

杨浪

一个要成为架构师的男人

博客园 首页 新随笔 联系 订阅 管理

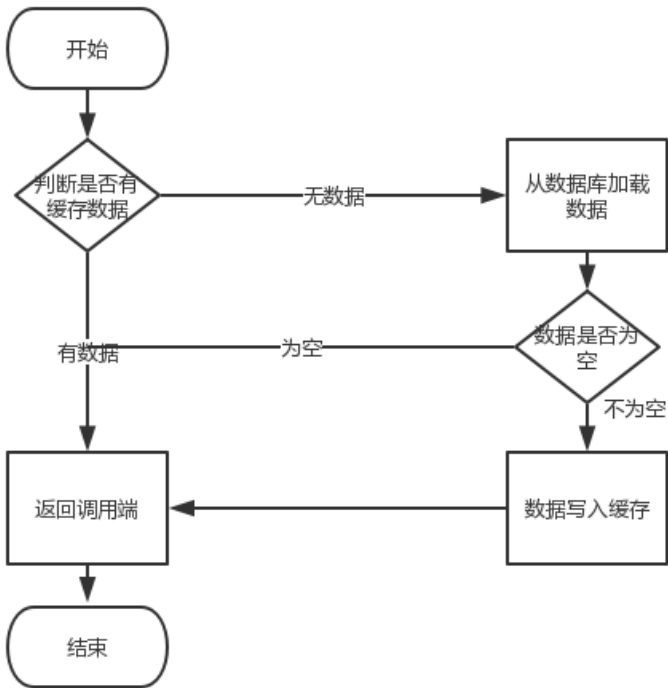
随笔 - 680 文章 - 163 评论 - 81

# 分布式之数据库和缓存双写一致性方案解析

## 引言

### 为什么写这篇文章？

首先，缓存由于其高并发和高性能的特性，已经在项目中被广泛使用。在读取缓存方面，大家没啥疑问，都是按照下图的流程来进行业务操作。



但是在更新缓存方面，对于更新完数据库，是更新缓存呢，还是删除缓存。又或者是先删除缓存，再更新数据库，其实大家存在很大的争议。目前没有一篇全面的博客，对这几种方案进行解析。于是博主战战兢兢，顶着被大家喷的风险，写了这篇文章。

### 文章结构

本文由以下三个部分组成

- 1、讲解缓存更新策略
- 2、对每种策略进行缺点分析
- 3、针对缺点给出改进方案

## 正文

先做一个说明，从理论上来说，给缓存设置过期时间，是保证最终一致性的解决方案。这种方案下，我们可以对存入缓存的数据设置过期时间，所有的写操作以数据库为准，对缓存操作只是尽最大努力即可。也就是说如果数据库写成功，缓存更新失败，那么只要到达过期时间，则后面的读请求自然会从数据库中读取新值然后回填缓存。因此，接下来讨论的思路不依赖于给缓存设置过期时间这个方案。在这里，我们讨论**三种**更新策略：

- 1. 先更新数据库，再更新缓存

2. 先删除缓存，再更新数据库
3. 先更新数据库，再删除缓存

应该没人问我，为什么没有先更新缓存，再更新数据库这种策略。

### (1)先更新数据库，再更新缓存

这套方案，大家是普遍反对的。为什么呢？有如下两点原因。

#### 原因一（线程安全角度）

同时有请求A和请求B进行更新操作，那么会出现

- (1) 线程A更新了数据库
- (2) 线程B更新了数据库
- (3) 线程B更新了缓存
- (4) 线程A更新了缓存

这就出现请求A更新缓存应该比请求B更新缓存早才对，但是因为网络等原因，B却比A更早更新了缓存。这就导致了脏数据，因此不考虑。

#### 原因二（业务场景角度）

有如下两点：

- (1) 如果你是一个写数据库场景比较多，而读数据场景比较少的业务需求，采用这种方案就会导致，数据压根还没读到，缓存就被频繁的更新，浪费性能。
- (2) 如果你写入数据库的值，并不是直接写入缓存的，而是要经过一系列复杂的计算再写入缓存。那么，每次写入数据库后，都再次计算写入缓存的值，无疑是浪费性能的。显然，删除缓存更为适合。

接下来讨论的就是争议最大的，先删缓存，再更新数据库。还是先更新数据库，再删缓存的问题。

### (2)先删缓存，再更新数据库

该方案会导致不一致的原因是。同时有一个请求A进行更新操作，另一个请求B进行查询操作。那么会出现如下情形：

- (1) 请求A进行写操作，删除缓存
- (2) 请求B查询发现缓存不存在
- (3) 请求B去数据库查询得到旧值
- (4) 请求B将旧值写入缓存
- (5) 请求A将新值写入数据库

上述情况就会导致不一致的情形出现。而且，如果不采用给缓存设置过期时间策略，该数据永远都是脏数据。

那么，**如何解决呢？**采用延时双删策略

伪代码如下

```
public void write(String key, Object data) {  
    redis.delKey(key);  
    db.updateData(data);  
    Thread.sleep(1000);  
    redis.delKey(key);  
}
```

转化为中文描述就是

- (1) 先淘汰缓存
- (2) 再写数据库（这两步和原来一样）
- (3) 休眠1秒，再次淘汰缓存

这么做，可以将1秒内所造成的缓存脏数据，再次删除。

**那么，这个1秒怎么确定的，具体该休眠多久呢？**

针对上面的情形，读者应该自行评估自己的项目的读数据业务逻辑的耗时。然后写数据的休眠时间则在读数据业务逻辑的耗时基础上，加几百ms即可。这么做的目的，就是确保读请求结束，写请求可以删除读请求造成的缓存脏数据。

**如果你用了mysql的读写分离架构怎么办？**

ok，在这种情况下，造成数据不一致的原因如下，还是两个请求，一个请求A进行更新操作，另一个请求B进行查询操作。

- (1) 请求A进行写操作，删除缓存
- (2) 请求A将数据写入数据库了，
- (3) 请求B查询缓存发现，缓存没有值
- (4) 请求B去从库查询，这时，还没有完成主从同步，因此查询到的是旧值

- (5) 请求B将旧值写入缓存
- (6) 数据库完成主从同步，从库变为新值

上述情形，就是数据不一致的原因。还是使用双删延时策略。只是，睡眠时间修改为在主从同步的延时时间基础上，加几百ms。

### 采用这种同步淘汰策略，吞吐量降低怎么办？

ok，那就将第二次删除作为异步的。自己起一个线程，异步删除。这样，写的请求就不用沉睡一段时间了，再返回。这么做，加大吞吐量。

### 第二次删除,如果删除失败怎么办？

这是个非常好的问题，因为第二次删除失败，就会出现如下情形。还是有两个请求，一个请求A进行更新操作，另一个请求B进行查询操作，为了方便，假设是单库：

- (1) 请求A进行写操作，删除缓存
- (2) 请求B查询发现缓存不存在
- (3) 请求B去数据库查询得到旧值
- (4) 请求B将旧值写入缓存
- (5) 请求A将新值写入数据库
- (6) 请求A试图去删除请求B写入对缓存值，结果失败了。

ok,这也就是说。如果第二次删除缓存失败，会再次出现缓存和数据库不一致的问题。

### 如何解决呢？

具体解决方案，且看博主对第(3)种更新策略的解析。

### (3)先更新数据库，再删缓存

首先，先说一下。老外提出了一个缓存更新套路，名为《Cache-Aside pattern》。其中就指出

- **失效**：应用程序先从cache取数据，没有得到，则从数据库中取数据，成功后，放到缓存中。
- **命中**：应用程序从cache中取数据，取到后返回。
- **更新**：先把数据存到数据库中，成功后，再让缓存失效。

另外，知名社交网站facebook也在论文《Scaling Memcache at Facebook》中提出，他们用的也是先更新数据库，再删缓存的策略。

### 这种情况不存在并发问题么？

不是的。假设这会有两个请求，一个请求A做查询操作，一个请求B做更新操作，那么会有如下情形产生

- (1) 缓存刚好失效
- (2) 请求A查询数据库，得一个旧值
- (3) 请求B将新值写入数据库
- (4) 请求B删除缓存
- (5) 请求A将查到的旧值写入缓存

ok，如果发生上述情况，确实是会发生脏数据。

### 然而，发生这种情况的概率又有多少呢？

发生上述情况有一个先天性条件，就是步骤（3）的写数据库操作比步骤（2）的读数据库操作耗时更短，才有可能使得步骤（4）先于步骤（5）。可是，大家想想，数据库的读操作的速度远快于写操作的（不然做读写分离干嘛，做读写分离的意义就是因为读操作比较快，耗资源少），因此步骤（3）耗时比步骤（2）更短，这一情形很难出现。

假设，有人非要抬杠，有强迫症，一定要解决怎么办？

### 如何解决上述并发问题？

首先，给缓存设有效时间是一种方案。其次，采用策略（2）里给出的异步延时删除策略，保证读请求完成以后，再进行删除操作。

### 还有其他造成不一致的原因么？

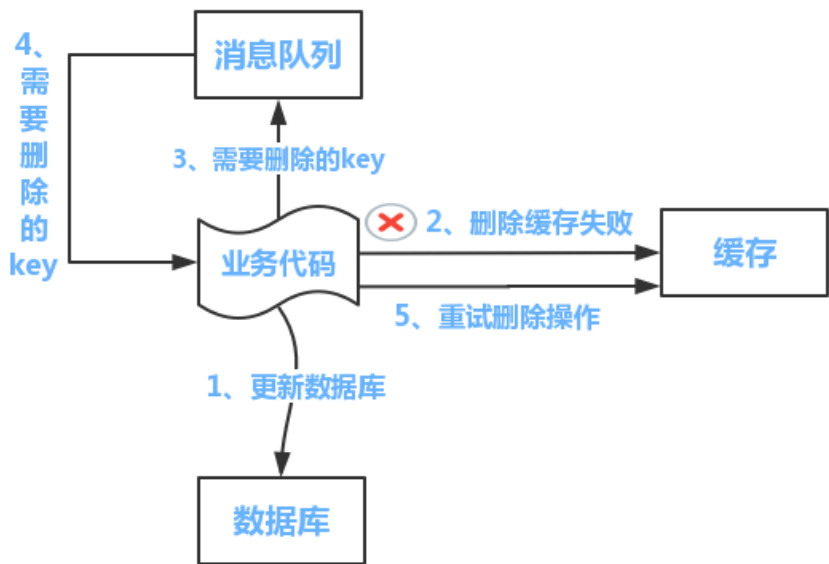
有的，这也是缓存更新策略（2）和缓存更新策略（3）都存在的一个问题，如果删缓存失败了怎么办，那不是会有不一致的情况出现么。比如一个写数据请求，然后写入数据库了，删缓存失败了，这会就出现不一致的情况了。这也是缓存更新策略（2）里留下的最后一个疑问。

### 如何解决？

提供一个保障的重试机制即可，这里给出两套方案。

#### 方案一：

如下图所示

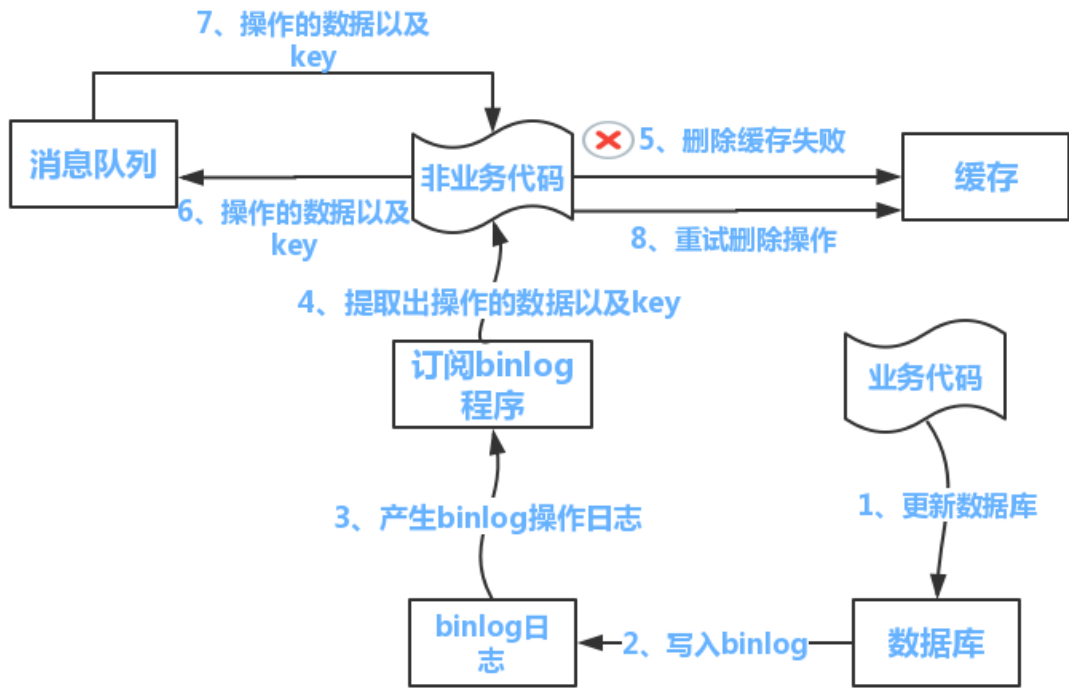


流程如下所示

- (1) 更新数据库数据;
- (2) 缓存因为种种问题删除失败
- (3) 将需要删除的key发送至消息队列
- (4) 自己消费消息，获得需要删除的key
- (5) 继续重试删除操作，直到成功

然而，该方案有一个缺点，对业务线代码造成大量的侵入。于是有了方案二，在方案二中，启动一个订阅程序去订阅数据库的binlog，获得需要操作的数据。在应用程序中，另起一段程序，获得这个订阅程序传来的信息，进行删除缓存操作。

方案二：



流程如下图所示：

- (1) 更新数据库数据

- (2) 数据库会将操作信息写入binlog日志当中
- (3) 订阅程序提取出所需要的数据以及key
- (4) 另起一段非业务代码，获得该信息
- (5) 尝试删除缓存操作，发现删除失败
- (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，重试操作。

**备注说明：**上述的**订阅binlog程序**在mysql中有现成的中间件叫canal，可以完成订阅binlog日志的功能。至于oracle中，博主目前不知道有没有现成中间件可以使用。另外，重试机制，博主采用的是消息队列的方式。如果对一致性要求不是很高，直接在程序中另起一个线程，每隔一段时间去重试即可，这些大家可以灵活自由发挥，只是提供一个思路。

## 总结

本文其实是对目前互联网中已有的一致性方案，进行了一个总结。对于先删缓存，再更新数据库的更新策略，还有方案提出维护一个内存队列的方式，博主看了一下，觉得实现异常复杂，没有必要，因此没有必要在文中给出。最后，希望大家有所收获。

## 参考文献

- 1、[主从DB与cache一致性](#)
- 2、[缓存更新的套路](#)

分类: [.Net](#), [数据库](#)

好文要顶

关注我

收藏该文

杨浪

关注 - 48

粉丝 - 132

1

0

[+加关注](#)

« 上一篇: [分布式之redis复习精讲](#)  
» 下一篇: [用ASP.NET Core 2.0 建立规范的 REST API -- DELETE, UPDATE, PATCH 和 Log](#)

posted @ 2018-05-28 10:00 [杨浪](#) 阅读(4642) 评论(0) [编辑](#) [收藏](#)

[刷新评论](#) [刷新页面](#) [返回顶部](#)

### 发表评论

编辑

预览

B

支持 Markdown

提交评论 退出 订阅评论 我的博客

[Ctrl+Enter]快捷键提交

- 相关博文：
- [ArrayList实现原理（JDK1.8）](#)
  - [十分钟掌握Pandas\(上\)——来自官网API](#)
  - [Java并发之synchronized关键字和Lock接口](#)
  - [Aso.NetCore的配置系统Configuration](#)
  - [kalilinux修改更新源和更新命令](#)
- » [更多推荐...](#)

公告

昵称：[杨浪](#)  
园龄：[5年4个月](#)  
粉丝：[132](#)  
关注：[48](#)  
[+加关注](#)

<	2021年1月						>
日	一	二	三	四	五	六	
27	28	29	30	31	1	2	
3	4	5	6	<a href="#">7</a>	<a href="#">8</a>	9	
10	11	12	13	14	15	16	
17	18	19	20	21	22	23	
24	25	26	27	28	29	30	
31	1	2	3	4	5	6	

搜索

常用链接

[我的随笔](#)  
[我的评论](#)  
[我的参与](#)  
[最新评论](#)  
[我的标签](#)

我的标签

[Quartz.Net\(3\)](#)  
[Redis\(2\)](#)  
[MySql\(2\)](#)  
[Python\(1\)](#)  
[Windows Terminal\(1\)](#)  
[容器化\(1\)](#)  
[设计模式\(1\)](#)

随笔分类

[.Net\(173\)](#) [.net core\(14\)](#) [ABP\(3\)](#)  
[ADO.NET\(6\)](#) [Asp.Net Core\(25\)](#)  
[AutoMapper\(8\)](#) [Django\(3\)](#) [docker\(3\)](#)  
[EF\(21\)](#) [EF Core\(4\)](#) [Fiddler\(2\)](#)  
[Javascript\(13\)](#) [JQuery\(9\)](#) [Json\(3\)](#)  
[Linux\(33\)](#) [更多](#)

随笔档案

[2021年1月\(2\)](#)  
[2020年12月\(1\)](#)  
[2020年10月\(1\)](#)  
[2020年9月\(2\)](#)  
[2020年8月\(4\)](#)  
[2020年7月\(2\)](#)  
[2020年6月\(12\)](#)  
[2020年3月\(3\)](#)  
[2020年2月\(3\)](#)  
[2020年1月\(4\)](#)  
[2019年12月\(16\)](#)  
[2019年11月\(26\)](#)  
[2019年10月\(16\)](#)  
[2019年9月\(3\)](#)  
[2019年8月\(3\)](#)  
[更多](#)

文章分类

[.Net\(51\)](#)  
[.net core\(13\)](#)  
[ABP\(4\)](#)  
[Asp .Net MVC\(8\)](#)  
[docker\(5\)](#)  
[Java、 \(1\)](#)  
[Linux\(5\)](#)  
[Web 前端\(5\)](#)  
[操作系统\(2\)](#)  
[架构\(34\)](#)  
[其他\(14\)](#)  
[数据结构和算法\(2\)](#)  
[数据库\(19\)](#)  
[微服务\(11\)](#)

最新评论

1. [Re:sql性能优化实例](#)

sqlserver吗

--Flame540
2. [Re:SSH-keygen用法](#)

整理得不错

--Flame540
3. [Re:Redis中切换db](#)

C# 代码如何切换DB

--悠悠木子李
4. [Re:CentOS7安装MySQL8.0图文教程](#)

后排提示一下， 35、36步新密码是root

--It's\_OK
5. [Re:SQL 执行 底层原理（一）](#)

分享一波SQL执行原理详解的资料，需要的朋友  
可以进1160405674的困找我来拿无偿

--图灵学院在线

阅读排行榜

1. [【Python】Python处理csv文件\(103617\)](#)
2. [关于sql语句between and的边界问题\(82764\)](#)
3. [SSH-keygen用法\(77664\)](#)
4. [【Python】python文件名和文件路径操作\(74204\)](#)

5. 由于 Web 服务器上的“ISAPI 和 CGI 限制”列表设置，无法提供您请求的页面。(56500)

评论排行榜

- 1. CentOS7安装MySQL8.0图文教程(11)
- 2. 【第三方插件】使用Topshelf创建Windows服务(10)
- 3. 【懒人有道】在asp.net core中实现程序集注入(5)
- 4. SQL中INNER JOIN、LEFT JOIN、RIGHT JOIN、FULL JOIN区别(4)
- 5. abp vNext微服务框架分析(3)

推荐排行榜

- 1. CentOS7安装MySQL8.0图文教程(11)
- 2. abp vNext微服务框架分析(8)
- 3. 【第三方插件】使用Topshelf创建Windows服务(7)
- 4. 【Python】Python处理csv文件(6)
- 5. SQL中INNER JOIN、LEFT JOIN、RIGHT JOIN、FULL JOIN区别(4)