样避免了占有大号资源的进程去申请小号资源。

13.3、死锁避免

动态地检测资源分配状态，以确保系统处于安全状态，只有处于安全状态时才会进行资源的分配。所谓安全状态是指：即使所有进程突然请求需要的所有资源，也能存在某种对进程的资源分配顺序，使得每一个进程运行完毕。

> 银行家算法

13.4、死锁解除

死锁解除的方法：

- 利用抢占：挂起某些进程，并抢占它的资源。但应防止某些进程被长时间挂起而处于饥饿状态；
- 利用回滚：让某些进程回退到足以解除死锁的地步，进程回退时自愿释放资源。要求系统保持进程的历史信息，设置还原点；

- 利用杀死进程：强制杀死某些进程直到死锁解除为止，可以按照优先级进行。

14、分页和分段有什么区别?

- 页式存储：用户空间划分为大小相等的部分称为页（page），内存空间划分为同样大小的区域称为页框，分配时以页为单位，按进程需要的页数分配，逻辑上相邻的页物理上不一定相邻；
- 段式存储：用户进程地址空间按照自身逻辑关系划分为若干个段（segment）（如代码段，数据段，堆栈段），内存空间被动态划分为长度不同的区域，分配时以段为单位，每段在内存中占据连续空间，各段可以不相邻；
- 段页式存储：用户进程先按段划分，段内再按页划分，内存划分和分配按页。

区别：

- 目的不同：分页的目的是管理内存，用于虚拟内存以获得更大的地址空间；分段的目的是满足用户的需要，使程序和数据可以被划分为逻辑上独立的地址空间；
- 大小不同：段的大小不固定，由其所完成的功能决定；页的大小固定，由系统决定；
- 地址空间维度不同：分段是二维地址空间（段号+段内偏移），分页是一维地址空间（每个进程一个页表/多级页表，通过一个逻辑地址就能找到对应的物理地址）；
- 分段便于信息的保护和共享；分页的共享收到限制；
- 碎片：分段没有内碎片，但会产生外碎片；分页没有外碎片，但会产生内碎片（一个页填不满）

15、什么是虚拟内存?

每个程序都拥有自己的地址空间，这个地址空间被分成大小相等的页，这些页被映射到物理内存；但不需要所有的页都在物理内存中，当程序引用到不在物理内存中的页时，由操作系统将缺失的部分装入物理内存。这样，对于程序来说，逻辑上似乎有很大的内存空间，只是实际上有一部分是存储在磁盘上，因此叫做虚拟内存。

虚拟内存的优点是让程序可以获得更多的可用内存。

15.1、如何进行地址空间到物理内存的映射?

内存管理单元（MMU）管理着逻辑地址和物理地址的转换，其中的页表（Page table）存储着页（逻辑地址）和页框（物理内存空间）的映射表，页表中还包含包含有效位（是在内存还是磁盘）、访问位（是否被访问过）、修改位（内存中是否被修改过）、保护位（只读还是可读写）。逻辑地址：页号+页内地址（偏移）；每个进程一个页表，放在内存，页表起始地址在PCB/寄存器中。

16、有哪些页面置换算法?

在程序运行过程中，如果要访问的页面不在内存中，就发生缺页中断从而将该页调入内存中。此时如果内存已无空闲空间，系统必须从内存中调出一个页面到磁盘中来腾出空间。页面置换算法的主要目标是使页面置换频率最低（也可以说缺页率最低）。

- **最佳页面置换算法** OPT（Optimal replacement algorithm）：置换以后不需要或者最远的将来才需要的页面，是一种理论上的算法，是最优策略；
- **先进先出** FIFO：置换在内存中驻留时间最长的页面。缺点：有可能将那些经常被访问的页面也被换出，从而使缺页率升高；
- **第二次机会算法** SCR：按FIFO选择某一页面，若其访问位为1，给第二次机会，并将访问位置0；
- **时钟算法** Clock：SCR中需要将页面在链表中移动（第二次机会的时候要将这个页面从链表头移到链表尾），时钟算法使用环形链表，再使用一个指针指向最老的页面，避免了移动页面的开销；
- **最近未使用算法** NRU（Not Recently Used）：检查访问位R、修改位M，优先置换R=M=0，其次是（R=0, M=1）；
- **最近最少使用算法** LRU（Least Recently Used）：置换出未使用时间最长的一页；实现方式：维护时间戳，或者维护一个所有页面的链表。当一个页面被访问时，将这个页面移到链表表头。这样就能保证链表表尾的页面是最近最久未访问的。
- **最不经常使用算法** NFU：置换出访问次数最少的页面

16.1、局部性原理

- 时间上：最近被访问的页在不久的将来还会被访问；
- 空间上：内存中被访问的页周围的页也很可能被访问。

16.2、什么是颠簸现象

颠簸本质上是指频繁的页调度行为。进程发生缺页中断时必须置换某一页。然而，其他所有的页都在使用，它置换一个页，但又立刻再次需要这个页。因此会不断产生缺页中断，导致整个系统的效率急剧下降，这种现象称为颠簸。内存颠簸的解决策略包括：

- 修改页面置换算法；
- 降低同时运行的程序的数量；
- 终止该进程或增加物理内存容量。

17、磁盘调度

过程：磁头（找到对应的盘面）；磁道（一个盘面上的同心圆环，寻道时间）；扇区（旋转时间）。为减小寻道时间的调度算法：

- 先来先服务
- 最短寻道时间优先
- 电梯算法：电梯总是保持一个方向运行，直到该方向没有请求为止，然后改变运行方向。

数据库

1、事务的概念和特性?

概念：事务（Transaction）是一个操作序列，不可分割的工作单位，以BEGIN TRANSACTION开始，以ROLLBACK/COMMIT结束

特性（ACID）：

- **原子性**（Atomicity）：逻辑上是不可分割的操作单元，事务的所有操作要么全部提交成功，要么全部失败回滚（用回滚日志实现，反向执行日志中的操作）；
- **一致性**（Consistency）：事务的执行必须使数据库保持一致性状态。在一致性状态下，所有事务对一个数据的读取结果都是相同的；
- **隔离性**（Isolation）：一个事务所做的修改在最终提交以前，对其它事务是不可见的（并发执行的事务之间不能相互影响）；
- **持久性**（Durability）：一旦事务提交成功，对数据的修改是永久性的

2、会出现哪些并发一致性问题?

- **丢失修改**：一个事务对数据进行了修改，在事务提交之前，另一个事务对同一个数据进行了修改，覆盖了之前的修改；
- **脏读**（Dirty Read）：一个事务读取了被另一个事务修改、但未提交（进行了回滚）的数据，造成两个事务得到的数据不一致；
- **不可重复读**（Nonrepeatable Read）：在同一个事务中，某查询操作在一个时间读取某一行数据和之后一个时间读取该行数据，发现数据已经发生修改（针对**update**操作）；
- **幻读**（Phantom Read）：当同一查询多次执行时，由于其它事务在这个数据范围内执行了插入操作，会导致每次返回不同的结果集（和不可重复读的区别：针对的是一个数据整体/范围；并且针对**insert/delete**操作）

3、数据库的四种隔离级别?

- **未提交读**（Read Uncommited）：在一个事务提交之前，它的执行结果对其它事务也是可见的。会导致脏读、不可重复读、幻读；
- **提交读**（Read Commited）：一个事务只能看见已经提交的事务所作的改变。可避免脏读问题；
- **可重复读**（Repeatable Read）：可以确保同一个事务在多次读取同样的数据时得到相同的结果。（MySQL的默认隔离级别）。可避免不可重复读；
- **可串行化**（Serializable）：强制事务串行执行，使之不可能相互冲突，从而解决幻读问题。可能导致大量的超时现象和锁竞争，实际很少使用。

4、什么是乐观锁和悲观锁?

- 悲观锁：认为数据随时会被修改，因此每次读取数据之前都会上锁，防止其它事务读取或修改数据；应用于**数据更新比较频繁**的场景；
- 乐观锁：操作数据时不会上锁，但是更新时会判断在此期间有没有别的事务更新这个数据，若被更新过，则失败重试；适用于**读多写少**的场景。乐观锁的实现方式有：
    - 加一个版本号或者时间戳字段，每次数据更新时同时更新这个字段；
    - 先读取想要更新的字段或者所有字段，更新的时候比较一下，只有字段没有变化才进行更新

5、常见的封锁类型?

1. 意向锁是 InnoDB 自动加的， 不需用户干预。
2. 对于 UPDATE、 DELETE 和 INSERT 语句， InnoDB会自动给涉及数据集加排他锁（X);
3. 对于普通 SELECT 语句，InnoDB 不会加任何锁;
4. 事务可以通过以下语句显式给记录集加共享锁或排他锁： **共享锁（S）**：SELECT * FROM table_name WHERE ... LOCK IN SHARE MODE。 其他 session 仍然可以查询记录，并也可以对该记录加 share mode 的共 享锁。但是如果当前事务需要对该记录进行更新操作，则很有可能造成死锁。 **排他锁（X)**：SELECT * FROM table_name WHERE ... FOR UPDATE。其他 session 可以查询该记录，但是不能对该记录加共享锁或排他锁，而是等待获得锁

- **排它锁**（Exclusive Lock）/ X锁：事务对数据加上X锁时，只允许此事务读取和修改此数据，并且其它事务不能对该数据加任何锁;
- **共享锁**（Shared Lock）/ S锁：加了S锁后，该事务只能对数据进行读取而不能修改，并且其它事务只能加S锁，不能加X锁
- **意向锁**（Intention Locks）：
    - 一个事务在获得某个**数据行**对象的 S 锁之前，必须先获得**整个表**的 IS 锁或更强的锁;
    - 一个事务在获得某个数据行对象的 X 锁之前，必须先获得整个表的 IX 锁;
    - IS/IX 锁之间都是兼容的;
    - 好处：如果一个事务想要对整个表加X锁，就需要先检测是否有其它事务对该表或者该表中的某一行加了锁，这种检测非常耗时。有了意向锁之后，只需要检测整个表是否存在IX/IS/X/S锁就行了

锁的作用：用于管理对共享资源的并发访问，保证数据库的完整性和一致性

5.1、封锁粒度的概念

MySQL 中提供了两种封锁粒度：**行级锁**以及**表级锁**。

封锁粒度小：

- 好处：锁定的数据量越少，发生锁争用的可能就越小，系统的**并发程度**就越高;
- 坏处：**系统开销**大（加锁、释放锁、检查锁的状态都需要消耗资源）

5.2、MySQL加锁

```
SELECT ... LOCK In SHARE MODE;
SELECT ... FOR UPDATE;
```

6、什么是三级封锁协议?

- 一级封锁协议：事务在修改数据之前必须先对其加X锁，直到事务结束才释放。可以解决丢失修改问题（两个事务不能同时对一个数据加X锁，避免了修改被覆盖）;
- 二级封锁协议：在一级的基础上，事务在读取数据之前必须先加S锁，读完后释放。可以解决脏读问题（如果已经有事务在修改数据，就意味着已经加了X锁，此时想要读取数据的事务并不能加S锁，也就无法进行读取，避免了读取脏数据）;
- 三级封锁协议：在二级的基础上，事务在读取数据之前必须先加S锁，直到事务结束才能释放。可以解决不可重复读问题（避免了在事务结束前其它事务对数据加X锁进行修改，保证了事务期间数据不会被其它事务更新）

7、什么是两段锁协议?

事务必须严格分为两个阶段对数据进行**加锁和解锁**的操作，第一阶段加锁，第二阶段解锁。也就是说一个事务中一旦释放了锁，就不能再申请新锁了。

可串行化调度是指，通过并发控制，使得并发执行的事务结果与某个串行执行的事务结果相同。事务遵循两段锁协议是保证可串行化调度的充分条件。

8、什么是 MVCC?

多版本并发控制（Multi-Version Concurrency Control, MVCC），MVCC在每行记录后面都保存有两个隐藏的列，用来存储**创建版本号**和**删除版本号**。

- 创建版本号：创建一个数据行时的事务版本号（**事务版本号**：事务开始时的系统版本号；系统版本号：每开始一个新的事务，系统版本号就会自动递增）；
- 删除版本号：删除操作时的事务版本号；
- 各种操作：
    - 插入操作时，记录创建版本号；
    - 删除操作时，记录删除版本号；
    - 更新操作时，先记录删除版本号，再新增一行记录创建版本号；
    - 查询操作时，要符合以下条件才能被查询出来：删除版本号未定义或大于当前事务版本号（删除操作是在当前事务启动之后做的）；创建版本号小于或等于当前事务版本号（创建操作是事务完成或者在事务启动之前完成）

通过版本号减少了锁的争用，**提高了系统性能**；可以实现**提交读**和**可重复读**两种隔离级别，未提交读无需使用MVCC

快照读与当前读

使用 MVCC 读取的是快照中的数据，这样可以减少加锁所带来的开销：

```
select * from table ...;
```

当前读读取的是最新的数据，需要加锁。以下第一个语句需要加 S 锁，其它都需要加 X 锁：

```
select * from table where ? lock in share mode;
select * from table where ? for update;
insert;
update;
delete;
```

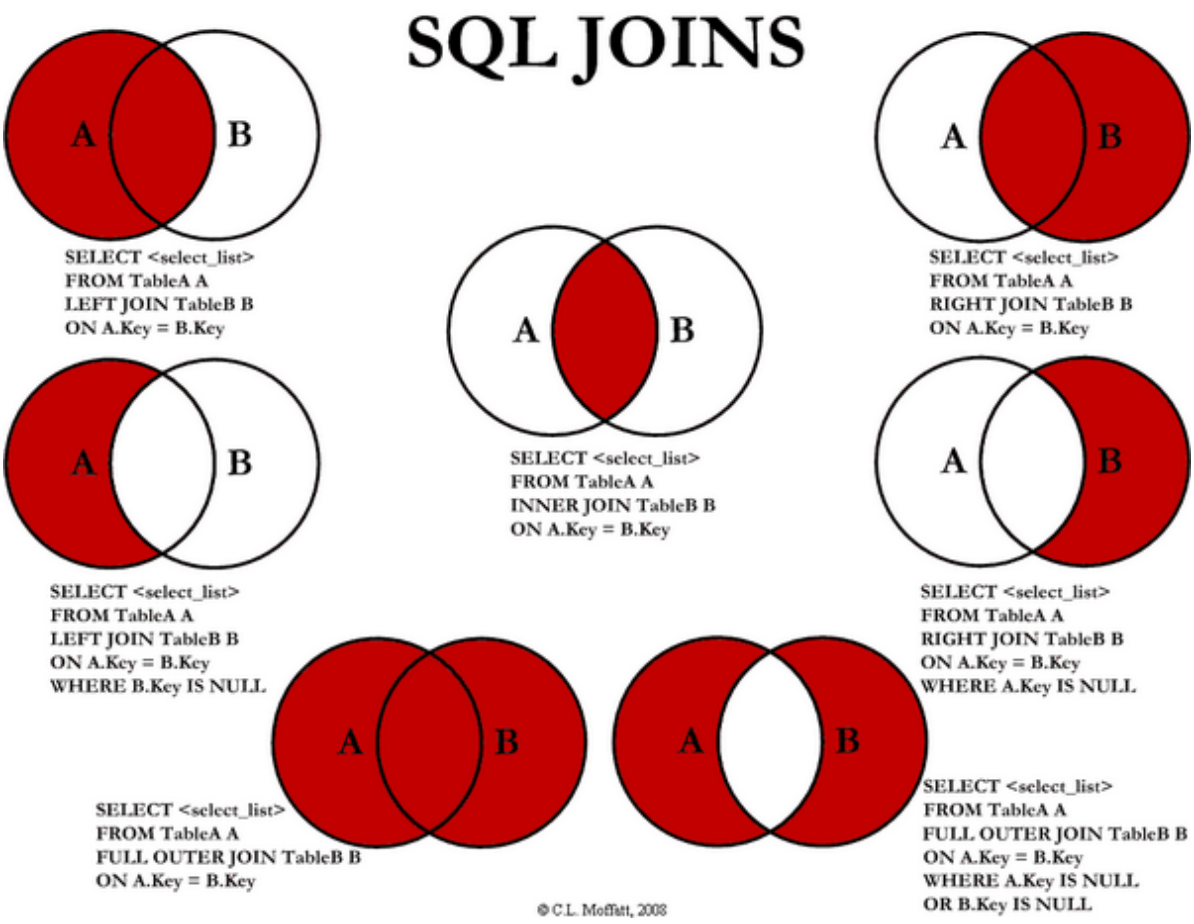
9、数据库的范式?

- **第一范式**（1NF，Normal Form）：**属性不应该是可分的。**举例：如果将“电话”作为一个属性（一列），是不符合1NF的，因为电话这个属性可以分解为家庭电话和移动电话...如果将“移动电话”作为一个属性，就符合1NF；
- **第二范式** 2NF：每个非主属性**完全依赖**于主属性集（候选键集）；
    - B完全依赖于A，就是说A中的所有属性唯一决定B，属性少了就不能唯一决定，属性多了则有冗余（叫依赖不叫完全依赖）。举例：（学号，课程名）这个主属性集可以唯一决定成绩，但是对于学生姓名这个属性，（学号，课程名）这个属性集就是冗余的，所以学生姓名不完全依赖于（学号，课程名）这一属性集；
    - 主属性集/候选码集：某一组属性能够唯一确定其它的属性（主键就是从候选键集中选的一个键），而其子集不能，这样的属性组中的属性就是主属性；不在候选码集中的属性成为非主属性；
    - 可以通过分解来满足 2NF：将（学号，课程名，成绩）做成一张表；（学号，学生姓名）做成另一张表，避免大量的数据冗余； 满足1NF后，要求表中的所有列，都必须依赖于主键，而不能有任何一列与主键没有关系，也就是说一个表只描述一件事情；
- **第三范式** 3NF：在 2NF 的基础上，非主属性**不传递依赖**于主属性
    - 传递依赖：如果C依赖于B，B依赖于A，那么C传递依赖于A；
    - 3NF在2NF的基础上，消除了非主属性之间的依赖；比如一个表中，主属性有（学号），非主属性有（姓名，院系，院长名），可以看到院长名这个非主属性依赖于院系，传递依赖于学号。消除的办法是分解。 必须先满足第二范式（2NF），要求：表中的每一列只与主键直接相关而不是间接相关，（表中的每一列只能依赖于主键）；

9.1、不符合范式会出现哪些异常

- 冗余数据：某些同样的数据多次出现（如学生姓名）；
- 修改异常：修改了一个记录中的信息，另一个记录中相同的信息却没有修改；
- 删除异常：删除一个信息，那么也会丢失其它信息（删除一个课程，丢失了一个学生的信息）；
- 插入异常：无法插入（插入一个还没有课程信息的学生）

10、列举几种表连接方式?



- 内连接（Inner Join）：仅将两个表中满足连接条件的行组合起来作为结果集
    - 自然连接：只考虑属性相同的元组对；
    - 等值连接：给定条件进行查询
- 外连接（Outer Join）
    - 左连接：左边表的所有数据都有显示出来，右边的表数据只显示共同有的那部分，没有对应的部分补NULL；
    - 右连接：和左连接相反；
    - 全外连接（Full Outer Join）：查询出左表和右表所有数据，但是去除两表的重复数据
- 交叉连接（Cross Join）：返回两表的笛卡尔积（对于所含数据分别为m、n的表，返回m*n的结果）

11、什么是存储过程？有哪些优缺点?

存储过程是事先经过编译并存储在数据库中的一段SQL语句的集合。想要实现相应的功能时，只需要调用这个存储过程就行了（类似于函数，输入具有输出参数）。

优点：

- 预先编译，而不需要每次运行时编译，提高了数据库执行**效率**；
- 封装了一系列操作，对于一些数据交互比较多的操作，相比于单独执行SQL语句，可以**减少网络通信量**；
- 具有**可复用性**，减少了数据库开发的工作量；
- **安全性高**，可以让没有权限的用户通过存储过程间接操作数据库；
- 更**易于维护**

缺点：

- **可移植性差**，存储过程将应用程序绑定到了数据库上；
- **开发调试复杂**：没有好的IDE；
- **修改复杂**，需要重新编译，有时还需要更新程序中的代码以更新调用

12、Drop/Delete/Truncate的区别?

- **Delete**用来删除表的全部或者**部分数据**，执行delete之后，用户**需要提交**之后才会执行，会触发表上的DELETE**触发器**（包含一个OLD的虚拟表，可以只读访问被删除的数据），DELETE之后表结构还在，删除很慢，一行一行地删，因为会记录日志，可以利用日志还原数据；
- **Truncate**删除表中的所有数据，这个操作**不能回滚**，也不会触发这个表上的触发器。操作比DELETE快很多（直接把表drop掉，再创建一个新表，删除的数据不能找回）。如果表中有自增（AUTO_INCREMENT）列，则重置为1；
- **Drop**命令从数据库中**删除表**，所有的数据行，索引和约束都会被删除；不能回滚，不会触发触发器；

12.1、什么是触发器?

触发器（TRIGGER）是由事件（比如INSERT/UPDATE/DELETE）来触发运行的操作（不能被直接调用，不能接收参数）。在数据库里以独立的对象存储，用于**保证数据完整性**（比如可以检验或转换数据）。

12.2、有哪些约束类型?

约束（Constraint）类型：主键（Primary Key）约束，唯一约束（Unique），检查约束，非空约束，外键（Foreign Key）约束。

13、什么是视图？什么是游标？

- 视图：从数据库的基本表中通过查询选取出来的数据组成的**虚拟表**（数据库中存放视图的定义）。可以对其进行增/删/改/查等操作。视图是对若干张基本表的引用，一张虚表，查询语句执行的结果，不存储具体的数据（基本表数据发生了改变，视图也会跟着改变）；可以跟基本表一样，进行增删改查操作(ps:增删改操作有条件限制)；如连表查询产生的视图无法进行，对视图的增删改会影响原表的数据。好处：
    - 通过只给用户访问视图的权限，保证数据的**安全性**；
    - **简化**复杂的SQL操作，隐藏数据的复杂性（比如复杂的连接）；
- 游标（Cursor）：用于定位在查询返回的**结果集的特定行**，以对特定行进行操作。使用游标可以方便地对结果集进行移动遍历，根据需要滚动或对浏览/修改任意行中的数据。主要用于交互式应用。

14、数据库索引的实现原理（B+树）

14.1、使用B树和B+树的比较

InnoDB的索引使用的是B+树实现，B+树对比B树的好处：

- IO次数少：B+树的中间结点只存放索引，数据都存在叶结点中，因此中间结点可以存更多的数据，让索引树更加矮胖；
- 范围查询效率更高：B树需要中序遍历整个树，只B+树需要遍历叶结点中的链表；
- 查询效率更加稳定：每次查询都需要从根结点到叶结点，路径长度相同，所以每次查询的效率都差不多

14.2、使用B树索引和哈希索引的比较

哈希索引能以 O(1) 时间进行查找，但是只支持精确查找，无法用于部分查找和范围查找，无法用于排序与分组；B树索引支持大于小于等于查找，范围查找。哈希索引遇到大量哈希值相等的情况后查找效率会降低。哈希索引不支持数据的排序。

15、使用索引的优点

- 大大加快了数据的**检索速度**；
- 可以显著减少查询中**分组和排序**的时间；
- 通过创建唯一性索引，可以保证数据库表中每一行数据的唯一性；
- 将随机 I/O 变为**顺序 I/O**（B+Tree 索引是有序的，会将相邻的数据都存储在一起）

缺点：建立和维护索引耗费时间空间，更新索引很慢。

16、哪些情况下索引会失效?

- 以“%(表示任意0个或多个字符)”开头的LIKE语句；
- OR语句前后没有同时使用索引；
- 数据类型出现隐式转化（如varchar不加单引号的话可能会自动转换为int型）；
- 对于多列索引，必须满足 **最左匹配原则**/最左前缀原则 (最左优先，eg：多列索引col1、col2和col3，则 索引生效的情形包括 col1或col1，col2或col1，col2，col3)；
- 如果MySQL估计全表扫描比索引快，则不使用索引（比如非常小的表）

17、在哪些地方适合创建索引?

- 某列经常作为最大最小值;
- 经常被查询的字段;
- 经常用作表连接的字段;
- 经常出现在ORDER BY/GROUP BY/DISDINCT后面的字段

17.1、创建索引时需要注意什么?

- 只应建立在**小字段**上，而不要对大文本或图片建立索引（一页存储的数据越多一次IO操作获取的数据越大效率越高）;
- 建立索引的字段应该**非空**，在MySQL中，含有空值的列很难进行查询优化，因为它们使得索引、索引的统计信息以及比较运算更加复杂。应该用0、一个特殊的值或者一个空串代替NULL;
- 选择**数据密度大**（唯一值占总数的百分比很大）的字段作索引

18、索引的分类?

- 普通索引
- 唯一索引 UNIQUE：索引列的值必须唯一，但允许有空值;
- 主键索引 PRIMARY KEY：必须唯一，不允许空值（是一种特殊的唯一索引；MySQL创建主键时默认为聚集索引，但主键也可以是非聚集索引）;
- 单列索引和多列索引/复合索引（Composite）：索引的列数;
- 覆盖（Covering）索引：索引包含了所有满足查询所需要的数据，查询的时候只需要读取索引而不需要回表读取数据;
- 聚集（Clustered）索引/非聚集索引：对磁盘上存放数据的物理地址重新组织以使这些数据按照指定规则排序的一种索引（数据的物理排列顺序和索引排列顺序一致）。因此每张表只能创建一个聚集索引（因为要改变物理存储顺序）。优点是查询速度快，因为可以直接按照顺序得到需要数据的物理地址。缺点是进行修改的速度较慢。对于需要经常搜索范围的值很有效。非聚集索引只记录逻辑顺序，并不改变物理顺序;
- 分区索引（？）
- 虚拟索引（Virtual）：模拟索引的存在而不用真正创建一个索引，用于快速测试创建索引对执行计划的影响。没有相关的索引段，不增加存储空间的使用

19、MySQL的两种存储引擎 InnoDB 和 MyISAM 的区别?

- InnoDB**支持事务**，可以进行Commit和Rollback;
- MyISAM 只支持表级锁，而 InnoDB 还**支持行级锁**，提高了并发操作的性能;
- InnoDB **支持外键**;
- MyISAM **崩溃**后发生损坏的概率比 InnoDB 高很多，而且**恢复的速度**也更慢;
- MyISAM 支持**压缩**表和空间数据索引，InnoDB需要更多的内存和存储;
- InnoDB 支持在线**热备份**

19.1、应用场景

- **MyISAM** 管理非事务表。它提供高速存储和检索（MyISAM强调的是性能，每次查询具有原子性，其执行速度比InnoDB更快），以及全文搜索能力。如果表比较小，或者是只读数据（有大量的SELECT），还是可以使用MyISAM;
- **InnoDB** 支持事务，并发情况下有很好的性能，基本可以替代MyISAM

19.2、热备份和冷备份

- 热备份：在数据库运行的情况下备份的方法。优点：可按表或用户备份，备份时数据库仍可使用，可恢复至任一时间点。但是不能出错
- 冷备份：数据库正常关闭后，将关键性文件复制到另一位置的备份方式。优点：操作简单快速，恢复简单

20、如何优化数据库?

20.1、SQL 语句的优化

> 分析慢查询日志：记录了在MySQL中响应时间超过阀值long_query_time的SQL语句，通过日志去找出IO大的SQL以及发现未命中索引的SQL

> 使用 Explain 进行分析：通过explain命令可以得到表的读取顺序、数据读取操作的操作类型、哪些索引可以使用、**哪些索引被实际使用**、表之间的引用以及**被扫描的行数**等问题；

- 应尽量避免在 where 子句中使用`!=`、`<`、`>`操作符或对字段进行null值判断，否则将引擎放弃使用索引而进行全表扫描；
- 只返回必要的列：最好不要使用 SELECT * 语句；
- 只返回必要的行：使用 LIMIT 语句来限制返回的数据；
- 将一个大连接查询分解成对每一个表进行一次单表查询，然后在应用程序中进行关联，这样做的好处有：
    - 让缓存更高效。对于连接查询，如果其中一个表发生变化，那么整个查询缓存就无法使用。而分解后的多个查询，即使其中一个表发生变化，对其它表的查询缓存依然可以使用；
    - 分解成多个单表查询，这些单表查询的缓存结果更可能被其它查询使用到，从而减少冗余的查询；
    - 减少锁竞争

20.2、索引的优化

注意会引起索引失效的情况，以及在适合的地方建立索引

20.3、数据库表结构的优化

- 设计表时遵循**三大范式**；
- 选择合适的**数据类型**：尽可能不要存储NULL字段；使用简单的数据类型（int, varchar/text）；
- 表的**水平切分**（Sharding）：将同一个表中的记录拆分到多个结构相同的表中（策略：哈希取模；根据ID范围来分）。当一个表的数据不断增多时，Sharding 是必然的选择，它可以将数据分布到集群的不同节点上，从而缓解单个数据库的压力；
- 表的**垂直切分**：将一张表按列切分成多个表。可以将不常用的字段单独放在同一个表中；把大字段独立放入一个表中；或者把经常使用的字段（关系密切的）放在一张表中。垂直切分之后业务更加清晰，系统之间整合或扩展容易，数据维护简单

20.4、系统配置的优化

- 操作系统：增加TCP支持的队列数；
- MySQL配置文件优化：缓存池大小和个数设置

20.5、硬件的优化

- 磁盘性能：固态硬盘；
- CPU：多核且高频；
- 内存：增大内存

21、什么是主从复制？实现原理是什么？

主从复制（Replication）是指数据可以从一个MySQL数据库主服务器复制到一个或多个从服务器，从服务器可以复制主服务器中的所有数据库或者特定的数据库，或者特定的表。默认采用异步模式。

实现原理：

- 主服务器 **binary log dump 线程**：将主服务器中的数据更改（增删改）日志写入 Binary log 中；
- 从服务器 **I/O 线程**：负责从主服务器读取binary log，并写入本地的 Relay log；
- 从服务器 **SQL 线程**：负责读取 Relay log，解析出主服务器已经执行的数据更改，并在从服务器中重新执行（Replay），保证主从数据的一致性

21.1、为什么要主从复制？

- 读写分离：主服务器负责写，从服务器负责读
    - 缓解了锁的争用，即使主服务器中加了锁，依然可以进行读操作；
    - 从服务器可以使用 MyISAM，提升查询性能以及节约系统开销；
    - 增加冗余，提高可用性
- 数据实时备份，当系统中某个节点发生故障时，可以方便的故障切换
- 降低单个服务器磁盘I/O访问的频率，提高单个机器的I/O性能