Redis笔记

1：

redis lettuce redision

三种客户端：

redistemplate 支持lettuce

redission做分布式的支持比较好;

2：

还是这个强啊

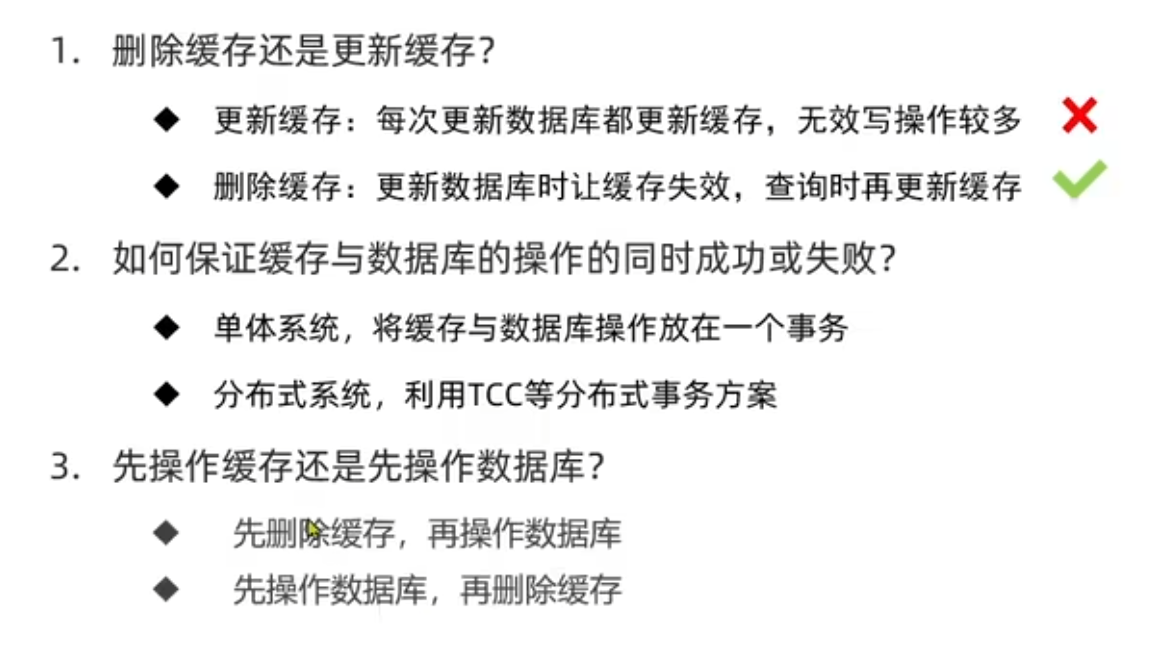
3：数据一致性





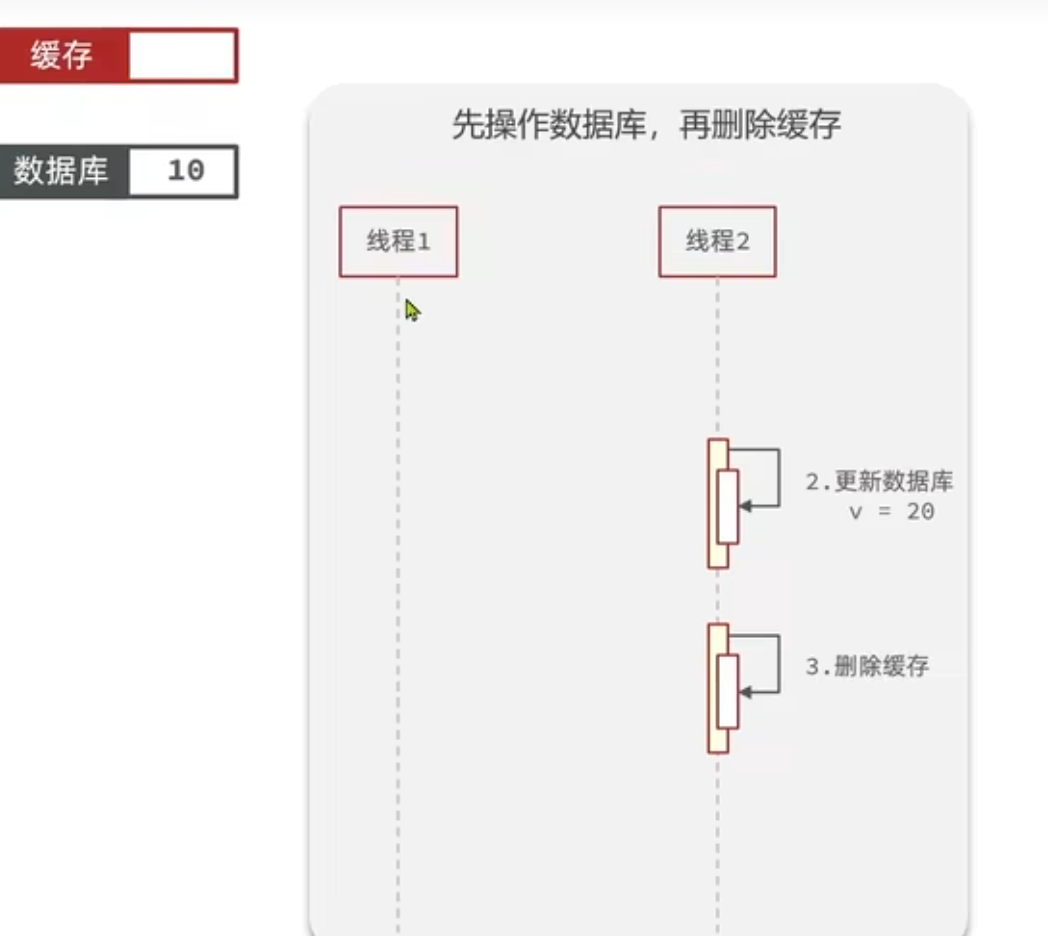
三种策略选一个 业务自行更新， 其他线程更新，或者 抽象成一个事务；

确定删除缓存；





删缓存很快，更新数据库很慢， 所以 缓存10 数据库20；



1：查的时候没有数据，1查数据 获得了10

2：访问数据库 设置成20

1：设置缓存为10 ；

也不安全；但是 认为 1查数据库很快，所以不大可能会发生这种事情；

有点牵强，

4：

缓存雪崩

randomttl

5：缓存击穿

就是访问空

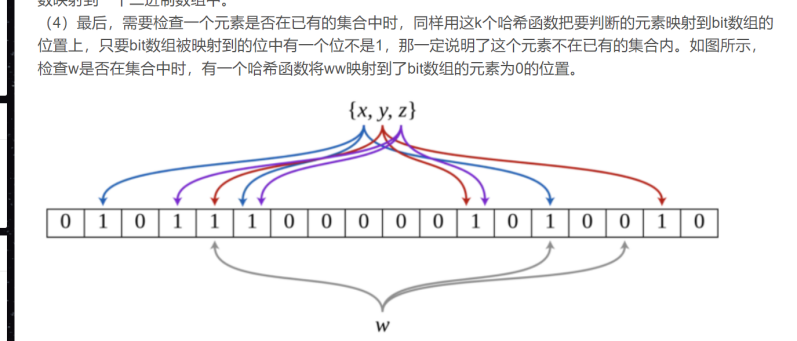
缓存中存着null 或者 布隆过滤器

5：缓存穿透

加互斥锁， 获得锁失败 原地自旋

或者逻辑过期， 获得锁成功的重建， 获得锁失败的返回旧数据；

6：布隆过滤器



使用k个函数 依次hash ，只要检测到为0说明一定不存在， 否则可能存在；k越多 巧合的概率就小了

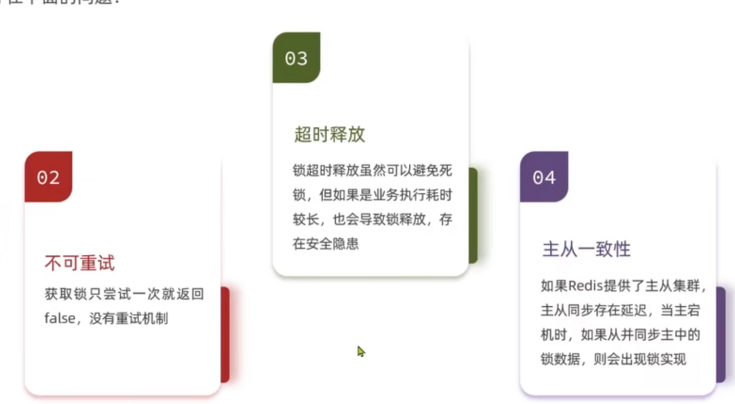
7：分布式锁

set lock tid1 EX NX

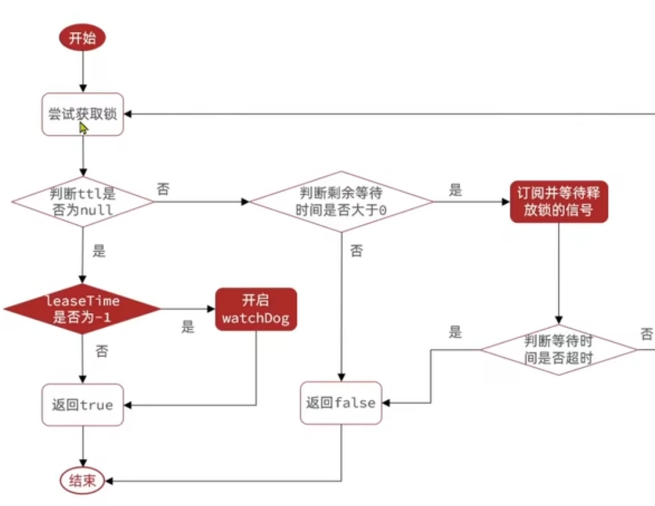
因为可能执行过程要崩溃 所以要Ex，

tid1是为了防止误删的， 只有thid相同 才可以删除

不可重试 不可重入；



redission 使用hashkey 来做锁， 如果有 hashkey+1



会有一个看门狗 持续通知 锁的剩余时间；

如果获取失败，则订阅该消息， 当超时 返回FALSE 否则 收到订阅消息，再次尝试获得锁

主从一致性：

所有的节点都获取成功了才算成功；

8：

rdb的时机

save的时候 阻塞

bgsave fork 一个新的线程 把数据备份下来；

触发rdb的频率的时候

aof

每秒， 每次， 或者缓冲区满了的时候执行；

rdb 系统占用大 重建快 文件小 信息少

aof 系统占用小 重建慢 文件大 信息大

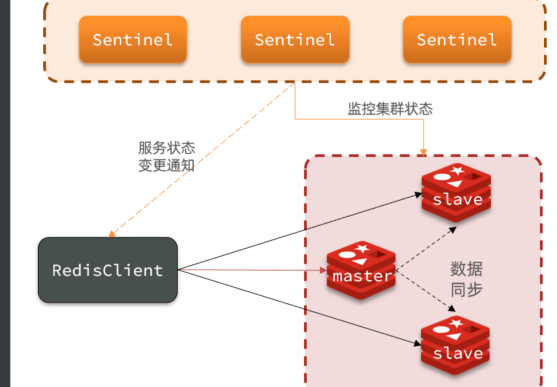
可以压缩；

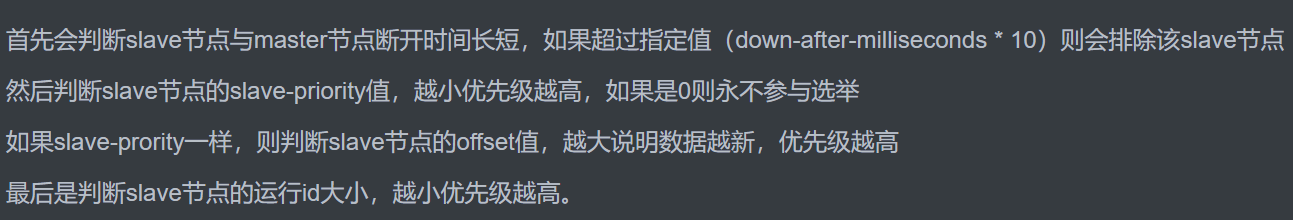
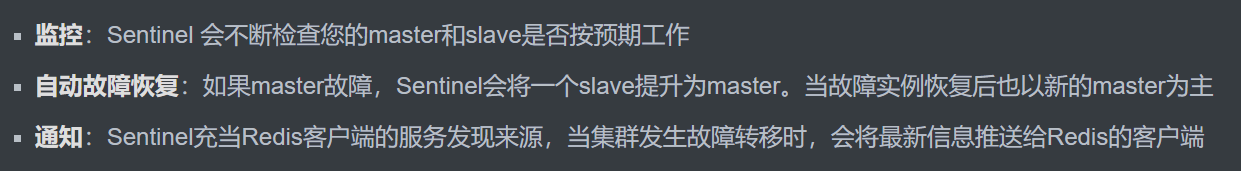
9：

redis主从的增量同步



10：

哨兵集群；



在master发生故障的时候， 选择合适的slave

根据slave-prioriy 和offset offset越大 说明数据越新，越适合做新的master

11：分片集群

slot 16384个插槽，

不同的key 通过{key}来控制他们落入同样的插槽之中；



ziplist 和quicklist

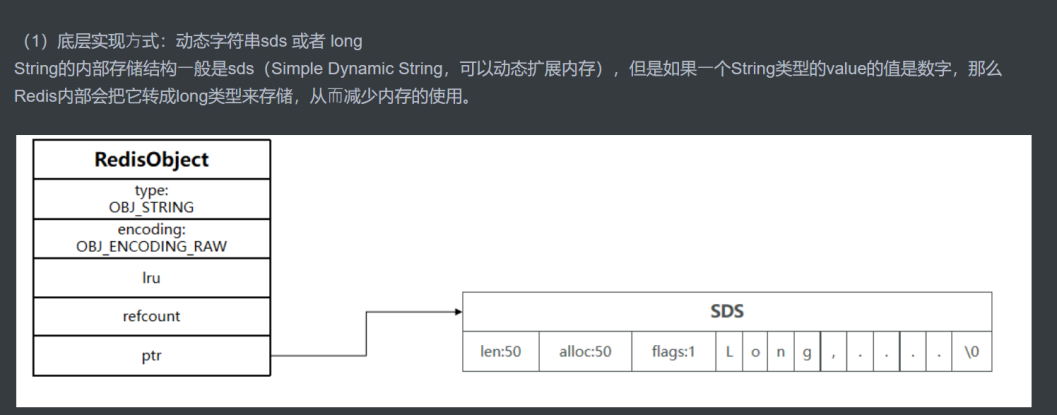
和skiplist

跳表；

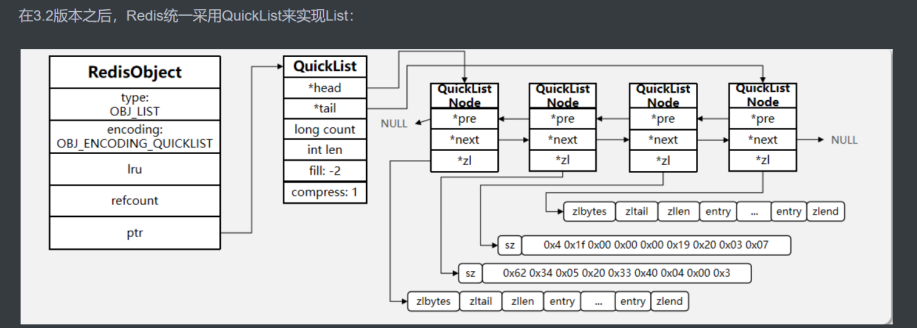
redisobject 为数据；

使用到的数据结构

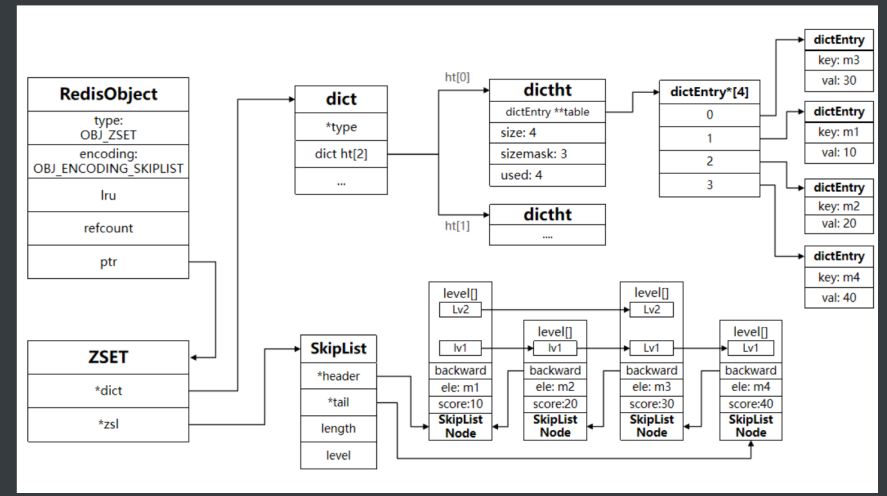
String：sds

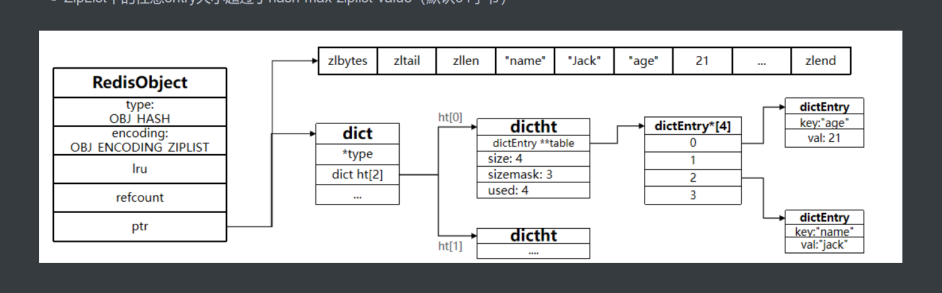


List：使用quicklist



zset：skiplist hash





12：

存在两套链表， 一个存放所有的键 另一个存放有ttl的键

过期策略：

内存过期，访问的时候再删除

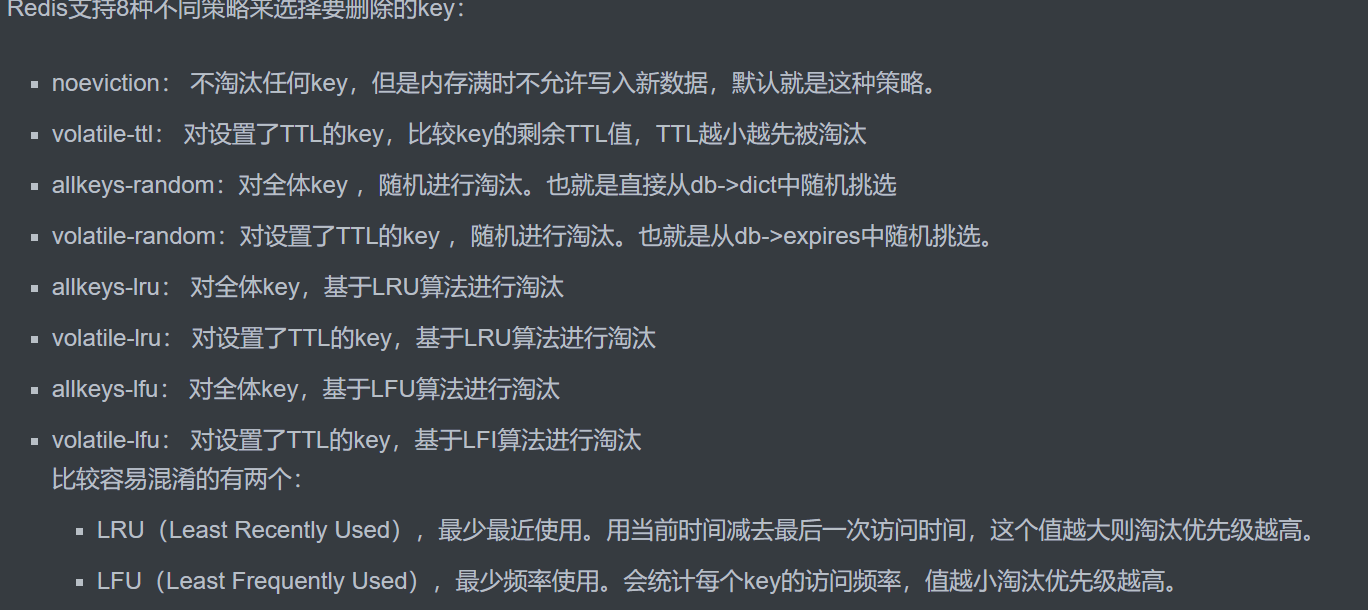
定时清除：设置一个定时器

定期清除；定期随机删除；

13：内存淘汰策略：

LRU 最近最少使用；

LFU 最少频率使用；最不常用的就淘汰了；



14： keys太慢 阻塞了所有线程怎么办；

建议使用Scan代替 keys

使用方法参考：Redis 用scan模糊匹配key，避免阻塞\_梦~'-CSDN博客

用SCAN cursor [MATCH pattern] [COUNT count]命令以迭代的方式进行key遍历（限制单次查询的key数量）。

这个 count 不是限定返回结果的数量，而是限定服务器单次遍历的字典槽位数量(约等于)。

缺点：

1.同一个元素可能会被返回多次。 处理重复元素的工作交由应用程序负责， 比如说， 可以考虑将迭代返回的元素仅仅用于可以安全地重复执行多次的操作上。

2.如果一个元素是在迭代过程中被添加到数据集的， 又或者是在迭代过程中从数据集中被删除的， 那么这个元素可能会被返回， 也可能不会，redis scan对返回结果集只提供有限的保证。

3.元素如果在迭代过程中被删除了，可能不会被返回。

4. 将想要禁止的命令，按以下格式加入配置文件中: