《我想进大厂》之Redis夺命连环11问

原创 科技缪缪 艾小仙 2020-10-08

收录于话题
#面试大全 29 #文章精选汇总 78

这是面试题系列第三篇--redis专题。

说说Redis基本数据类型有哪些吧

1. 字符串：redis没有直接使用C语言传统的字符串表示，而是自己实现的叫做简单动态字符串SDS的抽象类型。C语言的字符串不记录自身的长度信息，而SDS则保存了长度信息，这样将获取字符串长度的时间由O(N)降低到了O(1)，同时可以避免缓冲区溢出和减少修改字符串长度时所需的内存重分配次数。
2. 链表linkedlist：redis链表是一个双向无环链表结构，很多发布订阅、慢查询、监视器功能都是使用到了链表来实现，每个链表的节点由一个listNode结构来表示，每个节点都有指向前置节点和后置节点的指针，同时表头节点的前置和后置节点都指向NULL。
3. 字典hashtable：用于保存键值对的抽象数据结构。redis使用hash表作为底层实现，每个字典带有两个hash表，供平时使用和rehash时使用，hash表使用链地址法来解决键冲突，被分配到同一个索引位置的多个键值对会形成一个单向链表，在对hash表进行扩容或者缩容的时候，为了服务的可用性，rehash的过程不是一次性完成的，而是渐进式的。
4. 跳跃表skiplist：跳跃表是有序集合的底层实现之一，redis中在实现有序集合键和集群节点的内部结构中都是用到了跳跃表。redis跳跃表由zskiplist和zskiplistNode组成，zskiplist用于保存跳跃表信息（表头、表尾节点、长度等），zskiplistNode用于表示表跳跃节点，每个跳跃表的层高都是1-32的随机数，在同一个跳跃表中，多个节点可以包含相同的分值，但是每个节点的成员对象必须是唯一的，节点按照分值大小排序，如果分值相同，则按照成员对象的大小排序。
5. 整数集合intset：用于保存整数值的集合抽象数据结构，不会出现重复元素，底层实现为数组。
6. 压缩列表ziplist：压缩列表是为节约内存而开发的顺序性数据结构，他可以包含多个节点，每个节点可以保存一个字节数组或者整数值。

基于这些基础的数据结构，redis封装了自己的对象系统，包含字符串对象string、列表对象list、哈希对象hash、集合对象set、有序集合对象zset，每种对象都用到了至少一种基础的数据结构。

redis通过encoding属性设置对象的编码形式来提升灵活性和效率，基于不同的场景redis会自动做出优化。不同对象的编码如下：

1. 字符串对象string：int整数、embstr编码的简单动态字符串、raw简单动态字符串
2. 列表对象list：ziplist、linkedlist
3. 哈希对象hash：ziplist、hashtable
4. 集合对象set：intset、hashtable
5. 有序集合对象zset：ziplist、skiplist

Redis为什么快呢?

redis的速度非常的快，单机的redis就可以支撑每秒10几万的并发，相对于mysql来说，性能是mysql的几十倍。速度快的原因主要有几点：

1. 完全基于内存操作
2. C语言实现，优化过的数据结构，基于几种基础的数据结构，redis做了大量的优化，性能极高
3. 使用单线程，无上下文的切换成本
4. 基于非阻塞的IO多路复用机制

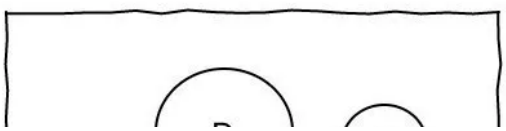
那为什么Redis6.0之后又改用多线程呢?

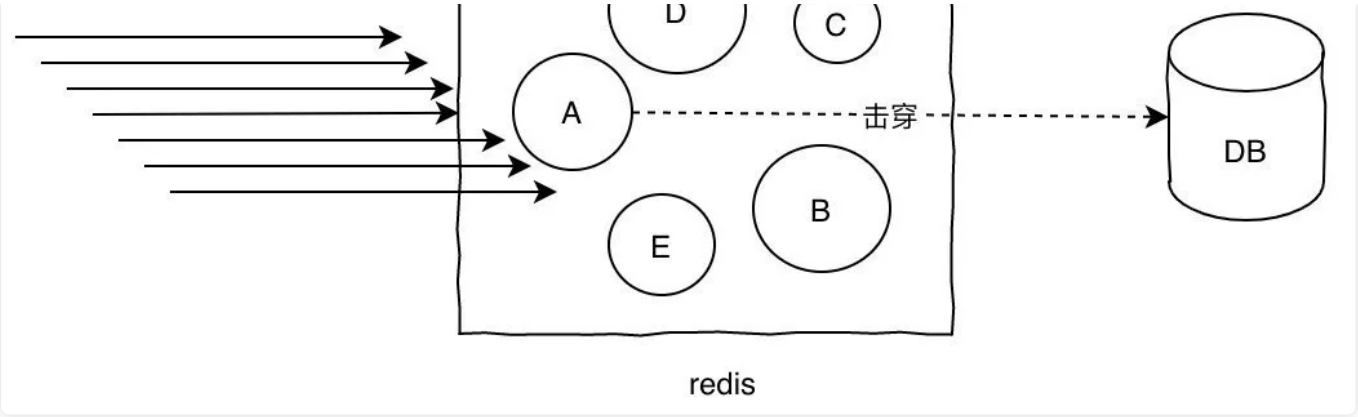
redis使用多线程并非是完全摒弃单线程，redis还是使用单线程模型来处理客户端的请求，只是使用多线程来处理数据的读写和协议解析，执行命令还是使用单线程。

这样做的目的是因为redis的性能瓶颈在于网络IO而非CPU，使用多线程能提升IO读写的效率，从而整体提高redis的性能。

知道什么是热key吗？热key问题怎么解决?

所谓热key问题就是，突然有几十万的请求去访问redis上的某个特定key，那么这样会造成流量过于集中，达到物理网卡上限，从而导致这台redis的服务器宕机引发雪崩。





针对热key的解决方案：

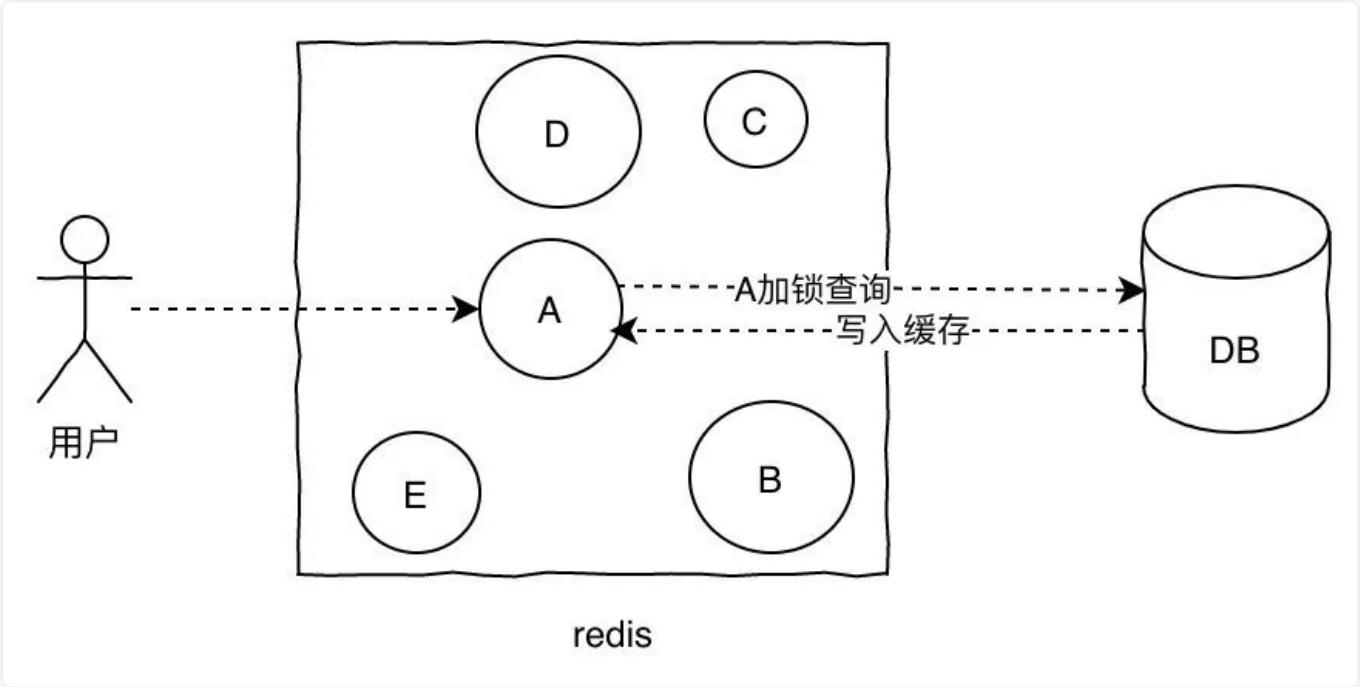
1. 提前把热key打散到不同的服务器，降低压力
2. 加入二级缓存，提前加载热key数据到内存中，如果redis宕机，走内存查询

什么是缓存击穿、缓存穿透、缓存雪崩？

缓存击穿的概念就是单个key并发访问过高，过期时导致所有请求直接打到db上，这个和热key的问题比较类似，只是说的点在于过期导致请求全部打到DB上而已。

解决方案：

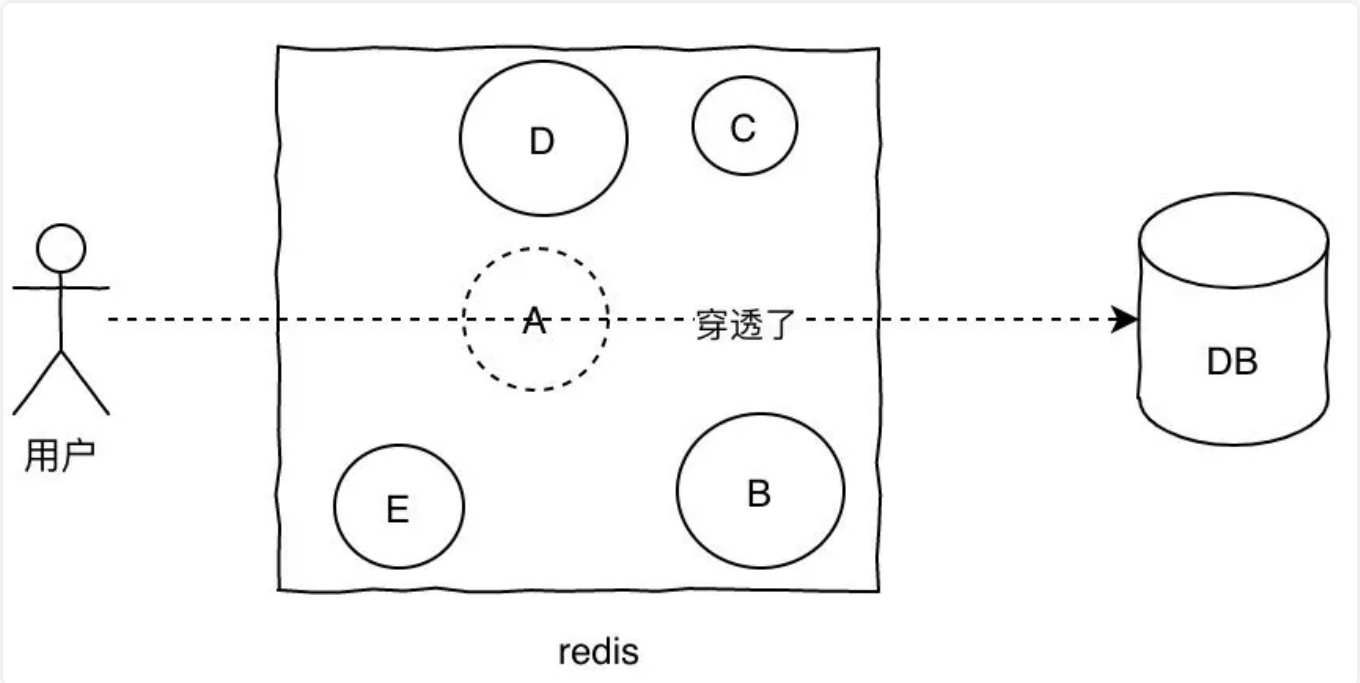
1. 加锁更新，比如请求查询A，发现缓存中没有，对A这个key加锁，同时去数据库查询数据，写入缓存，再返回给用户，这样后面的请求就可以从缓存中拿到数据了。
2. 将过期时间组合写在value中，通过异步的方式不断的刷新过期时间，防止此类现象。



https://tva

缓存穿透

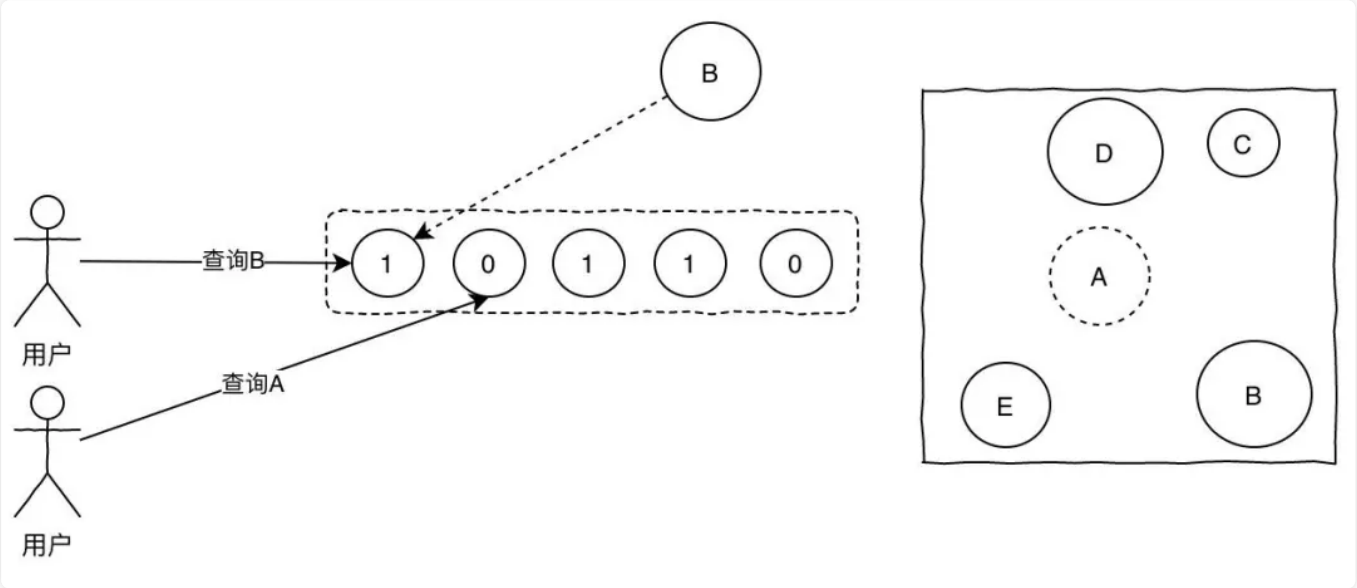
缓存穿透是指查询不存在缓存中的数据，每次请求都会打到DB，就像缓存不存在一样。



针对这个问题，加一层布隆过滤器。布隆过滤器的原理是在你存入数据的时候，会通过散列函数将它映射为一个位数组中的K个点，同时把他们置为1。

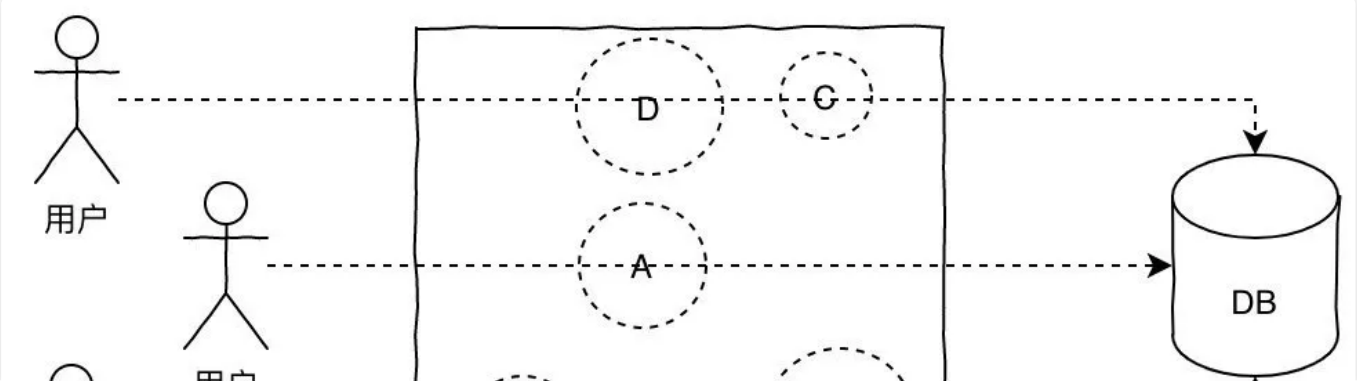
这样当用户再次来查询A，而A在布隆过滤器值为0，直接返回，就不会产生击穿请求打到DB了。

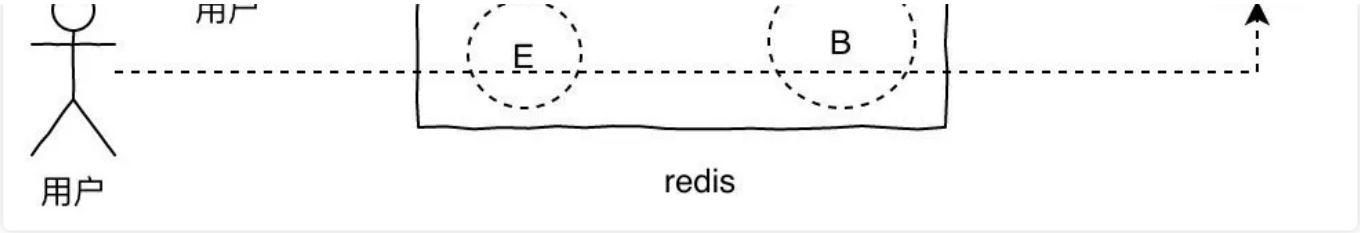
显然，使用布隆过滤器之后会有一个问题就是误判，因为它本身是一个数组，可能会有多个值落到同一个位置，那么理论上来说只要我们的数组长度够长，误判的概率就会越低，这种问题就根据实际情况来就好了。



缓存雪崩

当某一时刻发生大规模的缓存失效的情况，比如你的缓存服务宕机了，会有大量的请求进来直接打到DB上，这样可能导致整个系统的崩溃，称为雪崩。雪崩和击穿、热key的问题不太一样的是，他是指大规模的缓存都过期失效了。





针对雪崩几个解决方案：

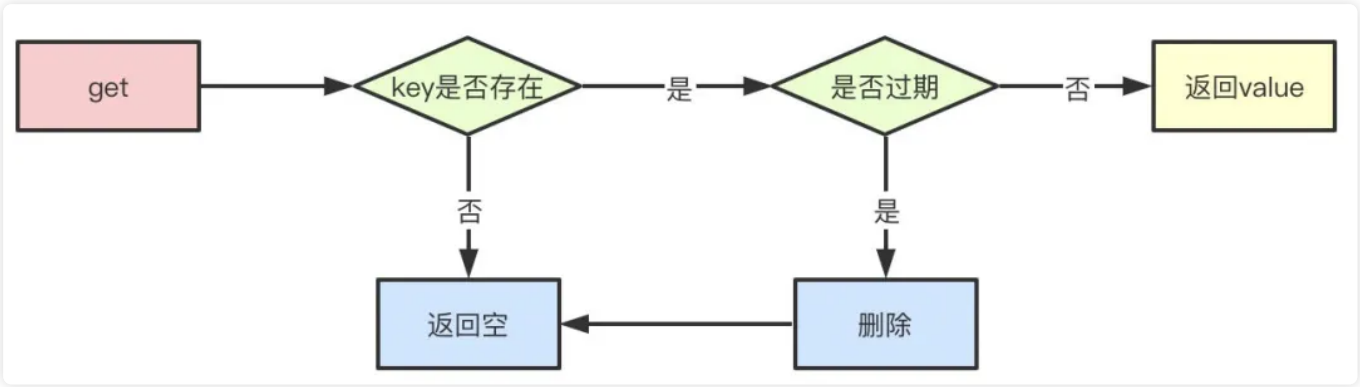
1. 针对不同key设置不同的过期时间，避免同时过期
2. 限流，如果redis宕机，可以限流，避免同时刻大量请求打崩DB
3. 二级缓存，同热key的方案。

Redis的过期策略有哪些?

redis主要有2种过期删除策略

惰性删除

惰性删除指的是当我们查询key的时候才对key进行检测，如果已经达到过期时间，则删除。显然，他有一个缺点就是如果这些过期的key没有被访问，那么他就一直无法被删除，而且一直占用内存。



定期删除

定期删除指的是redis每隔一段时间对数据库做一次检查，删除里面的过期key。由于不可能对所有key去做轮询来删除，所以redis会每次随机取一些key去做检查和删除。

那么定期+惰性都没有删除过期的key怎么办？

假设redis每次定期随机查询key的时候没有删掉，这些key也没有做查询的话，就会导致这些key一直保存在redis里面无法被删除，这时候就会走到redis的内存淘汰机制。

1. volatile-lru：从已设置过期时间的key中，移出最近最少使用的key进行淘汰
2. volatile-ttl：从已设置过期时间的key中，移出将要过期的key
3. volatile-random：从已设置过期时间的key中随机选择key淘汰
4. allkeys-lru：从key中选择最近最少使用的进行淘汰
5. allkeys-random：从key中随机选择key进行淘汰
6. noeviction：当内存达到阈值的时候，新写入操作报错

持久化方式有哪些？有什么区别？

redis持久化方案分为RDB和AOF两种。

RDB持久化可以手动执行也可以根据配置定期执行，它的作用是将某个时间点上的数据库状态保存到RDB文件中，RDB文件是一个压缩的二进制文件，通过它可以还原某个时刻数据库的状态。由于RDB文件是保存在硬盘上的，所以即使redis崩溃或者退出，只要RDB文件存在，就可以用它来恢复还原数据库的状态。

可以通过SAVE或者BGSAVE来生成RDB文件。

SAVE命令会阻塞redis进程，直到RDB文件生成完毕，在进程阻塞期间，redis不能处理任何命令请求，这显然是不合适的。

BGSAVE则是会fork出一个子进程，然后由子进程去负责生成RDB文件，父进程还可以继续处理命令请求，不会阻塞进程。

AOF

AOF和RDB不同，AOF是通过保存redis服务器所执行的写命令来记录数据库状态的。

AOF通过追加、写入、同步三个步骤来实现持久化机制。

1. 当AOF持久化处于激活状态，服务器执行完写命令之后，写命令将会被追加append到aof_buf缓冲区的末尾
2. 在服务器每结束一个事件循环之前，将会调用flushAppendOnlyFile函数决定是否要将aof_buf的内容保存到AOF文件中，可以通过配置appendfsync来决定。

```
always ##aof_buf内容写入并同步到AOF文件
everysec ##将aof_buf中内容写入到AOF文件，如果上次同步AOF文件时间距离现在超过1秒，则再次对AOF文件进行同步
no ##将aof_buf内容写入AOF文件，但是并不对AOF文件进行同步，同步时间由操作系统决定
```

如果不设置，默认选项将会是everysec，因为always来说虽然最安全（只会丢失一次事件循环的写命令），但是性能较差，而everysec模式只不过会可能丢失1秒钟的数据，而no模式的效率和everysec相仿，但是会丢失上次同步AOF文件之后的所有写命令数据。

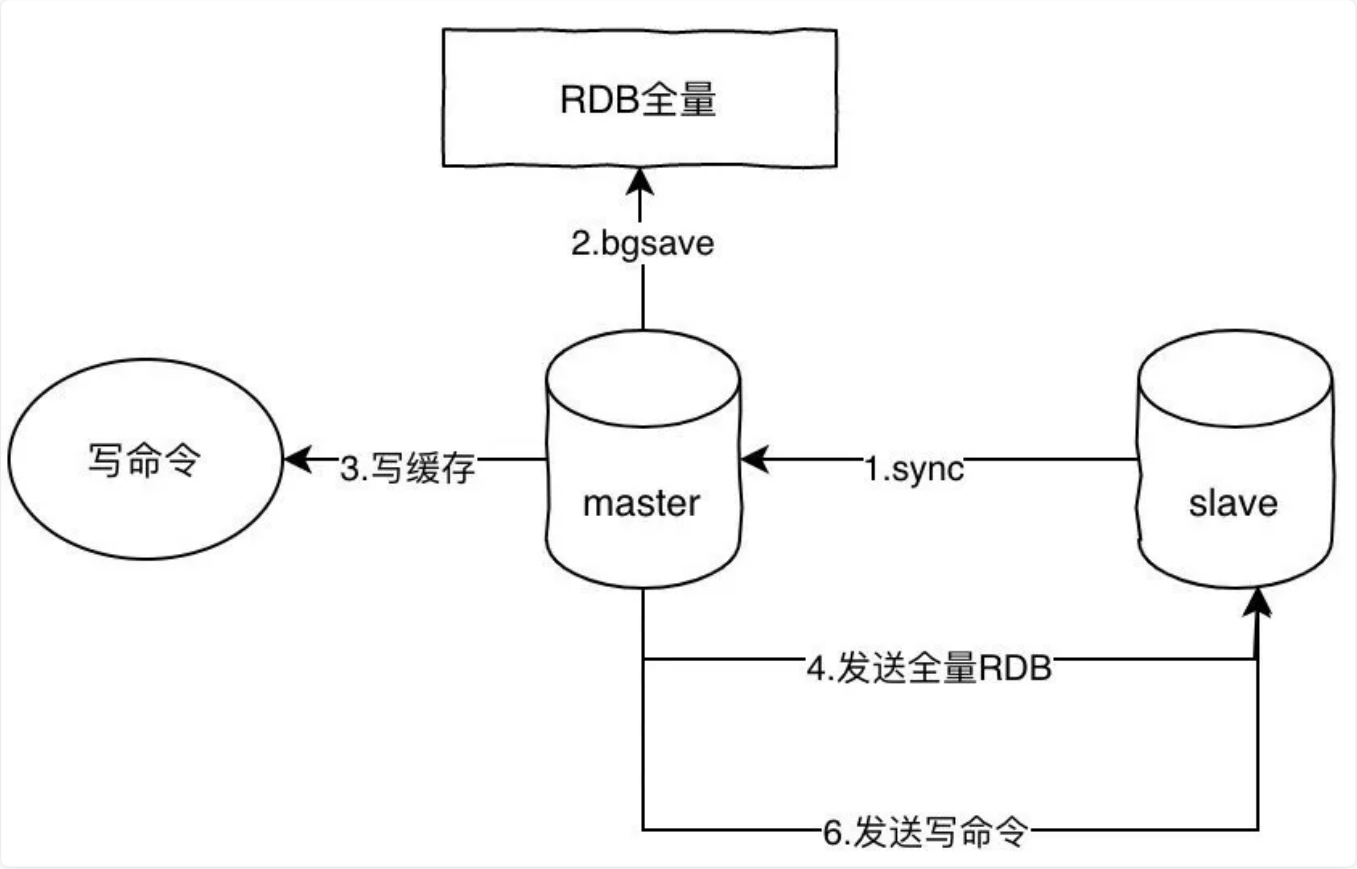
怎么实现Redis的高可用?

要想实现高可用，一台机器肯定是不够的，而redis要保证高可用，有2个可选方案。

主从架构

主从模式是最简单的实现高可用的方案，核心就是主从同步。主从同步的原理如下：

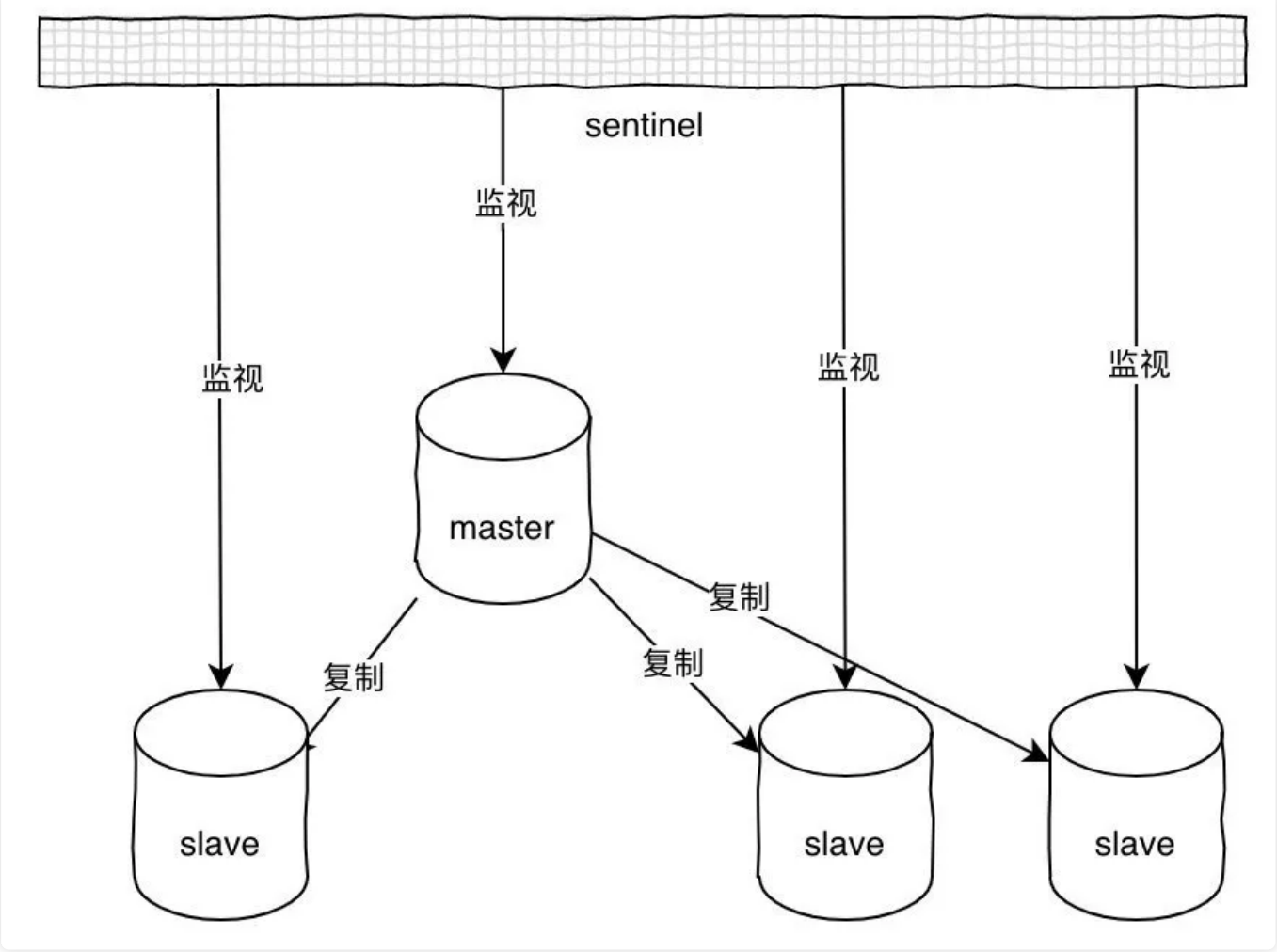
1. slave发送sync命令到master
2. master收到sync之后，执行bgsave，生成RDB全量文件
3. master把slave的写命令记录到缓存
4. bgsave执行完毕之后，发送RDB文件到slave，slave执行
5. master发送缓存中的写命令到slave，slave执行



这里我写的这个命令是sync，但是在redis2.8版本之后已经使用psync来替代sync了，原因是sync命令非常消耗系统资源，而psync的效率更高。

哨兵

基于主从方案的缺点还是很明显的，假设master宕机，那么就不能写入数据，那么slave也就失去了作用，整个架构就不可用了，除非你手动切换，主要原因就是因为没有自动故障转移机制。而哨兵(sentinel)的功能比单纯的主从架构全面的多了，它具备自动故障转移、集群监控、消息通知等功能。



哨兵可以同时监视多个主从服务器，并且在被监视的master下线时，自动将某个slave提升为master，然后由新的master继续接收命令。整个过程如下：

1. 初始化sentinel，将普通的redis代码替换成sentinel专用代码
2. 初始化masters字典和服务器信息，服务器信息主要保存ip:port，并记录实例的地址和ID
3. 创建和master的两个连接，命令连接和订阅连接，并且订阅sentinel:hello频道
4. 每隔10秒向master发送info命令，获取master和它下面所有slave的当前信息
5. 当发现master有新的slave之后，sentinel和新的slave同样建立两个连接，同时每个10秒发送info命令，更新master信息
6. sentinel每隔1秒向所有服务器发送ping命令，如果某台服务器在配置的响应时间内连续返回无效回复，将会被标记为下线状态
7. 选举出领头sentinel，领头sentinel需要半数以上的sentinel同意
8. 领头sentinel从已下线的的master所有slave中挑选一个，将其转换为master
9. 让所有的slave改为从新的master复制数据
10. 将原来的master设置为新的master的从服务器，当原来master重新回复连接时，就变成了新master的从服务器

sentinel会每隔1秒向所有实例（包括主从服务器和其他sentinel）发送ping命令，并且根据回复判断是否已经下线，这种方式叫做主观下线。当判断为主观下线时，就会向其他监视的sentinel询问，如果超过半数的投票认为已经是下线状态，则会标记为客观下线状态，同时触发故障转移。

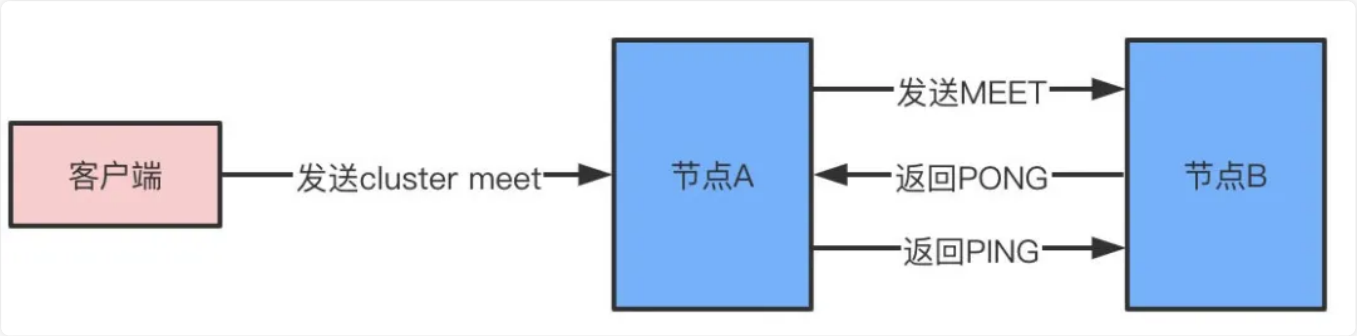
能说说redis集群的原理吗?

如果说依靠哨兵可以实现redis的高可用，如果还想在支持高并发同时容纳海量的数据，那就需要redis集群。redis集群是redis提供的分布式数据存储方案，集群通过数据分片sharding来进行数据的共享，同时提供复制和故障转移的功能。

节点

一个redis集群由多个节点node组成，而多个node之间通过cluster meet命令来进行连接，节点的握手过程：

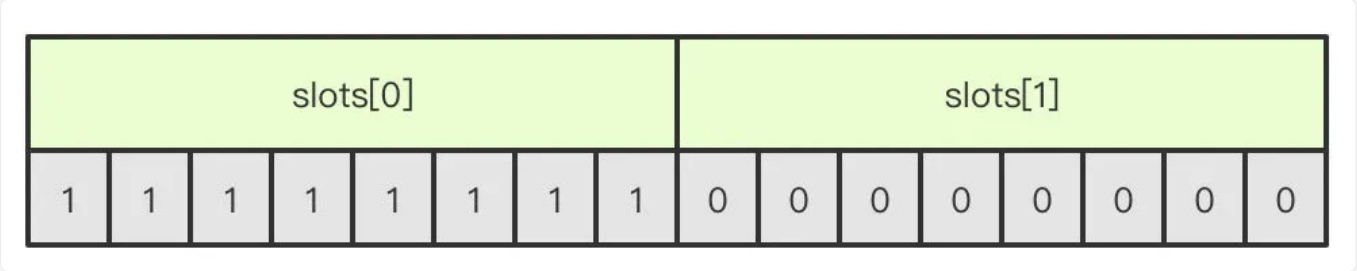
1. 节点A收到客户端的cluster meet命令
2. A根据收到的IP地址和端口号，向B发送一条meet消息
3. 节点B收到meet消息返回pong
4. A知道B收到了meet消息，返回一条ping消息，握手成功
5. 最后，节点A将会通过gossip协议把节点B的信息传播给集群中的其他节点，其他节点也将和B进行握手



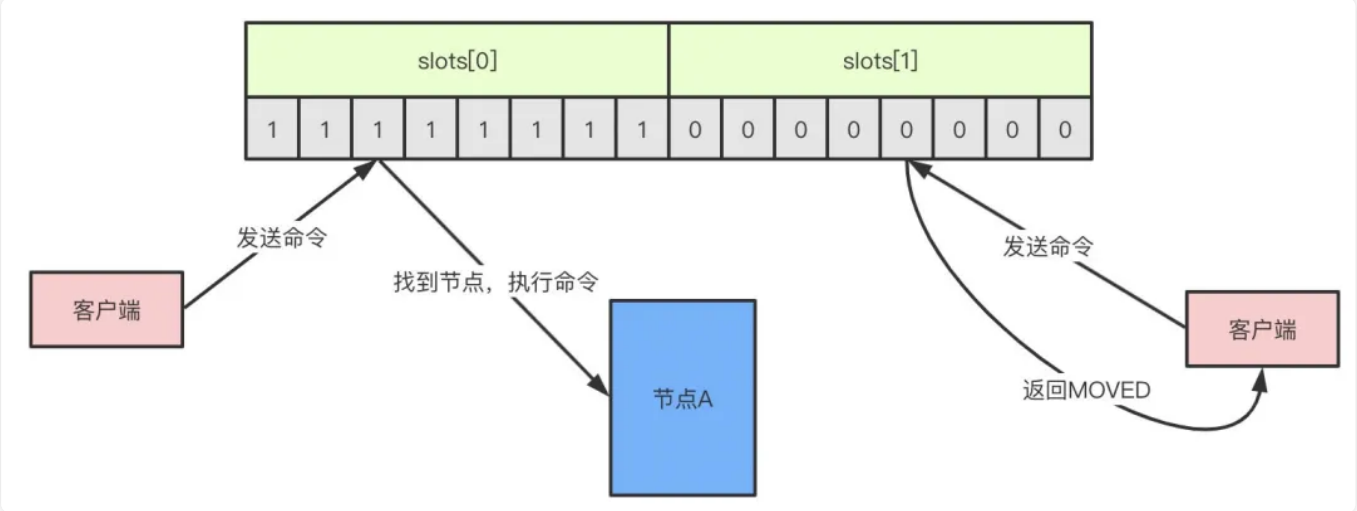
槽slot

redis通过集群分片的形式来保存数据，整个集群数据库被分为16384个slot，集群中的每个节点可以处理0-16384个slot，当数据库16384个slot都有节点在处理时，集群处于上线状态，反之只要有一个slot没有得到处理都会处理下线状态。通过cluster addslots命令可以将slot指派给对应节点处理。

slot是一个位数组，数组的长度是16384/8=2048，而数组的每一位用1表示被节点处理，0表示不处理，如图所示的话表示A节点处理0-7的slot。



当客户端向节点发送命令，如果刚好找到slot属于当前节点，那么节点就执行命令，反之，则会返回一个MOVED命令到客户端指引客户端转向正确的节点。（MOVED过程是自动的）



如果增加或者移出节点，对于slot的重新分配也是非常方便的，redis提供了工具帮助实现slot的迁移，整个过程是完全在线的，不需要停止服务。

故障转移

如果节点A向节点B发送ping消息，节点B没有在规定的时间内响应pong，那么节点A会标记节点B为pfail疑似下线状态，同时把B的状态通过消息的形式发送给其他节点，如果超过半数以上的节点都标记B为pfail状态，B就会被标记为fail下线状态，此时将会发生故障转移，优先从复制数据较多的从节点选择一个成为主节点，并且接管下线节点的slot，整个过程和哨兵非常类似，都是基于Raft协议做选举。

了解Redis事务机制吗?

redis通过MULTI、EXEC、WATCH等命令来实现事务机制，事务执行过程将一系列多个命令按照顺序一次性执行，并且在执行期间，事务不会被中断，也不会去执行客户端的其他请求，直到所有命令执行完毕。事务的执行过程如下：

1. 服务端收到客户端请求，事务以MULTI开始
2. 如果客户端正处于事务状态，则会把事务放入队列同时返回给客户端QUEUED，反之则直接执行这个命令
3. 当收到客户端EXEC命令时，WATCH命令监视整个事务中的key是否有被修改，如果有则返回空回复到客户端表示失败，否则redis会遍历整个事务队列，执行队列中保存的所有命令，最后返回结果给客户端

WATCH的机制本身是一个CAS的机制，被监视的key会被保存到一个链表中，如果某个key被修改，那么REDIS_DIRTY_CAS标志将会被打开，这时服务器会拒绝执行事务。

- END -

往 期 推 荐

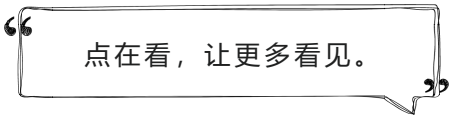
《我想进大厂》之mysql夺命连环13问

再深入一点|binlog和relay-log到底长啥样？

面试官：哪些场景会产生OOM？怎么解决？

《我想进大厂》之MQ夺命连环11问

长按二维码识别关注



收录于话题 #面试大全·29个

上一篇
《我想进大厂》之Dubbo普普通通9问

下一篇
面试官：哪些场景会产生OOM？怎么解决？