# Redis

## redis与memcached的区别

1. redis支持的数据结构更多，string、list、set、hash。memcached仅支持string
2. redis支持数据持久化，可以将内存中的数据保持在磁盘中。memcached放在内存里面。
3. Memcached是多线程，非阻塞IO复用的网络模型；Redis使用单线程的多路 IO 复用模型。

## redis设置过期时间的删除

如果redis的过期时间到了，采用定期删除和惰性删除的方式进行删除。

### 定期删除

redis默认是每隔 100ms 就随机抽取一些设置了过期时间的key，检查其是否过期，如果过期就删除。注意这里是随机抽取的。为什么要随机呢？你想一想假如 redis 存了几十万个 key ，每隔100ms就遍历所有的设置过期时间的 key 的话，就会给 CPU 带来很大的负载！

### 惰性删除

定期删除可能会导致很多过期 key 到了时间并没有被删除掉。所以就有了惰性删除。假如你的过期 key，靠定期删除没有被删除掉，还停留在内存里，除非你的系统去查一下那个 key，才会被redis给删除掉。这就是所谓的惰性删除，也是够懒的哈！

但是仅仅通过设置过期时间还是有问题的。我们想一下：如果定期删除漏掉了很多过期 key，然后你也没及时去查，也就没走惰性删除，此时会怎么样？如果大量过期key堆积在内存里，导致redis内存块耗尽了。怎么解决这个问题呢？ redis 内存淘汰机制。

## redis内存淘汰机制

redis在自己的配置文件中提供了6种淘汰机制

1. 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
2. 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
3. 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。
4. 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（这个是最常用的）
5. allkeys-random：从数据集（server.db[i].dict）中任意选择数据淘汰
6. no-eviction：禁止驱逐数据，也就是说当内存不足以容纳新写入数据时，新写入操作会报错。这个应该没人使用吧！
7. volatile-lfu：从已设置过期时间的数据集(server.db[i].expires)中挑选最不经常使用的数据淘汰
8. allkeys-lfu：当内存不足以容纳新写入数据时，在键空间中，移除最不经常使用的key。

## redis持久化机制(怎么保证 redis 挂掉之后再重启数据可以进行恢复)

redis支持两种不同的持久化策略，一个是快照、一个是追加文件

### 快照持久化(默认方式)

Redis可以通过创建快照来获得存储在内存里面的数据在某个时间点上的副本。Redis创建快照之后，可以对快照进行备份，可以将快照复制到其他服务器从而创建具有相同数据的服务器副本（Redis主从结构，主要用来提高Redis性能），还可以将快照留在原地以便重启服务器的时候使用。

快照持久化是Redis默认采用的持久化方式，在redis.conf配置文件中默认有此下配置：

1. save 900 1 #在900秒(15分钟)之后，如果至少有1个key发生变化，Redis就会自动触发BGSAVE命令创建快照。
2. save 300 10 #在300秒(5分钟)之后，如果至少有10个key发生变化，Redis就会自动触发BGSAVE命令创建快照。
3. save 60 10000 #在60秒(1分钟)之后，如果至少有10000个key发生变化，Redis就会自动触发BGSAVE命令创建快照。

### 追加文件(AOF)持久化，主流的方案

开启AOF持久化后每执行一条会更改Redis中的数据的命令，Redis就会将该命令写入硬盘中的AOF文件。AOF文件的保存位置和RDB文件的位置相同，都是通过dir参数设置的，默认的文件名是appendonly.aof。

1. appendfsync always #每次有数据修改发生时都会写入AOF文件,这样会严重降低Redis的速度

2. appendfsync everysec #每秒钟同步一次，显示地将多个写命令同步到硬盘，推荐，就算是丢失，也只是丢失了一秒的数据

3. appendfsync no #让操作系统决定何时进行同步

## redis事务

事务提供了一种将多个命令请求打包，然后一次性、按顺序地执行多个命令的机制，并且在事务执行期间，服务器不会中断事务而改去执行其他客户端的命令请求，它会将事务中的所有命令都执行完毕，然后才去处理其他客户端的命令请求。

Redis命令在事务中可能会执行失败，但是Redis事务不会回滚，而是继续会执行余下的命令

这是因为：

1. 只有当发生语法错误(这个问题在命令队列时无法检测到)了，Redis命令才会执行失败, 或对keys赋予了一个类型错误的数据：这意味着这些都是程序性错误，这类错误在开发的过程中就能够发现并解决掉，几乎不会出现在生产环境。
2. 由于不需要回滚，这使得Redis内部更加简单，而且运行速度更快。

## 缓存雪崩和缓存穿透问题解决方案

### 缓存雪崩

1. 什么是缓存雪崩

缓存同一时间大面积的失效，所以，后面的请求都会落到数据库上，造成数据库短时间内承受大量请求而崩掉。

1. 解决方法

事前：尽量保证整个 redis 集群的高可用性，发现机器宕机尽快补上。选择合适的内存淘汰策略。

事中：本地ehcache缓存 + hystrix限流&降级，避免MySQL崩掉。

事后：利用 redis 持久化机制保存的数据尽快恢复缓存

### 缓存穿透

1. 什么是缓存穿透

缓存穿透说简单点就是大量请求的 key 根本不存在于缓存中，导致请求直接到了数据库上，根本没有经过缓存这一层

1. 解决办法
2. 最基本的就是添加校验条件，不满足key格式的请求直接返回，不去请求缓存。
3. 设置缓存无效key，如果缓存和数据中都查不出一个key，就将这个key写入到redis中并且设置一个过期时间，这样做可以一定程度上缓解缓存穿透，但不能避免。
4. 布隆过滤器，把所有的请求key放到布隆过滤器中，当用户请求过来，我会先判断用户发来的请求的值是否存在于布隆过滤器中。不存在的话，直接返回请求参数错误信息给客户端，存在的话才会走下面的流程。

## 如何解决 Redis 的并发竞争 Key 问题

利用redis分布式锁

## 如何保证缓存与数据库双写时的数据一致性

读的时候，先读缓存，缓存没有的话，就读数据库，然后取出数据后放入缓存，同时返回响应。

更新的时候，先更新数据库，然后再删除缓存。

这种情况下一定存在缓存不一致问题。

一般来说，如果允许缓存可以稍微的跟数据库偶尔有不一致的情况，也就是说如果你的系统不是严格要求 “缓存+数据库” 必须保持一致性的话，最好不要做这个方案，即：读请求和写请求串行化，串到一个内存队列里去。

串行化可以保证一定不会出现不一致的情况，但是它也会导致系统的吞吐量大幅度降低，用比正常情况下多几倍的机器去支撑线上的一个请求。

## redis分布式锁

redis单节点分布式锁，通过一个key去set值，当值存在的话，就说明有在使用。使用完了再删除掉。

redis多节点的分布式锁，但使用的是redisson的trylock，使用的是redlock算法：

1. 尝试顺序的在n个实例上申请锁，当大于等于n/2个master上申请到锁的时候，如果锁的过期时间大于获取锁的时间，那么这个锁就申请成功了。
2. 锁申请到了之后，锁真正的过期时间应该是锁的过期时间-获取锁的时间。
3. 如果锁申请失败了，那么它就会在少部分申请成功锁的 master 节点上执行释放锁的操作，重置状态。

### redis是单线程还是多线程

redis一种使用单线程模型的，采用kv键值对在内存里面存储的nosql模式的存储系统。

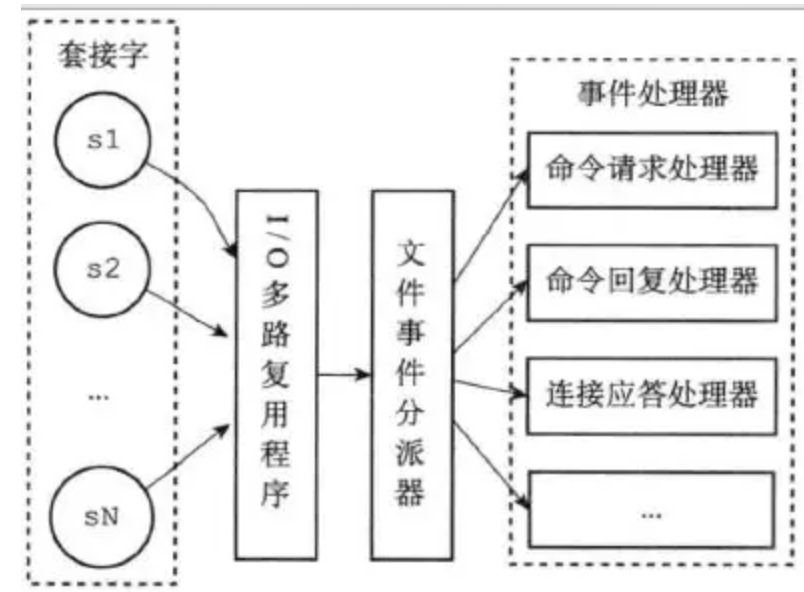
单线程模型的理解：

redis处理数据的模块可以分为两个部分：一个是io多路复用模块，一个是事件处理模块。这两个模块都是单线程处理的(但是处理的线程不是同一个)。

IO多路复用模块：

redis通过io多路复用模块同时监听多个socket连接，并根据socket中数据关联的不同的操作来关联不同的事件处理器。

其中I/O多路复用程序通过队列向文件事件分派器传送socket



### redis为什么这么快

1. redis的操作基于内存
2. 数据结构简单
3. 采用单线程，避免了不必要的上下文切换和竞争条件，也不存在多线程带来的CPU消耗，不用去考虑锁的问题。
4. 使用了io多路复用模型，非阻塞的io。