## mySQL里有2000w数据，redis中只存20w的数据，如何保证redis中的数据都是热点数据？

redis 内存数据集大小上升到一定大小的时候，就会施行数据淘汰策略。

通过配置redis.conf文件中maxmemory的值来开启内存淘汰功能。当客户端发起了需要申请更多内存的命令，redis检查内存使用情况，如果已经使用的内存大于maxmemory，则开始根据用户配置的不同淘汰策略来淘汰一些数据，从而换取一定的内存。

配置# maxmemory-policy noeviction来选择不同的策略。Redis支持运行时修改淘汰策略，不需要重启redis实例而实时的调整内存淘汰策略。



redis 提供 6种数据淘汰策略。

volatile-lru：从已设置过期时间的数据集（server.db[i].expires）中挑选最近最少使用的数据淘汰

volatile-ttl：从已设置过期时间的数据集（server.db[i].expires）中挑选将要过期的数据淘汰

volatile-random：从已设置过期时间的数据集（server.db[i].expires）中任意选择数据淘汰

allkeys-lru：从数据集（server.db[i].dict）中挑选最近最少使用的数据淘汰

allkeys-random：从数据集（server.db[i].dict）中任意选择数据淘汰

no-enviction（驱逐）：禁止驱逐数据

### lru

在服务器配置中保存了 lru 计数器 server.lrulock，会定时（redis 定时程序 serverCorn()）更新，server.lrulock 的值是根据 server.unixtime 计算出来的。

另外，从 struct redisObject 中可以发现，每一个 redis 对象都会设置相应的 lru。可以想象的是，每一次访问数据的时候，会更新 redisObject.lru。

**LRU 数据淘汰机制是这样的**：在数据集中随机挑选几个键值对，取出其中 lru 最大的键值对淘汰。所以，你会发现，**redis 并不是保证取得所有数据集中最近最少使用（LRU）的键值对，而只是随机挑选的几个键值对中的。**

### TTL

redis 数据集数据结构中保存了键值对过期时间的表，即 redisDb.expires。和 LRU 数据淘汰机制类似，TTL 数据淘汰机制是这样的：从过期时间的表中随机挑选几个键值对，取出其中 ttl 最大的键值对淘汰。同样你会发现，**redis 并不是保证取得所有过期时间的表中最快过期的键值对，而只是随机挑选的几个键值对中的。**

## Memcached和Redis区别

1 memcached将数据全部存在内存中，断电后，数据全部消失。redis除了将数据存放在内存中，还支持数据持久化。

2 memcached对数据类型支持相对简单，键值对，而redis还支持list set hash等数据类型

3 redis支持数据的备份，即一主多从

## redis常见的性能问题

1.Master写内存快照，save命令调度rdbSave函数，会阻塞主线程的工作，当快照比较大时对性能影响是非常大的，会间断性暂停服务，所以Master最好不要写内存快照。

2.Master AOF持久化，如果不重写AOF文件，这个持久化方式对性能的影响是最小的，但是AOF文件会不断增大，AOF文件过大会影响Master重启的恢复速度。

3.Master调用BGREWRITEAOF重写AOF文件，AOF在重写的时候会占大量的CPU和内存资源，导致服务load过高，出现短暂服务暂停现象。

4.Redis主从复制的性能问题，第一次Slave向Master同步的实现是：Slave向Master发出同步请求，Master先dump出rdb文件，然后将rdb文件全量传输给slave，然后Master把缓存的命令转发给Slave，初次同步完成。第二次以及以后的同步实现是：Master将变量的快照直接实时依次发送给各个Slave。不管什么原因导致Slave和Master断开重连都会重复以上过程。Redis的主从复制是建立在内存快照的持久化基础上，只要有Slave就一定会有内存快照发生。虽然Redis宣称主从复制无阻塞，但由于磁盘io的限制，如果Master快照文件比较大，那么dump会耗费比较长的时间，这个过程中Master可能无法响应请求，也就是说服务会中断，对于关键服务，这个后果也是很可怕的。

总结：

1.Master最好不要做任何持久化工作，包括内存快照和AOF日志文件，特别是不要启用内存快照做持久化。

2.如果数据比较关键，某个Slave开启AOF备份数据，策略为每秒同步一次。

3.为了主从复制的速度和连接的稳定性，Slave和Master最好在同一个局域网内。

4.尽量避免在压力较大的主库上增加从库

5.为了Master的稳定性，主从复制不要用图状结构，用单向链表结构更稳定，即主从关系为：Master<--Slave1<--Slave2<--Slave3.......，这样的结构也方便解决单点故障问题，实现Slave对Master的替换，也即，如果Master挂了，可以立马启用Slave1做Master，其他不变。