71.MySQL主从同步的一致性怎么保证？MySQL主从同步多长时间同步一次？

一般情况保证最终一致性即可，但有些场景必须保证强一致性，可用全同步复制。

1、异步模式（默认方式）

异步模式下，主节点执行完客户端提交的事务后立即提交事务并返回给客户端，并不关心 log dump 线程是否成功地将将此次事务写进 binglog 并且发送给从库。

假如执行事务的主线程提交事务后，log dump 线程还未来得及写入 binlog，此时系统宕机，则会造成 binglog 中没有保存刚才提交的事务，造成主从数据不一致。

```
    优点：异步模式下，主线程不用关系同步操作，性能最好。
    缺点：可能导致主从数据的不一致。
2、半同步复制
    半同步方式，当主库在执行完客户端提交的事务后不是立即提交事务，而是等待 log dump 线程将此次事务同步到binlog 发送给从库，
        并且至少一个从库成功保存到其relay log中，此时主库的才提交事务并返回客户端。

    优点：相比于异步模式，半同步方式一定程度上保证了数据同步的可靠性。
    缺点：增加了主库响应客户端的延时，延时至少为一个 TCP/IP 的往返时间，即 binglog 发送给从库至收到从库的响应时间。
3、全同步复制（类似于分布式事务二阶段提交的思想）
    全同步方式，当主库在执行完客户端提交的事务后，必须等待此次的binlog发送给从库，并且所有从库成功地执行完该事务后，主库才能返回客户端。其与半同步复制的区别如下：
        半同步下，主库等待binlog写入到从库的relay log即可返回，全同步方式下，必须等到从库执行事务成功。
        半同步下，至少一个从库响应后主库即可返回客户端，全同步下必须等待所有的从库返回。
    优点：对比半同步复制方式，全同步复制方式数据一致性的可靠性进一步提高
    缺点：执行事务时，主库需要等待所有的从库执行成功后才能返回，所以会大大提高主库的响应时间。

默认一分钟同步一次
```

72.僵尸进程和孤儿进程？

一：僵尸进程（有害）

一个进程使用fork创建子进程，如果子进程退出，而父进程并没有调用wait或waitpid获取子进程的状态信息，那么子进程的进程描述符仍然保存在系统中。这种进程称之为僵死进程

我们知道在unix/linux中，正常情况下子进程是通过父进程创建的，子进程在创建新的进程。子进程的结束和父进程的运行是一个异步过程，

即父进程永远无法预测子进程到底什么时候结束，如果子进程一结束就立刻回收其全部资源，那么在父进程内将无法获取子进程的状态信息。

因此，UNIX提供了一种机制可以保证父进程可以在任意时刻获取子进程结束时的状态信息：

1、在每个进程退出的时候，内核释放该进程所有的资源，包括打开的文件，占用的内存等。

但是仍然为其保留一定的信息（包括进程号the process ID，退出状态the termination status of the process，运行时间the amount of CPU time taken by the process等）

2、直到父进程通过wait / waitpid来取时才释放. 但这样就导致了问题，如果进程不调用wait / waitpid的话，那么保留的那段信息就不会释放，

其进程号就会一直被占用，但是系统所能使用的进程号是有限的，如果大量的产生僵死进程，将因为没有可用的进程号而导致系统不能产生新的进程.

此即为僵尸进程的危害，应当避免。

任何一个子进程(init除外)在exit()之后，并非马上就消失掉，而是留下一个称为僵尸进程(Zombie)的数据结构，等待父进程处理。

这是每个子进程在结束时都要经过的阶段。如果子进程在exit()之后，父进程没有来得及处理，这时用ps命令就能看到子进程的状态是“Z”。

如果父进程能及时处理，可能用ps命令就来不及看到子进程的僵尸状态，但这并不等于子进程不经过僵尸状态。

如果父进程在子进程结束之前退出，则子进程将由init接管。init将会以父进程的身份对僵尸状态的子进程进行处理。

二：孤儿进程（无害）

孤儿进程：一个父进程退出，而它的一个或多个子进程还在运行，那么那些子进程将成为孤儿进程。孤儿进程将被init进程(进程号为1)所收养，并由init进程对它们完成状态收集工作。

孤儿进程是没有父进程的进程，孤儿进程这个重任就落到了init进程身上，init进程专门负责处理孤儿进程的善后工作。

每当出现一个孤儿进程的时候，内核就把孤 儿进程的父进程设置为init，而init进程会循环地wait()它的已经退出的子进程。

这样，当一个孤儿进程凄凉地结束了其生命周期的时候，init进程就会代表党和政府出面处理它的一切善后工作。因此孤儿进程并不会有什么危害。

73.为什么linux为什么要有父子进程？

除了第一个进程，任何进程都是fork出来的。子进程fork出父进程的一个副本。

为了方便管理，因为子进程会继承父进程的属性和权限，而父进程也可以系统地管理子进程。这样不管是从安全性还是管理难易程度都是非常好的。

74.进程的创建过程

一旦操作系统发现了要求创建新进程的事件后，便调用进程创建原语create()按下述步骤创建一个新进程。

1）申请空白PCB。为新进程申请获得唯一的数字标识符，并从PCB集合中索取一个空白PCB。

2） 为新进程分配资源。为新进程的程序和数据以及用户栈分配必要的内存空间。显然，此时操作系统必须知道新进程所需要的内存大小。

3） 初始化进程控制块。PCB的初始化包括：

①初始化标识信息，将系统分配的标识符和父进程标识符，填入新的PCB中。

②初始化处理机状态信息，使程序计数器指向程序的入口地址，使栈指针指向栈顶。

③初始化处理机控制信息，将进程的状态设置为就绪状态或静止就绪状态，对于优先级，通常是将它设置为最低优先级，除非用户以显式的方式提出高优先级要求。

4） 将新进程插入就绪队列，如果进程就绪队列能够接纳新进程，便将新进程插入到就绪队列中。

75.tcp的Accept队列。

TCP服务端会维护两个队列，一个是SYN队列（未连接队列），一个是Accept队列（已连接队列）。

当收到客户端发来的SYN报文后，把该连接放入SYN队列中，然后回复SYN+ACK给客户端。

当服务端收到了客户端的第三次握手（ACK）后，就把这个连接放入Accept队列中，此时连接建立完成，三次握手完成。

当应用程序调用Accept（ ）函数时，内核会从Accept队列中选择一个正确的连接取出返回给应用程序。

76.socket编程中的accept()方法作用，会引起惊群吗？

accept()：在一个socket 接受一个连接

accept函数主要用于服务器端，一般位于listen函数之后，默认会阻塞进程，直到有一个客户请求连接，建立好连接后，

它返回的一个新的套接字 socketfd_new ，此后，服务器端即可使用这个新的套接字socketfd_new与该客户端进行通信，而sockfd 则继续用于监听其他客户端的连接请求。

惊群效应是什么惊群效应（thundering herd）是指多进程（多线程）在同时阻塞等待同一个事件的时候（休眠状态），如果等待的这个事件发生，

那么他就会唤醒等待的所有进程（或者线程），但是最终却只能有一个进程（线程）获得这个时间的“控制权”，对该事件进行处理，

而其他进程（线程）获取“控制权”失败，只能重新进入休眠状态，这种现象和性能浪费就叫做惊群效应。

一般都是socket的accept()会导致惊群，很多个进程都block在server socket的accept()，一但有客户端进来，所有进程的accept()都会返回，但是只有一个进程会读到数据，就是惊群。
Linux 2.6 版本之前，监听同一个 socket 的进程会挂在同一个等待队列上，当请求到来时，会唤醒所有等待的进程。
Linux 2.6 版本之后，Linux解决accept惊群的方法是，引入一个排他性标志位（WQ_FLAG_EXCLUSEVE），

用户进程 task 对 listen socket 进行 accept 操作，如果这个时候如果没有新的 connect 请求过来，用户进程 task 会阻塞睡眠在 listent fd 的睡眠队列上。

这个时候，用户进程 Task 会被设置 WQ_FLAG_EXCLUSIVE 标志位，并加入到 listen socket 的睡眠队列尾部

(这里要确保所有不带 WQ_FLAG_EXCLUSIVE 标志位的 non-exclusive waiters 排在带 WQ_FLAG_EXCLUSIVE 标志位的 exclusive waiters 前面)。

根据前面的唤醒逻辑，一个新的 connect 到来，内核只会唤醒一个用户进程 task 就会退出唤醒过程，从而不存在了"惊群"现象。

77.hashmap链表转换红黑树？为什么选择8？

当链表长度大于或等于阈值（默认为 8）的时候，如果同时还满足容量大于或等于 MIN_TREEIFY_CAPACITY（默认为 64）的要求，就会把链表转换为红黑树。

同样，后续如果由于删除或者其他原因调整了大小，当红黑树的节点小于或等于 6 个以后，又会恢复为链表形态。

通常如果 hash 算法正常的话，那么链表的长度也不会很长，那么红黑树也不会带来明显的查询时间上的优势，反而会增加空间负担。
所以通常情况下，并没有必要转为红黑树，所以就选择了概率非常小，小于千万分之一概率，也就是长度为 8 的概率，把长度 8 作为转化的默认阈值。

78.select、poll、epoll区别？

1、select连接数有限制，最大处理连接数不超过1024，epoll连接数很大。

2、select是线性轮询，总连接数很大时性能下降严重。epoll是基于回调callback的事件驱动，不会随着FD数目的增加效率下降。

3、select数组，poll链表，epoll红黑树

4、每次调用selcet、poll每次调用都要把fd集合从用户态拷入内核态。epoll只要拷贝一次。

79.MySQL查询一条语句的执行流程？MySQL服务的构成？

执行流程：

```
构成：
    1 连接层
        提供链接协议(socket,tcp/ip) #这里的socket也不是网络连接的socket,mysql的socket连接只能连接本地
        验证用户的合法性(用户名,密码,白名单)
        提供一个专用连接线程(接收sql,返回结果,将sql语句交给sql层继续处理)
    2 SQL层
        接收到sql语句（判断语法、判断语义、判断语句类型[DML、DDL、DCL、DQL]）
        数据库对象授权检查
        解析SQL语句，生成多种执行计划，MySQL没法直接执行SQL语句，必须解析成执行计划，运行执行计划，最终生成如何去磁盘找数据的方式
        优化器，选择它认为成本最低的执行计划
        执行器，根据优化器的选，按照优化器建议执行sql语句，得到去哪儿找sql语句需要访问的数据：A、具体在哪个数据文件上的哪个数据页中 B、将以上结果充送给下层继续处理
        接收存储引擎层的数据，结构化成表的形式，通过连接层提供的专用线程，将表数据返回给用户
        提供查询缓存，缓存之前查询的数据，假如查询的表是一个经常变动的表，查询缓存不要设置太大， query_cache使用memcached或Redis代替
        日志记录(binlog)
    3 存储引擎层
        接收上层的执行结果
        取出磁盘文件和相应数据
        返回给sql层，结构化之后生成表格，由专用线程返回给客户端
        存储引擎用于：
            存储数据（将SQL语句做的修改转储到磁盘上）
            检索数据（把存进去的数据在提取出来）
```

80.SELECT(0)、SELECT 0、Select Count(0)、Select Count(1)？

SELECT(0),是一个函数
返回当前工作区的编号
SELECT 0 是一个命令
选定未被使用的且最小的可以使用的工作区为当前工作区

```
Select Count(0)和Select Count(1)完全一样
1、一般情况下，Select Count (*)和Select Count(1)两着返回结果是一样的
2、 假如表没有主键(Primary key)，那么count(1)比count(*)快
```

81.系统调用和函数调用有什么区别？

1.使用INT和IRET指令，内核和应用程序使用的是不同的堆栈，因此存在堆栈的切换，从用户态切换到内核态，从而可以使用特权指令操控设备
2.依赖于内核，不保证移植性
3.在用户空间和内核上下文环境间切换，开销较大

4. 是操作系统的一个入口点
函数调用
1.使用CALL和RET指令，调用时没有堆栈切换
2.平台移植性好
3.属于过程调用，调用开销较小
4.一个普通功能函数的调用

82.聚簇索引和非聚簇索引的区别以及各自的优缺点？

聚集索引,表中存储的数据按照索引的顺序存储,检索效率比普通索引高,但对数据新增/修改/删除的影响比较大

非聚集索引,不影响表中的数据存储顺序,检索效率比聚集索引低,对数据新增/修改/删除的影响很小

83.如何打破双亲委派？哪些框架是打破了双亲委派？

如果不想打破双亲委派模型，就重写ClassLoader类中的findClass()方法即可，无法被父类加载器加载的类最终会通过这个方法被加载。

而如果想打破双亲委派模型则需要重写loadClass()方法。

```
典型的打破双亲委派模型的框架和中间件有tomcat与osgi。
```

84.线程池超过核心线程的线程会被回收，如何实现？

阻塞队列的timeout参数

当非核心线程去尝试从队列中获取任务时，获取不到任务会阻塞，如果超过了阻塞队列锁设置的timeout超时参数，即可认为是闲置的非核心线程，回收即可。

85.红黑树的特性？应用场景？

（1）每个节点或者是黑色，或者是红色。
（2）根节点是黑色。
（3）每个叶子节点（NIL）是黑色。 [注意：这里叶子节点，是指为空(NIL或NULL)的叶子节点！]
（4）如果一个节点是红色的，则它的子节点必须是黑色的。
（5）从一个节点到该节点的子孙节点的所有路径上包含相同数目的黑节点。

注意：
(01) 特性(3)中的叶子节点，是只为空(NIL或null)的节点。
(02) 特性(5)，确保没有一条路径会比其他路径长出俩倍。因而，红黑树是相对是接近平衡的二叉树。

```
红黑树的应用：
1、java8 hashmap中链表转红黑树。
    优势：时间复杂度从O(n)-->O(logn) ，且自旋开销较其他树较低（不用整体平衡）。
2、epoll在内核中的实现，用红黑树管理事件块（文件描述符）。
    优势：
        1、因为内核态需要维护一个长久存放fd的数据结构，而fd变动十分频繁，且需要支持快速查询，且所以红黑树很适合。
        2、红黑树可以判断是否是重复的fd
3、Java的TreeMap实现
    优势：相对与hashMap优势，内部key保持有序，且支持自定义排序比较器。
    适用场景，对数据需要排序统计
```

86.如何排查死锁？

jps + jstack。使用jstack工具，是jdkk自带的线程堆栈分析工具。

第一：在windons命令窗口，使用 jps -l

```
第二：使用jstack -l 12316
```

87.磁盘调度算法？

1.先到先服务
　如果访问的磁道很分散，性能上就很差，因为寻道时间过长
2.最短寻道时间优先算法
　可能造成某些请求饥饿，原因是磁头在一小块区域来回移动。
3.扫描算法（电梯算法）
　磁头在一个方向上移动，访问所有未完成的请求，知道磁头到达该方向上的最后一个磁道，才调换方向。
　中间部分磁道比其他部分响应的频率高，即每个磁道的响应频率存在差异。
4.循环扫描算法
　总是按相同的方向扫描，磁道只响应一个方向上的请求，使得每个磁道的响应频率基本一致。
　只有磁头朝某个特定方向移动时，才处理磁道访问请求，复位磁头时直接快速移动至最靠边缘的磁道，返回途中不处理任何请求。
5.LOCK与C-LOCK算法
　磁头在移动到最远的请求位置，立即反向移动。磁头不用移动到磁盘最始端和最末端才开始调换方向。
　LOCK算法，反向移动的途中会响应请求。
　C-LOCK算法，反向移动的途中不会响应请求。

88.为什么选用json序列化而不使用java的JDK序列化？

1.jdk序列化无法跨平台。（解码端也要用java的jdk反序列化）
2.有安全风险。（二进制文件流可能被篡改，而json可以用https：SSL加密，没有被篡改）
3.相比于json序列化，序列更长，传输解析更慢。

89.索引选择性差导致全表搜索解决办法？

1.通过将索引组合提高选择性（业务相关）
　select * from 护士表 where 科室 = '妇科' and 性别 = '女'。
2.引入搜索引擎，如Es或者Solr（更换数据源）
　将护士表导入ElasticSearch，Es基于分片多线程检索，解决查询慢问题。
3.强制使用索引（有时有奇效，以实际运行为准）
　explain select * from question force index（answer） where answer = 'A';
4.增加缓存，提高全表扫描速度。（钞能力）（Redis同理）
　就是给更多的内存去加载,利用内存的高吞吐解决查询慢的问题
　innodb_buffer_pool_size = 16G
　innodb_buffer_poll_instances = 2;

90.查找文件里是否包含某个字符串？

Grep 命令
　参数： -I ：忽略大小写 -c ：打印匹配的行数 -n：打印包含匹配项的行和行标 -v ：查找不包含匹配项的行

```
例子： 在access.log里查找“makehtml_archives”
    grep -n makehtml_archives access.log
```