缓存架构设计细节二三事

本文主要讨论这么几个问题：

（1）“缓存与数据库”需求缘起

（2）“淘汰缓存”还是“更新缓存”

（3）缓存和数据库的操作时序

（4）缓存和数据库架构简析

一、需求缘起

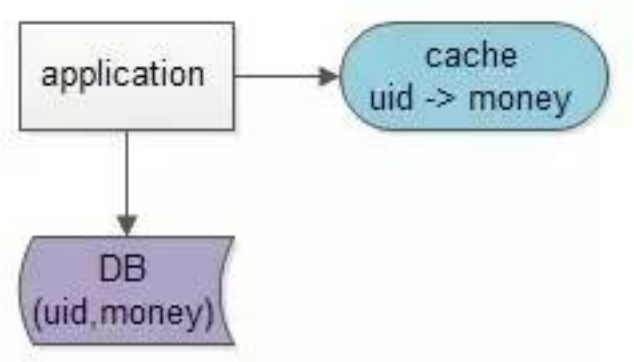
场景介绍

缓存是一种提高系统读性能的常见技术，对于读多写少的应用场景，我们经常使用缓存来进行优化。

例如对于用户的余额信息表account(uid, money)，业务上的需求是：

（1）查询用户的余额，SELECT money FROM account WHERE uid=XXX，占99%的请求

（2）更改用户余额，UPDATE account SET money=XXX WHERE uid=XXX，占1%的请求



由于大部分的请求是查询，我们在缓存中建立uid到money的键值对，能够极大降低数据库的压力。

读操作流程

有了数据库和缓存两个地方存放数据之后（uid->money），每当需要读取相关数据时（money），操作流程一般是这样的：

（1）读取缓存中是否有相关数据，uid->money

（2）如果缓存中有相关数据money，则返回【这就是所谓的**数据命中“hit”**】

（3）如果缓存中没有相关数据money，则从数据库读取相关数据money【这就是所谓的**数据未命中“miss”**】，放入缓存中uid->money，再返回

缓存的命中率 = 命中缓存请求个数/总缓存访问请求个数 = hit/(hit+miss)

上面举例的余额场景，99%的读，1%的写，这个缓存的命中率是非常高的，会在95%以上。

那么问题来了

当数据money发生变化的时候：

（1）是更新缓存中的数据，还是淘汰缓存中的数据呢？

（2）是先操纵数据库中的数据再操纵缓存中的数据，还是先操纵缓存中的数据再操纵数据库中的数据呢？

（3）缓存与数据库的操作，在架构上是否有优化的空间呢？

这是本文关注的三个核心问题。

二、更新缓存 VS 淘汰缓存

什么是更新缓存：数据不但写入数据库，还会写入缓存

什么是淘汰缓存：数据只会写入数据库，不会写入缓存，只会把数据淘汰掉

更新缓存的优点：缓存不会增加一次miss，命中率高

淘汰缓存的优点：简单（我去，更新缓存我也觉得很简单呀，楼主你太敷衍了吧）

那到底是选择更新缓存还是淘汰缓存呢，主要取决于“更新缓存的复杂度”。

例如，上述场景，只是简单的把余额money设置成一个值，那么：

（1）淘汰缓存的操作为deleteCache(uid)

（2）更新缓存的操作为setCache(uid, money)

更新缓存的代价很小，此时我们应该更倾向于更新缓存，以保证更高的缓存命中率

如果余额是通过很复杂的数据计算得出来的，例如业务上除了账户表account，还有商品表product，折扣表discount

account(uid, money)

product(pid, type, price, pinfo)

discount(type, zhekou)

业务场景是用户买了一个商品product，这个商品的价格是price，这个商品从属于type类商品，type类商品在做促销活动要打折扣zhekou，购买了商品过后，这个余额的计算就复杂了，需要：

（1）先把商品的品类，价格取出来：SELECT type, price FROM product WHERE pid=XXX

（2）再把这个品类的折扣取出来：SELECT zhekou FROM discount WHERE type=XXX

（3）再把原有余额从缓存中查询出来money = getCache(uid)

（4）再把新的余额写入到缓存中去setCache(uid, money-price*zhekou)

更新缓存的代价很大，此时我们应该更倾向于淘汰缓存。

however，淘汰缓存操作简单，并且带来的副作用只是增加了一次cache miss，建议作为通用的处理方式。

三、先操作数据库 vs 先操作缓存

OK，当写操作发生时，假设淘汰缓存作为对缓存通用的处理方式，又面临两种抉

择：

（1）先写数据库，再淘汰缓存

（2）先淘汰缓存，再写数据库

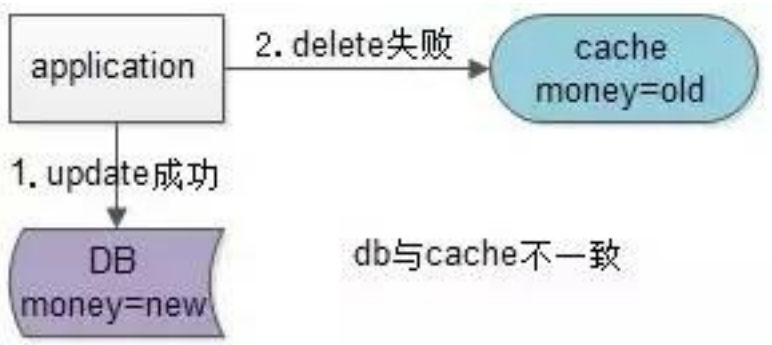
究竟采用哪种时序呢？

还记得在《冗余表如何保证数据一致性》文章（点击查看）里“究竟先写正表还是先写反表”的结论么？

对于一个不能保证事务性的操作，一定涉及“哪个任务先做，哪个任务后做”的问题，解决这个问题的方向是：

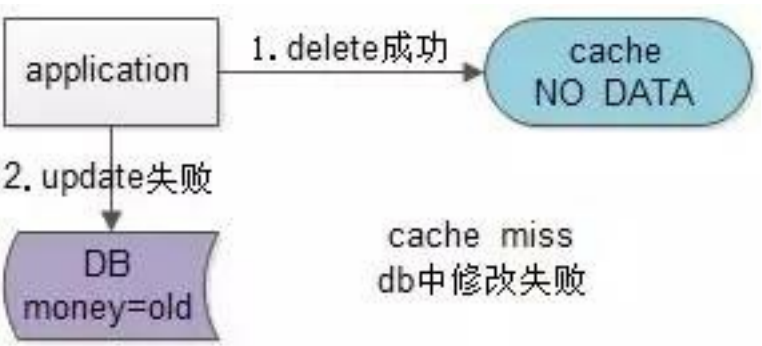
如果出现不一致，谁先做对业务的影响较小，就谁先执行。

由于写数据库与淘汰缓存不能保证原子性，谁先谁后同样要遵循上述原则。



假设先写数据库，再淘汰缓存

：第一步写数据库操作成功，第二步淘汰缓存失败，则会出现DB中是新数据，Cache中是旧数据，数据不一致。

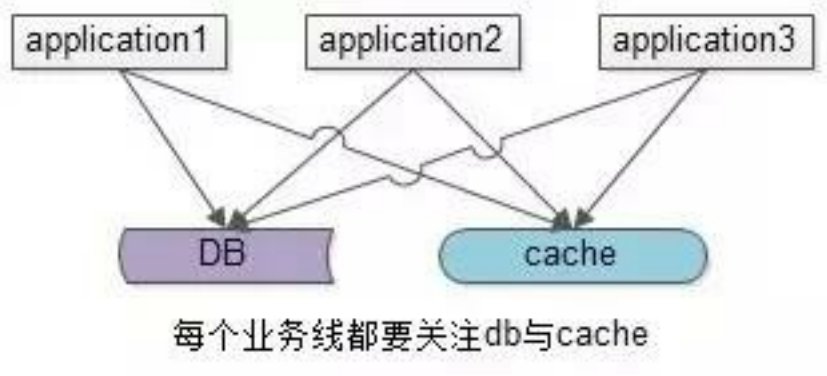


假设先淘汰缓存，再写数据库：第一步淘汰缓存成功，第二步写数据库失败，则只会

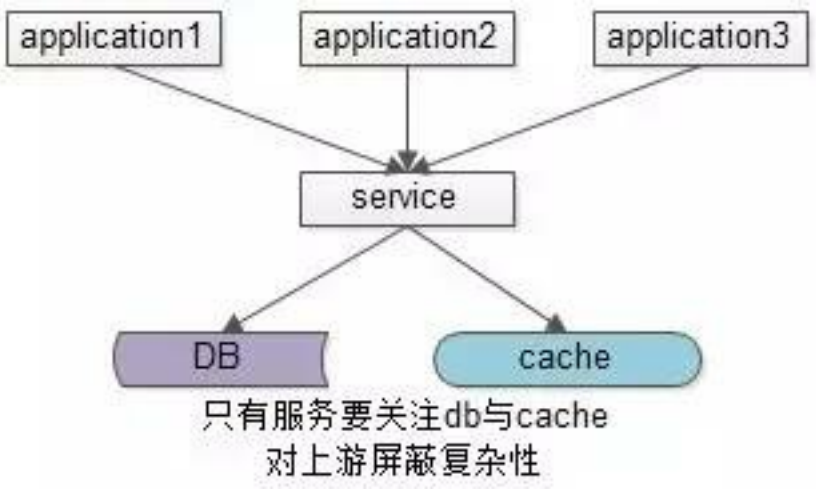
引发一次Cache miss。

结论：数据和缓存的操作时序，结论是清楚的：先淘汰缓存，再写数据库。

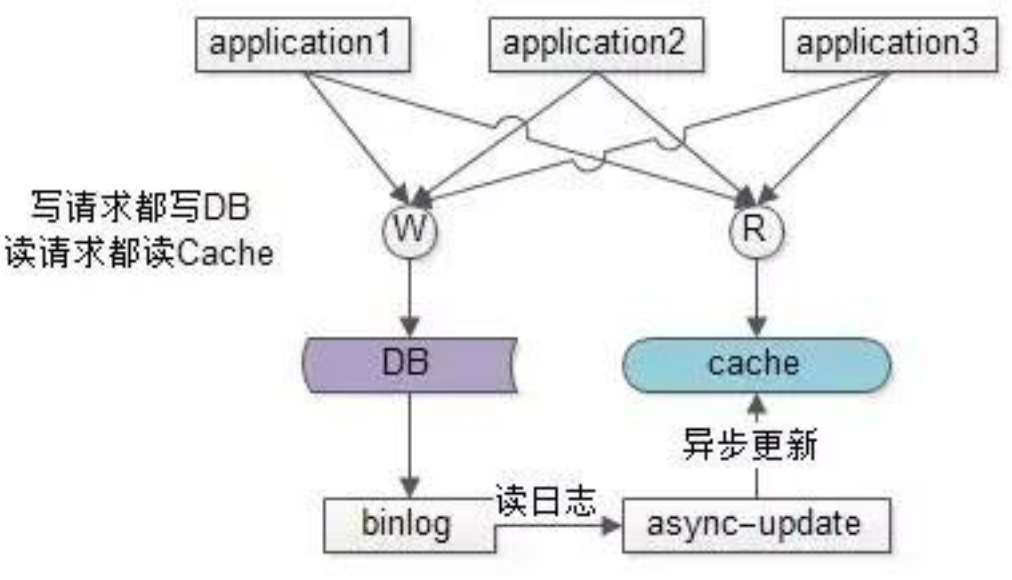
四、缓存架构优化



上述缓存架构有一个缺点：业务方需要同时关注缓存与DB，有没有进一步的优化空间呢？有两种常见的方案，一种主流方案，一种非主流方案（一家之言，勿拍）。



主流优化方案是**服务化**：加入一个服务层，向上游提供帅气的数据访问接口，向上游屏蔽底层数据存储的细节，这样业务线不需要关注数据是来自于cache还是DB。



非主流方案是异步缓存更新：业务线所有的写操作都走数据库，所有的读操作都总缓存，由一个异步的工具来做数据库与缓存之间数据的同步，具体细节是：

（1）要有一个init cache的过程，将需要缓存的数据全量写入cache

（2）如果DB有写操作，异步更新程序读取binlog，更新cache

在（1）和（2）的合作下，cache中有全部的数据，这样：

（a）业务线读cache，一定能够hit（很短的时间内，可能有脏数据），无需关注数据库

（b）业务线写DB，cache中能得到异步更新，无需关注缓存

这样将大大简化业务线的调用逻辑，存在的缺点是，如果缓存的数据业务逻辑比较复杂，async-update异步更新的逻辑可能也会比较复杂。

五、其他未尽事宜

本文只讨论了缓存架构设计中需要注意的几个细节点，如果数据库架构采用了一主多从，读写分离的架构，在特殊时序下，还很可能引发数据库与缓存的不一致，这个不一致如何优化，后续的文章再讨论吧。

六、结论强调

（1）淘汰缓存是一种通用的缓存处理方式

（2）先淘汰缓存，再写数据库的时序是毋庸置疑的

（3）服务化是向业务方屏蔽底层数据库与缓存复杂性的一种通用方式

欢迎加入我的社群或关注公众号“架构师之路”进行讨论。

W3Cschool（www.w3cschool.cn）最大的技术知识分享与学习平台

此篇内容来自于w3cschool.cn网站用户上传并发布。