十大排序：

冒泡排序，O(n^2)，O(1);

选择排序，O(n^2)，O(1);

插入排序，O(n^2)，O(1);

快速排序，O(nlogn)，O(logn)-O(n);

归并排序，O(nlogn)， O(n);

堆排序， O(nlogn)，O(1);

希尔排序，O(n^1.5)，O(1);

桶排序，O(n+k),O(n+k),O(n^2)，O(n);

计数排序，O(nlgn)，O(n),O(nlgn)，O(n);

基数排序，O(nk),O(nk),O(nk)，O(n+k)。

六大设计模式：（创建、行为、结构型各两个）

**作用：降低对象之间的耦合，增加程序的可复用性、可扩展性和可维护性。**

**单例模式**：全局共享一个实例，且只需要被初始化一次的场景，如数据库，全局配置，全局任务池。具体分为**预先加载线程安全的饿汉模式**和**延迟加载线程不安全的懒汉模式**。

**Go中的单例模式懒汉模式的实现：**

package singleton

import (

"sync"

)

type singleton struct {}

var instance \*singleton

var once sync.Once

func GetInstance() \*singleton {

once.Do(func() {

instance = &singleton{}

//这样是并发安全的，once里面的操作在全局中只执行一次})return instance

}

}

**Go中的单例模式饿汉模式的实现：**

type singleton struct {}

var instance = new(singleton)

func GetInstance() \*singleton{

return instance

}

**工厂模式**：类似Java的构造函数。最常用的是弄一个方法根据输入参数构造出想要的实例，无需知道其创建细节。比如 stu1:=student（id，name，call）。

**策略模式**：把具体的行为方法抽象成带参数的接口，给接口传入不同的对象可以实现应对不同场景的策略。比如 person.attack(石头)。

**模板模式**：类似java的父类接口，有两个主要特征继承和多态，子类既能继承父类功能达到复用的效果，又能拓展自己的具体内容（类似java的方法重写）。

**代理模式**：为另一个对象提供一个替身或者占位符，以控制对这个对象的访问，如火车站和售票窗口都实现了售票的方法，售票窗口可以代理火车站的售票。

**选项模式**：结构体参数很多，期望创建一个携带默认值的结构体变量，选择性修改其中一些参数的值。

linux指令：

cd切换目录，ls罗列目录中的子目录和文件，cp复制文件或目录,mv移动文件或重命名，pwd显示当前目录，

rm删除档案或目录，rmdir删除空目录，mkdir创建新目录，touch创建文件，clear清屏，kill杀进程，find目录中查找文件，grep文件中搜索内容，diff比较两个文件，df查看磁盘；

man显示手册页，chmod修改用户权限，chown改变文件所属用户组，tail查看测试项目的日志，cat查看文件内容，top查看所进程情况并持续监视（如平均负载、cpu和内存消耗）,

ps列出正在运行的进程，ps -ef查看进程,

ifconfig查看IP（旧版本），netstat显示各种网络相关信息（查看端口情况），ip addr显示linux的ip地址（新版本）

linux查看文件权限：

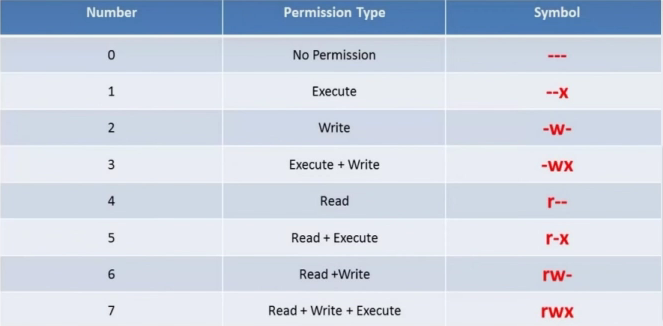
1.Linux的文件权限的描述符必定是十位的字符（例如：drwxr-xrw-）。

2.其中第一个字符‘d’表示是文件夹，‘-‘表示是文件；往后每三位为一组则分为rwx、r-x、rw-，分别对应文件所有者（user）、组（group）、其他人（other）的权限。

3. r（表示可读）、w（表示可写）、x（表示可执行）、-（表示没有该权限）。

4.修改文件的权限的命令：chmod permissions filename（用数字表示）或chmod ugo filename（用ugo和运算符表示）。

（1）用数字表示：



其中，每个数字可由二进制形式来解释含义，如：7=二进制的111，即同时拥有rwx权限。

（2）用ugo和运算符表示：

例：u+x代表给文件所有者加上可执行的权限；

ug+x代表给文件所有者和组都加上可执行的权限；

o-r代表去除其他人的可读权限；

g=rw代表赋予组可读可写的权限，并覆盖掉以前的权限。

数据库：

1. MySQL字符集：



PS: **DDL语句**：数据定义语句(增删改表)；**DML语句**(增删改行数据)；**DQL语句**(查询语句)

1、数据库三范式：

答：列的原子性，非主属性完全依赖主属性不能部分依赖，不能出现传递依赖。

2、InnoDB和MyISAM的区别：

答：InnoDB支持事务，外键，行级锁，MyISAM不支持；

InnoDB是聚簇索引而MyISAM是非聚簇索引；

InnoDB不支持全文索引且不保存表的具体行数而MyISAM支持且保存。

3、MySQL执行查询过程：

答：1.连接器管理客户端的连接，

2.查询缓存是否命中，

3.分析器分析语法，

4.优化器生成执行计划（包括索引选择），

5.执行器操作引擎并返回结果。

在这里存储引擎负责提供读写接口来存储数据。

4、索引是什么? 索引优缺点?

答：索引是一种加快检索速度的数据结构，类似于目录可以快速定位数据，缺点是创建和维护索引需要消耗时间和空间。

5、MySQL索引类型划分？

答：按存储结构划分：B+树索引，B树索引，R树索引，哈希索引，全文索引；

按应用划分：普通索引，唯一索引，联合索引，

按数据存储和索引键值逻辑关系划分：聚簇索引，非聚簇索引。

6、索引底层实现?为什么使用B+树,而不是B树,BST(二叉排序树),AVL(平衡二叉树),红黑树等等?

答：B+树索引的底层数据结构是B+树。

**B+树**：B+树的数据都存储在叶子结点中，分支结点均为索引，方便扫库，只需要扫一遍叶子结点即可，但是B树因为其分支结点同样存储着数据，我们要找到具体的数据，需要进行一次中序遍历按序来扫，所以B+树更加适合在区间查询的情况，

**不用B树：**B树不管叶子节点还是非叶子节点，都会保存数据，这样导致在非叶子节点中能保存的指针数量变少，指针少的情况下要保存大量数据，只能增加树的高度，导致IO操作变多，查询性能变低。

**不用Hash树：**适合**等值查询**，不支持范围查询和排序操作。

其他树同理，树高，IO代价高。

PS: InnoDB里的B+树中的指针为6字节，索引（默认主键，且类型为bigint）为8字节，所以一组为14字节。InnoDB中一个节点为一页，一页为16KB

则：第一层：16x1024➗14＝1170个索引指针

第二层：1170x1170＝1368900个索引指针

第三层：1368900x16x1024/每条数据的大小=总共存储的数据量

第n层：1170的n-1次方x16x1024/每条数据的大小=总共存储的数据量

7、什么是聚簇索引和非聚簇索引?

答：聚簇索引的叶子节点存储的是整行数据；

非聚簇索引的叶子节点存储的是主键的值，还需要回表才能找到整行数据的值。

8、非聚簇索引一定会回表吗?

答：不一定，覆盖索引不会回表，即查询语句所要求的字段全部命中了索引。

9、什么是联合索引?为什么需要注意联合索引中的字段顺序?

答：就是使用多个字段建立一个索引，索引会先按照第一个字段排序，在第一个字段值相等的情况下才按第二个字段排序，以此类推，所以使用索引时要按照字段顺序逐个使用才能命中索引，因此可以把查询频率较高的字段放在第一位。

10、什么是最左前缀原则?

答：联合索引中，利用索引列中最左的字段优先进行匹配，若最左的字段不存在则不走索引，若跳过某一列，则后面的字段不走索引，且针对>、<的范围查询时，右侧的列索引失效。

11、什么是前缀索引?

答：在对字符串创建索引时，可以使用字符串从左开始的部分字符创建索引, 减少空间和维护的成本, 但是也会降低索引的选择性。（选择性：不重复的索引值占总数据的比例）

\*\*12、什么是索引下推?

答：联合查询时，没有索引下推的话在定位完第一个字段索引后需要一条条进行回表查询, 然后再判断其他字段是否满足条件；而索引下推可以对索引中包含的字段先进行判断过滤, 然后再进行后续操作, 减少了回表次数，提高查询效率节约IO开销。

（如：select id from student where name=’小明’ and age=18;在这里查询完name字段如果没有索引下推的话会根据id值一个一个回表查询age字段是否符合）

13、如何查看MySQL语句是否使用到索引?

答：EXPLAIN SQL语句重点看possible\_key: 可能用到的索引(可以查看是否有冗余索引)；key: 真正使用到的索引。

\*\*14、为什么建议使用自增主键作为索引?

答：索引维护可能造成页分裂, 自增主键减少数据的移动和页分裂。

\*\*15、建立索引的原则？

答：**针对以下字段建立索引：**

1.索引字段非空，

2.索引字段小占用空间少（使用前缀索引），

3.不参与计算，

4.选择性接近1区分度高的字段；

**建立索引的原则：**

1.最左前缀原则，

2.尽量使用联合索引减少单列索引；

3.尽量拓展索引不要新建索引让索引冗余；

4.控制索引数量，索引数量太多会影响增删改效率。

16、InnoDB逻辑存储结构？

表空间 -> 段 -> 区 -> 页 -> 行

表空间：一张表对应一个表空间，用于存储记录、索引等数据

段：分为数据段、索引段、回滚段。数据段就是B+树的叶子节点，索引段即为B+树的非叶子节点

区：表空间的单元结构，每个区的大小为1M。

页：InnoDB 存储引擎磁盘管理的最小单元，每个页的大小默认为16KB。

行：存储表中一行行的数据。

17、InnoDB内存结构？

答：包含Buffer Pool(缓冲池)、Change Buffer(更改缓冲区)、Log Buffer(日志缓冲区)、Adaptive Hash lndex(自适应哈希索引)。

**Buffer Pool：**缓存磁盘上经常操作的真实数据，在执行增删改查操作时，先操作缓冲池中的数据(若缓冲池没有数据，则从磁盘加载并缓存)，然后再以一定频率刷新到磁盘，从而减少磁盘I/O，加快处理速度。

**Change Buffer：**增删改的数据没有在Buffer Pool中，不会直接操作磁盘，而会将数据变更存在Change Buffer 中，在未来数据被读取时，再将数据合并恢复到Buffer Pool中，再将合并后的数据刷新到磁盘中。

**Log Buffer：**保存要写入磁盘中的log日志数据，日志缓冲区的日志会定期刷新到磁盘中。

**Adaptive Hash lndex：**如果观察到hash索引可以提升速度，则建立hash索引。

18、InnoDB后台线程？



19、什么是事务？

答：一组操作的集合称为事务。事务内的操作全部执行, 要么都不执行。

20、事务的四个特征？

答：原子性（**Atomicity**），一致性（**Consistency**），

隔离性（**Isolation**），持久性（**Durability**）。

21、MySQL四种隔离级别？

| **隔离级别** | **脏读** | **不可重复读** | **幻读** |
| --- | --- | --- | --- |
| 读未提交 | √ | √ | √ |
| 读已提交 | × | √ | √ |
| 可重复读(默认) | × | × | √ |
| 串行化 | × | × | × |

PS：√代表会发生，×代表不会发生。

22、什么是脏读? 不可重复读? 幻读?

答：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；

一个事务先后读取同一条记录，但两次读取的数据不同；

同一事务两次读取的中间有另一事务对数据进行插入或删除，导致第二次读取到一些原本不存在的数据或无法读取到第一次读到的数据。

23、事务是如何实现的(原理) ?

答：redo log 实现持久性；undo log 实现原子性；锁和MVCC实现隔离性；redolog和undolog共同实现了一致性。

24、介绍下MySQL事务日志? redo log和undo log?

答：InnoDB的事务日志包括重做日志和回滚日志。

重做日志是将事务中用户操作的任何最新数据备份到一个地方，一旦故障重启可以根据重做日志进行重做，实现事务的持久性；（备份新数据）

回滚日志则是在事务开始时就把需操作的数据备份到一个地方，可以通过回滚得到数据的旧版本实现了事务的原子性。（备份旧数据）

25、什么是binlog?

答：只记录数据库表结构和表数据的变更的二进制日志，而不会记录select这类操作。可以用来做**数据恢复和主从复制。**

26、了解MVCC吗?说下什么是MVCC?

答：多版本并发控制，通过快照读的方式实现，也就是保存同一条数据的不同历史版本链。可以解决数据库读-写并发冲突，而不必通过加锁的形式来解决，提高了数据库并发性能。

27、MVCC实现原理? 有什么好处?

答：**实现原理**：主要依赖于每行数据的两个隐藏字段（最近修改事务id和回滚指针）、undolog和readview实现。

Readview维护了一个当前活跃事务id数组，回滚指针指向该行数据修改前的历史版本（存储在undo log中），同时用事务ID来记录历史版本号，这样在同一事务中只要回溯到同一个事务ID就可以解决不可重复读问题。

**好处**：而不必通过加锁的方式来解决读写冲突，极大增加了数据库的并发性。

28、RC（读已提交read commited）和RR（可重复读repeatable read）级别下MVCC的差异？

**RC**：每次SELECT查询前都生成一个Read View

**RR**：事务开启后第一次SELECT数据时生成一个Read View，之后复用前面的Read View。

29、MySQL如何解决幻读问题？

答：InnoDB在可重复读隔离级别(默认隔离级别)下通过加Next-key lock解决幻读问题。在select完之后给查询范围内的行数据上行锁，给指定范围的间隙上间隙锁。

30、快照读和当前读？

答：当前读：读取到的是数据的最新版本。update、delete、insert都是当前读。

快照读：读取到的是数据快照，即数据的历史版本。普通的查询select是快照读。

31、为什么需要加锁?

答：在对同一数据并发存取时不加锁可能导致存取的数据有误，破坏数据库的一致性。

32、MySQL锁粒度?

答：全局锁，表锁，行锁。

（行锁包括：记录锁、间隙锁、next-key lock(行锁和间隙锁的组合)、插入意向锁、隐式锁）

33、MySQL有哪些锁?

答：共享锁（读锁）和排它锁（写锁）。

34、乐观锁和悲观锁是什么?如何实现?

答：乐观锁假定没有发生并发冲突，只在事务提交时才检查是否违反数据库一致性，通过版本号机制和CAS算法实现；

悲观锁则假定最坏情况认为会产生并发冲突，事先通过锁机制屏蔽掉一切可能违反数据库一致性的操作，通过数据库锁机制实现。

读多写少则用乐观锁，写多则用悲观锁。

35、InnoDB的行锁是如何实现的?

答：InnoDB 行锁是通过给索引上的索引项加锁来实现的。

36、什么是两阶段锁协议?

答：在InnoDB事务中, 行锁是需要的时候才加上的, 但并不是不需要了就立刻释放, 而是等到事务结束时才释放. 这就是两阶段锁协议。

场景题：假如有一张表，有一千多万行数据，想要查询倒数第 100 行，此时使用 LIMIT 比较慢，有什么解决方案？

答：子查询+覆盖索引。即子查询先走索引（直接走主键id的唯一索引）得到一系列id值，再用得到的id值回表得到一整行数据。

**操作系统：**

**进程和线程**

1、进程有哪些状态？

答：创建、就绪、运行（执行）、终止、阻塞。

2、同步与异步的区别是什么？

同步与异步的区别在于调用结果的通知方式上。

同步：执行完函数或方法后，一直阻塞等待系统返回值或消息，接收到返回值或消息后才往下执行其他的命令。

异步：执行完函数或方法后，不必阻塞性地等待返回值或消息。

3、孤儿进程,僵尸进程,守护进程？

孤儿进程：父进程已退出而子进程还在运行，则该子进程称为孤儿进程。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。

僵尸进程：如果子进程退出，但还没有被父进程调用wait或waitpid获取状态信息，那么子进程的进程描述符仍然保存在系统中。这种进程称之为僵尸进程。

守护进程：守护进程是一种独立于控制终端，不能直接和用户交互，运行在后台保证系统安全的特殊进程。

4、进程和线程的区别？

答：（1）进程是资源管理的基本单位，线程是程序执行的基本单位。

（2）每个进程拥有自己的资源而线程共享其隶属进程的资源。

（3）创建、切换、撤销进程的开销远大于线程的开销。

5、线程的分类？

答：内核级线程，用户级线程（协程）。

6、协程与线程的区别？

答：（1）线程和进程都是同步机制，而协程是异步机制；

（2）线程是抢占式，而协程是非抢占式的；

（3）一个线程可以有多个协程；

（4）协程不被操作系统内核管理，而完全是由程序控制；

（5）协程能保留上一次调用时的状态。

7、并发和并行有什么区别？

答：并发是通过足够快的切换速度在宏观上表现为在一段时间内能同时运行多个程序；

并行则是在同一时刻，有多个任务在执行，是物理上的多个进程同时进行。

8、进程与线程的切换流程？

答：（1）切换页表以使用新的地址空间

（2）切换内核栈和硬件上下文。

对于线程切换，第1步是不需要做的，第2步是进程和线程切换都要做的。

因为每个进程都有自己的虚拟地址空间，而线程是共享所在进程的虚拟地址空间的，因此同一个进程中的线程进行线程切换时不涉及虚拟地址空间的转换。

9、什么是用户态和内核态？

答：内核态：内核态运行的程序可以访问计算机的任何数据和资源。处于内核态的 CPU 可以从一个程序切换到另外一个程序，并且独占 CPU 不会发生CPU抢占情况。

用户态：用户态运行的程序只能受限地访问内存，只能直接读取用户程序的数据，并且不允许访问外围设备，用户态下的 CPU 不允许独占，也就是说 CPU 能够被其他程序获取。

10、 用户态和内核态是如何切换的?

答：用户态切换到内核态主要是通过系统调用、异常、中断

（1）系统调用：用户态进程通过系统调用申请使用操作系统提供的服务程序完成工作。

（2）异常：用户态下的程序运行时发生了异常，CPU就会从用户态切换到内核态，比如缺页异常。

（3）外围设备的中断：当外围设备完成用户请求的操作后，会向CPU发出相应的中断信号。

内核态切换到用户态是通过设置程序状态字PSW

11、中断的处理过程?

答：保护现场，开中断，中断处理，关中断，恢复现场。

PS:缺页中断时，操作系统通过查找页表得知所缺页在磁盘的物理地址。

12、中断和轮询有什么区别？

答：轮询是CPU对特定设备的轮流询问，中断是通过特定事件提醒CPU；轮询效率低等待时间长，中断容易遗漏问题。

13、进程调度策略有哪几种？

答：先来先服务，短作业优先，最短剩余时间优先，高响应比优先，时间片轮转，优先级调度，多级反馈队列。

14、什么是临界区，如何解决冲突？

答：一次仅允许一个进程使用的资源称为临界资源，每个进程中访问临界资源的那段程序称为临界区。

解决冲突：（1）一次仅允许一个进程进入临界区，如已有进程进入自己的临界区，则其它所有试图进入临界区的进程必须等待；(一次进一个，排队)

（2）进入临界区的进程要在有限时间内退出。(进去一定时间要出来)

（3）如果进程不能进入自己的临界区，则应让出CPU，避免进程出现“忙等”现象。(进不去要放弃)

15、同步和互斥的区别：

同步：不同进程的若干程序片断按照一定的先后次序来运行。或不同进程必须按照一定的先后次序访问资源。

互斥：当某个进程在运行时，其它进程不能运行，只能等到该进程运行完这个程序片段后才可以运行。或一次只能有一个进程访问资源。

16、进程间通信方式有哪些？

答：管道，信号，信号量+共享内存，消息队列，socket。

17、线程间的通信方式

（1）使用全局变量：主要由于多个线程可能更改全局变量，因此全局变量最好声明为volatile

（2）使用消息实现通信

（3）使用事件CEvent类实现线程间通信

18、线程同步的方式有哪些？

答：临界区，互斥量，信号量，事件。

**死锁相关问题**

19、什么是死锁？

答：并发过程中，多个进程持有一部分资源又互相等待对方让出资源，导致整个系统陷入停滞。

20、死锁产生的条件？

互斥条件：一个资源一次只能被一个进程使用。

请求与保持条件：进程得不到所请求的资源时，自身阻塞的同时保持所持有的资源不释放。

不剥夺条件：进程所获得的资源在未使用完之前，不能由其他进程强行夺走，只能主动释放。

循环等待条件：存在进程资源的循环等待链。

21、如何处理死锁问题

死锁预防：确保死锁发生的四个必要条件中至少有一个不成立

死锁避免：银行家算法（让循环等待条件不成立）

死锁检测：

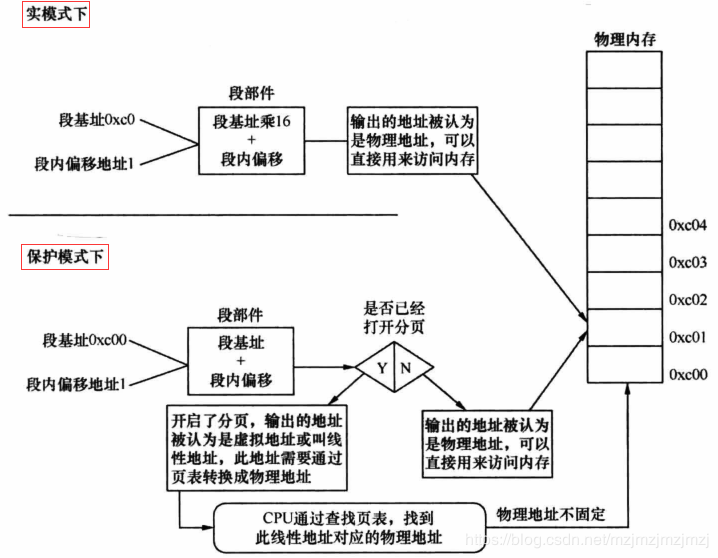
死锁解除：（1）资源抢占（2）进程终止法（3）进程回退法

鸵鸟策略：忽略死锁

**内存管理**

22、物理地址、逻辑地址、有效地址、线性地址、虚拟地址的区别?

答:物理地址就是内存中真正的地址，在保护模式下段基址 + 段内偏移被称为线性地址,一旦开启分页离散存储，线性地址也叫虚拟地址，段内偏移地址就是逻辑地址也叫有效地址。



段基址+段内偏移量（逻辑地址、有效地址）🡺分段查询🡺线性地址（虚拟地址）🡺分页查询🡺物理地址

23、什么是分页？

答：把一个进程的内存空间划分为大小相等的块称为页离散存储在实际内存中，同时用一个页表来记录页号与物理块号的映射关系，目的是最大限度利用内存空间。

24、什么是分段？

答：把一个进程按用户意愿有逻辑性地划分，然后段表来记录段号与物理块号的映射关系。

25、分页和分段有什区别？

答：分页对程序员是透明的，大小是不可变的，而分段是程序员根据逻辑需求灵活划分的；分页地址空间是一维的，给出逻辑地址即可，而分段是二维的，需给出段号和段内偏移量；分页是为了最大化利用内存，分段是有助于数据的共享和保护。

26、什么是交换空间？

答：当内存资源不足时，操作系统会把一些不常用的页转移到硬盘的一块空间上，以释放内存空间。硬盘上的那块空间叫做交换空间,而这一过程被称为交换。物理内存和交换空间的总容量就是虚拟内存的可用容量。

27、什么是虚拟内存？

答：操作系统通过交换操作让物理内存扩充成更大的逻辑内存，从而让程序获得更多的可用内存。

28、虚拟内存的实现方式有哪些?

答：虚拟内存的实现需要建立在离散分配的内存管理方式的基础上，主要有3种方式：请求分页存储管理，请求分段存储管理，请求段页式存储管理。

29、页面替换算法有哪些？

答：最优算法，最近未使用算法，先进先出算法，第二次机会算法，时钟算法

30、为什么虚拟地址空间切换会比较耗时？

答：每个进程有自己的虚拟地址空间，从虚拟地址映射到物理地址需要查找页表，这个过程会比较慢，所以会用cache来缓存频繁的地址映射加快速度。一旦切换进程使用新的页表，前面缓存的cache也就失效，缓存命中率降低导致虚拟地址映射到物理地址的速度变慢，表现出来就是程序运行变慢。

31、什么是缓冲区溢出？有什么危害？

答：缓冲区溢出是指当计算机向缓冲区填充数据时超出了缓冲区本身的容量，溢出的数据覆盖在合法数据上。会造成程序崩溃或跳转执行恶意代码。

**I/O管理**

32、讲一讲IO多路复用？

答：IO多路复用是指内核一旦发现进程指定的一个或者多个IO条件准备读取，它就通知该进程。（**内核** 发现进程指定的IO条件准备读取 **通知该进程**）

33、硬链接和软链接有什么区别？

答：硬链接就是在目录下创建一个条目，记录着文件名与 inode 编号，这个 inode 就是源文件的 inode。（类似于对象引用）

软链接文件保存着源文件所在的绝对路径，在读取时会定位到源文件上。（就像Windows的快捷方式）

34、Unix 常见的IO模型？

答：阻塞式IO模型，非阻塞式IO模型，IO复用式IO模型，信号驱动式IO模型，异步IO式IO模型。

35、select、poll 和 epoll 之间的区别?

答：（1）select 仅仅知道有 I/O 事件发生，但并不知道是哪几个流，所以只能无差别轮询所有流，时间复杂度为O（n）；

（2）poll 本质上和 select 没有区别， 但是它没有最大连接数的限制，原因是它是基于链表来存储的；

（3）epoll 可以理解为 event poll，不同于无差别轮询，它是事件驱动的，时间复杂度为O（1）。

36.介绍一下epoll的水平触发和边缘触发？

答：

**水平触发：当缓冲区有数据时，就会发出通知让内核调用read()读取。**

优点：保证了数据的完整输出；

缺点：当数据较大时，需要不断从用户态和内核态切换，消耗了大量的系统资源，影响服务器性能；

应用场景：应用较少，一般用于连接请求较少及客户端发送的数据量较少的服务器，可一次性接收所有数据。

**边缘触发：当缓冲区的状态发生变化时才会发出通知让内核调用read()读取，比如缓冲区中来了新的数据。边沿触发通常与非阻塞IO一起使用**，其工作模式为：epoll\_wait触发一次，在while循环内非阻塞IO读取数据，直到缓冲区数据为空（保证了数据的完整性），内核才会继续调用epoll\_wait等待事件发生。

边沿触发（EPOLLET）+非阻塞IO的优缺点如下：

优点：每次epoll\_wait只用触发一次，就可以读取缓冲区的所有数据，工作效率高，大大提升了服务器性能；

缺点：数据量很小时，至少需要调用两次非阻塞IO函数，而边沿触发只用调用一次。

**计算机网络**

1、什么是网络协议，为什么要对网络协议分层 ?

答：计算机在通信过程中要遵循的一些约定好的规则。方便实现和维护，层与层互不影响，有利于标准化制定。

2、说说 HTTP 常用的状态码及其含义？

答：1xx信息性状态码，2xx成功状态码，3xx重定向状态码，4xx客户端错误状态码，5xx服务端错误状态码。

典型404无法找到客户端请求的资源。301永久重定向（会缓存），302临时重定向（不会缓存）。

3、HTTP 常用的请求方式，区别和用途？



4、请简单说一下你了解的端口及对应的服务？



53、80、443、1080、3306

5、说下计算机网络体系结构

答：OSI七层模型：物理层，链路层，网络层，传输层，会话层，表示层，应用层。通常在四层模型或五层模型里后三层统称应用层。

6、如何理解 HTTP 协议是无状态的

答：协议对于事务处理没有记忆能力，不对通信状态进行保存，服务器无法判断用户身份。即打开一个服务器上的网页和上一次打开这个服务器上的网页之间没有任何联系。服务器不会去记住你是谁，所以是无状态协议。

7、 从浏览器地址栏输入 url 到显示主页的过程？

答：1、查缓存直到DNS 解析，查找域名对应的 IP 地址；2、与服务器通过三次握手，建立 TCP 连接；3、向服务器发送 HTTP 请求；4、服务器处理请求，返回网页内容；5、浏览器解析并渲染页面；6、TCP 四次挥手，连接结束。

8、说下 HTTP/1.0，1.1，2.0 的区别

答：HTTP/1.0默认是短连接，服务完立即断开；HTTP/1.1 默认长连接，该连接可以被多个请求复用；HTTP/2.0 采用多路复用。

9、长连接和短连接？

答：短连接每进行一次HTTP操作，就建立一次连接，任务结束就中断。长连接传输数据的TCP连接不会关闭，之后再进行访问就直接用已建立的连接。

10、POST 和 GET 有哪些区别？

答：GET只产生一个TCP包，POST可能产生两个；GET只能进行URL编码，参数包含在URL中用&连接起来，参数数据类型为ASCII字符，POST支持多种编码方式，参数不限制数据类型，通过request body传递；GET的请求和参数会被缓存起来同时支持收藏为书签而POST不行。

11、在交互过程中如果数据传送完了，还不想断开连接怎么办，怎么维持？

答：在 HTTP 中响应体的Connection字段指定为keep-alive。

12、HTTP 如何实现长连接？在什么时候会超时？

答：在 HTTP 中响应体的Connection字段指定为keep-alive。从 HTTP/1.1 以后，连接默认都是长连接。超时问题：HTTP 一般会有 httpd 守护进程，里面可以设置keep-alive timeout，当 tcp 连接闲置超过这个时间就会关闭。

13、HTTP 与 HTTPS 的区别？

答：HTTP存在以下问题：

1.请求信息明文传输容易窃取，

2.对方身份未确认容易冒充，

3.数据完整性未校验容易篡改。

Https 是身披 SSL(安全套接层) 的 HTTP，可以解决这些缺点实现安全传输，具体区别体现在不同端口，消耗资源，证书需要，报文加密。

14、Https 流程是怎样的？

答：1、客户端发起 Https 请求，连接到服务器的 443 端口。

2、服务器将自己的数字证书发送给客户端，证书里面有服务器的公钥，私钥由服务器持有。3、客户端收到数字证书进行验证，合法则用里面的公钥对自己生成的对称密钥进行加密。4、然后客户端将公钥加密后的密钥发送到服务器。

5、服务器接收到客户端发来的密文密钥之后，用自己之前保留的私钥对其进行非对称解密，解密之后就得到客户端的密钥。

6、这样，接下来客户端和服务端可以用这个对称密钥对传输数据进行加密和解密。

15、说说什么是数字签名？什么是数字证书？

答：将重要信息（公钥和个人信息）经过 Hash 摘要算法加密形成消息摘要，再将消息摘要交给可信的第三方认证中心（CA机构），用它的私钥对消息摘要加密，形成数字签名。公钥和个人信息、数字签名共同构成数字证书。当然，要验证证书合法性同样找CA机构用他们的私钥去解密看看里面的信息是不是一样。

16、对称加密与非对称加密有什么区别？

答：对称加密：指加密和解密使用同一密钥，优点是运算速度较快，缺点是如何安全将密钥传输给另一方。非对称加密：指的是加密和解密使用不同的密钥（即公钥和私钥）。公钥与私钥是成对存在的，如果用公钥对数据进行加密，只有对应的私钥才能解密。常见的非对称加密算法有 RSA。

17、说说 DNS 的解析过程？（迭代查询）

答：（1）、查找浏览器的缓存看有没有对应的IP地址，

（2）、将请求发往给本地 DNS 服务器查找有没有对应的IP地址，

（3）、本地 DNS 服务器向根域名服务器发送请求，根域名服务器返回负责.com的顶级域名服务器的 IP 地址的列表。

（4）、本地 DNS 服务器再向其中一个负责.com的顶级域名服务器发送一个请求，返回负责.baidu的权限域名服务器的 IP 地址列表。

（5）、本地 DNS 服务器再向其中一个权限域名服务器发送一个请求，返回域名所对应的 IP 地址。

18、什么是 CSRF 攻击，如何避免？

答：CSRF是跨站请求伪造，是一种挟制用户在当前已登录的 Web 应用程序上执行非本意的操作的攻击方法。方法：检查referer、添加校验token。

19、聊聊五层计算机网络体系结构中，每一层对应的网络协议有哪些？

答：网络层：IP协议，ARP协议，RIP协议（路由信息交换协议），ICMP协议；

传输层：TCP（传输控制协议）,UDP（用户数据报文协议）；

应用层：HTTP（超文本传输协议），FTP（文本传输协议），SMTP（简单邮件传输协议），DNS（域名系统），SSH（安全外壳协议）。

20、TCP 和 UDP 分别对应的常见应用层协议有哪些？

答：TCP：HTTP，FTP，SMTP，TELNET，SSH；

UDP：DNS，TFTP，SNMP。

21、说说 WebSocket 与 socket 的区别？

答：Socket 其实就是等于IP 地址 + 端口 + 协议，是网络编程的标准接口，支持不同进程间的通信。

WebSocket 是一个持久化的协议，用来解决http 不支持持久化连接的问题，是应用层通信协议。

22、什么是 DoS、DDoS、DRDoS 攻击？

答：引起拒绝服务的攻击；分布式拒绝服务，不同位置的多个攻击者同时向一个或几个目标发动攻击；分布式反射拒绝服务，该方式靠的是发送大量带有被害者 IP 地址的数据包给攻击主机，然后攻击主机对 IP 地址源做出大量回应，从而形成拒绝服务攻击。

23、什么是 XSS 攻击，如何避免?

答：跨站脚本攻击，攻击者往 Web 页面里插入恶意 html 代码，当用户浏览该页面时，嵌入 Web 里面的 html 代码会被执行，从而达到恶意攻击用户的特殊目的。

解决：1.对输入过滤，2.限制输入长度，3.对链接跳转要校验内容，4.HTML转义。

24、Http 请求的过程与原理

答：1、查缓存直到DNS 解析，查找域名对应的 IP 地址；2、与服务器通过三次握手，建立 TCP 连接；3、向服务器发送 HTTP 请求；4、服务器处理请求，返回网页内容；5、浏览器解析并渲染页面；6、TCP 四次挥手，连接结束。

25、forward 和 redirect 的区别？

答：forward 是直接转发方式，客户端只发出一次请求，由服务器第二个信息资源响应该请求； redirect 是间接转发方式，实际是两次 HTTP 请求，服务器端在响应第一次请求的时候，让浏览器再向另外一个 URL 发出请求，从而达到转发的目的。

26、聊聊 SQL 注入？

答：通过客户端传入特殊字符来欺骗服务端执行恶意指令，如''or'1'='1'。解决：不相信任何外部输入参数，过滤参数中含有的一些数据库关键词。

27、Session 和 Cookie 的区别？

答：Cookie：在客户端保存用户信息，安全性较差，并且浏览器会限制cookie数量。

Session：在服务端记录用户状态，安全性较高，但占用服务器性能。

Token：在客户端保存认证授权状态，适合前后端分离的项目(处于不同服务器)。

28、IP 地址有哪些分类？

答：五类。IP 地址 = 网络号 + 主机号，A类：0开头，前8位为网络号；；B类：10开头，前16位为网络号；C类：110开头，前24位为网络号；D类：1110开头；E类：11110开头。

29、说下 ARP 协议的工作过程？

答：1、每台主机有自己的ARP列表记录 IP 地址和 MAC 地址的对应关系；

2、当源主机发送数据包时会先检查自己的ARP列表看看有没有对应的mac地址，如果没有就向本地网段发起一个 ARP 请求的广播包，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，就会忽略此数据包；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；

4、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。

30、有了 IP 地址，为什么还要用 MAC 地址？

答：一开始只有Mac地址没有IP地址，随着网络中的设备越来越多，整个路由过程越来越复杂，便出现了子网的概念。IP地址的出现与子网有关，对于同一个子网上的设备，IP 地址的前缀都是一样的，这样路由器通过 IP 地址的前缀就知道设备在在哪个子网上，方便路由器将数据包扔给对应的子网。IP 地址可以比作为地址，MAC 地址为收件人，在一次通信过程中，两者是缺一不可的。只用IP地址因为可以自行更改管理起来会很麻烦。

31、聊聊保活计时器的作用（客户端连接成功之后发生故障怎么处理）

答：TCP有个保活计时器，服务器每收到一次客户的数据，就重置保活计时器，时间的设置通常是两个小时。若两个小时都没有收到客户端的数据，服务端就发送一个探测报文段，以后则每隔 75 秒钟发送一次。若连续发送 10 个探测报文段后仍然无客户端的响应，服务端就认为客户端出了故障，接着就关闭这个连接。

（两小时没收到客户端数据就发一次探测报文段，每75秒发一次，连续10次后仍无响应视为故障，关闭连接）

32、如果服务器出现了大量 CLOSE\_WAIT 状态如何解决。

答：服务器端收到客户端发送的FIN后，TCP 协议栈就会自动发送 ACK，接着进入CLOSE\_WAIT状态。但是如果服务器端不执行 socket 的 close() 操作，那么就没法进入 LAST\_ACK, 导致大量连接处于 CLOSE\_WAIT 状态。所以，如果服务器出现了大量CLOSE\_WAIT状态，一般是程序 Bug，或者关闭 socket 不及时。

33、URI 和 URL 的区别

答：URI是统一资源标志符，主要作用是唯一标识一个资源。URL是统一资源定位符，主要作用是提供资源的路径。（例如：URI 像是身份证，可以唯一标识一个人，而 URL 更像一个住址，可以通过 URL 找到这个人。）

34、ICMP 协议的功能

答：控制消息协议，是一种面向无连接的协议，用于传输出错报告控制信息。日常使用得比较多的ping，就是基于 ICMP 的。

35、说下 ping 的原理

答：ping常用于测试目的主机是否可达及了解其有关状态。工作过程：向目的主机发送多个ICMP回送请求报文，根据目的主机返回的回送报文的时间和成功响应的次数估算出数据包往返时间及丢包率。

36、请详细介绍一下 TCP 的三次握手机制

答：第一次握手客户端发送连接请求报文，发送完毕后，就进入 SYN\_SENT 状态；第二次握手服务端发送连接确认报文，发送完毕后，服务器端就由监听状态进入 SYN\_RCV 状态；第三次握手客户端发出连接确认报文，发送完毕后，客户端进入 ESTABLISHED 状态，当服务器端接收到这个包时，也进入 ESTABLISHED 状态。

37、TCP 握手为什么是三次，为什么不能是两次？不能是四次？

答：两次不可靠，四次太多余。（两次握手的话，服务端不能确保自己的连接确认报文被客户端成功接收到，不能保证自己的发送能力和客户端的接受能力正常）

第一次握手，Server知道Client的发送能力和自己的接收能力是正常的；

第二次握手，Client知道Server的发送和接收能力和自己的发送和接收能力是正常的，但是Server还不知道自己的发送能力和客户端的接收能力正常与否；

第三次握手，Client回馈，让Server知道自己的发送能力和Client的接收能力正常。

总之就是让双方都确认对方和自己的接收和发送功能是正常的。

38、说说 TCP 四次挥手过程

答：第一次挥手：客户端发送连接释放报文，发送完毕后，客户端进入FIN\_WAIT\_1状态；第二次挥手：服务器发出确认收到报文，发送完毕后，服务器端进入CLOSE\_WAIT状态，客户端接收到这个确认包之后，进入FIN\_WAIT\_2状态；

第三次挥手：服务端关闭连接，发送连接释放报文，发送完毕后，服务器端进入LAST\_ACK状态，等待来自客户端的最后一个 ACK；

第四次挥手： 客户端接收到来自服务器端的关闭请求，发回确认收到报文，客户端进入 TIME\_WAIT 状态，等待了某个固定时间（两个最大段生命周期）之后，没有收到服务器端的 ACK ，认为服务器端已经正常关闭连接，于是自己也关闭连接，进入 CLOSED 状态。服务器端接收到这个确认包之后，关闭连接，进入 CLOSED 状态。

39、TCP 挥手为什么需要四次呢？

答：客户端发出断开连接请求时，服务端可能还有数据没有发送完毕，所以客户端发出连接释放通知，服务端确认收到后，客户端就进入半关闭状态（只收消息不发消息），服务端把未发完的数据发送完毕后，发送连接释放通知，客户端确认后就关闭连接，刚好四次。

40、TCP 四次挥手过程中，为什么需要等待 2MSL, 才进入 CLOSED 关闭状态？

答：最主要是保证客户端发送的最后一个 ACK 报文段能够到达服务端。这样如果最后一个 ACK 报文段丢失，服务端会超时重传这个 FIN+ACK 报文段，而客户端就能在 2MSL 时间内（超时 + 1MSL 传输）收到这个重传的 FIN+ACK 报文段。其次就是防止已失效的连接请求报文段出现在本连接中。

41、为什么会产生粘包和拆包呢?

答：粘包：1.要发送的数据小于TCP发送缓冲区大小，TCP将多次写入缓冲区的数据一性

性发送；

2.接收端没有及时读取接收缓冲区中的数据。

拆包：1.发送端要发送的数据大于 TCP 发送缓冲区剩余空间大小。

2.发送端要发送的数据大于最大报文长度，TCP在传输前就进行拆包。

解决方案：

1.发送端将每个数据包封装为固定长度，在数据尾部增加特殊字符进行分割

2.将数据分为两部分，一部分是头部，一部分是内容体；其中头部结构大小固定，且有一个字段声明内容体的大小。

42、聊聊 TCP 的流量控制

答：流量控制：一种可以让发送端根据接收端的实际接收能力控制发送数据量的机制，主要通过滑动窗口实现。也就是每次发送数据都会根据接收端的剩余接收空间不断发送直到剩余接收空间为0，停止发送并开启一个定时任务，每隔一段时间就去询问接受方，直到接收空间大于 0，才继续开始发送。

43、说说半连接队列和 SYN Flood 攻击的关系

答：TCP 进入三次握手前，服务端会从CLOSED状态变为LISTEN状态, 同时在内部创建了两个队列：半连接队列（SYN 队列）和全连接队列（ACCEPT 队列）。当服务端发出第二次握手后，需要等待客户端的最后一个ack，该连接会被放入半连接队列去等待。三次握手完成该连接就会从半连接队列放到全连接队列。SYN Flood 是一种典型的 DDos 攻击，它在短时间内，伪造不存在的 IP 地址, 向服务器大量发起连接请求却又不回复最后一个ack，让服务器的连接一直等待最后一个ack，导致服务器上建立大量的半连接半连接队列满了，这就无法处理正常的 TCP 请求。

44、聊聊 TCP 的滑动窗口

答：为了解决发送端根据返回信息来决定发送与否效率低的问题，TCP 引入了窗口其实也就是发送缓冲区，TCP 报文首部有个字段win控制窗口大小，窗口大小值表示无需等待确认应答，而可以继续发送数据的最大值。接受方每次收到数据包，在发送确认报文的时候，同时告诉发送方，自己的缓存区还有多少空余空间，缓冲区的空余空间，我们就称之为接受窗口大小。这就是 win。

45、TCP 的拥塞控制

答：流量控制是作用于接收者的，拥塞控制是作用于网络的，防止过多的数据包注入到网络中，避免出现网络负载过大的情况，类似让水管达到最佳充满状态又不撑爆水管。

主要有这几种算法实现：慢开始算法、拥塞避免、快速重传、快速恢复算法。

46、请简述 TCP 和 UDP 的区别

答：TCP：面向连接的可靠交付，以字节流传输，效率低，耗费资源多，适用于对通信数据严格的场景，如文件传输。

UDP：无连接的尽最大努力交付，以数据报文传输，效率高，耗费资源少，适用于对通信速度要求高的场景，如在线视频。

47、说说 TCP 是如何确保可靠性的呢？

答：（1）TCP连接和断开具有可靠性，由三次握手，四次挥手机制实现。

（2）TCP是有状态的，它会记录哪些数据发送了，哪些数据被接收了，哪些没有被接受，并且保证数据包按序到达，保证数据传输不出差错。

（3）TCP是可控制的，具体有超时重传、流量控制、拥塞控制等机制。

48、说说 TCP 报文首部有哪些字段，其作用又分别是什么？

答：16位源端口号，目的端口号；32位序号，确认号；4位头部长度，6位标志位（URG,ACK,PSH,RST,SYN,FIN），16 位窗口大小；16 位校验和，16 位紧急指针。

49、说说 TCP 的重传机制

答：重传包括：

超时重传（没有收到确认则重传）、

快速重传（连续收到三个冗余ACK代表有包丢失）、

带选择确认的重传（SACK）（告诉发送方自己丢失包之后接收到的数据包）、

重复 SACK（告诉发送方自己已接收哪些数据包）

四种。

50、UDP首部

答：UDP的首部只有8个字节，源端口号、目的端口号、长度和校验和各两个字节。

java基础

1、JDK 和 JRE 有什么区别？

答：JDK是Java开发包，包括开发环境和运行环境；JRE是Java运行环境。

2、== 和 equals 的区别是什么？

答：== 对于基本类型来说是值比较，对于引用类型来说是比较的是引用（或者说地址）；equals 因为被很多类重写成了值比较（本质就是 == ），所以一般情况下 equals 比较的是值是否相等。

3、两个对象的 hashCode() 相同，则 equals() 也一定为 true，对吗？

答：不对，两个对象的 hashCode() 相同，equals() 不一定 true。比如出现两个对象的哈希冲突。

4、final 在 Java 中有什么作用？

答：final 修饰的类叫最终类，该类不能被继承。final 修饰的方法不能被重写。final 修饰的变量叫常量，常量必须初始化，初始化之后值就不能被修改。

5、Java 中的 Math. round(-1. 5) 等于多少？

答：等于 -1，因为在数轴上取值时，中间值（0.5）向右取整，所以正 0.5 是往上取整，负 0.5 是直接舍弃。

6、String 属于基础的数据类型吗？

答：String 不属于基础类型，属于对象。

7、Java 中操作字符串都有哪些类？它们之间有什么区别？

答：String、StringBuffer、StringBuilder。String 和 StringBuffer、StringBuilder 的区别在于 String 声明的是不可变的对象，每次操作都会生成新的 String 对象，然后将指针指向新的 String 对象，而 StringBuffer、StringBuilder 可以在原有对象的基础上进行操作，所以在经常改变字符串内容的情况下最好不要使用 String。StringBuffer 和 StringBuilder 最大的区别在于，StringBuffer 是线程安全的，而 StringBuilder 是非线程安全的，但 StringBuilder 的性能却高于 StringBuffer，所以在单线程环境下推荐使用 StringBuilder，多线程环境下推荐使用 StringBuffer。

8、String str="i"与 String str=new String("i")一样吗？

答：不一样，一个分配到常量池中，一个分到堆内存中。

9、如何将字符串反转？

答：使用 StringBuilder 或者 stringBuffer 的 reverse() 方法。

10、抽象类必须要有抽象方法吗？

答：不需要，抽象类不一定非要有抽象方法，也可以正常运行。

11、普通类和抽象类有哪些区别？

答：普通类不能包含抽象方法，抽象类可以包含抽象方法。抽象类不能直接实例化，普通类可以直接实例化。

12、抽象类能使用 final 修饰吗？

答：不能，定义抽象类就是让其他类继承然后拓展的。

13、接口和抽象类有什么区别？

答：实现：抽象类的子类使用 extends 来继承而接口必须使用 implements 来实现接口。构造函数：抽象类可以有构造函数而接口不能有。实现数量：类可以实现很多个接口，但是只能继承一个抽象类。访问修饰符：接口中的方法默认使用 public 修饰；抽象类中的方法可以是任意访问修饰符。

14、Java 中 IO 流分为几种？

答：按功能来分：输入流、输出流。按类型来分：字节流和字符流。字节流和字符流的区别是：字节流按 8 位传输以字节为单位输入输出数据，字符流按 16 位传输以字符为单位输入输出数据。

15、BIO、NIO、AIO 有什么区别？

答：BIO：Block IO 同步阻塞式 IO，平常使用的传统 IO，模式简单使用方便，并发处理能力低。NIO：Non IO 同步非阻塞 IO，是传统 IO 的升级，实现了多路复用。AIO：Asynchronous IO 是 NIO 的升级，也叫 NIO2，实现了异步非堵塞 IO ，异步 IO 的操作基于事件和回调机制。

16、Java 容器都有哪些？

答：Java 容器分为 Collection 和 Map 两大类，其下又有很多子类。

17、Collection 和 Collections 有什么区别？

答：Collection 是一个集合接口，它提供了对集合对象进行基本操作的通用接口方法，所有集合都是它的子类，比如 List、Set 等。Collections 是一个包装类，包含了很多静态方法，不能被实例化，就像一个工具类，比如提供的排序方法：Collections. sort(list)。

18、List、Set、Map 之间的区别是什么？

答：List、Set、Map 的区别主要体现在两个方面：元素是否有序、是否允许元素重复。

19、HashMap 和 Hashtable 有什么区别？

答：存储：HashMap 允许 key 和 value 为 null，而 Hashtable 不允许。线程安全：Hashtable 是线程安全的，而 HashMap 是非线程安全的。

20、如何决定使用 HashMap 还是 TreeMap？

答：对于在 Map 中插入、删除、定位一个元素这类操作，HashMap 是最好的选择，因为相对而言 HashMap 的插入会更快，但如果你要对一个 key 集合进行有序的遍历，那 TreeMap 是更好的选择。

21、说一下 HashMap 的实现原理？

答：HashMap 基于 Hash 算法实现的，当传入 key 时，HashMap 会根据 key. hashCode() 计算出 hash 值，根据 hash 值将 value 保存在 bucket 里。当计算出的 hash 值相同时，我们称之为 hash 冲突，HashMap 的做法是用链表和红黑树存储相同 hash 值的 value。当 hash 冲突的个数比较少时，使用链表否则使用红黑树。

22、说一下 HashSet 的实现原理？

答：HashSet 是基于 HashMap 实现的，HashSet 底层使用 HashMap 来保存所有元素，因此 HashSet 的实现比较简单，相关 HashSet 的操作，基本上都是直接调用底层 HashMap 的相关方法来完成，HashSet 不允许重复的值。

23、ArrayList 和 LinkedList 的区别是什么？

答：ArrayList 是动态数组的数据结构实现，而 LinkedList 是双向链表的数据结构实现。ArrayList 比 LinkedList 在随机访问的时候效率要高，在非首尾的增加和删除操作时LinkedList 要比 ArrayList 效率要高。

24、如何实现数组和 List 之间的转换？

答：数组转 List：使用 Arrays. asList(array) 进行转换。List 转数组：使用 List 自带的 toArray() 方法。

25、ArrayList 和 Vector 的区别是什么？

答：线程安全：Vector（矢量队列）是线程安全的，而 ArrayList 是非线程安全的。性能：ArrayList 在性能方面要优于 Vector。扩容：ArrayList 和 Vector 都会根据实际的需要动态的调整容量，只不过在 Vector 扩容每次会增加 1 倍，而 ArrayList 只会增加 50%。

26、Array 和 ArrayList 有何区别？

答：Array 可以存储基本数据类型和对象，ArrayList 只能存储对象。Array 是指定固定大小的，而 ArrayList 大小是自动扩展的。Array 内置方法没有 ArrayList 多。

27、在 Queue 中 poll()和 remove()有什么区别？

答：相同点：都是返回第一个元素，并在队列中删除返回的对象。不同点：如果没有元素 poll()会返回 null，而 remove()会直接抛出异常。

28、哪些集合类是线程安全的？

答：Vector、Hashtable、Stack 都是线程安全的，而像 HashMap 则是非线程安全的，。

29、迭代器 Iterator 是什么？

答：Iterator 接口提供遍历任何 Collection 的接口。我们可以从一个 Collection 中使用迭代器方法来获取迭代器实例。

30、Iterator 有什么特点？

答：Iterator 的特点是更加安全，因为它可以确保，在当前遍历的集合元素被更改的时候，就会抛出异常。

31、Iterator 和 ListIterator 有什么区别？

答：Iterator 可以遍历 Set 和 List 集合，而 ListIterator 只能遍历 List。Iterator 只能单向遍历，而 ListIterator 可以双向遍历（向前/后遍历）。ListIterator 从 Iterator 接口继承，然后添加了一些额外的功能，比如添加一个元素、替换一个元素、获取前面或后面元素的索引位置。

32、怎么确保一个集合不能被修改？

答：可以使用 Collections. unmodifiableCollection(Collection c) 方法来创建一个只读集合，这样改变集合的任何操作都会抛出异常。

33、Java创建线程有哪几种方式？

答：有三种方式：继承 Thread 重写 run 方法；实现 Runnable 接口；实现 Callable 接口。

34、说一下 runnable 和 callable 有什么区别？

答：runnable 没有返回值，callable 可以拿到有返回值，callable 可以看作是 runnable 的补充。

35、线程有哪些状态？

答：NEW 尚未启动，RUNNABLE 正在执行中，BLOCKED 阻塞的（被同步锁或者IO锁阻塞），WAITING 永久等待状态，TIMED\_WAITING 等待指定的时间重新被唤醒的状态，TERMINATED 执行完成。

36、sleep() 和 wait() 有什么区别？

答：类的不同：sleep() 来自 Thread，wait() 来自 Object。释放锁：sleep() 不释放锁；wait() 释放锁。用法不同：sleep() 时间到会自动恢复；wait() 可以使用 notify()/notifyAll()直接唤醒。

37、notify()和 notifyAll()有什么区别？

答：notifyAll()会唤醒所有的线程，notify()之后唤醒一个线程。notifyAll() 调用后，会将全部线程由等待池移到锁池，然后参与锁的竞争，竞争成功则继续执行，如果不成功则留在锁池等待锁被释放后再次参与竞争。而 notify()只会唤醒一个线程，具体唤醒哪一个线程由虚拟机控制。

38、线程的 run() 和 start() 有什么区别？

答：start() 方法用于启动线程，run() 方法用于执行线程的运行时代码。run() 可以重复调用，而 start() 只能调用一次。

39、创建线程池有哪几种方式？

答：有七种。名字记不住。

40、线程池都有哪些状态？

答：RUNNING：这是最正常的状态，接受新的任务，处理等待队列中的任务。SHUTDOWN：不接受新的任务提交，但是会继续处理等待队列中的任务。STOP：不接受新的任务提交，不再处理等待队列中的任务，中断正在执行任务的线程。TIDYING：所有的任务都销毁了，workCount 为 0，线程池的状态在转换为 TIDYING 状态时，会执行钩子方法 terminated()。TERMINATED：terminated()方法结束后，线程池的状态就会变成这个。

41、线程池中 submit() 和 execute() 方法有什么区别？

答：execute()：只能执行 Runnable 类型的任务。submit()：可以执行 Runnable 和 Callable 类型的任务。Callable 类型的任务可以获取执行的返回值，而 Runnable 执行无返回值。

42、在 Java 程序中怎么保证多线程的运行安全？

答：方法一：使用安全类，比如 Java. util. concurrent 下的类。方法二：使用自动锁 synchronized。方法三：使用手动锁 Lock。

43、ThreadLocal 是什么？有哪些使用场景？

答：ThreadLocal 为每个使用该变量的线程提供独立的变量副本，所以每一个线程都可以独立地改变自己的副本，而不会影响其它线程所对应的副本。ThreadLocal 的经典使用场景是数据库连接和 session 管理等。

44、什么是反射？

答：反射是在运行状态中，对于任意一个类，都能够知道这个类的所有属性和方法；对于任意一个对象，都能够调用它的任意一个方法和属性；这种动态获取的信息以及动态调用对象的方法的功能称为 Java 语言的反射机制。

45、什么是 Java 序列化？什么情况下需要序列化？

答：Java 序列化是为了保存各种对象在内存中的状态，并且可以把保存的对象状态再读出来。以下情况需要使用 Java 序列化：想把的内存中的对象状态保存到一个文件中或者数据库中时候；想用套接字在网络上传送对象的时候；想通过RMI（远程方法调用）传输对象的时候。

46、动态代理是什么？有哪些应用？

答：动态代理是运行时动态生成代理类。动态代理的应用有 spring aop、hibernate 数据查询、测试框架的后端 mock、rpc，Java注解对象获取等。

47、怎么实现动态代理？

答：JDK 原生动态代理和 cglib 动态代理。JDK 原生动态代理是基于接口实现的，而 cglib 是基于继承当前类的子类实现的。

48、为什么要使用克隆？

答：克隆的对象可能包含一些已经修改过的属性，而 new 出来的对象的属性都还是初始化时候的值，所以当需要一个新的对象来保存当前对象的“状态”就靠克隆方法了。

49、如何实现对象克隆？

答：实现 Cloneable 接口并重写 Object 类中的 clone() 方法。实现 Serializable 接口，通过对象的序列化和反序列化实现克隆，可以实现真正的深度克隆。

50、深拷贝和浅拷贝区别是什么？

答：浅克隆：当对象被复制时只复制它本身和其中包含的值类型的成员变量，而引用类型的成员对象并没有复制。深克隆：除了对象本身被复制外，对象所包含的所有成员变量也将复制。

51、JSP 和 servlet 有什么区别？

答：

JSP 是 servlet 技术的扩展，本质上就是 servlet 的简易方式。servlet 和 JSP 最主要的不同点在于，servlet 的应用逻辑是在 Java 文件中，并且完全从表示层中的 html 里分离开来，而 JSP 的情况是 Java 和 html 可以组合成一个扩展名为 JSP 的文件。JSP 侧重于视图，servlet 主要用于控制逻辑。

52、throw 和 throws 的区别？

答：throw：是真实抛出一个异常。throws：是声明可能会抛出一个异常。

53、final、finally、finalize 有什么区别？

答：final：是修饰符，如果修饰类，此类不能被继承；如果修饰方法和变量，则表示此方法和此变量不能在被改变，只能使用。

finally：是 try{} catch{} finally{} 最后一部分，表示不论发生任何情况都会执行，finally 部分可以省略，但如果 finally 部分存在，则一定会执行 finally 里面的代码。

finalize：是 Object 类的一个方法，在垃圾收集器执行的时候会调用被回收对象的此方法。

54、try-catch-finally 中哪个部分可以省略？

答：try-catch-finally 其中 catch 和 finally 都可以被省略，但是不能同时省略，也就是说有 try 的时候，必须后面跟一个 catch 或者 finally。

55、try-catch-finally 中，如果 catch 中 return 了，finally 还会执行吗？

答：finally 一定会执行，即使是 catch 中 return 了，catch 中的 return 会等 finally 中的代码执行完之后，才会执行。

Go基础

1、golang 中 make 和 new 的区别？

答：作用对象不同，new给int,string,数组分配内存；返回类型不同，new返回变量指针而make返回变量本身；new分配空间被清零而make分配空间后会进行初始化。

2、数组和切片的区别？

答：定义方式不一样，一个值类型一个引用类型；初始化方式不一样，数组是定长，需要指定大小且不能改变；在函数传递中，数组切片都是值传递，只不过一个复制数组一个复制携带底层数组地址指针的结构体。

3、for range 的时候它的地址会发生变化么？

答：在 for a,b := range c 遍历中， a 和 b 在内存中只会存在一份，即之后每次循环时遍历到的数据都是以值覆盖的方式赋给 a 和 b，a，b 的内存地址始终不变。

4、go defer，多个 defer 的顺序，defer 在什么时机会修改返回值？

答：首先return，其次return value，最后defer。多个 defer 调用顺序是defer栈。defer可以修改函数最终返回值，修改时机：有名返回值或者函数返回指针。

5、uint 类型溢出问题？

答：超过最大存储值如uint8最大是255，如果再加1可能等于0，也可能出现其他难以预估的结果。

6、能介绍下 rune 类型吗？

答：rune相当于int32，golang字符串底层是byte数组相当于int8，用来处理ASCII字符，而中文字符占3个字节，适合用rune来处理。

7、golang 中解析 tag 是怎么实现的？反射原理是什么？

答：tag 是结构体支持标记，go 中解析的 tag 是通过反射实现的，反射是指计算机程序在运行时可以访问、检测和修改它本身状态或行为的一种能力或动态知道给定数据对象的类型和结构，并有机会修改它。反射将接口变量转换成反射对象 Type 和 Value；反射可以通过反射对象 Value 还原成原先的接口变量；反射可以用来修改一个变量的值，前提是这个值可以被修改。

8、调用函数传入结构体时，应该传值还是指针？

答：Go 的函数参数传递都是值传递。值传递是在调用函数时将实际参数复制一份传递到函数中，这样在函数中如果对参数进行修改，将不会影响到实际参数。传指针是引用传递，是指在调用函数时将实际参数的地址传递到函数中，那么在函数中对参数所进行的修改，将影响到实际参数。

9、讲讲 Go 的 slice 底层数据结构和一些特性？

答：Go 的 slice 底层数据结构是由一个 array 指针指向底层数组，len 表示切片长度，cap 表示切片容量。slice 的主要特性是自动扩容，如何扩容bababa。。。

10、讲讲 Go 的 select 底层数据结构和一些特性？

答：go 的 select 结构组成主要是由 case 语句和执行的函数组成，为 golang 提供了多路 IO 复用机制，和其他的 IO 复用一样，无差别轮询用于检测是否有读写事件 ready。特性有：select 操作至少要有一个 case 语句。select 仅支持管道，而且是单协程操作。每个 case 语句仅能处理一个管道，要么读要么写。多个 case 语句的执行顺序是随机的。存在 default 语句，select 将不会阻塞，但是存在 default 会影响性能。

11、讲讲 Go 的 defer 底层数据结构和一些特性？

答：每个 defer 语句都对应一个\_defer 实例，多个实例使用指针连接起来形成一个单链表，每次插入\_defer 实例，均插入到链表的头部，取也是从头部取出，类似栈形成后进先出的效果。特性：延迟函数的参数是 defer 语句出现的时候就已经确定了的。多个defer语句遵循先进后出。延迟函数可能操作主函数的返回值。申请资源后立即使用 defer 关闭资源是个好习惯。

12、单引号，双引号，反引号的区别？

答：单引号，表示byte类型或rune类型，对应 uint8和int32类型，默认是 rune 类型。双引号，才是字符串，实际上是字符数组。反引号，表示字符串字面量，但不支持任何转义序列。

13、map 使用注意的点，是否并发安全？

答：map的类型是map[key]，其中key必须是可比较的，map[key]函数返回结果可以是一个值，也可以是两个值。map是无序的，一定要先初始化，否则panic。map类型是容易发生并发访问问题的。

14、map 循环是有序的还是无序的？

答：无序的。map 因扩张?重新哈希时，各键值项存储位置都可能会发生改变，顺序自然也没法保证了。

15、 map 中删除一个 key，它的内存会释放么？

答：如果删除的元素是值类型，map的内存不会自动释放。如果删除的元素是引用类型，如指针，slice，map，chan等，map的内存会自动释放，但释放的内存是子元素应用类型的内存占用。将map设置为nil后，内存被回收。

16、怎么处理对 map 进行并发访问？有没有其他方案？ 区别是什么？

答：使用内置sync.Map。使用读写锁实现并发安全map。

17、nil map 和空 map 有何不同？

答：nil map 是未初始化，空map是长度为空。可以对未初始化的map进行取值，但取出来的东西是空：不能对未初始化的map进行赋值，这样将会抛出一个异常：通过fmt打印map时，空map和nil map结果是一样的，都为map[]。所以，这个时候别断定map是空还是nil，而应该通过map == nil来判断。

18、map 的数据结构是什么？是怎么实现扩容？

答：golang 中 map 是一个 kv 对集合。底层使用 hash table，用链表来解决冲突。发生哈希冲突时哈希表以 bmap 为最小粒度挂载，一个 bmap 可以放 8 个 kv。也就是说map的底层哈希表是由若干个 挂载了bmap的 bucket（即链表里的第一个bmap） 组成的数组（所以map底层哈希表也叫bmap数组），每个 bucket 都采用链表结构来挂载bmap，而一个 bmap 可以放 8 个 kv。map容量与装载因子有关，装载因子=哈希表元素个数/桶的数量即bmap数组的长度。装载因子越大，意味着bmap数组越短或者哈希表元素越多，哈希表空闲位置越少，越容易发生冲突，哈希表性能越低。所以要尽量降低装载因子，源码里设置阈值为6.5，当存到map的元素过多使得装载因子接近阈值，map就会进行扩容。还有另一种情况，溢出桶数量过多也会扩容。扩容时会新建一个bmap数组，逐步将旧数组的数据迁移至新数组，然后去除旧数组的引用，让GC来清除内存。总之，map扩容就是加大bmap数组也就是哈希表的长度，降低冲突几率。

19、slices能作为map类型的key吗？

答：不能，map类型的key必须是可比较的。切片、maps、functions无法比较。

20、context 结构是什么样的？context 使用场景和用途？

答：Go 的 Context 的数据结构包含 Deadline，Done，Err，Value。Deadline 方法返回一个Time，表示当前 Context 应该结束的时间，Done 方法当 Context 被取消或者超时时候返回的一个 close 的 channel，告诉给 context 相关的函数要停止当前工作然后返回了，Err 表示 context 被取消的原因，Value 方法表示 context 实现共享数据存储的地方，是协程安全的。主要使用场景：1：上下文控制，2：多个 goroutine 之间的数据交互等，3：超时控制：到某个时间点超时，过多久超时。

21、channel 是否线程安全？锁用在什么地方？

答：是线程安全，锁用在可能产生并发冲突的地方。设计理念：不要通过共享内存的方式进行通信，而应该通过通信的方式共享内存。

22、go channel 的底层实现原理 （数据结构）

答：管道的源码中可以看到：其结构体封装有很多字段，关键的有，buf 指向底层循环数组，只有缓冲型的 channel 才有。sendx，recvx（发送、接收指针） 均指向底层循环数组，表示当前可以发送和接收的元素位置索引值。sendq，recvq （发送、接收队列）分别表示被阻塞的 goroutine 队列，这些 goroutine 由于尝试读取 channel 或向 channel 发送数据而被阻塞。lock 用来保证每个读 channel 或写 channel 的操作都是原子的。

23、nil、关闭的 channel、有数据的 channel，再进行读、写、关闭会怎么样？

答：nil即未初始化的管道，读写阻塞，关闭panic。已关闭的管道，可读，写和关闭panic。正常管道，正常关闭，读写正常或阻塞。

24、向 channel 发送数据和从 channel 读数据的流程是什么样的？

答：向 channel 写数据的流程：如果管道等待接收队列不为空，说明有协程在等着读取管道的数据，也就是缓冲区中没有数据或者没有缓冲区，此时直接从等待接收队列取出 G,并把数据写入，最后把该 G 唤醒，结束发送过程； 如果缓冲区中有空余位置，将数据写入缓冲区，结束发送过程； 如果缓冲区中没有空余位置，将待发送数据写入 G，将当前 G 加入管道的等待发送队列，并进入睡眠，等待被读 goroutine 唤醒。向 channel 读数据的流程：如果等待发送队列不为空，且没有缓冲区，直接从等待发送队列中取出 G，把 G 中数据读出，最后把 G 唤醒，结束读取过程； 如果等待发送队列不为空，此时说明缓冲区已满，从缓冲区中首部读出数据，把 G 中数据写入缓冲区尾部，把 G 唤醒，结束读取过程； 如果缓冲区中有数据，则从缓冲区取出数据，结束读取过程；如果没数据可读，则将当前 goroutine 加入等待接收队列，进入睡眠，等待被写 goroutine 唤醒。

25、讲讲 Go 的 chan 底层数据结构和主要使用场景

答：管道的源码中可以看到：其结构体封装有很多字段，关键的有，buf 指向底层循环数组，只有缓冲型的 channel 才有。sendx，recvx（发送、接收指针） 均指向底层循环数组，表示当前可以发送和接收的元素位置索引值。sendq，recvq （发送、接收队列）分别表示被阻塞的 goroutine 队列，这些 goroutine 由于尝试读取 channel 或向 channel 发送数据而被阻塞。lock 用来保证每个读 channel 或写 channel 的操作都是原子的。主要使用场景：消息传递、消息过滤，信号广播，事件订阅与广播，请求、响应转发，任务分发，结果汇总，并发控制，限流，同步与异步。

26、什么是 GMP？

答：

27、进程、线程、协程有什么区别？

答：

28、抢占式调度是如何抢占的？

答：

29、M 和 P 的数量问题？

答：

30、除了 mutex 以外还有那些方式安全读写共享变量？

答：

31、Go 如何实现原子操作？

答：

32、Mutex 是悲观锁还是乐观锁？悲观锁、乐观锁是什么？

答：

33、Mutex 有几种模式？

答：

34、goroutine 的自旋占用资源如何解决

答：

35、

Redis：

**概述**

\*1. Redis是什么？简述它的优缺点？

答：Redis是Key-Value类型的内存数据库，整个数据库加载在内存当中操作，定期通过异步操作把数据flush到硬盘上进行保存。

优点：1.读写性能极高2.支持数据持久化3.支持事务4.数据结构丰富5.支持主从复制。

缺点：1.数据库容量受物理内存的限制，2.主从复制时可能出现主机故障而从机未能及时复制数据。

\*2. Redis为什么这么快？

答：1.使用内存存储，没有磁盘IO上的开销；

2.使用单个线程处理请求，避免线程切换和锁资源争用；

3.使用多路复用IO技术，将epoll作为I/O多路复用技术的实现；

4.提供可以直接使用的优化的数据结构；

5.使用不同的底层模型：Redis直接自己构建了虚拟内存机制 。

3. Redis相比Memcached有哪些优势？

答：1.数据类型：Memcached所有的值均是简单的字符串，Redis支持更为丰富的数据类型；

2.持久化：Redis支持数据落地持久化存储；

3.集群模式：Redis提供主从同步机制，以及集群部署能力，Memcached没有原生的集群模式；

4.性能对比：Redis的速度比Memcached快很多；

5.网络IO模型：Redis使用单线程的多路 IO 复用模型，Memcached使用多线程的非阻塞IO模式；

6.Redis支持服务器端的数据操作，而Memcached通常在客户端处理数据。

\*4. 为什么要用 Redis 做缓存？

答：从高并发上来说，直接操作缓存能够承受的请求是远远大于直接访问数据库的；

从高性能上来说，操作缓存就是直接操作内存，所以存取数据速度相当快。

5. 为什么要用 Redis 而不用 map/guava 做缓存?

答：缓存分为本地缓存和分布式缓存。以java为例，使用自带的map或者guava实现的是本地缓存，最主要的特点是轻量以及快速，生命周期随着jvm的销毁而结束，并且在多实例的情况下，每个实例都需要各自保存一份缓存，缓存不具有一致性。使用Redis或memcached之类的称为分布式缓存，在多实例的情况下，各实例共用一份缓存数据，缓存具有一致性。

\*6. Redis的常用场景有哪些?

答：缓存：不仅能够提升网站访问速度，还能大大降低数据库的压力。

排行榜：Redis提供的有序集合数据类构能实现排行应用。

计数器：Redis提供的incr命令来实现计数器功能。

分布式会话：搭建以Redis等内存数据库为中心的session服务。

分布式锁：可以利用Redis的setnx功能来编写分布式的锁。

社交网络：Redis的set可以很方便得出微博的共同好友。

消息系统：Redis提供了发布/订阅及阻塞队列功能，能实现一个简单的消息队列系统。

\*7. Redis的数据类型有哪些？

答：五种常见数据类型：

String，最常见；

Hash，键值对；

Set，无序不可重复；

List，有序可重复；

SortedSet，有序的set；

三种特殊数据类型：

Bitmap：位图，一个以位为单位的数组，数组中的每个单元只能存0或者1，适合统计数据的二值状态；

HyperLogLog，是一种用于统计基数的数据集合类型；

Geospatial ：主要用于存储地理位置信息，并对存储的信息进行操作，适用场景如朋友的定位、附近的人、打车距离计算等。

PS：

7.1 zset底层数据结构? 简单说说跳表底层的数据结构?

答：zset底层数据结构是 **压缩列表** 和 **哈希表+跳表**，

**压缩列表**=列表长度+尾部偏移量+列表元素个数+普通数组+列表结束标识

**跳表**=普通链表+多级索引；通过多级索引的跳转实现快速查找元素

7.2 什么时候采用压缩列表、什么时候采用跳表呢?

答：同时满足两个条件时使用压缩列表，其他使用跳表：

1.元素数量小于128个。2.所有元素的长度都小于64字节。

7.3 跳表的时间复杂度

答：因为采用二分法思想，渐进时间复杂度为O（logN）。

7.4 简单描述一下跳表如何查找某个元素呢?

答：从第最上层开始，找到每一层中最后一个小于目标元素值的元素，然后进入下一层（若已找到目标元素则直接进入下一层），直到在最后一层找到目标元素。

7.5 简单描述一下跳表如何插入某个元素呢?

答：从最上层开始找，找到最底层最后一个小于插入值的元素，在其后面插入目标元素，然后执行随机函数（随机函数取值为0则不加，为1则加）看看是否在最底层的上一层也插入目标元素，一直重复直到不再往上插入。

7.6 简单描述一下跳表如何删除某个元素呢?

答：从最上层开始找，找到最底层中的目标元素，然后将其整个tower（即往上的层级的相同元素）删除。

7.7 zset为什么用跳表而不用二叉树或者红黑树呢?

答：1.用跳表实现**范围查找效率更高**，只需要找到区间起点依次往后遍历；

2.用跳表实现**简单易懂容易维护**，不用考虑树的旋转等操作。

**持久化**

\*8. Redis持久化机制？

答：持久化是指将Redis中的数据写入到磁盘空间中，Redis提供了两种不同的持久化方法，一种叫快照RDB，另一种叫只追加文件AOF。

RDB：在指定的时间间隔内将内存中的数据集快照写入磁盘(Snapshot)，它恢复时是将快照文件直接读到内存里,性能高但安全性低。

AOF：以独立日志的方式记录每次写命令，并在 Redis 重启时在重新执行 AOF 文件中的命令以达到恢复数据的目的。AOF 的主要作用是解决数据持久化的实时性。性能低但保证数据安全。具体有三种机制：每修改同步，每秒同步，不同步。

9. 如何选择合适的持久化方式

答：1.如果是数据不那么敏感，且可以从其他地方重新生成补回的，那么可以关闭持久化。

2.如果是数据比较重要，不想再从其他地方获取，且可以承受数分钟的数据丢失，比如缓存等，那么可以只使用RDB。

3.如果是用做内存数据库，要使用Redis的持久化，建议是RDB和AOF都开启，RDB方式更适合做数据的备份，AOF可以保证数据的不丢失。

10. Redis持久化数据和缓存怎么做扩容？

答：如果Redis被当做缓存使用，使用一致性哈希实现动态扩容缩容。如果Redis被当做一个持久化存储使用，必须使用固定的keys-to-nodes映射关系，节点的数量一旦确定不能变化。否则的话(即Redis节点需要动态变化的情况），必须使用可以在运行时进行数据再平衡的一套系统，而当前只有Redis集群可以做到这样。

\*11. Redis过期键的删除策略

答：Redis的过期删除策略就是：惰性删除和定期删除两种策略配合使用。

**惰性删除**：只有在访问数据的时候，才去检查当前键值是否过期。优点是简单，缺点是删除过期键不及时，造成了一定的空间浪费。

**定期删除**：周期性的抽样部分key判断是否过期，过期则执行删除。关键是检测周期时间的设置问题。检测周期太短对CPU不友好，检测周期太长对内存不友好。

**定时删除**：在设置某个key 的过期时间同时，我们创建一个定时器，让定时器在该过期时间到来时，立即执行对其进行删除的操作。对内存最友好但对CPU最不友好。

12. Redis key的过期时间和永久有效分别怎么设置？

答：通过expire或pexpire命令，客户端可以以秒或毫秒的精度为数据库中的某个键设置生存时间。

**缓存异常**

13.Redis内存淘汰策略和缓存更新策略？

答：内存淘汰：当Redis的内存达到设置的阈值后，Redis会主动挑选部分key删除以释放更多内存。

8种淘汰策略：

**Noeviction(不淘汰)**:不淘汰任何key,但是内存满时不允许写入新数据，默认就是这种策略。

**volatile-ttl(根据TTL)**:对设置了TTL的key,比较key的剩余TTL值，TTL越小越先被淘汰。

**随机淘汰：**

**allkeys-random**:对全体key,随机进行淘汰。

**volatile-random**:对设置了TTL的key,随机进行淘汰。

**LRU算法（最少最近使用）：**

**allkeys-lru**:对全体key,基于LRU算法进行淘汰。

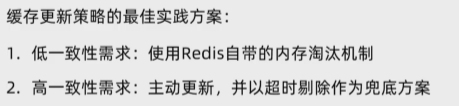
**volatile-lru**:对设置了TTL的key,基于LRU算法进行淘汰。

**LFU算法（最少频率使用）**

**allkeys-lfu**:对全体key,基于LFU算法进行淘汰。

**volatile-lfu**:对设置了TTL的key,基于LFI算法进行淘汰。

缓存更新策略：



\*14. 如何保证缓存与数据库双写时的数据一致性？（重要）

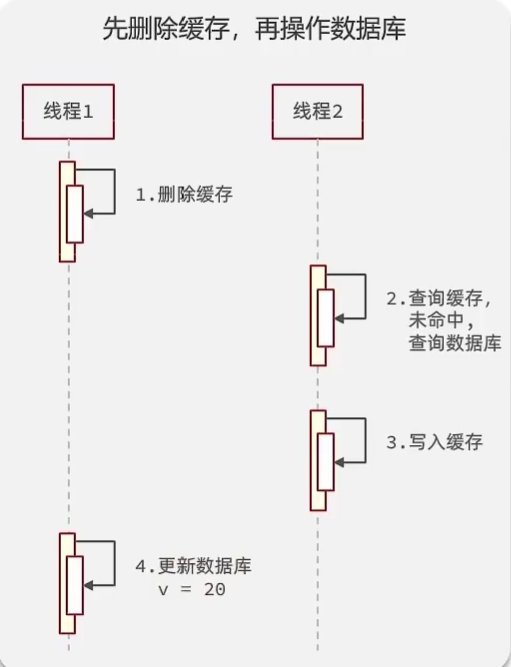
答：更新缓存：无效写操作较多；

删除缓存：更新数据库时删缓存，查询数据库时再更新缓存（采用删除缓存更优）

共有2种方案：1.先删除缓存，后更新数据库；2.先更新数据库，后删除缓存（推荐这种，错误概率更小）。

15. 先删除缓存，后更新数据库方案（错误概率大）

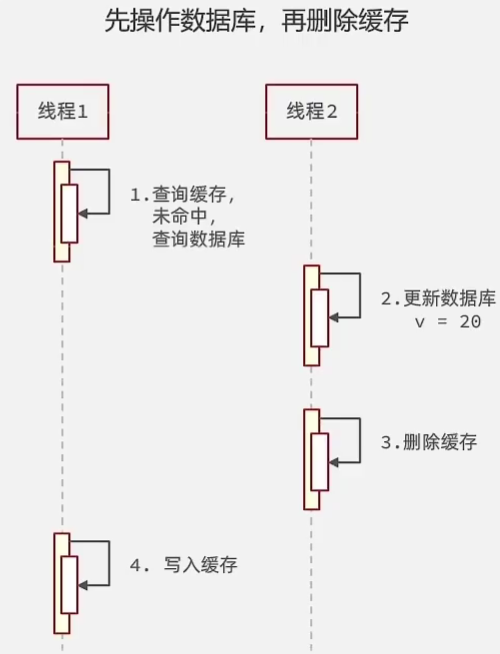
答：问题：容易在读写并发操作时导致数据不一致的情形出现。



方案一：延时双删。（1）先淘汰缓存（2）再写数据库（3）休眠1秒，再次淘汰缓存。这么做，可以将1秒内所造成的缓存脏数据，再次删除。

方案二：更新与读取操作进行异步串行化。

16. 先更新数据库，后删除缓存（推荐，错误概率小）



答：问题1：线程1查询完数据库后准备写入缓存时，线程2更新了数据库并且删除缓存，这样线程1写入的缓存就是脏数据。

问题2：比如更新数据库成功了，但是在删除缓存时出错，那么此时再读取缓存时会读取到原先的脏数据。

解决方案：利用消息队列进行删除的补偿。系统接收到消息队列发送的消息后再次对 Redis 进行删除操作。

\*17. 什么是缓存击穿?

答：缓存击穿跟缓存雪崩有点类似，缓存雪崩是大规模的key失效，而缓存击穿是某个热点的key失效，大并发集中对其进行请求，就会造成大量请求读缓存没读到数据，从而导致高并发访问数据库，引起数据库压力剧增。这种现象就叫做缓存击穿。

解决方案：

1. 在缓存失效后，通过互斥锁来控制读数据写缓存的线程数量。**优点**：没有额外内存消耗，，保证一致性，实现简单。**缺点**：线程需要等待，影响性能，有死锁风险。

2. 热点数据缓存物理永不过期，逻辑过期：

（1）物理不过期，针对热点key不设置过期时间；

（2）逻辑过期，把过期时间存在key对应的value里，如果查询结果发现过期了，则返回过期的旧数据，并开启一个后台的异步线程对缓存进行重新构建，重新设置逻辑过期时间。

**优点**：线程无需等待，性能好。**缺点**：消耗内存，不保证数据一致性，实现复杂。

\*18. 什么是缓存穿透?

答：缓存穿透是指用户请求的数据在缓存中不存在即没有命中，同时在数据库中也不存在，导致用户每次请求该数据都要去数据库中查询一遍。它和缓存击穿的根本区别在于传进来的key是不存在的。如果有恶意攻击者不断请求系统中不存在的数据，会导致短时间大量请求落在数据库上，造成数据库压力过大，甚至导致数据库承受不住而宕机崩溃。

解决方法：

1.将无效的key存放进Redis中并设置value="null"，这样就直接返回不需要再查询数据库了。**优点**：实现简单，方便维护。**缺点**：消耗内存，可能造成数据库与缓存短期不一致。

2.使用布隆过滤器，查询Redis前先去布隆过滤器查询 key 是否存在，如果不存在就直接返回，不让其访问redis和数据库，从而避免了对底层存储系统的查询压力。**优点**：内存占用少，没有多余key。**缺点**：实现复杂，存在误判可能（布隆过滤器查询不到则一定不存在，布隆过滤器查询到的话大概率存在但也可能不存在。）

总结：空key少、重复率高的则用第一种方法；空key多且重复率低则用第二种方法。

\*19. 什么是缓存雪崩?

答：如果缓存在某一个时刻出现大规模的key失效，那么就会导致大量的请求打在了数据库上面，导致数据库压力巨大，如果在高并发的情况下，可能瞬间就会导致数据库宕机。造成缓存雪崩的关键在于同一时间的大规模的key失效，

主要有两种可能：第一种是可能采用了相同的过期时间，第二种是Redis宕机。

解决方案：

1.均匀过期：设置不同的过期时间，避免相同的过期时间导致缓存雪崩，造成大量数据库的访问；

2.多级缓存：第一级缓存失效的基础上，访问二级缓存，每一级缓存的失效时间都不同；

3.热点数据缓存永远不过期；

4.利用Redis集群提高服务的高可用性，防止Redis宕机导致缓存雪崩的问题。可以使用 主从+ 哨兵，Redis集群来避免 Redis 全盘崩溃的情况；

5.给缓存业务添加降级限流策略：当redis集群出现故障或宕机时拒绝服务，返回error。

\*20. 什么是缓存预热?

答：缓存预热是指系统上线后，提前将相关的缓存数据加载到缓存系统。避免在用户请求的时候，先查询数据库，然后再将数据缓存的问题，用户可直接查询事先被预热的缓存数据。如果不进行预热，那么Redis初始状态数据为空，系统上线初期，对于高并发的流量，都会访问到数据库中， 对数据库造成流量的压力。

\*21. 什么是缓存降级？

答：缓存降级是指缓存失效或缓存服务器挂掉的情况下，不去访问数据库，直接返回默认数据或访问服务的内存数据。降级一般是有损的操作，所以尽量减少降级对于业务的影响程度。在进行降级之前要对系统进行梳理，看看系统是不是可以丢卒保帅；从而梳理出哪些必须誓死保护，哪些可降级。

**线程模型**

\*22. Redis为何选择单线程？

答：1.避免过多的上下文切换开销，程序始终运行在进程中单个线程内，没有多线程切换的场景；

2.避免同步机制的开销，不用考虑数据同步的问题，不用引入同步机制；

3.实现简单，方便维护，底层数据结构的设计都不用考虑线程安全问题。

23. Redis真的是单线程？

答：在Redis 6.0之前，其核心网络模型使用的是单线程。

**Redis6.0引入多线程I/O，只是用来处理网络数据的读写和协议的解析，而执行命令依旧是单线程。**

24. Redis 6.0为何引入多线程？

答：因为Redis是基于内存的操作，性能已经足够优秀，而网络 I/O 瓶颈则越来越明显。要提升 Redis的性能主要方向在于优化网络 I/O 模块，Redis的单线程模式会导致系统消耗很多 CPU 时间在网络 I/O 上从而降低吞吐量，故引入多线程处理网络数据的读写和协议解析，而命令执行依然是单线程的。

25. Redis 6.0 采用多线程后，性能的提升效果如何？

答：Redis 作者曾提到：Redis 6 引入的多线程 IO 特性对性能提升至少是一倍以上。

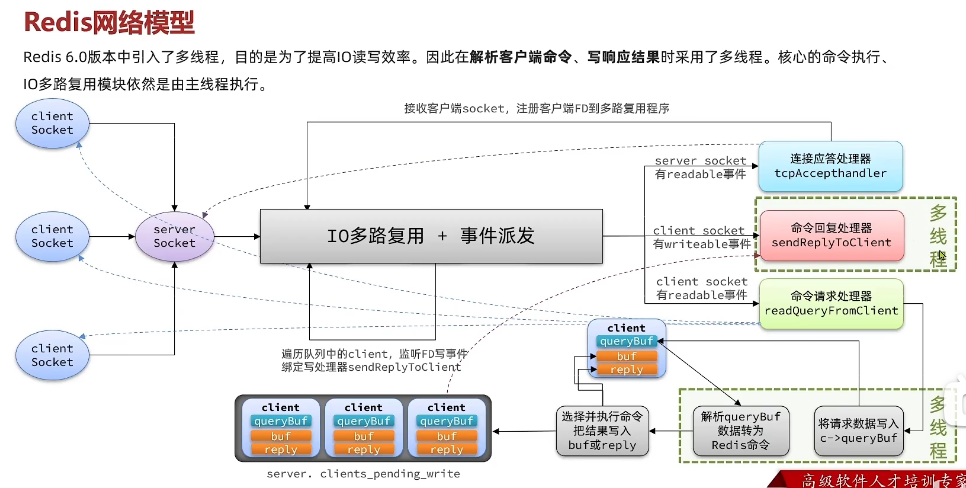
26. 介绍下Redis的线程模型和实现机制？

答：线程模型：redis在IO多路复用模块和命令执行上是单线程，而解析客户端命令和写响应结果使用多线程。

实现机制：

1.redis主线程采用IO多路复用机制同时监听多个socket，将产生事件的socket压入内存队列，

1. 事件分派器根据 socket 上的事件类型来选择对应的事件处理器进行处理。其中主要包括连接应答处理器、命令请求处理器、命令回复处理器。
2. **命令请求处理器**通过**多线程**将请求数据写入缓冲区并将数据解析为redis命令。
3. Redis主线程执行命令并把处理结果写入buf或reply。
4. **命令回复处理器**通过**多线程**将处理结果返回给客户端。



27. Redis 6.0开启多线程后，是否会存在线程并发安全问题？

答：Redis 的多线程部分只是用来处理网络数据的读写和协议解析，执行命令仍然是单线程顺序执行，所以不存在线程并发安全问题。

28. Redis 6.0 与 Memcached 多线程模型的对比

答：**相同点**:都采用了 Master 线程 -Worker 线程的模型。

**不同点**：Memcached 的命令处理在work多线程上，模型更加简单，是传统意义上的多线程，会涉及线程切换、锁竞争、线程安全等问题。

Redis 的命令处理在Master单线程上，虽然一定程度上增加了模型复杂度，但也解决了线程并发安全等问题。

**事务**

29. Redis事务的概念

答：Redis的事务并不是我们传统意义上理解的事务，Redis**事务的执行严格意义上并不是原子性的，**可以理解为一个**打包的批量执行脚本，**中间某条指令的失败不会导致前面已做指令的回滚，也不会造成后续的指令不做。

30. Redis事务的三个阶段

答：1.multi 开启事务

2.大量操作指令进入队列中

3.exec执行事务块内命令，截止此处一个事务已经结束。

其中，discard表示取消事务，watch表示监视一个或多个key，如果事务执行前key被改动，事务将打断。unwatch取消监视。

31. Redis事务相关命令

答：Redis事务功能是通过MULTI、EXEC、DISCARD和WATCH 四个原语实现的

**WATCH**：监控一个或多个键，一旦其中有一个键被修改（或删除），之后的事务就不

会执行，监控一直持续到EXEC命令。

**MULTI**：开启一个事务，之后客户端可以继续向服务器发送任意多条命令，这些命令

不会立即被执行，而是被放到一个队列中，当EXEC命令被调用时，所有队列中的命令才会被执行。

**EXEC**：执行所有事务块内的命令。返回事务块内所有命令的返回值，按命令执行的先

后顺序排列。当操作被打断时，返回空值 nil 。

**DISCARD**：清空事务队列，并放弃执行事务，并且客户端会从事务状态中退出。

**UNWATCH**：取消watch对所有key的监控。

32. Redis事务支持隔离性吗?

答：Redis 是单进程程序，并且它保证在执行事务时，不会对事务进行中断，事务可以运行直到执行完所有事务队列中的命令为止。因此，**Redis 的事务是总是带有隔离性的**。

33. Redis为什么不支持事务回滚？

答：Redis 命令只会因为错误的语法而失败，或是命令用在了错误类型的键上面，失败的命令是由编程错误造成的，而这些错误应该在开发的过程中被发现，而不应该出现在生产环境中。因为不需要对回滚进行支持，所以 Redis 的内部可以保持简单且快速。

34. Redis事务其他实现

答：1.基于Lua脚本，Redis可以保证脚本内的命令一次性按顺序执行，其同时也不提供事务运行错误的回滚，执行过程中如果部分命令运行错误，剩下的命令还是会继续运行完。

2.基于中间标记变量，通过另外的标记变量来标识事务是否执行完成，读取数据时先读取该标记变量判断是否事务执行完成。但这样会需要额外写代码实现，比较繁琐。

**主从、哨兵、集群**

36. Redis常见使用方式有哪些？

答：①Redis单机②Redis主从架构③Redis Sentinel(哨兵)；④Redis Cluster。

Redis单机：适用于数据量少，主要承载高并发高性能的场景，缓存不大的情况。

主从架构：主从实例间数据实时同步，并且提供数据持久化和备份策略，具备高可靠性和读写分离。

哨兵模式：master 节点挂掉后，哨兵进程会主动选举新的 master，可用性高，但是每个节点存储的数据是一样的，浪费内存空间。数据量不是很多，集群规模不是很大，需要自动容错容灾的时候使用。

Redis cluster:针对海量数据+高并发+高可用的场景，如果是海量数据，如果数据量很大，建议使用Redis cluster，所有master的容量总和就是Redis cluster可缓存的数据容量。

37. 介绍下Redis Sentinel（哨兵）

答：优：能够实现故障转移和高可用；缺：每个节点存储内容相同，浪费内存资源。

1. 哨兵模式三大作用：**监控、故障转移、通知**。其部署架构主要包括①Redis Sentinel集群②Redis数据集群。
2. **监控**：由一个或者多个Sentinel实例组成系统可以监视任意多个主服务器，以及这些主服务器属下的所有从服务器，实现过程：

①sentinel节点每隔1秒向redis数据集群的每个实例发送ping命令，

②如果某节点发现某实例在规定时间内未响应则认为该实例**主观下线**，

③若超过指定数量（默认为sentinel实例数量的一半）认为该实例主观下线则判定该 实例**客观下线**。

1. **故障转移**：客户端连接Redis的时候，先连接哨兵，哨兵会告诉客户端Redis主节点的地址，然后客户端连接上Redis并进行后续的操作。当主节点宕机的时候，哨兵监测到主节点宕机，会重新推选出某个表现良好的从节点成为新的主节点，然后通过发布订阅模式通知其他的从服务器，让它们切换主机。实现过程：

①选定一个slave作为新的master，执行slave of no one

②让所有节点执行slave of 新master IP地址

③修改故障节点配置，添加slave of 新master IP地址

（4）**通知**：当集群发生故障转移时，会将最新master节点IP地址通知给redis客户端

38. 故障转移时会从剩下的slave选举一个新的master，被选举为master的标准是什么？

答：1. Slave节点与master断开时间超过指定值则排除该节点；

1. Slave节点的slave-priority值越小优先级越高，0则不参与选举；
2. Slave节点的offset值越大优先级越高；
3. Slave节点的id越小优先级越高。

39. 介绍下Redis Cluster

答：Redis Cluster实现了redis的分布式存储，对数据进行分片，每台redis存储不同内容，起到负载均衡作用，解决单机redis的内存容量和并发问题。

40. 了解主从复制的原理吗？

答：redis主从复制主要分为全量同步和增量同步。从节点首次连接时触发全量同步，非首次连接则出发增量同步。

全量同步流程：

slave节点请求增量同步

master节点判断replid,发现不一致，拒绝增量同步

master将完整内存数据生成RDB,发送RDB到slave

slave清空本地数据，加载master的RDB

master将RDB期间的命令记录在repl\_baklog,并持续将log中的命令发送给slave

slave执行接收到的命令，保持与master同步

增量同步流程：

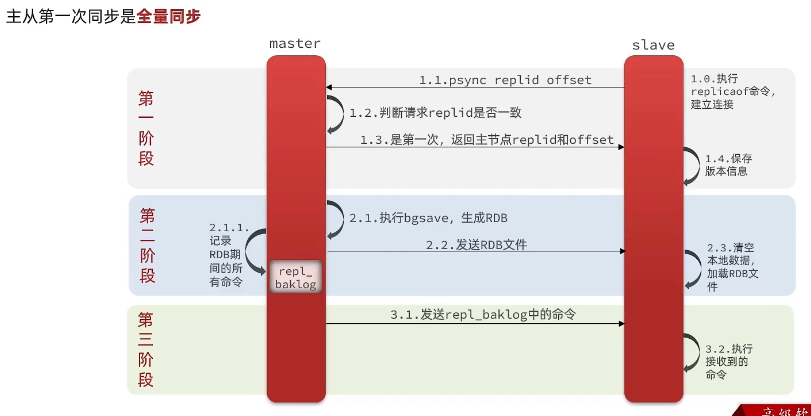
slave节点请求增量同步

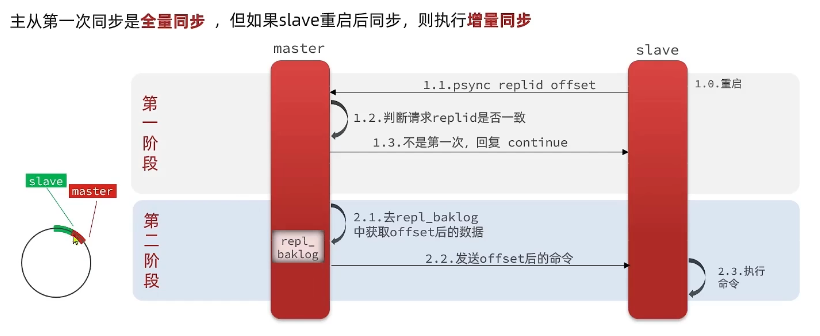
master节点判断replid一致，返回continue

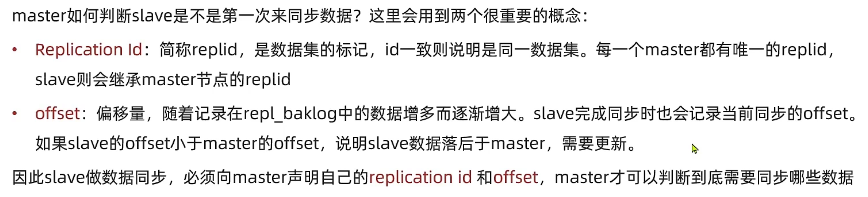
master将repl\_baklog中offset之后的命令发送给slave

slave执行接收到的命令，保持与master同步

注：repL\_baklog大小有上限，写满后会盖最早的数据。如果slave断开时间过久，导致尚未备份的数据被覆盖，则无法基于log做增量同步，只能再次全量同步。







Redis集群优化：

1.启用无磁盘复制，避免全量同步时的磁盘IO。

2.Redis.单节点上的内存占用不要太大，减少全量复制时RDB导致的过多磁盘IO。

3.适当提高缓冲区大小，避免全量同步。

4.限制一个master.上的slave节点数量。

41. 由于主从延迟导致读取到过期数据怎么处理？

答：**1.通过scan命令扫库。**当redis中的key被scan时，相当于访问了该key，同样也会做过期检测，充分发挥redis惰性删除的策略。这个方法能大大降低脏数据读取的概念。缺点也比较明显，会造成一定的数据压力。**2.升级到redis新版本。**在 redis3.2 版本后，redis做了改进，如果读取的数据已经过期了，从库虽然不会删除，但是会返回空值，避免了客户端读到过期数据。

42. Redis主从架构数据会丢失吗，为什么？

答：**1.异步复制导致数据丢失**：主节点数据未复制给从节点时就宕机了，从节点选举为主节点时丢失部分数据。

**2.脑裂导致数据丢失**：主节点由于网络问题脱离从节点，但主节点还在运行，此时从节点选举为主节点后就存在双主节点（脑裂现象），原主节点变为现主节点的从节点时进行全量复制，自己的数据会被清空，导致部分数据丢失。

43. 如何解决主从架构数据丢失的问题？

答：不可避免，只能减小概率。

1..修改配置文件min-slaves-to-write 1；min-slaves-max-lag 10

要求至少有1个slave，数据复制和同步的延迟不能超过10秒。一旦所有从节点数据复制和同步的延迟都超过了10秒钟，这个主节点就不会再接收任何请求。

2.对于客户端可以采取降级措施，将数据暂时写入本地缓存和磁盘中，在一段时间后重新写入master来保证数据不丢失；也可以将数据写入kafka消息队列，隔一段时间去消费kafka中的数据。

44. Redis cluster中是如何实现数据分布的？这种方式有什么优点？

答： 使用哈希槽分区方案。具备可拓展性和高容错性，解决数据倾斜问题。

注：

（1）**哈希取余算法**

每个key所落在的节点位置=hash(key)%N（N为机器数量）

优点：实现简单。缺点：难以扩容缩容；容错性差，有服务器宕机会导致数据重新洗牌。

（2）**一致性哈希算法分区**

**一致性哈希环**：把所有哈希值看成一个哈希环，范围[0，2^32-1]，

**节点IP映射**：节点在环上的位置=hash(IP)，

**落键规则**：计算出hash(key)，找到哈希环上对应的位置并顺时针走，第一个遇到的节点就是存储节点。

优点：解决容错性和可拓展性。缺点：有数据倾斜问题，节点分布不均匀导致数据缓存集中在某个节点上。

（3）**哈希槽分区**

一个集群只能有固定的16384个hash slot（哈希槽），把这些槽位分配给所有主节点。用CRC16算法，计算CRC16(key)%10384来确定槽位，进而确定节点位置。

优点：容易扩容缩容，容错性高，可解决数据倾斜问题。

45. Redis cluster节点间通信是什么机制？

答：采用gossip协议，每个主节点持有一份集群的元数据，每个节点都会频繁给其他节点发送ping，互相通过ping监测彼此健康状态并交换元数据。

优点：去中心化、可扩展、容错、一致性收敛、简单；缺点：消息延迟、消息冗余。

**分布式问题**

46. 什么是分布式锁？为什么用分布式锁？

答：满足分布式系统或集群模式下多进程可见并且互斥的锁。传统意义上的锁只能保证单机的时候有效，分布式集群环境就无能为力了；而分布式锁可以用来控制分布式系统之间同步访问共享资源。

分布式锁需要满足的特性：

多进程可见：集群环境中不同进程都可以获取锁资源

互斥：同一时刻只有一个线程可以获取同一锁资源

高可用：在分布式场景下，一小部分服务器宕机不影响正常使用

高性能：加锁这一操作对业务性能影响不能太大

安全性：锁要有过期机制，防止客户端还未释放锁时宕机或者网络不可达时产生死锁

47. 常见的分布式锁有哪些解决方案？

答：实现分布式锁目前有三种流行方案，MySQL、Redis、ZooKeeper



48. Redis实现分布式锁

（1）Redis分布式锁的三个核心要素：

①加锁。使用setnx来加锁，在redis中设置一个key代表加锁。返回1，说明key原本不存在，该线程成功得到了锁；返回0，说明key已经存在，该线程加锁失败；

②解锁。执行del指令，删除该key即解锁。

③锁超时。如果一个得到锁的线程在执行任务的过程中挂掉，来不及显式地释放锁，将会导致死锁，别的进程无法再获取锁。所以setnx的key必须设置一个超时时间，以保证即使没有被显式释放，这把锁也要在一段时间后自动释放。

（2）上述分布式锁存在的问题

①SETNX 和 EXPIRE 非原子性

某一个进程刚执行setnx，成功得到了锁。还未来得及执行expire命令，节点就挂掉了。此时这把锁就没有设置过期时间，别的线程就再也无法获得该锁。

②锁误解除

某一个线程成功获取到了锁，在锁过期时线程还在执行中，导致该线程执行完毕释放锁时，错误地释放了其他线程加的锁。

③超时解锁导致并发问题

A线程成功获取到了锁，在锁过期时线程还在执行中，当B线程加锁并执行时，AB两个线程处于并发执行状态。

④redis分布式锁不可重入问题

在同一线程内，对某个key加锁在还没解锁之前无法再次加锁。（无法适配递归场景）

⑤无法等待锁释放，不可重试。

加锁失败时会立即返回，无法再次尝试，客户端无法得知锁何时被释放。

（3）解决措施

①Redis2.6.12版本为set指令增加了可选参数，可原子性设置key的过期时间

②del释放锁之前判断的锁是不是自己加的锁。在加锁的时候把当前线程的id当做value，可生成一个 UUID 标识当前线程，在删除之前验证key对应的value是不是自己线程的id。

③将过期时间设置足够长或获取锁时自动开启一个定时任务，每隔一定时间刷新过期时间，总时间超过一定值之后不再续期（watch dog机制）。

④对锁进行重入计数，加锁时加 1，解锁时减 1，当计数归 0 时释放锁。

⑤使用自旋锁，加锁失败时隔一段时间重新尝试加锁，超过一定时间返回失败；使用 Redis 的发布订阅功能，当获取锁失败时，订阅锁释放消息，获取锁成功后当锁释放时，发送锁释放消息。

49. Redis如何做内存优化？

答：1.控制key数量，尽量使用合适的redis数据结构，减少key数量节省内存。

2.缩减键值对象，key尽量短，value存放序列化后的业务对象，业务对象尽量去掉非必要的属性，选择高效的序列化工具。

3.合理设置过期时间、内存策略和持久化策略。

50. 如果现在有个读超高并发的系统，用Redis来抗住大部分读请求，你会怎么设计？

答：先看我们有的方案：单体、主从+哨兵（包含读写分离）、redis cluster集群

①如果读请求每秒几万，可以用单体redis；

②如果读请求每秒十万以上，可以用主从架构+哨兵模式。哨兵集群主要是提高缓存架构的可用性，解决单点故障问题，读写分离机制解决读请求的高并发。

③如果缓存数据超过1T，可以用redis cluster，用redis集群分布式存储解决缓存容量问题。