Redis开发规范

1.冷热数据分离，不要将所有数据全部都放到Redis中

虽然Redis支持持久化，但是Redis的数据存储全部都是在内存中的，成本昂贵。建议根据业务只将高频热数据存储到Redis中【QPS大于5000】，对于低频冷数据可以使用MySQL/ElasticSearch/MongoDB等基于磁盘的存储方式，不仅节省内存成本，而且数据量小在操作时速度更快、效率更高！

2.不同的业务数据要分开存储

不要将不相关的业务数据都放到一个Redis实例中，建议新业务申请新的单独实例。因为Redis为单线程处理，独立存储会减少不同业务相互操作的影响，提高请求响应速度；同时也避免单个实例内存数据量膨胀过大，在出现异常情况时可以更快恢复服务！

3.存储的Key一定要设置超时时间

如果应用将Redis定位为缓存Cache使用，对于存放的Key一定要设置超时时间！因为若不设置，这些Key会一直占用内存不释放，造成极大的浪费，而且随着时间的推移会导致内存占用越来越大，直到达到服务器内存上限！另外Key的超时长短要根据业务综合评估，而不是越长越好！

4.对于必须要存储的大文本数据一定要压缩后存储

对于大文本【超过500字节】写入到Redis时，一定要压缩后存储！大文本数据存入Redis，除了带来极大的内存占用外，在访问量高时，很容易就会将网卡流量占满，进而造成整个服务器上的所有服务不可用，并引发雪崩效应，造成各个系统瘫痪！

5.线上Redis禁止使用Keys正则匹配操作

Redis是单线程处理，在线上KEY数量较多时，操作效率极低【时间复杂度为O(N)】，该命令一旦执行会严重阻塞线上其它命令的正常请求，而且在高QPS情况下会直接造成Redis服务崩溃！如果有类似需求，请使用scan命令代替！

6.可靠的消息队列服务

Redis List经常被用于消息队列服务。假设消费者程序在从队列中取出消息后立刻崩溃，但由于该消息已经被取出且没有被正常处理，那么可以认为该消息已经丢失，由此可能会导致业务数据丢失，或业务状态不一致等现象发生。为了避免这种情况，Redis提供了RPOPLPUSH命令，消费者程序会原子性的从主消息队列中取出消息并将其插入到备份队列中，直到消费者程序完成正常的处理逻辑后再将该消息从备份队列中删除。同时还可以提供一个守护进程，当发现备份队列中的消息过期时，可以重新将其再放回到主消息队列中，以便其它的消费者程序继续处理。

7.谨慎全量操作Hash、Set等集合结构

在使用HASH结构存储对象属性时，开始只有有限的十几个field，往往使用HGETALL获取所有成员，效率也很高，但是随着业务发展，会将field扩张到上百个甚至几百个，此时还使用HGETALL会出现效率急剧下降、网卡频繁打满等问题【时间复杂度O(N)】,此时建议根据业务拆分为多个Hash结构；或者如果大部分都是获取所有属性的操作,可以将所有属性序列化为一个STRING类型存储！同样在使用SMEMBERS操作SET结构类型时也是相同的情况！

8.根据业务场景合理使用不同的数据结构类型

目前Redis支持的数据库结构类型较多：字符串（String），哈希（Hash），列表（List），集合（Set），有序集合（Sorted Set），Bitmap，HyperLogLog和地理空间索引（geospatial）等,需要根据业务场景选择合适的类型，常见的如：String可以用作普通的K-V、计数类；

Hash可以用作对象如商品、经纪人等，包含较多属性的信息；List可以用作消息队列、粉丝/关注列表等；Set可以用于推荐；Sorted Set可以用于排行榜等！