**redis和memcached什么区别？为什么高并发下有时单线程的redis比多线程的memcached效率要高？**

区别：

1.mc可缓存图片和视频。rd支持除k/v更多的数据结构;

2.rd可以使用虚拟内存，rd可持久化和aof灾难恢复，rd通过主从支持数据备份;

3.rd可以做消息队列。

原因：mc多线程模型引入了缓存一致性和锁，加锁带来了性能损耗。

**redis主从复制如何实现的？redis的集群模式如何实现？redis的key是如何寻址的？**

主从复制实现：主节点将自己内存中的数据做一份快照，将快照发给从节点，从节点将数据恢复到内存中。之后再每次增加新数据的时候，主节点以类似于mysql的二进制日志方式将语句发送给从节点，从节点拿到主节点发送过来的语句进行重放。

分片方式：

-客户端分片

-基于代理的分片

● Twemproxy

● codis

-路由查询分片

● Redis-cluster（本身提供了自动将数据分散到Redis Cluster不同节点的能力，整个数据集合的某个数据子集存储在哪个节点对于用户来说是透明的）

redis-cluster分片原理：Cluster中有一个16384长度的槽(虚拟槽)，编号分别为0-16383。每个Master节点都会负责一部分的槽，当有某个key被映射到某个Master负责的槽，那么这个Master负责为这个key提供服务，至于哪个Master节点负责哪个槽，可以由用户指定，也可以在初始化的时候自动生成，只有Master才拥有槽的所有权。Master节点维护着一个16384/8字节的位序列，Master节点用bit来标识对于某个槽自己是否拥有。比如对于编号为1的槽，Master只要判断序列的第二位（索引从0开始）是不是为1即可。这种结构很容易添加或者删除节点。比如如果我想新添加个节点D, 我需要从节点A、B、 C中得部分槽到D上。

**使用redis如何设计分布式锁？说一下实现思路？使用zk可以吗？如何实现？这两种有什么区别？**

redis:

1.线程A setnx(上锁的对象,超时时的时间戳t1)，如果返回true，获得锁。

2.线程B 用get获取t1,与当前时间戳比较,判断是是否超时,没超时false,若超时执行第3步;

3.计算新的超时时间t2,使用getset命令返回t3(该值可能其他线程已经修改过),如果t1==t3，获得锁，如果t1!=t3说明锁被其他线程获取了。

4.获取锁后，处理完业务逻辑，再去判断锁是否超时，如果没超时删除锁，如果已超时，不用处理（防止删除其他线程的锁）。

zk:

1.客户端对某个方法加锁时，在zk上的与该方法对应的指定节点的目录下，生成一个唯一的瞬时有序节点node1;

2.客户端获取该路径下所有已经创建的子节点，如果发现自己创建的node1的序号是最小的，就认为这个客户端获得了锁。

3.如果发现node1不是最小的，则监听比自己创建节点序号小的最大的节点，进入等待。

4.获取锁后，处理完逻辑，删除自己创建的node1即可。

区别:zk性能差一些，开销大，实现简单。

**知道redis的持久化吗？底层如何实现的？有什么优点缺点？**

RDB(Redis DataBase:在不同的时间点将redis的数据生成的快照同步到磁盘等介质上):内存到硬盘的快照，定期更新。缺点：耗时，耗性能(fork+io操作)，易丢失数据。

AOF(Append Only File：将redis所执行过的所有指令都记录下来，在下次redis重启时，只需要执行指令就可以了):写日志。缺点：体积大，恢复速度慢。

bgsave做镜像全量持久化，aof做增量持久化。因为bgsave会消耗比较长的时间，不够实时，在停机的时候会导致大量的数据丢失，需要aof来配合，在redis实例重启时，优先使用aof来恢复内存的状态，如果没有aof日志，就会使用rdb文件来恢复。Redis会定期做aof重写，压缩aof文件日志大小。Redis4.0之后有了混合持久化的功能，将bgsave的全量和aof的增量做了融合处理，这样既保证了恢复的效率又兼顾了数据的安全性。bgsave的原理，fork和cow, fork是指redis通过创建子进程来进行bgsave操作，cow指的是copy on write，子进程创建后，父子进程共享数据段，父进程继续提供读写服务，写脏的页面数据会逐渐和子进程分离开来。

**redis过期策略都有哪些？LRU算法知道吗？写一下java代码实现？**

过期策略:

定时过期(一key一定时器)，惰性过期：只有使用key时才判断key是否已过期，过期则清除。定期过期：前两者折中。

LRU:new LinkedHashMap<K, V>(capacity, DEFAULT\_LOAD\_FACTORY, true);

//第三个参数置为true，代表linkedlist按访问顺序排序，可作为LRU缓存；设为false代表按插入顺序排序，可作为FIFO缓存

LRU算法实现：1.通过双向链表来实现，新数据插入到链表头部；2.每当缓存命中（即缓存数据被访问），则将数据移到链表头部；3.当链表满的时候，将链表尾部的数据丢弃。

LinkedHashMap：HashMap和双向链表合二为一即是LinkedHashMap。HashMap是无序的，LinkedHashMap通过维护一个额外的双向链表保证了迭代顺序。该迭代顺序可以是插入顺序（默认），也可以是访问顺序。

**缓存穿透、缓存击穿、缓存雪崩解决方案？**

缓存穿透：指查询一个一定不存在的数据，如果从存储层查不到数据则不写入缓存，这将导致这个不存在的数据每次请求都要到DB去查询，可能导致DB挂掉。

解决方案：1.查询返回的数据为空，仍把这个空结果进行缓存，但过期时间会比较短；2.布隆过滤器：将所有可能存在的数据哈希到一个足够大的bitmap中，一个一定不存在的数据会被这个bitmap拦截掉，从而避免了对DB的查询。

缓存击穿：对于设置了过期时间的key，缓存在某个时间点过期的时候，恰好这时间点对这个Key有大量的并发请求过来，这些请求发现缓存过期一般都会从后端DB加载数据并回设到缓存，这个时候大并发的请求可能会瞬间把DB压垮。

解决方案：1.使用互斥锁：当缓存失效时，不立即去load db，先使用如Redis的setnx去设置一个互斥锁，当操作成功返回时再进行load db的操作并回设缓存，否则重试get缓存的方法。2.永远不过期：物理不过期，但逻辑过期（后台异步线程去刷新）。

缓存雪崩：设置缓存时采用了相同的过期时间，导致缓存在某一时刻同时失效，请求全部转发到DB，DB瞬时压力过重雪崩。与缓存击穿的区别：雪崩是很多key，击穿是某一个key缓存。

解决方案：将缓存失效时间分散开，比如可以在原有的失效时间基础上增加一个随机值，比如1-5分钟随机，这样每一个缓存的过期时间的重复率就会降低，就很难引发集体失效的事件。

**在选择缓存时，什么时候选择redis，什么时候选择memcached**

选择redis的情况：

      1、复杂数据结构，value的数据是哈希，列表，集合，有序集合等这种情况下，会选择redis, 因为memcache无法满足这些数据结构，最典型的的使用场景是，用户订单列表，用户消息，帖子评论等。

      2、需要进行数据的持久化功能，但是注意，不要把redis当成数据库使用，如果redis挂了，内存能够快速恢复热数据，不会将压力瞬间压在数据库上，没有cache预热的过程。对于只读和数据一致性要求不高的场景可以采用持久化存储

      3、高可用，redis支持集群，可以实现主动复制，读写分离，而对于memcache如果想要实现高可用，需要进行二次开发。

      4、存储的内容比较大，memcache存储的value最大为1M。

选择memcache的场景：

     1、纯KV,数据量非常大的业务，使用memcache更合适，原因是，

           a)memcache的内存分配采用的是预分配内存池的管理方式，能够省去内存分配的时间，redis是临时申请空间，可能导致碎片化。

           b)虚拟内存使用，memcache将所有的数据存储在物理内存里，redis有自己的vm机制，理论上能够存储比物理内存更多的数据，当数据超量时，引发swap,把冷数据刷新到磁盘上，从这点上，数据量大时，memcache更快

           c)网络模型，memcache使用非阻塞的IO复用模型，redis也是使用非阻塞的IO复用模型，但是redis还提供了一些非KV存储之外的排序，聚合功能，复杂的CPU计算，会阻塞整个IO调度，从这点上由于redis提供的功能较多，memcache更快些

           d) 线程模型，memcache使用多线程，主线程监听，worker子线程接受请求，执行读写，这个过程可能存在锁冲突。redis使用的单线程，虽然无锁冲突，但是难以利用多核的特性提升吞吐量。

**缓存与数据库不一致怎么办**

假设采用的主存分离，读写分离的数据库，

如果一个线程A先删除缓存数据，然后将数据写入到主库当中，这个时候，主库和从库同步没有完成，线程B从缓存当中读取数据失败，从从库当中读取到旧数据，然后更新至缓存，这个时候，缓存当中的就是旧的数据。

发生上述不一致的原因在于，主从库数据不一致问题，加入了缓存之后，主从不一致的时间被拉长了

处理思路：在从库有数据更新之后，将缓存当中的数据也同时进行更新，即当从库发生了数据更新之后，向缓存发出删除，淘汰这段时间写入的旧数据。

**主从数据库不一致如何解决**

场景描述，对于主从库，读写分离，如果主从库更新同步有时差，就会导致主从库数据的不一致

1、忽略这个数据不一致，在数据一致性要求不高的业务下，未必需要时时一致性

2、强制读主库，使用一个高可用的主库，数据库读写都在主库，添加一个缓存，提升数据读取的性能。

3、选择性读主库，添加一个缓存，用来记录必须读主库的数据，将哪个库，哪个表，哪个主键，作为缓存的key,设置缓存失效的时间为主从库同步的时间，如果缓存当中有这个数据，直接读取主库，如果缓存当中没有这个主键，就到对应的从库中读取。

**Redis常见的性能问题和解决方案**

  1、master最好不要做持久化工作，如RDB内存快照和AOF日志文件

  2、如果数据比较重要，某个slave开启AOF备份，策略设置成每秒同步一次

  3、为了主从复制的速度和连接的稳定性，master和Slave最好在一个局域网内

  4、尽量避免在压力大得主库上增加从库

  5、主从复制不要采用网状结构，尽量是线性结构，Master<--Slave1<----Slave2 ....

**Redis的数据淘汰策略有哪些**

voltile-lru 从已经设置过期时间的数据集中挑选最近最少使用的数据淘汰

voltile-ttl 从已经设置过期时间的数据库集当中挑选将要过期的数据

voltile-random 从已经设置过期时间的数据集任意选择淘汰数据

allkeys-lru 从数据集中挑选最近最少使用的数据淘汰

allkeys-random 从数据集中任意选择淘汰的数据

no-eviction 禁止驱逐数据

**Redis当中有哪些数据结构**

字符串String、字典Hash、列表List、集合Set、有序集合SortedSet。如果是高级用户，那么还会有，如果你是Redis中高级用户，还需要加上下面几种数据结构HyperLogLog、Geo、Pub/Sub。

**假如Redis里面有1亿个key，其中有10w个key是以某个固定的已知的前缀开头的，如果将它们全部找出来？**

使用keys指令可以扫出指定模式的key列表。

对方接着追问：如果这个redis正在给线上的业务提供服务，那使用keys指令会有什么问题？

这个时候你要回答redis关键的一个特性：redis的单线程的。keys指令会导致线程阻塞一段时间，线上服务会停顿，直到指令执行完毕，服务才能恢复。这个时候可以使用scan指令，scan指令可以无阻塞的提取出指定模式的key列表，但是会有一定的重复概率，在客户端做一次去重就可以了，但是整体所花费的时间会比直接用keys指令长。

**使用Redis做过异步队列吗，是如何实现的**

    使用list类型保存数据信息，rpush生产消息，lpop消费消息，当lpop没有消息时，可以sleep一段时间，然后再检查有没有信息，如果不想sleep的话，可以使用blpop, 在没有信息的时候，会一直阻塞，直到信息的到来。redis可以通过pub/sub主题订阅模式实现一个生产者，多个消费者，当然也存在一定的缺点，当消费者下线时，生产的消息会丢失。

**Redis如何实现延时队列**

   使用sortedset，使用时间戳做score, 消息内容作为key,调用zadd来生产消息，消费者使用zrangbyscore获取n秒之前的数据做轮询处理。