# 商户查询缓存

我们利用商户查询业务了解分析一下在企业中Redis缓存使用上的技巧，还有常见问题的解决方案。

## 什么是缓存：

缓存就是数据交换的缓冲区，称为Cache，是存储数据的临时地方，一般读写性能比较高！

比较常见的例子就是我们的计算机，我们知道计算机主要构成就是：CPU、内存、磁盘，CPU的运算能力发展到现在已经很恐怖了，运算能力已经远远超过了内存和磁盘读写数据的能力，但是CPU所造的任何的运算都需要先从内存或者磁盘里读到数据，放到自己的寄存器才可以做运算，正是因为内存和磁盘的读写能力远远低于CPU运算能力，所以计算机性能受到了一些限制，所以为了解决这个问题，人们就在CPU的内部添加了一个缓存：CPU告诉缓存，即CPU会把经常需要读写的一些数据放到CPU的缓存中去，当我们去做高速运算的时候就不必要每次等半天，从内存或磁盘里把数据读过来再运算，而是直接从CPU高速缓存里拿到数据进行运算，这样一来，可以充分地让CPU的运算能力得到释放！所以衡量一个CPU性能是否强大的一个重要标志就是CPU缓存的大小！

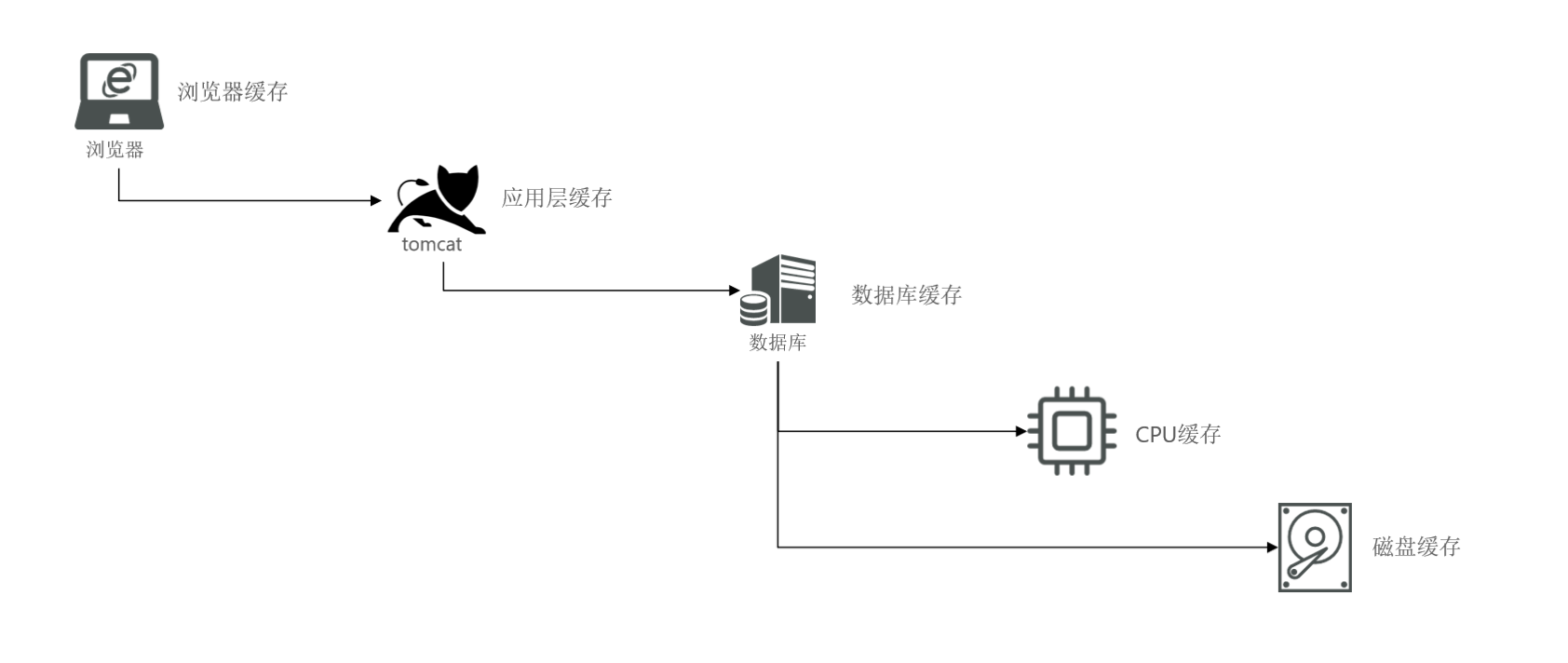
## Web应用每阶段的缓存举例

在我们的Web应用开发的过程中也离不开缓存，比如说一个Web应用，用户通过浏览器向Web应用发起请求，这个时候浏览器首先就可以建立缓存，浏览器能缓存的东西可多了，比如页面静态资源，我们访问一个页面，页面有很多的CSS，JS和图片，这些东西一般都是不变的，浏览器就可以把这些东西缓存在本地，这样就无需每次访问都要去加载这些数据了，大大降低网络延迟，提高页面响应的速度！

Tomcat中的Web应用可以添加应用层缓存，比如我们去创建一个Map，我们把数据库中查询到的数据放到这个Map中，以后再来的时候我直接从Map中读给你，这样一来不就减少了数据库查询，效率也就提升了，这就是应用层缓存的一个例子。

一般来说我们不会用map做应用层缓存，我们可以利用Redis，因为Redis本身读写效率非常强大！所以用Redis做应用层缓存再合适不过了！

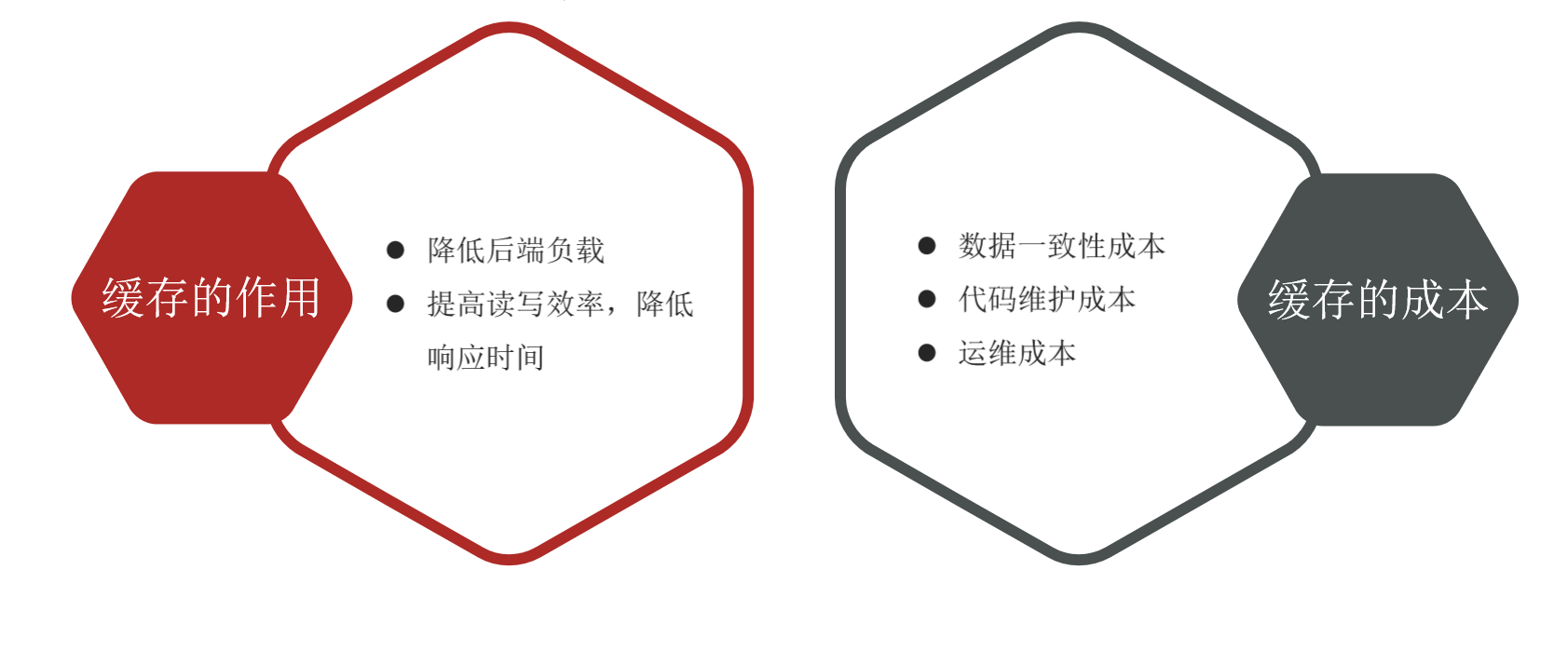
应用层缓存未命中的情况下，请求依然还会落到数据库，数据库层面依然可以添加缓存。数据库缓存什么呢？比如索引，MySQL数据库是一个聚簇索引，会对ID创建索引，这些索引数据就会缓存起来，当我们根据索引进行查询的时候，就可以在内存里快速检索得到结果，而不用每次都要去读写磁盘，效率就会大大提升，这就是数据库层面的缓存。当然最终数据去查找还是要落到磁盘，做一些复杂的排序、表关联还会用CPU去做运算，所以最终我们的数据库还会去访问CPU和磁盘，这个时候自然就会用到我们之前提到的CPU多级缓存和磁盘读写缓存。



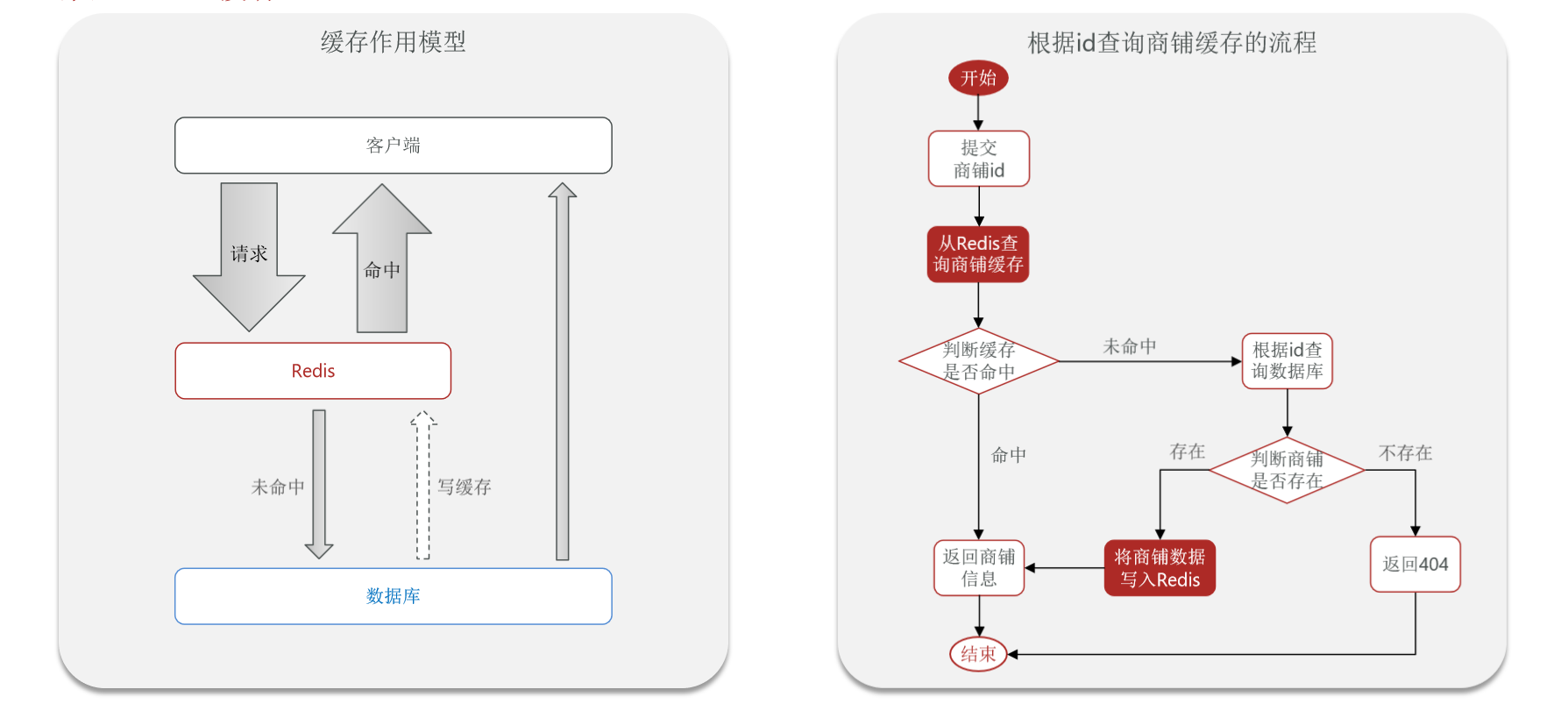
综上，在整个web开发的每一个阶段都可以添加缓存！即缓存的应用场景是非常丰富的！

## 缓存的作用和成本

但是缓存也不能乱用，任何东西都是一个双刃剑，有得必有失，用缓存给你带来好处的时候，也会带来一些成本！见下图：



## 查询商户功能添加Redis缓存



## 缓存更新策略

上面我们已经给商户查询的业务添加了缓存，大大降低了对于数据库的负载压力，不过就像我们上面说的那样，缓存是一把双刃剑，缓存在带来好处的时候，同时也引入了一些问题！

首先就是数据一致性问题，因为我们把数据同时保存在缓存和数据库中，如果我们对数据库数据做了修改，缓存并不知道也不会去更新，于是用户来查询的时候，查到的就是缓存中的旧数据，与数据库不一致，在很多业务场景下是不允许的。所以现在来学习一下缓存的更新策略，正是为了解决数据库数据和缓存数据不一致的问题！



### 内存淘汰：

不用自己维护，利用Redis的内存淘汰机制，当内存不足的时候自动淘汰部分数据，下次查询时候更新缓存。

这种机制其实原本是Redis用来解决内存不足的问题，因为我们知道Redis是基于内存存储的，内存不同于磁盘是，有限的，比较珍贵的，所以往往Redis内存会设置一个上限，如果说存储数据越来越多，终究会导致内存的不足，于是Redis里面就会有一个内存淘汰机制，这个机制我们可以自己去配置，有好几种不同的策略，，现在先按住不表，后面讲Redis原理的时候再来详细学习！不管什么策略，当内存不足的时候，就会触发这种策略，然后把一部分数据把他淘汰掉！这样一来就可以清空一部分内存空间。这种机制默认就开启了的，不用我们管。

内存淘汰在一定程度上可以保证数据一致性，你想当内存不足时把一部份数据淘汰了，这部分数据在Redis就没有了，这个时候如果用户来查询这部分数据，按照我们以前的写的业务逻辑，如果在Redis未命中，就查询数据库，然后把数据库数据写入Redis。同学们，这不就能做到数据一致了吗！但是这种一致性首先不是我们能控制的，内存淘汰的时候淘汰的是哪一部分数据？什么时候淘汰？我们都不知道！可能内存都很充足，很有可能很长一段时间数据都不会淘汰，这样一来，每次来查询这些不被淘汰的数据，都是查到的旧的数据！于是就语法保证一致性了，所以说内存淘汰只能在一定程度保证数据一致性。

### 超时剔除：

利用Redis的expire名利给缓存数据添加超时时间TTL，到期后会自动删除缓存数据，下次查询时更新缓存。

超时剔除保证数据一致性的强弱取决于设置TTL超时时间的长短！当然不能完全保证一致性，如果在超时时间内数据库数据发生了变化，此时Redis数据和数据库数据还是不一致！

### 主动更新：

编写业务逻辑，在修改数据库的同时，更新缓存。

### 缓存更新策略选择：

#### 低一致性场景的需求，可以采用内存淘汰机制，例如上个提交中的商铺类型的查询缓存。

#### 高一致性场景的需求：主动更新，并以超时剔除作为兜底方案。例如上个提交中的商铺详情的查询缓存

## 主动更新策略详解：



### 1：先操作数据库，再操作redis缓存；

### 2：删除Redis缓存，而不是修改Redis缓存。

### 问题1：删除缓存还是更新缓存？答：删除缓存！

更新缓存：每次更新数据库都更新缓存，无效写操作太多，比如上百次更新数据库，但是期间没有查询请求。

删除缓存：更新数据库时让缓存失效，查询时再更新缓存。

### 问题2：如何保证缓存与数据库的操作的同时成功或失败？

单体系统：将缓存和数据库操作放在同一个事务，因为缓存操作和数据库操作在用一个项目中，甚至是一个方法中！所以我们可以把缓存和数据库操作放在同一个事务。利用事务本身的特性就能保证同时成功和失败！

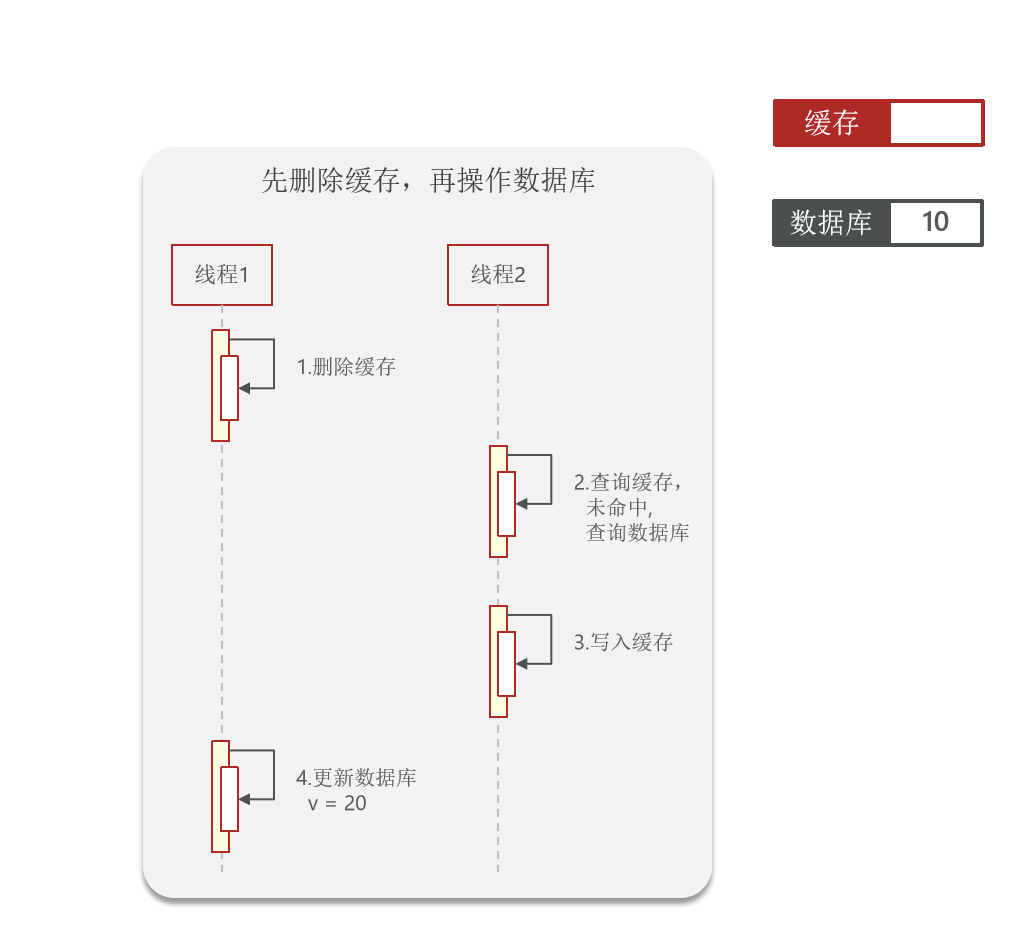
分布式系统：利用TCC等分布式事务方案。分布式系统我们的缓存操作和数据库操作很有可能实在不同服务中，就不得不用到分布式事务方案了。

### 问题3：先操作缓存还是先操作数据库？

以上两步明确之后，还要考虑线程安全的问题！我们在操作的时候因为有数据库操作和缓存操作这两个操作，那么这个时候在多线程并发的情况下，这两个操作之间可能会有多个线程同时执行，这个时候数据库操作和缓存操作的先后顺序就会带来不一样的线程安全问题！究竟是先删缓存在操作数据库呢？还是先操作数据库再删缓存呢？

#### 先删除缓存，再操作数据库：

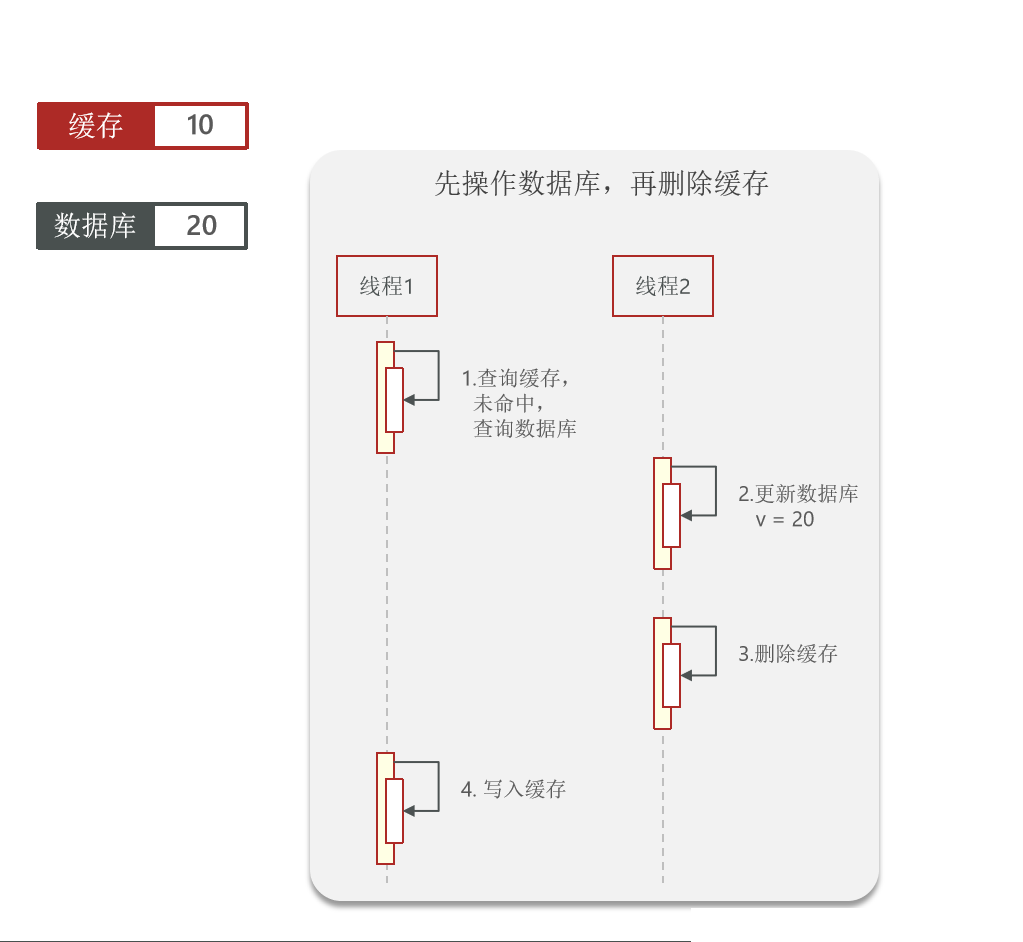
 特殊情况：线程1删除缓存，还未操作数据库，查询操作的线程2进来，查询缓存未命中，从数据库中查询并写入缓存，此时写入缓存的是旧数据，线程2结束，之后线程1再操作数据库，那么此时数据库与Redis的数据不一致！



此种特殊情况发生的概率是比较大的，因为线程1的写数据库操作是比较耗时的，而线程2的查询数据库操作是比较快的。

#### 先操作数据库，再删除缓存：

 特殊情况：当缓存失效了，查询操作的线程1进来，则缓存未命中，数据库中查询，查到后还未写入缓存，此时线程2进来更新数据库，然后删除缓存（此时缓存本来就没有），线程2结束，线程1继续执行写缓存操作，此时写入缓存的是旧数据，那么此时数据库与Redis的数据不一致！



此种特殊情况发生的概率是比较小的，因为需要线程1查询缓存时缓存恰好失效，线程1已经查完数据库了，余下的写入缓存的时间是很快的，此时还能让线程2完成更新数据库与删除缓存的操作，等线程2结束，线程1才继续写入缓存操作，概率太小了。

所以先操作数据库，再删除缓存，就算出现特殊情况，还有超时时间兜底，超时时间到了旧数据一样会删除。

### 缓存更新策略的最佳实践方案：

1：低一致性场景的需求，使用Redis自带的内存淘汰机制。

2：高一致性场景的需求：主动更新，并以超时剔除作为兜底方案。

2.1：读操作：

缓存命中则直接返回；

缓存未命中则查询数据库，并写入缓存，设定超时时间

2.2：写操作：

先写数据库，然后再删除缓存；

要确保数据库与缓存操作的原子性

## 给查询商铺的缓存添加超时剔除和主动更新策略