# 缓存不一致性

## 引用缓存的好处

   1）减库压力；2）提并发

## 引用缓存的问题

        1）处理逻辑变得复杂；2）缓存和数据库数据不一致

## 数据不一致的原因

      缓存与数据库操作不是原子操作，一方操作成功，另一方操作失败，造成数据不一致

## 更新策略

* 1. 先删缓存，再更库（常用）
  2. 先更库，再删缓存（常用）

### 4.0先更库，再更缓存（选择）

### 4.1先删缓存，再更库

（1）请求A删缓存

（2）请求B发现缓存不存在，去库查询旧值将旧值写入缓存

（3）请求A将新值写库

### 4.1.2缓存设过期时间

问题：时间不好把控，频繁请求数据库，出现缓存雪崩,不用

### 4.1.1延时双删

（1）删缓存

（2）写库

（3）休眠，再删缓存

问题：时间不好把控，吞吐量降低，业务侵入，不用

### 4.1.2延时异步双删重试

第二次删除失败，进行重试

不需休眠再删缓存。加大吞吐量

### 4.1.3缓存更新与读取操作进行串行化（不用）

**1. 更新数据时，根据数据的唯一标识将请求路由到一个jvm队列中，去更新数据库,**然后请求结束**。  
　　2. 读取数据的时候，先查缓存，发现数据不在缓存中，根据唯一标识路由之后发送同一个jvm内部的队列中，重新读取数据库后更新缓存,**最后请求结束**。**

缺点：极端情况很少，系统复杂度和开销增大

优化：如果发现**队列中已经有一个更新缓存的请求了，那么就不用再放个更新请求操作进去了，直接让后面的读请求阻塞个200ms左右(实际值可以根据服务的响应时间和机器的处理能力来计算)，然后再次查询缓存，如果缓存没有值就查数据库，拿到结果后不用更新缓存，直接返回给页面**。

问题：高并发的场景下

1、**读请求长时阻塞**

**可能数据更新很频繁，导致队列中积压了大量更新操作在里面，然后读请求会发生大量的超时，最后导致大量的请求直接走数据库。务必通过一些模拟真实的测试，看看更新数据的频率是怎样的。**

**另外一点，因为一个队列中，可能会积压针对多个数据项的更新操作，因此需要根据自己的业务情况进行测试，可能需要部署多个服务，每个服务分摊一些数据的更新操作。如果一个内存队列里居然会挤压 100 个商品的库存修改操作，每隔库存修改操作要耗费 10ms 去完成，那么最后一个商品的读请求，可能等待 10 \* 100 = 1000ms = 1s 后，才能得到数据，这个时候就导致读请求的长时阻塞。**

一定要做根据实际业务系统的运行情况，去进行一些压力测试，和模拟线上环境，去看看最繁忙的时候，内存队列可能会挤压多少更新操作，可能会导致最后一个更新操作对应的读请求，会 hang 多少时间，如果读请求在 200ms 返回，如果你计算过后，哪怕是最繁忙的时候，积压 10 个更新操作，最多等待 200ms，那还可以的。

如果**一个内存队列中可能积压的更新操作特别多，那么你就要加机器，让每个机器上部署的服务实例处理更少的数据，那么每个内存队列中积压的更新操作就会越少**。

其实根据之前的项目经验，一般来说，数据的写频率是很低的，因此实际上正常来说，在队列中积压的更新操作应该是很少的。像这种针对**读高并发、读缓存架构的项目，一般来说写请求是非常少的，每秒的 QPS 能到几百就不错了**。

实际粗略测算一下

如果一秒有 500 的写操作，如果分成 5 个时间片，每 200ms 就 100 个写操作，放到 20 个内存队列中，每个内存队列，可能就积压 5 个写操作。每个写操作性能测试后，一般是在 20ms 左右就完成，那么针对每个内存队列的数据的读请求，也就最多 hang 一会儿，200ms 以内肯定能返回了。

经过刚才简单的测算，我们知道，单机支撑的写 QPS 在几百是没问题的，如果写 QPS 扩大了 10 倍，那么就扩容机器，扩容 10 倍的机器，每个机器 20 个队列。

2、**读请求并发量过高**

做好压力测试，确保恰巧碰上上述情况的时候，还有一个风险，就是突然间大量读请求会在几十毫秒的延时 hang 在服务上，看服务能不能扛的住，需要多少机器才能扛住最大的极限情况的峰值。

但是因为并不是所有的数据都在同一时间更新，缓存也不会同一时间失效，所以每次可能也就是少数数据的缓存失效了，然后那些数据对应的读请求过来，并发量应该也不会特别大。

3、**多服务实例部署的请求路由**

可能这个服务部署了多个实例，那么必须保证说，执行数据更新操作，以及执行缓存更新操作的请求，都通过 Nginx 服务器路由到相同的服务实例上。

比如说，**对同一个商品的读写请求，全部路由到同一台机器上。可以自己去做服务间的按照某个请求参数的 hash 路由，也可以用 Nginx 的 hash 路由功能**。

4、**热点商品的路由问题，导致请求的倾斜**

万一某个商品的读写请求特别高，全部打到相同的机器的相同的队列里面去了，可能会造成某台机器的压力过大。就是说，因为只有在商品数据更新的时候才会清空缓存，然后才会导致读写并发，所以其实要根据业务系统去看，**如果更新频率不是太高的话，这个问题的影响并不是特别大**，但是的确可能某些机器的负载会高一些。

### 4.1.5分布式锁

缺点:并发低，和用缓存的目的矛盾，不用

### 4.2先更库，再删缓存

（1）缓存失效，请求A查库得旧值

（2）请求B将新值写库，删除缓存

（3）请求A将查到的旧值写入缓存

问题：很难出现读操作快于写操作。

### 4.2.1缓存设过期时间

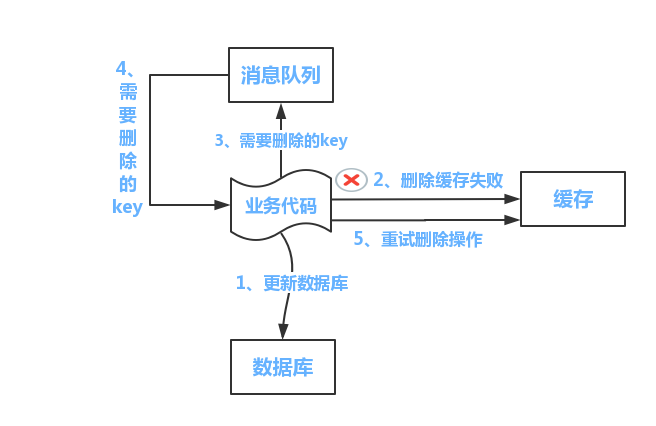
问题：时间不好把控，频繁请求数据库，出现缓存雪崩等问题,不用

### 4.2.2定期清缓存

问题：时间越短，数据一致性越高。但是时间越短，查数据库就会越频繁。因此时间应该根据业务也不好定，不用。

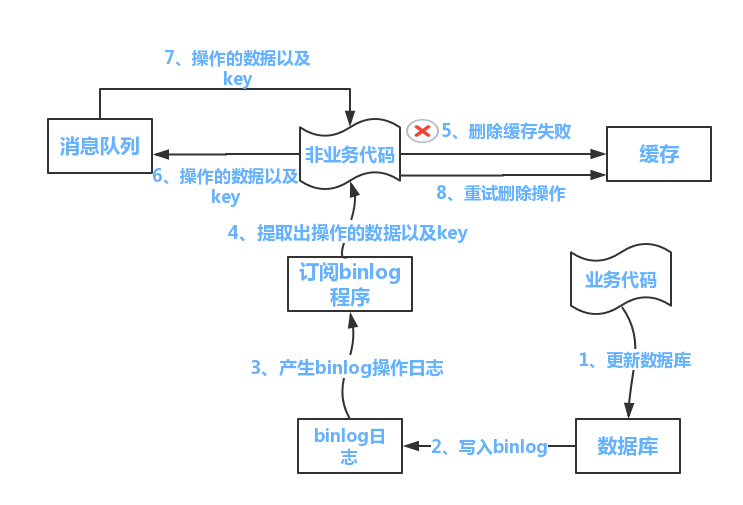
### 4.2.3异步延时删除,保证读请求完成以后，再进行删除

### 4.2.4异步延时删除重试（可用）



缺点：代码侵入

### 4.2.5基于订阅binlog的同步机制实现重试（常用）



（1）更新库，写入binlog日志，订阅程序提取出所需要的数据以及key，另起一段非业务代码，获得该信息

（5）删除缓存，删除失败

（6）将这些信息发送至消息队列

（7）重新从消息队列中获得该数据，重试删除。