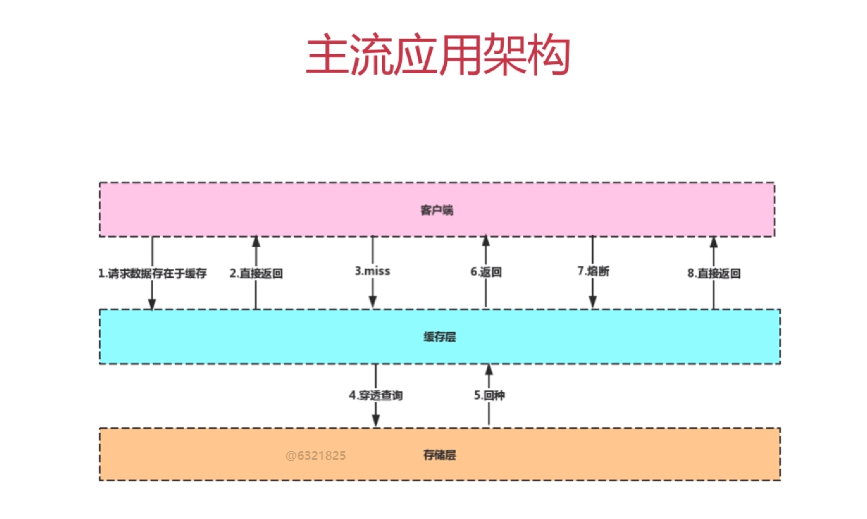
## Redis



1. redis和memcache的区别

|  |  |  |
| --- | --- | --- |
|  | Redis | Memcache |
| 数据类型 | 丰富 | 简单数据类型 |
| 持久化 | 支持 | 不支持 |
| 主从 | 支持 | 不支持 |
| 分片 | 支持 | 不支持 |

1. redis的数据类型

String: 最基本的数据类型，

命令set，get

Hash: String元素组成的字典，

命令 hset user name liqian age 24，hget user name/age

List: 有序列表，按String插入顺序排序，

命令lpush mylist aaa,lrange mylist 0 10

rpush mylist aaa，lpop mylist(移出list第一个元素)

该List类似于栈，后进先出，试用场景最新消息排行榜

Set: String无序集合，通过hash实现不允许重复

命令 sadd myset aaa，smembers myset

Sorted Set：通过分数为集合中成员从小到大排序

命令 zadd myzset 1 aaa，zrangebyscore myzset 0 10

HyperLogLog:用于计数

Geo：用于存储地理位置信息

1. 海量数据查询某固定前缀的key

使用keys 命令会造成redis客户端卡顿

使用scan 命令

格式scan [cursor] match [pattern] count [count]

每次执行返回一个新的cursor，并且返回的key的集合可能重复，返回的数量尽可能和 count一样，但不一定相等

使用该种方式不会造成服务器卡顿

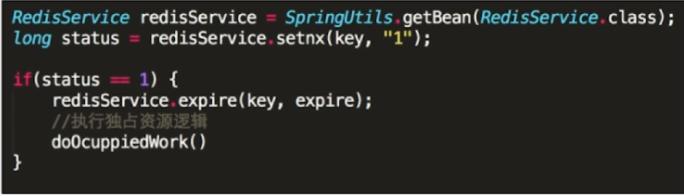
1. redis 实现分布式锁

setnx key指令设置key

expire key seconds指令为key指定存活时间

一个线程执行时先使用setnx指令设置一个key如果设置成功返回1则获得锁，如果设置 失败返回0，证明该锁已被占用，进入等待

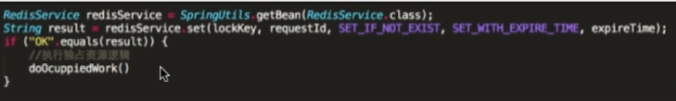
Eg：



使用该种方式存在问题假如setnx执行后redis挂掉那么key则永远不会过期，返回值一直为0，setnx和expire分开执行不是原子操作

新方法 set myKey test ex 10 nx

成功返回ok 失败返回nil



1. 如何解决大量key同时过期，会造成卡顿

只需要在key的过期时间上加一个随机值

1. redis做为异步队列
2. 可以使用list作为队列rpush生产消息lpop消费消息

缺点：没有等待队列里有值就直接消费

弥补：在应用层引入sleep机制去lpop重试

1. 使用 blpop key timeout

等待list,timeout秒直到超时

缺点：消费端只能有一个

1. 使用pub/sub主题订阅模式

Pub消息发送者，sub消息订阅者

subscribe mytopic 订阅mytopic频道

Publish mytopic hello world 向mytopic频道发布消息

缺点：消息发布是无状态的，无法保证消息订阅者一定接收到消息

1. redis 持久化

RDB （快照）：保存某个时间点的全量数据快照

AOF：保存写状态，记录下了除了查询以外所有变更数据库的指令

如果同时使用 RDB 和 AOF 两种持久化机制，那么在 redis 重启的时候，会使用 AOF 来重新构建数据，因为 AOF 中的数据更加完整。

# Redis的过期策略以及内存淘汰机制

分析:这个问题其实相当重要，比如你redis只能存5G数据，可是你写了10G，那会删5G的数据。怎么删的，这个问题思考过么？还有，你的数据已经设置了过期时间，但是时间到了，内存占用率还是比较高，有思考过原因么?

回答:redis采用的是定期删除+惰性删除策略。

为什么不用定时删除策略?

定时删除,用一个定时器来负责监视key,过期则自动删除。虽然内存及时释放，但是十分消耗CPU资源。在大并发请求下，CPU要将时间应用在处理请求，而不是删除key,因此没有采用这一策略.

定期删除+惰性删除是如何工作的呢?

定期删除，redis默认每个100ms检查，是否有过期的key,有过期key则删除。需要说明的是，redis不是每个100ms将所有的key检查一次，而是随机抽取进行检查(如果每隔100ms,全部key进行检查，redis岂不是卡死)。因此，如果只采用定期删除策略，会导致很多key到时间没有删除。于是，惰性删除派上用场。也就是说在你获取某个key的时候，redis会检查一下，这个key如果设置了过期时间那么是否过期了？如果过期了此时就会删除。采用定期删除+惰性删除就没其他问题了么?不是的，如果定期删除没删除key。然后你也没即时去请求key，也就是说惰性删除也没生效。这样，redis的内存会越来越高。那么就应该采用内存淘汰机制。

在redis.conf中有一行配置

# maxmemory-policy volatile-lru

该配置就是配内存淘汰策略的(什么，你没配过？好好反省一下自己)

1. noeviction：当内存不足以容纳新写入数据时，新写入操作会报错。应该没人用吧。
2. allkeys-lru：当内存不足以容纳新写入数据时，在键空间中，移除最近最少使用的key。推荐使用，目前项目在用这种。
3. allkeys-random：当内存不足以容纳新写入数据时，在键空间中，随机移除某个key。应该也没人用吧，你不删最少使用Key,去随机删。
4. volatile-lru：当内存不足以容纳新写入数据时，在设置了过期时间的键空间中，移除最近最少使用的key。这种情况一般是把redis既当缓存，又做持久化存储的时候才用。不推荐
5. volatile-random：当内存不足以容纳新写入数据时，在设置了过期时间的键空间中，随机移除某个key。依然不推荐
6. volatile-ttl：当内存不足以容纳新写入数据时，在设置了过期时间的键空间中，有更早过期时间的key优先移除。不推荐ps：如果没有设置 expire 的key, 不满足先决条件(prerequisites); 那么 volatile-lru, volatile-random 和 volatile-ttl 策略的行为, 和 noeviction(不删除) 基本上一致。
7. redis雪崩、穿透、击穿

**雪崩**：

缓存层承载了大量的请求，缓存层挂掉，所有的请求，就会到同时访问存储层，存储层的调用量就会暴增，造成存储层也会挂掉的情况，就叫雪崩。

**穿透**：

恶意访问缓存中不存在的数据，如id=-1，这样所有请求就会直接查询数据库，并发高的话，直接让数据库挂掉。

解决：每次只要查询库id=-1不存在就在缓存中设置一个控制 set -1 nuknown，下次恶意请求就会直接访问缓存。

**击穿**：

热点数据key失效的时候，造成访问该key的请求直接到库，造成数据库瘫痪。

解决办法：

1. 设置redis集群和DB集群的高可用，如果redis出现宕机情况，可以立即由别的机器顶替上来。这样可以防止一部分的风险。
2. 不同的key,可以设置不同的过期时间，让缓存失效的时间点不一致，尽量达到平均分布。
3. redis中设置永久不过期，这样就保证了，不会出现热点问题，也就是物理上不过期。
4. 在缓存失效后，通过加锁或者队列来控制读和写数据库的线程数量。比如：对某个key只允许一个线程查询数据和写缓存，其他线程等待。单机的话，可以使用synchronized或者lock来解决，如果是分布式环境，可以是用redis的setnx命令来解决。
5. Redis双写一致性解决

简单情况：先删除缓存，再更新数据库。如果数据库更新失败了，那么数据库中是旧数据，缓存中是空的，那么数据不会不一致。

复杂情况：先删除了缓存，要去修改数据库，还没修改。一个请求过来，去读缓存，发现缓存空了，去查询数据库，把旧数据放到了缓存中。随后数据变更的程序完成了数据库的修改。造成数据不一致。

解决方法：更新数据的时候，根据数据唯一标识，将操作发送到一个jvm内部队列

1. Redis的应用场景

Token的生成

短信验证码code

缓存热点数据

实现网页计数器

分布式锁

延迟操作（订单超时时间）