计算机网络常问面试题

1、OSI， TCP/IP，五层协议的体系结构

```
1. OSI分层（7层）：物理层、数据链路层、网络层、传输层、会话层、表示层、应用层。
2. TCP/IP分层（4层）：网络接口层、网际层、运输层、应用层。
3. 五层协议（5层）：物理层、数据链路层、网络层、运输层、应用层。
```

2、IP 地址的分类

```
1. A类地址：以0开头，第一个字节范围：0~127；
2. B类地址：以10开头，第一个字节范围：128~191；
3. C类地址：以110开头，第一个字节范围：192~223；
4. D类地址：以1110开头，第一个字节范围为224~239；
```

3、ARP 协议的工作原理

```
1.首先，每台主机都会在自己的ARP缓冲区中建立一个 ARP列表，以表示IP地址和MAC地址的对应关系。

2.当源主机需要将一个数据包要发送到目的主机时，会首先检查自己 ARP列表中是否存在该 IP地址对应的MAC地址，如果有，就直接将数据包发送到这个MAC地址；如果没有，就向本地网段发起一个ARP请求的广播包，查询此目的主机对应的MAC地址。此ARP请求数据包里包括源主机的IP地址、硬件地址、以及目的主机的IP地址。

3.网络中所有的主机收到这个ARP请求后，会检查数据包中的目的IP是否和自己的IP地址一致。如果不相同就忽略此数据包；如果相同，该主机首先将发送端的MAC地址和IP地址添加到自己的ARP列表中，如果ARP表中已经存在该IP的信息，则将其覆盖，然后给源主机发送一个 ARP响应数据包，告诉对方自己是它需要查找的MAC地址；源主机收到这个ARP响应数据包后，将得到的目的主机的IP地址和MAC地址添加到自己的ARP列表中，并利用此信息开始数据的传输。如果源主机一直没有收到ARP响应数据包，表示ARP查询失败。
```

4、TCP 的主要特点是什么?

1. 1．TCP 是面向连接的。（就好像打电话一样，通话前需要先拨号建立连接，通话结束后要挂机释放连接）；

   2．每一条 TCP 连接只能有两个端点，每一条 TCP 连接只能是点对点的（一对一）；

   3．TCP 提供可靠交付的服务。通过 TCP 连接传送的数据，无差错、不丢失、不重复、并且按序到达；

   4．TCP 提供全双工通信。TCP 允许通信双方的应用进程在任何时候都能发送数据。TCP 连接的两端都设有发送缓存和接收缓存，用来临时存放双方通信的数据；

   5．面向字节流。TCP 中的“流”（Stream）指的是流入进程或从进程流出的字节序列。“面向字节流”的含义是：虽然应用程序和 TCP 的交互是一次一个数据块（大小不等），但 TCP 把应用程序交下来的数据仅仅看成是一连串的无结构的字节流。

5、UDP 的主要特点是什么?

1. 1．UDP 是无连接的；

   2．UDP 使用尽最大努力交付，即不保证可靠交付，因此主机不需要维持复杂的链接状态（这里面有许多参数）；

   3．UDP 是面向报文的；

   4．UDP 没有拥塞控制，因此网络出现拥塞不会使源主机的发送速率降低（对实时应用很有用，如直播，实时视频会议等）；

   5．UDP 支持一对一、一对多、多对一和多对多的交互通信；

   6．UDP 的首部开销小，只有 8 个字节，比 TCP 的 20 个字节的首部要短。

6、TCP 和 UDP 的区别?

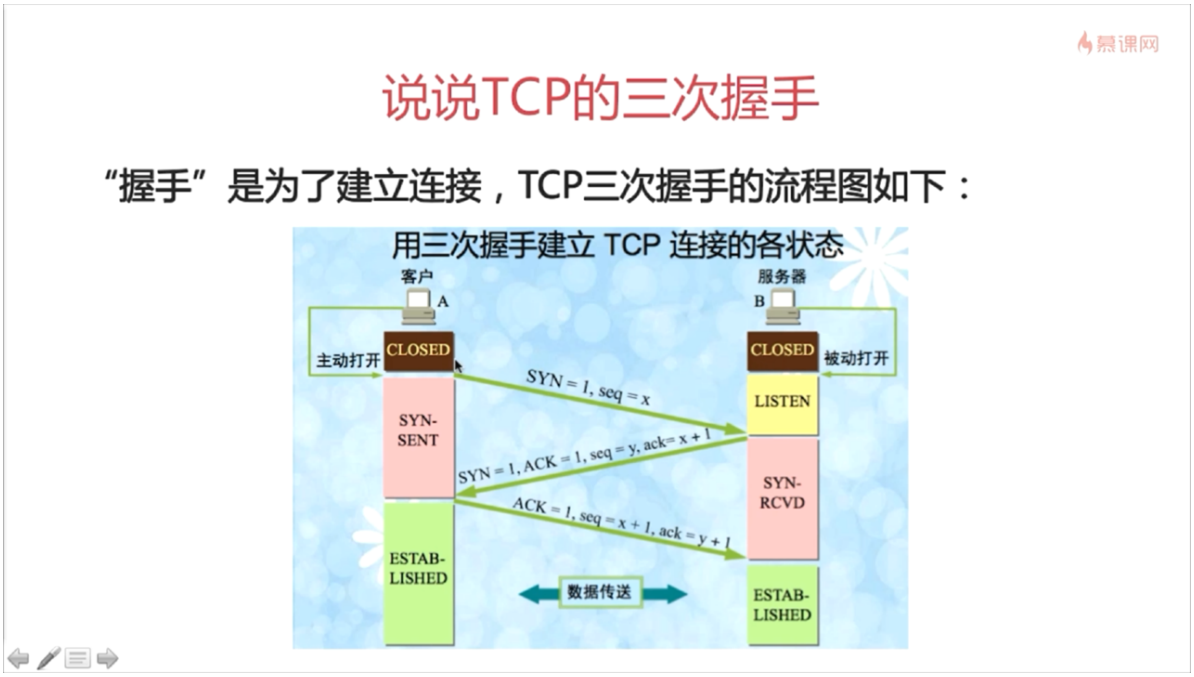
	TCP	UDP
是否面向连接	面向连接	无连接
传输可靠性	可靠	不可靠
传输形式	字节流	数据报文段
传输效率	慢	快
所需资源	多	少
应用场景	要求通信数据可靠 如：文件传输、邮件传输	要求通信速度高 如：域名转换、直播
首部字节	20-60	8

码农求职小助手

7、详细说下 TCP 三次握手的过程?

- **1. 三次握手**

TCP 建立连接的过程叫做握手，握手需要在客户和服务器之间交换三个 TCP 报文段。



最初客户端和服务端都处于 CLOSED(关闭) 状态。本例中 A（Client） 主动打开连接，B（Server）被动打开连接。

一开始，B 的 TCP 服务器进程首先创建传输控制块TCB，准备接受客户端进程的连接请求。然后服务端进程就处于 LISTEN(监听) 状态，等待客户端的连接请求。如有，立即作出响应。

第一次握手：A 的 TCP 客户端进程也是首先创建传输控制块 TCB。然后，在打算建立 TCP 连接时，向 B 发出连接请求报文段，这时首部中的同步位 SYN=1，同时选择一个初始序号 seq = x。TCP 规定，SYN 报文段（即 SYN = 1 的报文段）不能携带数据，但要消耗掉一个序号。这时，TCP 客户进程进入 SYN-SENT（同步已发送）状态。

第二次握手：B 收到连接请求报文后，如果同意建立连接，则向 A 发送确认。在确认报文段中应把 SYN 位和 ACK 位都置 1，确认号是 ack = x + 1，同时也为自己选择一个初始序号 seq = y。请注意，这个报文段也不能携带数据，但同样要消耗掉一个序号。这时 TCP 服务端进程进入 SYN-RCVD（同步收到）状态。

第三次握手：TCP 客户进程收到 B 的确认后，还要向 B 给出确认。确认报文段的 ACK 置 1，确认号 ack = y + 1，而自己的序号 seq = x + 1。这时 ACK 报文段可以携带数据。但如果不携带数据则不消耗序号，这种情况下，下一个数据报文段的序号仍是 seq = x + 1。这时，TCP 连接已经建立，A 进入 ESTABLISHED（已建立连接）状态。

8、为什么两次握手不可以呢?

1.为了防止已经失效的连接请求报文段突然又传送到了 B，因而产生错误。比如下面这种情况：A 发出的第一个连接请求报文段并没有丢失，而是在网路结点长时间滞留了，以致于延误到连接释放以后的某个时间段才到达 B。本来这是一个早已失效的报文段。但是 B 收到此失效的链接请求报文段后，就误认为 A 又发出一次新的连接请求。于是就向 A 发出确认报文段，同意建立连接。

2.对于上面这种情况，如果不进行第三次握手，B 发出确认后就认为新的运输连接已经建立了，并一直等待 A 发来数据。B 的许多资源就这样白白浪费了。

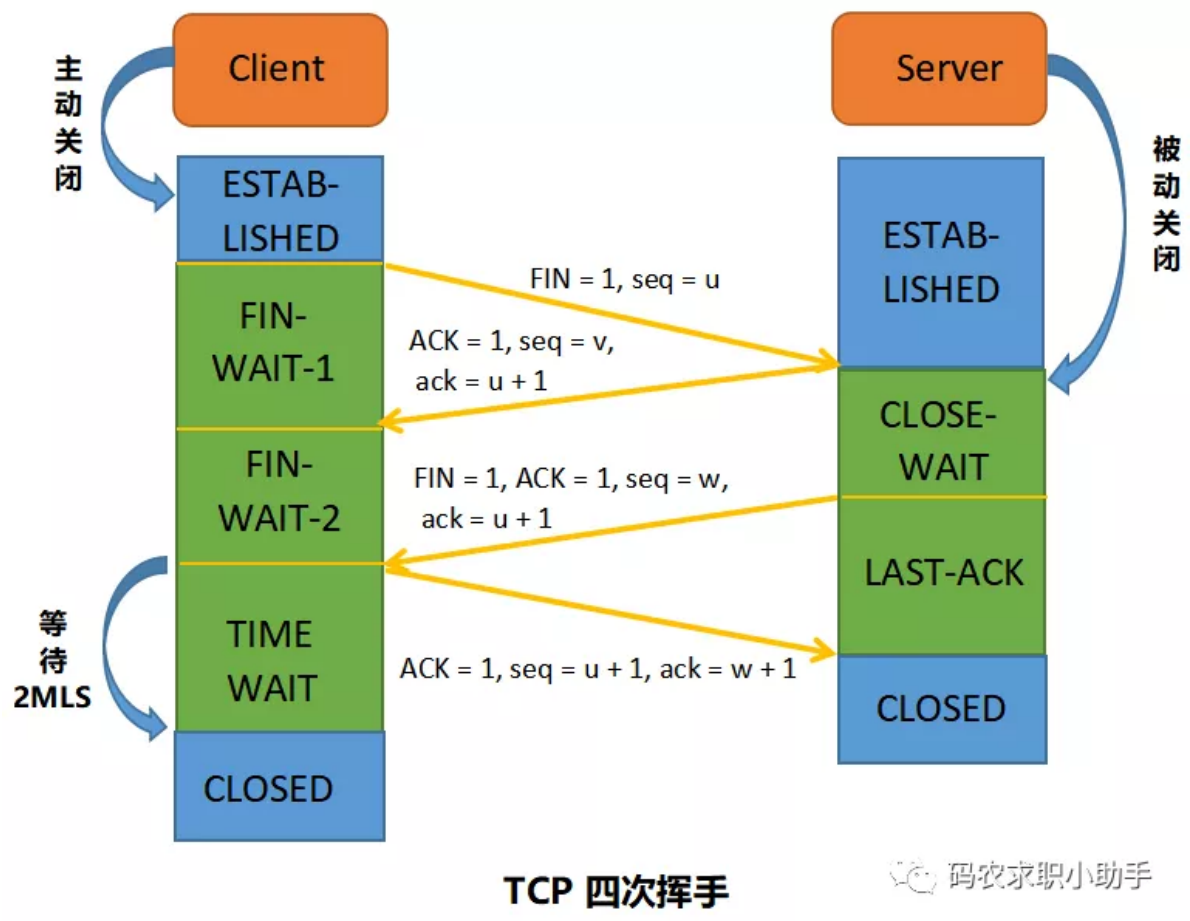
3.如果采用了三次握手，由于 A 实际上并没有发出建立连接请求，所以不会理睬 B 的确认，也不会向 B 发送数据。B 由于收不到确认，就知道 A 并没有要求建立连接。

9、为什么不需要四次握手?

有人可能会说 A 发出第三次握手的信息后在没有接收到 B 的请求就已经进入了连接状态，那如果 A 的这个确认包丢失或者滞留了怎么办？我们需要明白一点，完全可靠的通信协议是不存在的。在经过三次握手之后，客户端和服务端已经可以确认之前的通信状况，都收到了确认信息。所以即便再增加握手次数也不能保证后面的通信完全可靠，所以是没有必要的。

10、详细说下 TCP 四次挥手的过程?

据传输结束后，通信的双方都可以释放连接。现在 A 和 B 都处于 ESTABLISHED 状态。



第一次挥手：A 的应用进程先向其 TCP 发出连接释放报文段，并停止再发送数据，主动关闭 TCP 连接。A 把连接释放报文段首部的终止控制位 FIN 置 1，其序号 seq = u（等于前面已传送过的数据的最后一个字节的序号加 1），这时 A 进入 FIN-WAIT-1（终止等待1）状态，等待 B 的确认。请注意：TCP 规定，FIN 报文段即使不携带数据，也将消耗掉一个序号。

第二次挥手：B 收到连接释放报文段后立即发出确认，确认号是 ack = u + 1，而这个报文段自己的序号是 v（等于 B 前面已经传送过的数据的最后一个字节的序号加1），然后 B 就进入 CLOSE-WAIT（关闭等待）状态。TCP 服务端进程这时应通知高层应用进程，因而从 A 到 B 这个方向的连接就释放了，这时的 TCP 连接处于半关闭（half-close）状态，即 A 已经没有数据要发送了，但 B 若发送数据，A 仍要接收。也就是说，从 B 到 A 这个方向的连接并未关闭，这个状态可能会持续一段时间。A 收到来自 B 的确认后，就进入 FIN-WAIT-2(终止等待2)状态，等待 B 发出的连接释放报文段。

第三次挥手：若 B 已经没有要向 A 发送的数据，其应用进程就通知 TCP 释放连接。这时 B 发出的连接释放报文段必须使 FIN = 1。假定 B 的序号为 w（在半关闭状态，B 可能又发送了一些数据）。B 还必须重复上次已发送过的确认号 ack = u + 1。这时 B 就进入 LAST-ACK(最后确认)状态，等待 A 的确认。

第四次挥手：A 在收到 B 的连接释放报文后，必须对此发出确认。在确认报文段中把 ACK 置 1，确认号 ack = w + 1，而自己的序号 seq = u + 1（前面发送的 FIN 报文段要消耗一个序号）。然后进入 TIME-WAIT(时间等待) 状态。请注意，现在 TCP 连接还没有释放掉。必须经过时间等待计时器设置的时间 2MSL（MSL：最长报文段寿命）后，A 才能进入到 CLOSED 状态，然后撤销传输控制块，结束这次 TCP 连接。当然如果 B 一收到 A 的确认就进入 CLOSED 状态，然后撤销传输控制块。所以在释放连接时，B 结束 TCP 连接的时间要早于 A。

11、为什么 TIME-WAIT 状态必须等待 2MSL 的时间呢？（下面的回答A代表客户端，B代表服务端）

1. 1．为了保证 A 发送的最后一个 ACK 报文段能够到达 B。这个 ACK 报文段有可能丢失，因而使处在 LAST-ACK 状态的 B 收不到对已发送的 FIN + ACK 报文段的确认。B 会超时重传这个 FIN+ACK 报文段，而 A 就能在 2MSL 时间内（超时 + 1MSL 传输）收到这个重传的 FIN+ACK 报文段。接着 A 重传一次确认，重新启动 2MSL 计时器。

   2．防止已失效的连接请求报文段出现在本连接中。A 在发送完最后一个 ACK 报文段后，再经过时间 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。这样就可以使下一个连接中不会出现这种旧的连接请求报文段。

12、为什么第二次跟第三次不能合并，第二次和第三次之间的等待是什么？

当服务器执行第二次挥手之后，此时证明客户端不会再向服务端请求任何数据，但是服务端可能还正在给客户端发送数据（可能是客户端上一次请求的资源还没有发送完毕），所以此时服务端会等待把之前未传输完的数据传输完毕之后再发送关闭请求。

13、TCP 协议是如何保证可靠传输的？

1. 1．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；

   2．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；

   3．丢弃重复数据：对于重复数据，能够丢弃重复数据；

   4．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；

   5．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；

   6．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。

14、对 TCP 拥塞控制使用了哪些算法？

为了进行拥塞控制，TCP 发送方要维持一个拥塞窗口(cwnd) 的状态变量。拥塞控制窗口的大小取决于网络的拥塞程度，并且动态变化。发送方让自己的发送窗口取为拥塞窗口和接收方的接受窗口中较小的一个。

TCP 的拥塞控制采用了四种算法，即：慢开始、拥塞避免、快重传和快恢复。在网络层也可以使路由器采用适当的分组丢弃策略（如：主动队列管理 AQM），以减少网络拥塞的发生。

- **慢开始：**

慢开始算法的思路是当主机开始发送数据时，如果立即把大量数据字节注入到网络，那么可能会引起网络阻塞，因为现在还不知道网络的符合情况。经验表明，较好的方法是先探测一下，即由小到大逐渐增大发送窗口，也就是由小到大逐渐增大拥塞窗口数值。cwnd 初始值为 1，每经过一个传播轮次，cwnd 加倍。

- **拥塞避免：**

拥塞避免算法的思路是让拥塞窗口 cwnd 缓慢增大，即每经过一个往返时间 RTT 就把发送方的 cwnd 加 1。

- **快重传与快恢复：**

在 TCP/IP 中，快速重传和快恢复（fast retransmit and recovery，FRR）是一种拥塞控制算法，它能快速恢复丢失的数据包。没有FRR，如果数据包丢失了，TCP 将会使用定时器来要求传输暂停。在暂停的这段时间内，没有新的或复制的数据包被发送。有了 FRR，如果接收机接收到一个不按顺序的数据段，它会立即给发送机发送一个重复确认。如果发送机接收到三个重复确认，它会假定确认件指出的数据段丢失了，并立即重传这些丢失的数据段。

15、你对 HTTP 状态码有了解吗？

	类别	原因短语
1XX	Informational（信息性状态码）	接收的请求正在处理
2XX	Success（成功状态码）	请求正常处理完毕
3XX	Redirection（重定向状态码）	需要进行附加操作以完成请求
4XX	Client Error（客户端错误状态码）	服务器无法处理请求
5XX	Server Error（服务器错误状态码）	服务器处理请求出错

码农求职小助手

- **1XX 信息**

100 Continue ：表明到目前为止都很正常，客户端可以继续发送请求或者忽略这个响应。

- **2XX 成功**

```
1. 200 OK

2. 204 No Content ：请求已经成功处理，但是返回的响应报文不包含实体的主体部分。一般在只需要从客户端往服务器发送信息，而不需要返回数据时使用。

3. 206 Partial Content ：表示客户端进行了范围请求，响应报文包含由 Content-Range 指定范围的实体内容。
```

- **3XX 重定向**

```
1. 301 Moved Permanently ：永久性重定向；

2. 302 Found ：临时性重定向；

3. 303 See Other ：和 302 有着相同的功能，但是 303 明确要求客户端应该采用 GET 方法获取资源。

4. 304 Not Modified ：如果请求报文首部包含一些条件，例如：If-Match，If-Modified-Since，If-None-Match，If-Range，If-Unmodified-Since，如果不满足条件，则服务器会返回 304 状态码。

5. 307 Temporary Redirect ：临时重定向，与 302 的含义类似，但是 307 要求浏览器不会把重定向请求的 POST 方法改成 GET 方法。
```

- **4XX 客户端错误**

```
1. 400 Bad Request ：请求报文中存在语法错误。

2. 401 Unauthorized ：该状态码表示发送的请求需要有认证信息（BASIC 认证、DIGEST 认证）。如果之前已进行过一次请求，则表示用户认证失败。

3. 403 Forbidden ：请求被拒绝。

4. 404 Not Found
```

- **5XX 服务器错误**

```
1. 500 Internal Server Error ：服务器正在执行请求时发生错误；

2. 503 Service Unavailable ：服务器暂时处于超负载或正在进行停机维护，现在无法处理请求。
```

16、说下 GET 和 POST 的区别?

GET 和 POST 本质都是 HTTP 请求，只不过对它们的作用做了界定和适配，并且让他们适应各自的场景。

本质区别：GET 只是一次 HTTP请求，POST 先发请求头再发请求体，实际上是两次请求。

1. 从功能上讲，GET 一般用来从服务器上获取资源，POST 一般用来更新服务器上的资源；

2. 从 REST 服务角度上说，GET 是幂等的，即读取同一个资源，总是得到相同的数据，而 POST 不是幂等的，因为每次请求对资源的改变并不是相同的；进一步地，GET 不会改变服务器上的资源，而 POST 会对服务器资源进行改变；

3. 从请求参数形式上看，GET 请求的数据会附在 URL 之后，即将请求数据放置在 HTTP 报文的 请求头 中，以 ? 分割 URL 和传输数据，参数之间以 & 相连。特别地，如果数据是英文字母/数字，原样发送；否则，会将其编码为 application/x-www-form-urlencoded MIME 字符串(如果是空格，转换为+，如果是中文/其他字符，则直接把字符串用 BASE64 加密，得出如：%E4%BD%A0%E5%A5%BD，其中 %XX 中的 XX 为该符号以 16 进制表示的 ASCII)；而 POST 请求会把提交的数据则放置在是 HTTP 请求报文的 请求体 中；

4. 就安全性而言，POST 的安全性要比 GET 的安全性高，因为 GET 请求提交的数据将明文出现在 URL 上，而且 POST 请求参数则被包装到请求体中，相对更安全；

5. 从请求的大小看，GET 请求的长度受限于浏览器或服务器对 URL 长度的限制，允许发送的数据量比较小，而 POST 请求则是没有大小限制的。

17、HTTP 和 HTTPS 的区别?

1. 开销：HTTPS 协议需要到 CA 申请证书，一般免费证书很少，需要交费；

2. 资源消耗：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，需要消耗更多的 CPU 和内存资源；

3. 端口不同：HTTP 和 HTTPS 使用的是完全不同的连接方式，用的端口也不一样，前者是 80，后者是 443；

4. 安全性：HTTP 的连接很简单，是无状态的；HTTPS 协议是由 TSL+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。

18、HTTPS 的优缺点?

- **优点**:

1. 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；

2. 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；

3. 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。

- **缺点**:

1. HTTPS 协议握手阶段比较费时，会使页面的加载时间延长近 50%，增加 10% 到 20% 的耗电；

2. HTTPS 连接缓存不如 HTTP 高效，会增加数据开销和功耗，甚至已有的安全措施也会因此而受到影响；

3. SSL 证书需要钱，功能越强大的证书费用越高，个人网站、小网站没有必要一般不会用；

4. SSL 证书通常需要绑定 IP，不能在同一 IP 上绑定多个域名，IPv4 资源不可能支撑这个消耗；

5. HTTPS 协议的加密范围也比较有限，在黑客攻击、拒绝服务攻击、服务器劫持等方面几乎起不到什么作用。最关键的，SSL 证书的信用链体系并不安全，特别是在某些国家可以控制 CA 根证书的情况下，中间人攻击一样可行。

数据库面试题

1.事务四大特性

1.原子性（Atomicity）- 原子性是指事务包含的所有操作要么全部成功，要么全部失败回滚，因此事务的操作如果成功就必须要完全应用到数据库，如果操作失败则不能对数据库有任何影响。

2.一致性（Consistency）- 事务开始前和结束后，数据库的完整性约束没有被破坏。比如A向B转账，不可能A扣了钱，B却没收到。

3.隔离性（Isolation）- 隔离性是当多个用户并发访问数据库时，比如操作同一张表时，数据库为每一个用户开启的事务，不能被其他事务的操作所干扰，多个并发事务之间要相互隔离。

4.持久性（Durability）- 持久性是指一个事务一旦被提交了，那么对数据库中的数据的改变就是永久性的，即便是在数据库系统遇到故障的情况下也不会丢失提交事务的操作**。

2.多个事务读可能会道理以下问题

1.脏读：事务B读取事务A还没有提交的数据
2.不可重复读：，一行被检索两次，并且该行中的值在不同的读取之间不同时
3.幻读:当在事务处理过程中执行两个相同的查询，并且第二个查询返回的行集合与第一个查询不同时
4.这两个区别在于，不可重复读重点在一行，幻读的重点 ，返回 的集合不一样

3.MYSQL的两种存储引擎区别（事务、锁级别等等），各自的适用场景

引擎	特性
MYISAM	不支持外键，表锁，插入数据时，锁定整个表，查表总行数时，不需要全表扫描
INNODB	支持外键，行锁，查表总行数时，全表扫描

5.数据库三范式

级别	概念
1NF	属性不可分
2NF	非主键属性，完全依赖于主键属性
3NF	非主键属性无传递依赖

6.事务的并发问题

```
1、脏读：事务A读取了事务B更新的数据，然后B回滚操作，那么A读取到的数据是脏数据

2、不可重复读：事务 A 多次读取同一数据，事务 B 在事务A多次读取的过程中，对数据作了更新并提交，导致事务A多次读取同一数据时，结果因此本事务先后两次读到的数据结果会不一致。

3、幻读：幻读解决了不重复读，保证了同一个事务里，查询的结果都是事务开始时的状态（一致性）。
```

7.事务的隔离级别

事务隔离级别	脏读	不可重复读	幻读
读未提交 read-uncommitted	是	是	是
读已提交 read-committed	否	是	是
可重复读 repeatable-read	否	否	是
串行化 serializable	否	否	否

- **读未提交**：另一个事务修改了数据，但尚未提交，而本事务中的SELECT会读到这些未被提交的数据**脏读**
- **不可重复读**：事务 A 多次读取同一数据，事务 B 在事务A多次读取的过程中，对数据作了更新并提交，导致事务A多次读取同一数据时，结果因此本事务先后两次读到的数据结果会不一致。
- **可重复读**：在同一个事务里，SELECT的结果是事务开始时时间点的状态，因此，同样的SELECT操作读到的结果会是一致的。但是，会有**幻读**现象
- **串行化**：最高的隔离级别，在这个隔离级别下，不会产生任何异常。并发的事务，就像事务是在一个个按照顺序执行一样

8.MySQL常见的三种存储引擎（InnoDB、MyISAM、MEMORY）的区别?

虽然MySQL里的存储引擎不只是MyISAM与InnoDB这两个，但常用的就是两个。

两种存储引擎的大致区别表现在：

- ```
  - InnoDB支持事务，MyISAM不支持，这一点是非常之重要。事务是一种高级的处理方式，如在一些列增删改中只要哪个出错还可以回滚还原，而MyISAM就不可以了。

  - MyISAM适合查询以及插入为主的应用。

  ```
  ```

- InnoDB适合频繁修改以及涉及到安全性较高的应用。

- InnoDB支持外键，MyISAM不支持。

- 从MySQL5.5.5以后，InnoDB是默认引擎。

- InnoDB不支持FULLTEXT类型的索引。

- InnoDB中不保存表的行数，如select count(*) from table时，InnoDB需要扫描一遍整个表来计算有多少行，但是MyISAM只要简单的读出保存好的行数即可。注意的是，当count(*)语句包含where条件时MyISAM也需要扫描整个表。

- 对于自增长的字段，InnoDB中必须包含只有该字段的索引，但是在MyISAM表中可以和其他字段一起建立联合索引。

- DELETE FROM table时，InnoDB不会重新建立表，而是一行一行的 删除，效率非常慢。MyISAM则会重建表。

- InnoDB支持行锁（某些情况下还是锁整表，如 update table set a=1 where user like '%lee%'。

9.关于MySQL数据库提供的两种存储引擎，MyISAM与InnoDB选择使用：

- 1.- INNODB会支持一些关系数据库的高级功能，如事务功能和行级锁，MyISAM不支持。

  2.- MyISAM的性能更优，占用的存储空间少，所以，选择何种存储引擎，视具体应用而定。

  3.- 如果你的应用程序一定要使用事务，毫无疑问你要选择INNODB引擎。但要注意，INNODB的行级锁是有条件的。在where条件没有使用主键时，照样会锁全表。比如DELETE FROM mytable这样的删除语句。

  4.- 如果你的应用程序对查询性能要求较高，就要使用MyISAM了。MyISAM索引和数据是分开的，而且其索引是压缩的，可以更好地利用内存。所以它的查询性能明显优于INNODB。压缩后的索引也能节约一些磁盘空间。MyISAM拥有全文索引的功能，这可以极大地优化LIKE查询的效率。

10.MySQL的MyISAM与InnoDB两种存储引擎在，事务、锁级别，各自的适用场景?

事务处理上方面

- - MyISAM：强调的是性能，每次查询具有原子性,其执行数度比InnoDB类型更快，但是不提供事务支持。

  - InnoDB：提供事务支持事务，外部键等高级数据库功能。 具有事务(commit)、回滚(rollback)和崩溃修复能力(crash recovery capabilities)的事务安全(transaction-safe (ACID compliant))型表。

锁级别

- ```
  - MyISAM：只支持表级锁，用户在操作MyISAM表时，select，update，delete，insert语句都会给表自动加锁，如果加锁以后的表满足insert并发的情况下，可以在表的尾部插入新的数据。

  - InnoDB：支持事务和行级锁，是innodb的最大特色。行锁大幅度提高了多用户并发操作的新能。但是InnoDB的行锁，只是在WHERE的主键是有效的，非主键的WHERE都会锁全表的。
  ```

11.MySQL B+Tree索引和Hash索引的区别?

- ```
  - Hash索引结构的特殊性，其检索效率非常高，索引的检索可以一次定位;

  - B+树索引需要从根节点到枝节点，最后才能访问到页节点这样多次的IO访问;
  ```

12.那为什么大家不都用Hash索引而还要使用B+树索引呢?

Hash索引

```
1. Hash索引仅仅能满足"=","IN"和"<=>"查询，不能使用范围查询,因为经过相应的Hash算法处理之后的Hash值的大小关系，并不能保证和Hash运算前完全一样；

2. Hash索引无法被用来避免数据的排序操作，因为Hash值的大小关系并不一定和Hash运算前的键值完全一样；

3. Hash索引不能利用部分索引键查询，对于组合索引，Hash索引在计算Hash值的时候是组合索引键合并后再一起计算Hash值，而不是单独计算Hash值，所以通过组合索引的前面一个或几个索引键进行查询的时候，Hash索引也无法被利用；

4. Hash索引在任何时候都不能避免表扫描，由于不同索引键存在相同Hash值，所以即使取满足某个Hash键值的数据的记录条数，也无法从Hash索引中直接完成查询，还是要回表查询数据；

5. Hash索引遇到大量Hash值相等的情况后性能并不一定就会比B+树索引高。
```

13.B+树索引和哈希索引的明显区别是:

```
如果是等值查询，那么哈希索引明显有绝对优势，因为只需要经过一次算法即可找到相应的键值；当然了，这个前提是，键值都是唯一的。如果键值不是唯一的，就需要先找到该键所在位置，然后再根据链表往后扫描，直到找到相应的数据；如果是范围查询检索，这时候哈希索引就毫无用武之地了，因为原先是有序的键值，经过哈希算法后，有可能变成不连续的了，就没办法再利用索引完成范围查询检索；同理，哈希索引没办法利用索引完成排序，以及like ‘xxx%’ 这样的部分模糊查询（这种部分模糊查询，其实本质上也是范围查询）；哈希索引也不支持多列联合索引的最左匹配规则；B+树索引的关键字检索效率比较平均，不像B树那样波动幅度大，在有大量重复键值情况下，哈希索引的效率也是极低的，因为存在所谓的哈希碰撞问题。


```

14.有哪些锁（乐观锁悲观锁），select 时怎么加排它锁?

悲观锁（Pessimistic Lock）

悲观锁的特点是先获取锁，再进行业务操作，即“悲观”的认为获取锁是非常有可能失败的，因此要先确保获取锁成功再进行业务操作。通常所说的“一锁二查三更新”即指的是使用悲观锁。通常来讲在数据库上的悲观锁需要数据库本身提供支持，即通过常用的`select … for update`操作来实现悲观锁。当数据库执行`select for update`时会获取被`select`中的数据行的行锁，因此其他并发执行的`select for update`如果试图选中同一行则会发生排斥（需要等待行锁被释放），因此达到锁的效果。`select for update`获取的行锁会在当前事务结束时自动释放，因此必须在事务中使用。这里需要注意的一点是不同的数据库对`select for update`的实现和支持都是有所区别的，例如`oracle`支持`select for update no wait`，表示如果拿不到锁立刻报错，而不是等待，`MySQL`就没有`no wait`这个选项。另外`MySQL`还有个问题是`select for update`语句执行中所有扫描过的行都会被锁上，这一点很容易造成问题。因此如果在`MySQL`中用悲观锁务必要确定走了索引，而不是全表扫描。

乐观锁（Optimistic Lock）

乐观锁，也叫乐观并发控制，它假设多用户并发的事务在处理时不会彼此互相影响，各事务能够在不产生锁的情况下处理各自影响的那部分数据。在提交数据更新之前，每个事务会先检查在该事务读取数据后，有没有其他事务又修改了该数据。如果其他事务有更新的话，那么当前正在提交的事务会进行回滚。乐观锁的特点先进行业务操作，不到万不得已不去拿锁。即“乐观”的认为拿锁多半是会成功的，因此在进行完业务操作需要实际更新数据的最后一步再去拿一下锁就好。乐观锁在数据库上的实现完全是逻辑的，不需要数据库提供特殊的支持。一般的做法是在需要锁的数据上增加一个版本号，或者时间戳，然后按照如下方式实现：乐观锁（给表加一个版本号字段） 这个并不是乐观锁的定义，给表加版本号，是数据库实现乐观锁的一种方式。

15.数据库的读写分离、主从复制，主从复制分析的 7 个问题?

主从复制的几种方式

同步复制

- 所谓的同步复制，意思是master的变化，必须等待slave-1,slave-2,...,slave-n完成后才能返回。 这样，显然不可取，也不是MySQL复制的默认设置。比如，在WEB前端页面上，用户增加了条记录，需要等待很长时间。

异步复制

- 如同AJAX请求一样。master只需要完成自己的数据库操作即可。至于slaves是否收到二进制日志，是否完成操作，不用关心,MySQL的默认设置。

半同步复制

- master只保证slaves中的一个操作成功，就返回，其他slave不管。 这个功能，是由google为MySQL引入的。

16.主从复制分析的 7 个问题

问题1：master的写操作，slaves被动的进行一样的操作，保持数据一致性，那么slave是否可以主动的进行写操作?

> 假设slave可以主动的进行写操作，slave又无法通知master，这样就导致了master和slave数据不一致了。因此slave不应该进行写操作，至少是slave上涉及到复制的数据库不可以写。实际上，这里已经揭示了读写分离的概念。

问题2：主从复制中，可以有N个slave,可是这些slave又不能进行写操作，要他们干嘛?

> 以实现数据备份。类似于高可用的功能，一旦master挂了，可以让slave顶上去，同时slave提升为master。异地容灾，比如master在北京，地震挂了，那么在上海的slave还可以继续。主要用于实现scale out,分担负载,可以将读的任务分散到slaves上。【很可能的情况是，一个系统的读操作远远多于写操作，因此写操作发向master，读操作发向slaves进行操作】

问题3：主从复制中有master,slave1,slave2,...等等这么多MySQL数据库，那比如一个JAVA WEB应用到底应该连接哪个数据库?

> 当 然，我们在应用程序中可以这样，insert/delete/update这些更新数据库的操作，用connection(for master)进行操作，select用connection(for slaves)进行操作。那我们的应用程序还要完成怎么从slaves选择一个来执行select，例如使用简单的轮循算法。这样的话，相当于应用程序完成了SQL语句的路由，而且与MySQL的主从复制架构非常关联，一旦master挂了，某些slave挂了，那么应用程序就要修改了。能不能让应用程序与MySQL的主从复制架构没有什么太多关系呢？找一个组件，application program只需要与它打交道，用它来完成MySQL的代理，实现SQL语句的路由**。MySQL proxy并不负责，怎么从众多的slaves挑一个？可以交给另一个组件(比如haproxy)来完成。这就是所谓的MySQL READ WRITE SPLITE，MySQL的读写分离。

问题4：如果MySQL proxy , direct , master他们中的某些挂了怎么办?

> 总统一般都会弄个副总统，以防不测。同样的，可以给这些关键的节点来个备份。

问题5：当master的二进制日志每产生一个事件，都需要发往slave，如果我们有N个slave,那是发N次，还是只发一次?

> 如果只发一次，发给了slave-1，那slave-2,slave-3,...它们怎么办？显 然，应该发N次。实际上，在MySQL master内部，维护N个线程，每一个线程负责将二进制日志文件发往对应的slave。master既要负责写操作，还的维护N个线程，负担会很重。可以这样，slave-1是master的从，slave-1又是slave-2,slave-3,...的主，同时slave-1不再负责select。 slave-1将master的复制线程的负担，转移到自己的身上。这就是所谓的多级复制的概念。

问题6：当一个select发往MySQL proxy，可能这次由slave-2响应，下次由slave-3响应，这样的话，就无法利用查询缓存了。

> 应该找一个共享式的缓存，比如memcache来解决。将slave-2,slave-3,...这些查询的结果都缓存至mamcache中。

问题7：随着应用的日益增长，读操作很多，我们可以扩展slave，但是如果master满足不了写操作了，怎么办呢?

> scale on ?更好的服务器？ 没有最好的，只有更好的，太贵了。。。scale out ? 主从复制架构已经满足不了。可以分库【垂直拆分】，分表【水平拆分】。

17.使用explain优化sql和索引？（参考PPT）

18.MySQL慢查询怎么解决?

- **slow_query_log** 慢查询开启状态。
- **slow_query_log_file** 慢查询日志存放的位置（这个目录需要MySQL的运行帐号的可写权限，一般设置为MySQL的数据存放目录）。
- **long_query_time** 查询超过多少秒才记录。

19.MySQL都有什么锁，死锁判定原理和具体场景，死锁怎么解决?

MySQL都有什么锁

MySQL有三种锁的级别：页级、表级、行级。

- ```
  - 表级锁：开销小，加锁快；不会出现死锁；锁定粒度大，发生锁冲突的概率最高,并发度最低。
  - 行级锁：开销大，加锁慢；会出现死锁；锁定粒度最小，发生锁冲突的概率最低,并发度也最高。
  - 页面锁：开销和加锁时间界于表锁和行锁之间；会出现死锁；锁定粒度界于表锁和行锁之间，并发度一般
  ```

什么情况下会造成死锁

- ```
  - 所谓死锁：是指两个或两个以上的进程在执行过程中。
  - 因争夺资源而造成的一种互相等待的现象,若无外力作用,它们都将无法推进下去。
  - 此时称系统处于死锁状态或系统产生了死锁,这些永远在互相等竺的进程称为死锁进程。
  - 表级锁不会产生死锁.所以解决死锁主要还是针对于最常用的InnoDB。
  ```

死锁的解决办法

- ```
  - 查出的线程杀死 kill

  - 设置锁的超时时间，Innodb 行锁的等待时间，单位秒。可在会话级别设置，RDS 实例该参数的默认值为 50（秒）。
  ```

20.MySQL 高并发环境解决方案?

需求分析：互联网单位 每天大量数据读取，写入，并发性高。

- ```
  - 现有解决方式：水平分库分表，由单点分布到多点数据库中，从而降低单点数据库压力。
  - 集群方案：解决DB宕机带来的单点DB不能访问问题。
  - 读写分离策略：极大限度提高了应用中Read数据的速度和并发量。无法解决高写入压力。
  ```

21.数据库崩溃时事务的恢复机制（REDO日志和UNDO日志）?

Undo Log

- 1.Undo Log是为了实现事务的原子性，在MySQL数据库InnoDB存储引擎中，还用了Undo Log来实现多版本并发控制(简称：MVCC)。

  2.- 事务的原子性(Atomicity)事务中的所有操作，要么全部完成，要么不做任何操作，不能只做部分操作。如果在执行的过程中发生了错误，要回滚(Rollback)到事务开始前的状态，就像这个事务从来没有执行过。

  3.- 原理Undo Log的原理很简单，为了满足事务的原子性，在操作任何数据之前，首先将数据备份到一个地方（这个存储数据备份的地方称为UndoLog）。然后进行数据的修改。如果出现了错误或者用户执行了ROLLBACK语句，系统可以利用Undo Log中的备份将数据恢复到事务开始之前的状态。

之所以能同时保证原子性和持久化，是因为以下特点:

- 1.更新数据前记录Undo log。
  2.为了保证持久性，必须将数据在事务提交前写到磁盘。只要事务成功提交，数据必然已经持久化。
  3.Undo log必须先于数据持久化到磁盘。如果在G,H之间系统崩溃，undo log是完整的， 可以用来回滚事务。
  4.如果在A-F之间系统崩溃,因为数据没有持久化到磁盘。所以磁盘上的数据还是保持在事务开始前的状态。
  缺陷：每个事务提交前将数据和Undo Log写入磁盘，这样会导致大量的磁盘IO，因此性能很低。

Redo Log

- 原理和Undo Log相反，Redo Log记录的是新数据的备份。在事务提交前，只要将Redo Log持久化即可，不需要将数据持久化。当系统崩溃时，虽然数据没有持久化，但是Redo Log已经持久化。系统可以根据Redo Log的内容，将所有数据恢复到最新的状态。

操作系统面试题

1、进程和线程以及它们的区别

1.进程是对运行时程序的封装，是系统进行资源调度和分配的的基本单位，实现了操作系统的并发；

2.线程是进程的子任务，是CPU调度和分派的基本单位，用于保证程序的 实时性，实现进程内部的并发；

3.一个程序至少有一个进程，一个进程至少有一个线程，线程依赖于进程而存在；

4.进程在执行过程中拥有独立的内存单元，而多个线程共享进程的内存。

2、进程间的通信的几种方式

```
1.管道（pipe）及命名管道（named pipe）：管道可用于具有亲缘关系的父子进程间的通信，有名管道除了具有管道所具有的功能外，它还允许无亲缘关系进程间的通信；

2.信号（signal）：信号是一种比较复杂的通信方式，用于通知接收进程某个事件已经发生；

3.消息队列：消息队列是消息的链接表，它克服了上两种通信方式中信号量有限的缺点，具有写权限得进程可以按照一定得规则向消息队列中

4.共享内存：可以说这是最有用的进程间通信方式。它使得多个进程可以访问同一块内存空间，不同进程可以及时看到对方进程中对共享内存中数据得更新。这种方式需要依靠某种同步操作，如互斥锁和信号量等；

5.信号量：主要作为进程之间及同一种进程的不同线程之间得同步和互斥手段；

6.套接字：这是一种更为一般得进程间通信机制，它可用于网络中不同机器之间的进程间通信，应用非常广泛。
```

3、什么是死锁？死锁产生的条件？

1). 死锁的概念

在两个或者多个并发进程中，如果每个进程持有某种资源而又等待其它进程释放它或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。通俗的讲，就是两个或多个进程无限期的阻塞、相互等待的一种状态。

2). 死锁产生的四个必要条件

```
1.互斥：至少有一个资源必须属于非共享模式，即一次只能被一个进程使用；若其他申请使用该资源，那么申请进程必须等到该资源被释放为止；

2.占有并等待：一个进程必须占有至少一个资源，并等待另一个资源，而该资源为其他进程所占有；

3.非抢占：进程不能被抢占，即资源只能被进程在完成任务后自愿释放

4.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

3). 死锁的处理基本策略和常用方法

```
1.打破占有并等待条件

2.打破非抢占条件

3.打破循环等待条件
```

4．Linux中常用到的命令

```
显示文件目录命令ls 如ls
改变当前目录命令cd 如cd /home
建立子目录mkdir 如mkdir xiong
删除子目录命令rmdir 如rmdir /mnt/cdrom
删除文件命令rm 如rm /ucdos.bat
文件复制命令cp 如cp /ucdos /fox
获取帮助信息命令man 如man ls
显示文件的内容less 如less mwm.lx
重定向与管道type 如type readme>>direct，将文件readme的内容追加到文direct中
```

5．什么是死锁？其条件是什么？怎样避免死锁？

死锁的概念：在两个或多个并发进程中，如果每个进程持有某种资源而又都等待别的进程释放它或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。通俗地讲，就是两个或多个进程被无限期地阻塞、相互等待的一种状态。
死锁产生的原因主要是：□ 系统资源不足；，进程推进顺序非法。
产生死锁的必要条件：

```
(1)互斥（mutualexclusion），一个资源每次只能被一个进程使用；
(2)不可抢占（nopreemption），进程已获得的资源，在未使用完之前，不能强行剥夺；
(3)占有并等待（hold andwait），一个进程因请求资源而阻塞时，对已获得的资源保持不放；
(4)环形等待（circularwait），若干进程之间形成一种首尾相接的循环等待资源关系。
```

这四个条件是死锁的必要条件，只要系统发生死锁，这些条件必然成立，而只要上述条件之一不满足，就不会发生死锁。
死锁的解除与预防：理解了死锁的原因，尤其是产生死锁的四个必要条件，就可以最大可能地避免、预防和解除死锁。所以，在系统设计、进程调度等方面注意如何不让这四个必要条件成立，如何确定资源的合理分配算法，避免进程永久占据系统资源。此外，也要防止进程在处于等待状态的情况下占用资源。因此，对资源的分配要给予合理的规划。
死锁的处理策略：鸵鸟策略、预防策略、避免策略、检测与恢复策略。

6、进程的调度算法有哪些？

```
调度算法是指：根据系统的资源分配策略所规定的资源分配算法。常用的调度算法有：先来先服务调度算法、时间片轮转调度法、短作业优先调度算法、最短剩余时间优先、高响应比优先调度算法、优先级调度算法等等。
```

- **先来先服务调度算法**

```
先来先服务调度算法是一种最简单的调度算法，也称为先进先出或严格排队方案。当每个进程就绪后，它加入就绪队列。当前正运行的进程停止执行，选择在就绪队列中存在时间最长的进程运行。该算法既可以用于作业调度，也可以用于进程调度。先来先去服务比较适合于常作业（进程），而不利于段作业（进程）。
```

- **时间片轮转调度算法**

```
1 时间片轮转调度算法主要适用于分时系统。在这种算法中，系统将所有就绪进程按到达时间的先后次序排成一个队列，进程调度程序总是选择就绪队列中第一个进程执行，即先来先服务的原则，但仅能运行一个时间片。
```

- **短作业优先调度算法**

```
1 短作业优先调度算法是指对短作业优先调度的算法，从后备队列中选择一个或若干个估计运行时间最短的作业，将它们调入内存运行。 短作业优先调度算法是一个非抢占策略，他的原则是下一次选择预计处理时间最短的进程，因此短进程将会越过长作业，跳至队列头。
```

- **最短剩余时间优先调度算法**

```
1 最短剩余时间是针对最短进程优先增加了抢占机制的版本。在这种情况下，进程调度总是选择预期剩余时间最短的进程。当一个进程加入到就绪队列时，他可能比当前运行的进程具有更短的剩余时间，因此只要新进程就绪，调度程序就能可能抢占当前正在运行的进程。像最短进程优先一样，调度程序正在执行选择函数是必须有关于处理时间的估计，并且存在长进程饥饿的危险。
```

- **高响应比优先调度算法**

```
1 高响应比优先调度算法主要用于作业调度，该算法是对 先来先服务调度算法和短作业优先调度算法的一种综合平衡，同时考虑每个作业的等待时间和估计的运行时间。在每次进行作业调度时，先计算后备作业队列中每个作业的响应比，从中选出响应比最高的作业投入运行。
```

- **优先级调度算法**

```
1 优先级调度算法每次从后备作业队列中选择优先级最高的一个或几个作业，将它们调入内存，分配必要的资源，创建进程并放入就绪队列。在进程调度中，优先级调度算法每次从就绪队列中选择优先级最高的进程，将处理机分配给它，使之投入运行。
```

7、死锁产生的必要条件?

```
1 1．互斥条件：进程要求对所分配的资源进行排它性控制，即在一段时间内某资源仅为一进程所占用。
2
3 2．请求和保持条件：当进程因请求资源而阻塞时，对已获得的资源保持不放。
4
5 3．不剥夺条件：进程已获得的资源在未使用完之前，不能剥夺，只能在使用完时由自己释放。
6
7 4．环路等待条件：在发生死锁时，必然存在一个进程--资源的环形链。
```

8、解决死锁的基本方法?

```
1．预防死锁

2．避免死锁

3．检测死锁

4．解除死锁
```

9、怎么预防死锁?

```
1．破坏请求条件：一次性分配所有资源，这样就不会再有请求了；

2．破坏请保持条件：只要有一个资源得不到分配，也不给这个进程分配其他的资源：

3．破坏不可剥夺条件：当某进程获得了部分资源，但得不到其它资源，则释放已占有的资源；

4．破坏环路等待条件：系统给每类资源赋予一个编号，每一个进程按编号递增的顺序请求资源，释放则相反。
```

10、怎么避免死锁?

- **银行家算法**

```
当进程首次申请资源时，要测试该进程对资源的最大需求量，如果系统现存的资源可以满足它的最大需求量则按当前的申请量分配资源，否则就推迟分配。

当进程在执行中继续申请资源时，先测试该进程已占用的资源数与本次申请资源数之和是否超过了该进程对资源的最大需求量。若超过则拒绝分配资源。若没超过则再测试系统现存的资源能否满足该进程尚需的最大资源量，若满足则按当前的申请量分配资源，否则也要推迟分配。
```

- **安全序列**

```
是指系统能按某种进程推进顺序（P1，P2，P3，...，Pn），为每个进程 Pi 分配其所需要的资源，直至满足每个进程对资源的最大需求，使每个进程都可以顺序地完成。这种推进顺序就叫安全序列【银行家算法的核心就是找到一个安全序列】。
```

- **系统安全状态**

```
如果系统能找到一个安全序列，就称系统处于安全状态，否则，就称系统处于不安全状态。
```