主从DB与cache一致性优化

本文主要讨论这么几个问题：

（1）数据库主从延时为何会导致缓存数据不一致

（2）优化思路与方案

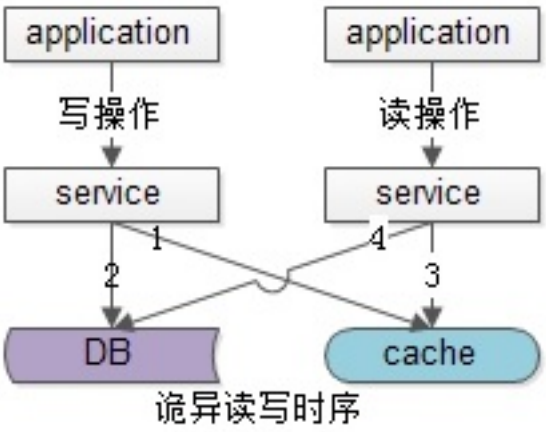
一、需求缘起

上一篇《缓存架构设计细节二三事》中有一个小优化点，在只有主库时，通过“串行化”的思路可以解决缓存与数据库中数据不一致。引发大家热烈讨论的点是“在主从同步，读写分离的数据库架构下，有可能出现脏数据入缓存的情况，此时串行化方案不再适用了”，这就是本文要讨论的主题。

二、为什么数据会不一致

为什么会读到脏数据，有这么几种情况：

（1）单库情况下，服务层的并发读写，缓存与数据库的操作交叉进行



诡异读写时序

虽然只有一个DB，在上述诡异异常时序下，也可能脏数据入缓存：

1）请求A发起一个写操作，第一步淘汰了cache，然后这个请求因为各种原因在服务层卡住了（进行大量的业务逻辑计算，例如计算了1秒钟），如上图步骤1

2）请求B发起一个读操作，读cache，cache miss，如上图步骤2

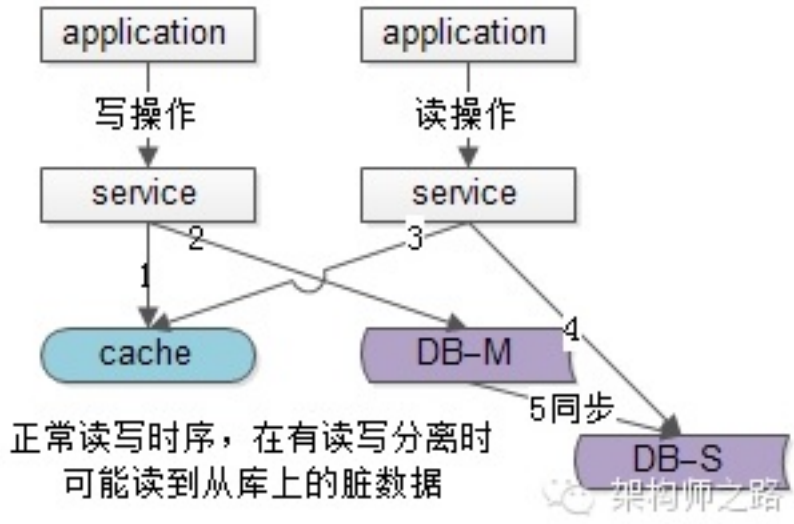
3）请求B继续读DB，读出来一个脏数据，然后脏数据入cache，如上图步骤3

4）请求A卡了很久后终于写数据库了，写入了最新的数据，如上图步骤4

这种情况虽然少见，但理论上是存在的， 后发起的请求B在先发起的请求A中间完成了。

（2）主从同步，读写分离的情况下，读从库读到旧数据

在数据库架构做了一主多从，读写分离时，更多的脏数据入缓存是下面这种情况：



1）请求A发起一个写操作，第一步淘汰了cache，如上图步骤1

2）请求A写数据库了，写入了最新的数据，如上图步骤2

3）请求B发起一个读操作，读cache，cache miss，如上图步骤3

4）请求B继续读DB，读的是从库，此时主从同步还没有完成，读出来一个脏数据，然后脏数据入cache，如上图步4

5）最后数据库的主从同步完成了，如上图步骤5

这种情况请求A和请求B的时序是完全没有问题的，是主动同步的时延（假设延时1秒钟）中间有读请求读从库读到脏数据导致的不一致。

那怎么来进行优化呢？

三、不一致优化思路

有同学说“那能不能先操作数据库，再淘汰缓存”，这个是不行的，在《缓存架构设计细节二三事》的文章中介绍过。

出现不一致的根本原因：

（1）单库情况下，服务层在进行1s的逻辑计算过程中，可能读到旧数据入缓存

（2）主从库+读写分离情况下，在1s钟主从同步延时过程中，可能读到旧数据入缓存

既然旧数据就是在那1s的间隙中入缓存的，**是不是可以在写请求完成后，再休眠1s，再次淘汰缓存**，就能将这1s内写入的脏数据再次淘汰掉呢？

答案是可以的。

写请求的步骤由2步升级为3步：

（1）先淘汰缓存

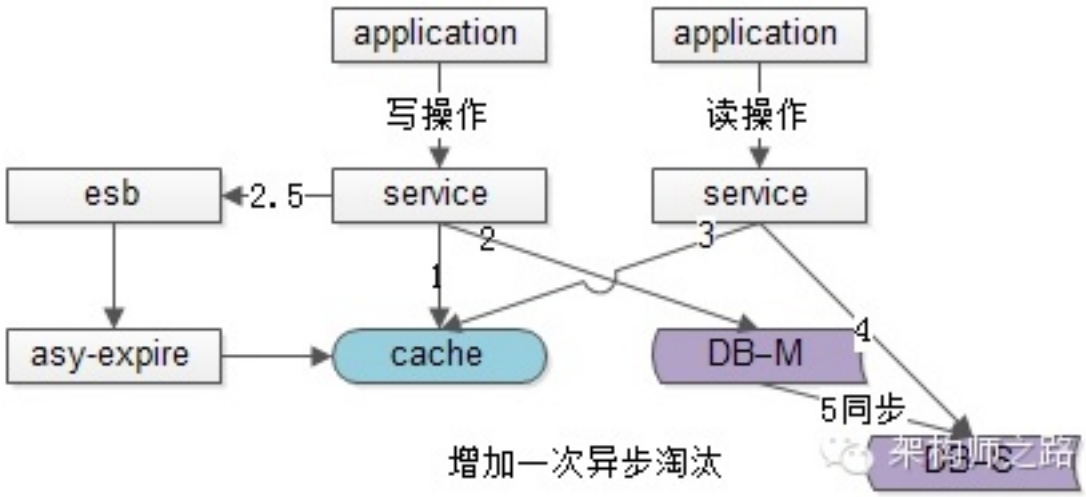
（2）再写数据库（这两步和原来一样）

（3）休眠1秒，再次淘汰缓存

这样的话，1秒内有脏数据如缓存，也会被再次淘汰掉，但带来的问题是：

（1）所有的写请求都阻塞了1秒，大大降低了写请求的吞吐量，增长了处理时间，业务上是接受不了的

再次分析，其实第二次淘汰缓存是“为了保证缓存一致”而做的操作，而不是“业务要求”，所以其实无需等待，**用一个异步的timer，或者利用消息总线异步的来做这个事情即可**：



写请求由2步升级为2.5步：

（1）先淘汰缓存

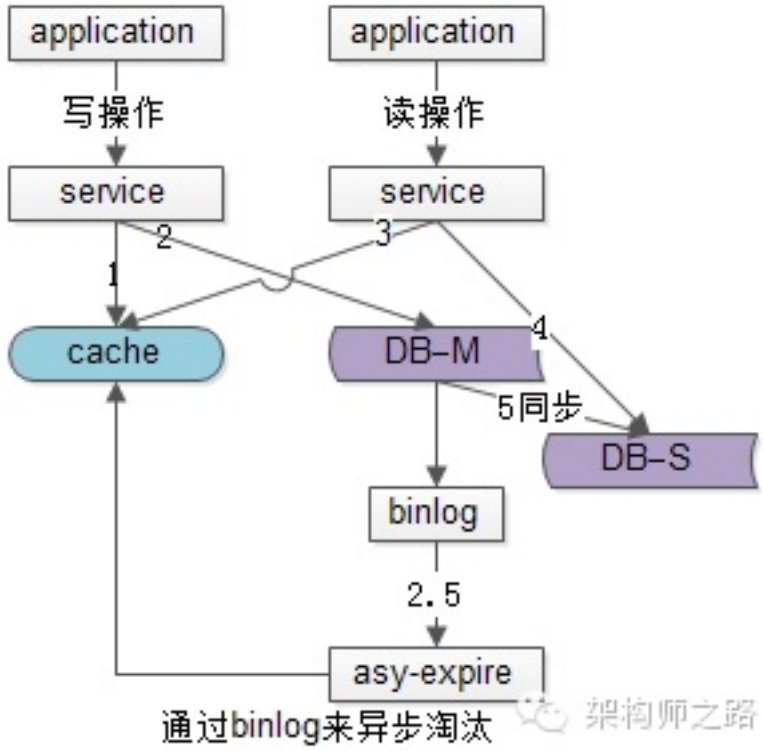
（2）再写数据库（这两步和原来一样）

（2.5）不再休眠1s，而是往消息总线esb发送一个消息，发送完成之后马上就能返回

这样的话，写请求的处理时间几乎没有增加，这个方法淘汰了缓存两次，因此被称为“缓存双淘汰”法。这个方法付出的代价是，缓存会增加1次cache miss（代价几乎可以忽略）。

而在下游，有一个异步淘汰缓存的消费者，在接收到消息之后，asy-expire在1s之后淘汰缓存。这样，即使1s内有脏数据入缓存，也有机会再次被淘汰掉。

上述方案有一个缺点，需要业务线的写操作增加一个步骤，有没有方案对业务线的代码没有任何入侵呢，是有的，这个方案在《细聊冗余表数据一致性》中也提到过，通过分析线下的binlog来异步淘汰缓存：



业务线的代码就不需要动了，新增一个线下的读binlog的异步淘汰模块，读取到binlog中的数据，异步的淘汰缓存。

提问：为什么上文总是说1s，这个1s是怎么来的？

回答：1s只是一个举例，需要根据业务的数据量与并发量，观察主从同步的时延来设定这个值。例如主从同步的时延为200ms，这个异步淘汰cache设置为258ms就是OK的。

四、总结

在“异常时序”或者“读从库”导致脏数据入缓存时，可以用二次异步淘汰的“缓存双淘汰”法来解决缓存与数据库中数据不一致的问题，具体实施至少有三种方案：

（1）**timer异步淘汰**（本文没有细讲，本质就是起个线程专门异步二次淘汰缓存）

（2）**总线异步淘汰**

（3）**读binlog异步淘汰**

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。