1、redis的数据结构分别有什么，各数据结构的底层原理如何实现（zset与set的区别，zset的底层实现。zset如何实现分页功能？（例子：ZRANGEBYSCORE zset01(key) 60 90 limit 2 2））

string：字符串

在redis中，其自己定义了一种字符串格式，叫做SDS（Simple Dynamic String），即简单动态字符串

list：列表

ziplist并不是一个类名，其结构是下面这样的：

ziplist类似一个封装的数组，通过zltail可以方便地进行追加和删除尾部数据、使用entries可以方便地计算长度，，但是其依然有数组的缺点，就是当插入和删除数据时会频繁地引起数据移动，

**quicklist：**quicklist是一个双向链表的结构，但是内部又涉及了ziplist，我们可以这么说，在宏观上，quicklist是一个双向链表，在微观上，每一个quicklist的节点都是一个ziplist

在redis.conf中，可以使用下面两个参数来进行优化：

* list-max-ziplist-size：表示每个quicklistNode的字节大小。默认为2，表示8KB
* list-compress-depth：表示quicklistNode节点是否要压缩。默认为0，表示不压缩

这种存储方式的优点和链表的优点一致，就是插入和删除的效率很高，而链表查询的效率又由ziplist来进行弥补，所以quicklist就成为了list数据结构的首选

hash：散列表

**zipmap：**其中相邻的两个字符串就分别是键和值, 就是查找的时间复杂度为O(n)，所以只能当作一个轻量级的hashmap来使用

Dict:

如果我们不想更深入的话了解到这种程度就可以了，其中真正存储数据的是dictEntry结构，很明显是一个链表，我们知道这是采用链式结构存储就足够了

这种方式会消耗较多的内存，所以一般数据较少时会采用轻量级的zipmap

set：无序集合

intset有一个数据升级的概念，比方说我们有一个16位整数的set，这时候插入了一个32位整数，所以就导致整个集合都升级为32位整数，但是反过来却不行，这也就是柔性数组的由来

如果集合过大，会采用dict的方式来进行存储

zset：有序集合

sorted set，是一个键值对的结构，其键被称为member，也就是集合元素（zset依然是set，所以member不能相同），其对应的值被称为score，是一个浮点数，可以理解为优先级，用于排列zset的顺序

其也有两种存储方式，一种是ziplist/zipmap的格式，这种方式我们就不过多介绍了，只需要了解这种格式将数据按照score的顺序排列即可

另一种存储格式是采用了skiplist，意为跳跃表，可以看成平衡树映射的数组，其查找的时间复杂度和平衡树基本没有差别，但是实现更为简单，形如下面这样的结构（图来源跳跃表的原理）：

2、redis集群原理是什么？

**其实就是分库分表，去中心化**

1、集群是如何判断是否有某个节点挂掉

　　首先要说的是，每一个节点都存有这个集群所有主节点以及从节点的信息。它们之间通过互相的ping-pong判断是否节点可以连接上。如果有一半以上的节点去ping一个节点的时候没有回应，集群就认为这个节点宕机了，然后去连接它的备用节点。

2、集群进入fail状态的必要条件

A、某个主节点和所有从节点全部挂掉，我们集群就进入faill状态。

B、如果集群超过半数以上master挂掉，无论是否有slave，集群进入fail状态.

C、如果集群任意master挂掉,且当前master没有slave.集群进入fail状态

3.redis集群去中心化（所有Master节点并发处理读写）

集群中原则每个Master节点都有一个或多个Slave节点。集群中所有的Master节点都可以进行读写数据，不分主次，记redis集群式去中心化的。每个Master节点与Slave节点间通过Goossip协议内部通信，异步复制。不用我们瞎操心（所有的redis节点彼此互联(PING-PONG机制),内部使用二进制协议优化传输速度和带宽.），但是异步赋值会导致某些特定情况下的数据丢失，即，redis集群不能保证数据的强一致性

4.redis集群的分区规则：虚拟槽分区(槽：slot)

RedisCluster采用此分区，所有的键根据哈希函数(CRC16[key]&16383)映射到0－16383槽内，共16384个槽位，每个节点维护部分槽及槽所映射的键值数据

哈希函数: Hash()=CRC16[key]&16383 按位与

redis用虚拟槽分区原因：解耦数据与节点关系，节点自身维护槽映射关系，分布式存储

5. redisCluster的缺陷（虚拟槽分区的缺点）

a，键的批量操作支持有限，比如mset, mget，如果多个键映射在不同的槽，就不支持了

b，键事务支持有限，当多个key分布在不同节点时无法使用事务，同一节点是支持事务

c，键是数据分区的最小粒度，不能将一个很大的键值对映射到不同的节点

d，不支持多数据库，只有0，select 0

e，复制结构只支持单层结构，不支持树型结构。

6.客户端与redis集群交互方式

由于Cluster架构中无Proxy层，客户端是直接与集群中的任意可用节点直接交互的，【话是这么说，但是一个请求是怎么找到集群中的一个节点的呢，redis有多种策略，一般使用CRC16去hash(key)计算改请求要分配到具体的哪一个节点上。然后才是 客户端与节点的直接操作】对象保存到Redis之前先经过CRC16哈希到一个指定的Node上，

3、redis主从切换的投票机制原理。

4、redis一般Qps是多少？

5、redis过期时间如何设置为毫秒？（PEXPIRE命令 后面单位为毫秒 key milliseconds）

6、mysql tinyint(1)代表什么含义？

7、mysql tinyint取值范围是多少？

8、JVM为什么要调Xss,有什么作用，默认值是多少（1024k）,（一般往小调），调整这个参数有什么作用？

9、JVM MetaSpace默认值是多少？

10、ConcurrentHashMap的底层原理。（CAS+Sychronized）

11、创建一个hashMap长度为1000，那么初始的长度应该设置为多少？（比如说，我们有1000个元素new HashMap(1000), 但是理论上来讲new HashMap(1024)更合适，不过上面已经说过，即使是1000，hashmap也自动会将其设置为1024。 但是new HashMap(1024)还不是更合适的，因为0.751000 < 1000, 也就是说为了让0.75 size > 1000, 我们必须这样new HashMap(2048)才最合适，既考虑了&的问题，也避免了resize的问题。）

12、Xms、Xmx为什么要调成一致？你一般设置成多少？

13、JDK 1.8默认的GC是什么？

14、GC日志看过吗？里面都有什么？

15、young区一般设置多少次GC后在进入old区？

16、cms、串行GC、并行GC、G1有什么区别？什么时候用cms？什么时候用G1？生产环境如何选择垃圾收集器？

17、mysql如何避免死锁？代码中如何写可以避免死锁？

18、mysql索引如何会失效？

19、什么时候要创建索引？什么时候不应该创建索引？

20、mysql你是如何优化的？

21、mysql优化主要看哪些参数？（explain）

22、Integer、Float值的比较可以用“==”吗？

**不要用==，尽量使用equals方法。**

23、索引除了BTREE外是否还用过其他索引？

索引是帮助mysql获取数据的数据结构。最常见的索引是Btree索引和Hash索引。

不同的引擎对于索引有不同的支持：Innodb和MyISAM默认的索引是Btree索引；而Mermory默认的索引是Hash索引。

我们在mysql中常用两种索引算法BTree和Hash，两种算法检索方式不一样，对查询的作用也不一样。

一、BTree

BTree索引是最常用的mysql数据库索引算法，因为它不仅可以被用在=,>,>=,<,<=和between这些比较操作符上，而且还可以用于like操作符，只要它的查询条件是一个不以通配符开头的常量，例如：

select \* from user where name like ‘jack%’;

select \* from user where name like ‘jac%k%’;

如果一通配符开头，或者没有使用常量，则不会使用索引，例如：

select \* from user where name like ‘%jack’;

select \* from user where name like simply\_name;

二、Hash

Hash索引只能用于对等比较，例如=,<=>（相当于=）操作符。由于是一次定位数据，不像BTree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IO访问，所以检索效率远高于BTree索引。

但为什么我们使用BTree比使用Hash多呢？主要Hash本身由于其特殊性，也带来了很多限制和弊端：

Hash索引仅仅能满足“=”,“IN”,“<=>”查询，不能使用范围查询。

联合索引中，Hash索引不能利用部分索引键查询。

对于联合索引中的多个列，Hash是要么全部使用，要么全部不使用，并不支持BTree支持的联合索引的最优前缀，也就是联合索引的前面一个或几个索引键进行查询时，Hash索引无法被利用。

Hash索引无法避免数据的排序操作

由于Hash索引中存放的是经过Hash计算之后的Hash值，而且Hash值的大小关系并不一定和Hash运算前的键值完全一样，所以数据库无法利用索引的数据来避免任何排序运算。

Hash索引任何时候都不能避免表扫描

Hash索引是将索引键通过Hash运算之后，将Hash运算结果的Hash值和所对应的行指针信息存放于一个Hash表中，由于不同索引键存在相同Hash值，所以即使满足某个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，还是要通过访问表中的实际数据进行比较，并得到相应的结果。

Hash索引遇到大量Hash值相等的情况后性能并不一定会比BTree高

对于选择性比较低的索引键，如果创建Hash索引，那么将会存在大量记录指针信息存于同一个Hash值相关联。这样要定位某一条记录时就会非常麻烦，会浪费多次表数据访问，而造成整体性能底下。

1. hash索引查找数据基本上能一次定位数据，当然有大量碰撞的话性能也会下降。而btree索引就得在节点上挨着查找了，很明显在数据精确查找方面hash索引的效率是要高于btree的；

2. 那么不精确查找呢，也很明显，因为hash算法是基于等值计算的，所以对于“like”等范围查找hash索引无效，不支持；

3. 对于btree支持的联合索引的最优前缀，hash也是无法支持的，联合索引中的字段要么全用要么全不用。提起最优前缀居然都泛起迷糊了，看来有时候放空得太厉害；

4. hash不支持索引排序，索引值和计算出来的hash值大小并不一定一致。

24、创建对象都有哪些方式？

1，new

Student s = new Student();

在堆储存区开辟了一块空间，其对象的引用存储在栈存储区上。

反射 reflect

java的反射机制是指，

获得类的Class 对象的3中方法：

1,类名.class (任意数据类型，都会有一个class 属性)

2,Class.forName("java.lang.String"); 类的全路径

3,类的实例化对象下，有getClass() 方法。

3，clone

调用clone,jvm就会创建一个新的对象，将前面对象的内容全部拷贝进去。用clone方法创建对象并不会调用任何构造函数。

前提，必须要实现Cloneable 接口，本地实现 protected native Object clone() throws CloneNotSupportedException;

Demo clone = (Demo) newInstance.clone();

4，反序列化

序列化：将堆内存中的java 对象通过某种方式，存储到磁盘上或者传输给其他网络节点，也就是java对象转成二进制。

反序列化：与序列化相反，再将序列化之后的二进制串再转成数据结构或对象。

25、线程都用了哪些设计模式？

多线程开发可以更好的发挥多核cpu性能，常用的多线程设计模式有：Future、Master-Worker、Guard Susperionsion、不变、生产者-消费者 模式；jdk除了定义了若干并发的数据结构，也内置了多线程框架和各种线程池； 锁（分为内部锁、重入锁、读写锁）、ThreadLocal、信号量等在并发控制中发挥着巨大的作用。

多线程设计模式：

1.Single Threaded Execution Pattern

[同一时刻只允许一个线程操作]

比喻：三个挑水的和尚，只能同一时间一个人过桥，不然都掉河里喂鱼了。

总结：在多个线程同时要访问的方法上加上synchronized关键字。

2.Immutable Pattern

[变量赋值一次后只能读取，不能改变。]

比喻：一夫多妻制，多个妻子共享一个丈夫。一旦赋值，任何一个妻子不能更改共享的 husband为其它人。

总结：将多线程共享的变量用final关键字修饰。

3.Guarded Suspension Pattern

[要等到我准备好哦，没准备好就在门口等着，准备好了再叫你。]

比喻：

GG：小伙子去MM家，敲门...

MM：我在换衣服，稍等。

GG：等待 【调用wait()方法挂起自己的线程】

MM：换好了，进来吧 【调用notify()或着notifyAll()方法通知GG】

总结：判断某个条件是否为真，如果条件成立则继续执行一步，如果条件不成立用wait()方法挂起当前线程，条件为真后由另一个线程用notify()或 着notifyAll()方法唤醒挂起的线程。

4.Balking Pattern

[有人在做了？哈哈，太好了，我就不过去了！]

比喻：饭店里我想好了要点的菜后，高高的举起手示意服务生过来，一个服务生准备过去的时候看到另外一个服务生已经过去了，哈哈已经有人过去了我就不过去了。

总结：设置一个volatile的共享变量，多个线程进入同一方法时，判断变量的值是否为真，如果为真说明有人已经在做这个工作了，线程返回，反之将变量赋值为真并执行。

5.Producer-Consumer Pattern

[生产-消费者，你生产蛋榚我来吃。]

比喻(1)：一群猪围着猪食槽的一头，塞进几块玉米饼后，群猪争先恐后从食槽头抢，谁抢到谁吃。

比喻(2)：一群猪围着猪食槽的头和尾，塞进几块玉米饼后，群猪争先恐后从食槽头和食槽尾抢，谁抢到谁吃。

总结：<1>生产者将生产出来的东西add(E e)到一个Queue,然后唤醒正在从Queue等东西的线程，用poll()方法从队列的头获取到东西，当Queue里的东西被取完，取东西的线程再次被挂起。

<2>生产者将生产出来的东西放入(方法很多，有add(E e)、addFirst(E e)、addLast(E e)等)一个Deque,然后唤醒正在从Queue等东西的线程，线程从Deque的头和尾取东西，当Deque里的东西被取完，取东西的线程再次被挂起。

6.Read-Write Lock Pattern

[学生抄的时候，不允许老师擦掉黑板上的字。]

比喻：老师在黑板上写了些字，下面的同学在拼命的抄，过会儿老师要写些新的字，写新字要擦掉原来的那些，问：都写完了么？如果都回答写完了，就擦掉原来的字写新的，如果还有一个回答没写完，就等待最后一个同学抄完再写。

总结：Jdk1.5及以上有专门的读写锁实现

java.util.concurrent.locks.ReentrantReadWriteLock.ReadLock；

java.util.concurrent.locks.ReentrantReadWriteLock.WriteLock；

7.Thread-Per-Message Pattern

[一任务一线程]

总结：一个客人一个妞服务，好是好，可是天朝扫黄太厉害，现在狼多肉少哇，用线程池吧，轮流服务。

8.Worker Thread Pattern

[同Producer-Consumer Pattern]

9.Future Pattern

[给您一张提货单，下午来取。]

比喻：

李刚：大钞票一摔，来只大蛋榚！

店员：对不起没这么大的，但现做，给您张单子，下午xx点来取。

李刚：取货单？？，上面都写了些啥？

订单号：SB01

客户名：李刚

已付款：250

取货时间：PM2:50

总结：调用某个方法时，这个方法可能需要请求其它系统，这个过程比较耗时，为了提高客户的体验需要方法立即返回，过一段时间再查询结果。

在程序里声明一个Hashmap,Key保存订单号,Value保存生产出的蛋榚，然后根据订单号取得对应的蛋榚。

10.Two-Phase Termination Pattern

[玩具收拾好，睡觉去]

比喻：小孩子在玩具，到点了妈妈喊：别玩了，睡觉去！

总结：一个线程在while(!isInterrupted()){..}循环中执行，另外一个线程判断某个条件达到后获得准备被结束线程的句柄，调用interrupt()

设置线程的中断状态。

11.Thread-Specific Storage Pattern

[线程私有物品保管箱]

总结：一个方法可能会被同一个线程访问多次，如果每访问一次就要声明一个数据库连接的Connection变量，则对程序的性能有影响。将Connection放在ThreadLocal里，这样每次访问就不必再产生一个Connection,同一个线程对应同一个Connection.

26、线程池原理？一般是如何设置核心线程数与最大线程数的？是否可以用Excutors创建？为什么？

27、快排的写法？

28、如何排查CPU高的进程？快速定位问题？

29、哪些操作会导致CPU占用高？

30、CAS全称是什么，原理是什么？他有什么作用，会导致什么问题，如何解决导致的问题？除了ABA还有什么问题？

31、voletile 关键字是做什么的？有什么特性？为什么禁止指令重排？

作用：

[内存一致性](https://www.jianshu.com/p/7b901261fc2d)

ThreadA再修改标志位flag的时候从主内存中刷新变量的最新值，同时将线程B工作内存中的flag变量置为不可靠状态(dirty)，那么下次ThreadB如果要使用flag标志位的时候就会从主内存中读取变量的最新值，从而保证了变量再不同线程中的一致性.

[防止指令重排](https://www.jianshu.com/p/7b901261fc2d) ：

[内存屏障](https://www.jianshu.com/p/7b901261fc2d)：  
volatile关键字可以实现变量在各线程中的一致性，并且具有禁止指令重排的功能。其实这两个特性是通过**内存屏障**来实现的.

内存屏障是jvm上的指令,jvm上还有其它指令例如

(1) lock:将主内存中的变量锁定，为一个线程所独占

(2) unclock:将lock加的锁定解除，此时其它的线程可以有机会访问此变量

(3) read:将主内存中的变量值读到工作内存当中

(4) load:将read读取的值保存到工作内存中的变量副本中。

(5) use:将值传递给线程的代码执行引擎

(6) assign:将执行引擎处理返回的值重新赋值给变量副本

(7) store:将变量副本的值存储到主内存中。

(8) write:将store存储的值写入到主内存的共享变量当中。

32、springMVC解释一下@Autowire注解底层是如何工作的？

33、threadLocal类有什么作用？

34、tomcat默认最大的线程数是多少？

maxThreads（最大线程数）：每一次HTTP请求到达Web服务，tomcat都会创建一个线程来处理该请求，那么最大线程数决定了Web服务可以同时处理多少个请求，默认200.

35、mysql默认最大连接数是多少？

accepCount（最大等待数）：当调用Web服务的HTTP请求数达到tomcat的最大线程数时，还有新的HTTP请求到来，这时tomcat会将该请求放在等待队列中，这个acceptCount就是指能够接受的最大等待数，默认100.如果等待队列也被放满了，这个时候再来新的请求就会被tomcat拒绝（connection refused）。

maxConnections（最大连接数）：这个参数是指在同一时间，tomcat能够接受的最大连接数。一般这个值要大于maxThreads+acceptCount。

36、我们在写输入的日志代码时有时候会写log.isinfoEnable(),不写这句与写这句判断有什么区别？

log4j中log.isDebugEnabled(), log.isInfoEnabled()和log.isTraceEnabled()作用

Debug,Info和Trace一般会打印比较详细的信息，而且打印的次数较多，如果我们不加log.isDebugEnabled()等  
进行预先判断，当系统loglevel设置高于Debug或Info或Trace时，虽然系统不会答应出这些级别的日志，但是每次还是会拼接  
参数字符串，影响系统的性能。

这3个方法是对对项目的优化方法，加这个方法目的地在于如果代码中存在连接字符串的情况，打印信息时会出现太多的拼接字符串影响系统性能。如果系统中是固定字符串加不加都可以。