11 | MySQL如何应对高并发（一）：使用缓存保护MySQL

李玥 · 后端存储实战课

你好，我是李玥。

通过前面几节课的学习，相信你对 MySQL 这类关系型数据库的能力，已经有了定量的认知。

我们知道，大部分面向公众用户的互联网系统，它的并发请求数量是和在线用户数量正相关的，而 MySQL 能承担的并发读写的量是有上限的，当系统的在线用户超过几万到几十万这个量级的时候，单台 MySQL 就很难应付了。

绝大多数互联网系统，都使用 MySQL 加上 Redis 这对儿经典的组合来解决这个问题。Redis 作为 MySQL 的前置缓存，可以替 MySQL 挡住绝大部分查询请求，很大程度上缓解了 MySQL 并发请求的压力。

Redis 之所以能这么流行，非常重要的一个原因是，它的 API 非常简单，几乎没有太多的学习成本。但是，要想在生产系统中用好 Redis 和 MySQL 这对儿经典组合，并不是一件很简单的事儿。我在《08 | 一个几乎每个系统必踩的坑儿：访问数据库超时》举的社交电商数据库

超时故障的案例，其中一个重要的原因就是，对缓存使用不当引发了缓存穿透，最终导致数据库被大量查询请求打死。

今天这节课，我们就来说一下，在电商的交易类系统中，如何正确地使用 Redis 这样的缓存系统，以及如何正确应对使用缓存过程中遇到的一些常见的问题。

更新缓存的最佳方式

要正确地使用好任何一个数据库，你都需要先了解它的能力和弱点，扬长避短。Redis 是一个使用内存保存数据的高性能 KV 数据库，它的高性能主要来自于：

1. 简单的数据结构；
2. 使用内存存储数据。

上节课我们讲到过，数据库可以分为执行器和存储引擎两部分，Redis 的执行器这一层非常的薄，所以 Redis 只能支持有限的几个 API，几乎没有聚合查询的能力，也不支持 SQL。它的存储引擎也非常简单，直接在内存中用最简单的数据结构来保存数据，你从它的 API 中的数据类型基本就可以猜出存储引擎中数据结构。

比如，Redis 的 LIST 在存储引擎的内存中的数据结构就是一个双向链表。内存是一种易失性存储，所以使用内存保存数据的 Redis 不能保证数据可靠存储。从设计上来说，Redis 牺牲了大部分功能，牺牲了数据可靠性，换取了高性能。但也正是这些特性，使得 Redis 特别适合用来做 MySQL 的前置缓存。

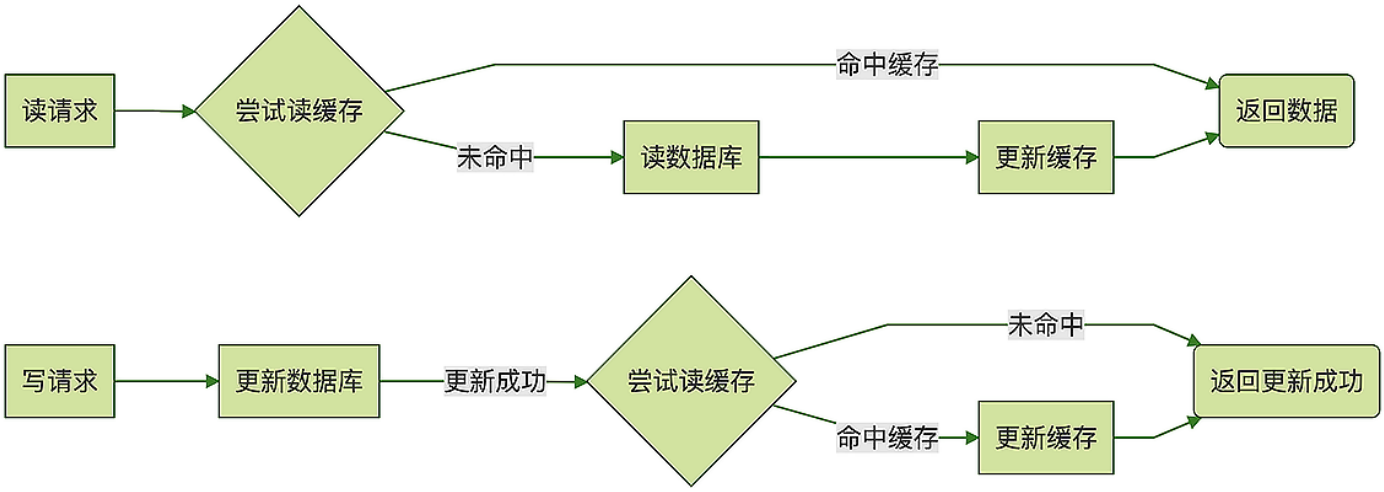
虽然说，Redis 支持将数据持久化到磁盘中，并且还支持主从复制，但你需要知道，**Redis 仍然是一个不可靠的存储，它在设计上天然就不保证数据的可靠性**，所以一般我们都使用 Redis 做缓存，很少使用它作为唯一的数据存储。

即使只是把 Redis 作为缓存来使用，我们在设计 Redis 缓存的时候，也必须要考虑 Redis 的这种“数据不可靠性”，或者换句话说，我们的程序在使用 Redis 的时候，要能兼容 Redis 丢数据的情况，做到即使 Redis 发生了丢数据的情况，也不影响系统的数据准确性。

我们仍然用电商的订单系统来作为例子说明一下，如何正确地使用 Redis 做缓存。在缓存 MySQL 的一张表的时候，通常直接选用主键来作为 Redis 中的 Key，比如缓存订单表，那就直接用订单表的主键订单号来作为 Redis 中的 key。

如果说，Redis 的实例不是给订单表专用的，还需要给订单的 Key 加一个统一的前缀，比如“orders:888888”。Value 用来保存序列化后的整条订单记录，你可以选择可读性比较好的 JSON 作为序列化方式，也可以选择性能更好并且更节省内存的二进制序列化方式，都是可以的。

然后我们来说，缓存中的数据要怎么来更新的问题。我见过很多同学都是这么用缓存的：在查询订单数据的时候，先去缓存中查询，如果命中缓存那就直接返回订单数据。如果没有命中，那就去数据库中查询，得到查询结果之后把订单数据写入缓存，然后返回。在更新订单数据的时候，先去更新数据库中的订单表，如果更新成功，再去更新缓存中的数据。



这其实是一种经典的缓存更新策略: **Read/Write Through**。这样使用缓存的方式有没有问题？绝大多数情况下可能都没问题。但是，在并发的情况下，有一定的概率会出现“脏数据”问题，缓存中的数据可能会被错误地更新成了旧数据。

比如，对同一条订单记录，同时产生了一个读请求和一个写请求，这两个请求被分配到两个不同的线程并行执行，读线程尝试读缓存没命中，去数据库读到了订单数据，这时候可能另外一个读线程抢先更新了缓存，在处理写请求的线程中，先后更新了数据和缓存，然后，拿着订单旧数据的第一个读线程又把缓存更新成了旧数据。

这是一种情况，还有比如两个线程对同一个条订单数据并发写，也有可能造成缓存中的“脏数据”，具体流程类似于我在之前“如何保证订单数据准确无误？”这节课中讲到的 ABA 问题。你不要觉得发生这种情况的概率比较小，出现“脏数据”的概率是和系统的数据量以及并发数量正相关的，当系统的数据量足够大并且并发足够多的情况下，这种脏数据几乎是必然会出现的。

我在“商品系统的存储该如何设计”这节课中，在讲解如何缓存商品数据的时候，曾经简单提到过缓存策略。其中提到的 Cache Aside 模式可以很好地解决这个问题，在大多数情况下是使用缓存的最佳方式。

Cache Aside 模式和上面的 Read/Write Through 模式非常像，它们处理读请求的逻辑是完全一样的，唯一的一个小差别就是，Cache Aside 模式在更新数据的时候，并不去尝试更新缓存，而是去删除缓存。

订单服务收到更新数据请求之后，先更新数据库，如果更新成功了，再尝试去删除缓存中订单，如果缓存中存在这条订单就删除它，如果不存在就什么都不做，然后返回更新成功。这条更新后的订单数据将在下次被访问的时候加载到缓存中。使用 Cache Aside 模式来更新缓存，可以非常有效地避免并发读写导致的脏数据问题。

注意缓存穿透引起雪崩

如果我们的缓存命中率比较低，就会出现大量“缓存穿透”的情况。缓存穿透指的是，在读数据的时候，没有命中缓存，请求“穿透”了缓存，直接访问后端数据库的情况。

少量的缓存穿透是正常的，我们需要预防的是，短时间内大量的请求无法命中缓存，请求穿透到数据库，导致数据库繁忙，请求超时。大量的请求超时还会引发更多的重试请求，更多的重试请求让数据库更加繁忙，这样恶性循环导致系统雪崩。

当系统初始化的时候，比如说系统升级重启或者是缓存刚上线，这个时候缓存是空的，如果大量的请求直接打过来，很容易引发大量缓存穿透导致雪崩。为了避免这种情况，可以采用灰度发布的方式，先接入少量请求，再逐步增加系统的请求数量，直到全部请求都切换完成。

如果系统不能采用灰度发布的方式，那就需要在系统启动的时候对缓存进行预热。所谓的缓存预热就是在系统初始化阶段，接收外部请求之前，先把最经常访问的数据填充到缓存里面，这样大量请求打过来的时候，就不会出现大量的缓存穿透了。

还有一种常见的缓存穿透引起雪崩的情况是，当发生缓存穿透时，如果从数据库中读取数据的时间比较长，也容易引起数据库雪崩。

这种情况我在《08 | 一个几乎每个系统必踩的坑儿：访问数据库超时》这节课中也曾经提到过。比如说，我们缓存的数据是一个复杂的数据库联查结果，如果在数据库执行这个查询需要 10 秒钟，那当缓存中这条数据过期之后，最少 10 秒内，缓存中都不会有数据。

如果这 10 秒内有大量的请求都需要读取这个缓存数据，这些请求都会穿透缓存，打到数据库上，这样很容易导致数据库繁忙，当请求量比较大的时候就会引起雪崩。

所以，如果说构建缓存数据需要的查询时间太长，或者并发量特别大的时候，Cache Aside 或者是 Read/Write Through 这两种缓存模式都可能出现大量缓存穿透。

对于这种情况，并没有一种方法能应对所有的场景，你需要针对业务场景来选择合适解决方案。比如说，可以牺牲缓存的时效性和利用率，缓存所有的数据，放弃 Read Through 策略所有的请求，只读缓存不读数据库，用后台线程来定时更新缓存数据。

小结

使用 Redis 作为 MySQL 的前置缓存，可以非常有效地提升系统的并发上限，降低请求响应时延。绝大多数情况下，使用 Cache Aside 模式来更新缓存都是最佳的选择，相比 Read/Write Through 模式更简单，还能大幅降低脏数据的可能性。

使用 Redis 的时候，还需要特别注意大量缓存穿透引起雪崩的问题，在系统初始化阶段，需要使用灰度发布或者其他方式来对缓存进行预热。如果说构建缓存数据需要的查询时间过长，

或者并发量特别大，这两种情况下使用 Cache Aside 模式更新缓存，会出现大量缓存穿透，有可能会引发雪崩。

顺便说一句，我们今天这节课中讲到的这些缓存策略，都是非常经典的理论，早在互联网大规模应用之前，这些缓存策略就已经非常成熟了，在操作系统中，CPU Cache 的缓存、磁盘文件的内存缓存，它们也都应用了我们今天讲到的这些策略。

所以无论技术发展的多快，计算机的很多基础的理论的知识都是相通的，你绞尽脑汁想出的解决工程问题的方法，很可能早都写在几十年前出版的书里。学习算法、数据结构、设计模式等等这些基础的知识，并不只是为了应付面试。

思考题

课后请你想一下，具体什么情况下，使用 Cache Aside 模式更新缓存会产生脏数据？欢迎你在评论区留言，通过一个例子来说明情况。

感谢阅读，如果你觉得今天的内容对你有帮助，也欢迎把它分享给你的朋友。

精选留言 (42)

李玥 置顶

2020-03-21

Hi，我是李玥。

这里回顾一下上节课的思考题：

课后请你选一种你熟悉的非关系型数据库，最好是支持 SQL 的，当然，不支持 SQL 有自己的查询语言也可以。比如说 HBase、Redis 或者 MongoDB 等等都可以，尝试分析一下查询的执行过程，对比一下它的执行器和存储引擎与 MySQL 有什么不同。

谈一下我的理解：

我们拿一个分布式数据库Hive来看一下它的执行器和存储引擎。严格来说，Hive并不是一个数据库，它只是一个执行器，它的存储引擎就是HDFS加上Map-Reduce。在Hive中，一条SQL的执行过程是和MySQL差不多的，Hive会解析SQL，生成并优化逻辑执行计划，然后它就会把逻辑执行计划交给Map-Reduce去执行了，后续生成并优化物理执行计划，在HDFS上执行查询这些事儿，都是Map-Reduce去干的。顺便说一下，Hive的执行引擎（严格来说是物理执行引擎）是可以替换的，所以就有了Hive on Spark，Hive on Tez这些。

34

Geek_3894f9
2020-03-21

数据加版本号，写库时自动增一。更新缓存时，只允许高版本数据覆盖低版本数据。

作者回复: 👍👍👍

共 19 条评论>

69

公号-技术夜未眠
2020-03-21

Cache Aside 在高并发场景下也会出现数据不一致。
读操作A，没有命中缓存，就会到数据库中取数据v1。
此时来了一个写操作B，将v2写入数据库，让缓存失效；
读操作A在把v1放入缓存，这样就会造成脏数据。因为缓存中是v1，数据库中是v2.

共 13 条评论>

52

GaGi
2020-03-21

对于Cache aside和read/write through而带来的数据不一致问题，工作中是这样解决：
a写线程，b读线程：
b线程：读缓存->未命中->上写锁>从db读数据到缓存->释放锁；
a线程：上写锁->写db->删除缓存/改缓存->释放锁；
这样来保证a，b线程并发读写缓存带来的脏数据问题；

作者回复: 👍👍👍

共 16 条评论>

40

AI
2020-07-16

老师，看了你对缓存模式的解读之后，跟我之前的理解感觉有比较大的冲突。

在文中，你提到的 Read/Write Through 策略，我看到的更多的是把这种方式称为Cache Aside。
而Cache Aside 和 Read/Write Throug的差别，也不是是否去删除缓存。

我看的一些文章，包括买的书籍，大部分都跟ehcache caching-patterns描述的意思差不多，总结如下：

Cache Aside

应用程序直接与DB、缓存交互，并负责对缓存的维护。

读数据时，先访问缓存，命中则直接返回。
如果不命中，则先查询DB，并将数据写到缓存，最后返回数据。

写数据时，同时更新DB和缓存。

Read-Through

应用程序只与缓存交互，而对DB的读取由缓存来代理。

读数据时，先访问缓存，命中则直接返回。
如果不命中，则由缓存查询DB，并将数据写到缓存，最后返回数据。

Write-Through

应用程序只与缓存交互，而对DB的写由缓存来代理。

写数据时，访问缓存，由缓存将数据写到DB，并将数据缓存起来。

例如使用Redis来缓存MySQL的数据，一般都是通过应用程序来直接与Redis、MySQL交互，我的理解是Cache Aside，包"是/否"删除Cache在内。

而Read-Through，像Guava LoadingCache，在load里面定义好访问DB的代码，后续的读操作都是直接与Cache交互了。

https://www.ehcache.org/documentation/3.8/caching-patterns.html
https://docs.oracle.com/cd/E15357_01/coh.360/e15723/cache_rtwtwbra.htm#COHDG5178
https://dzone.com/articles/using-read-through-amp-write-through-in-distribute
https://docs.microsoft.com/en-us/azure/architecture/patterns/cache-aside
《亿级流量网站架构核心技术》

共 2 条评论> 19

每天晒白牙
2020-03-22

思考题
Cache Aside 模式如何产生脏数据?
首先 Cache Aside 这种模式和 Read/Write Through 模式的读取操作一样，都是先尝试读缓存，如果命中直接返回;未命中的话读数据库，然后更新缓存。
写操作不是更新缓存，而是把缓存中的数据删掉
那怎么出现脏数据?
假设有下面两个线程对 key 分别进行读写操作
读线程 t1
写线程 t2
按照下面的流程进行操作
1. t1 读缓存未命中，然后从数据库中读到 value1
2. t2 更新 key 为 value2，并尝试删缓存(此时缓存中并没有)
3. t1 把从 db 中读到的 value1写回缓存

这时 db 中 key 的 value 为新数据 value2，缓存中为旧数据 value1，产生了不一致。
这种情况只发生在读线程从 bd 读到旧数据往 cache 中写前，有写线程更新了 db，然后读线程把老数据写回 cache

Read/Write Through 发生脏数据的情况
第一种情况是并发读写
对 key 进行读写的两个线程
读线程 t1

写线程 t2

按照如下时间顺序操作
1.t1 尝试读缓存但未命中，然后去 db 中读取到了数据 value1，但还未更新缓存兄弟的数据
2. t2 更新数据库 value2，更新成功后常识读取缓存，未命中，所以更新缓存为 value2
3.t1 线程继续把之前从 db 中读到的旧数据 value1 写回缓存
这样 db 中是新数据，但缓存中是旧数据

第二种情况是并发写
这种情况是db 中产生了 ABA 问题
比如有两个写线程 t1,t2，分别按下面的先后顺序操作
1.t1 尝试把 key 更新为 value1，但响应丢失
2.t2 尝试把 key 更新为 value2，还未响应结果
3.t1 发生重试操作
4.t2 响应成功
5.t1 响应成功
本来写应该按先后顺序的，t2后到，数据库和缓存中应该是 value2，但因为 t1 发生了重试，导致数据库和缓存中是 value1了，产生了ABA问题，解决办法是在更新时加上 version 版本号判断

所以没啥万能的方法，需要根据业务场景来制定方法

8

seg-上海
2020-04-25

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

共 1 条评论>

5

约书亚
2020-03-21

A,B两个进程
B read cache x=null
B read DB x=1

A write DB x=2
A delete cache
B write cache x=1
这时数据库里x=2，缓存中x=1，直到缓存过期之前一直是脏数据。这种概率算是比较小的了。

文章中提到用灰度来解决问题，似乎解决不了基于类似redis这种做分布式缓存时的问题。

共 2 条评论>

5

握了个大蚂蚱
2020-04-23

总结：
1.为什么先更新mysql再更新（删除）redis比反过来好?

降低了脏数据出现的概率，前者产生脏数据是由于并发，后者几乎是必然，只要先写再读的请求发生，都会造成脏数据：先把redis中的缓存清了，然后读请求读不到去数据库中找到并更新在redis中。
2.为什么aside cache比read/write through好?
也是降低了脏数据出现的概率。前者只有读写先后访问数据库，又调转顺序访问redis时redis中出现脏数据，这个概率很小，而并发写时相当于不操作redis；而后者在并发写的情况下也容易脏。

4

mickey
2020-04-18

老师好。文中写“订单服务收到更新数据请求之后，先更新数据库，如果更新成功了，再尝试去删除缓存中订单，如果缓存中存在这条订单就删除它，如果不存在就什么都不做，然后返回更新成功。这条更新后的订单数据将在下次被访问的时候加载到缓存中。”

请问，前面的读线程没命中，去数据库读到了订单数据，这是写进程进来完成后，读线程将原来读的脏数据生成了缓存，这样还是不能解决问题啊?

4

丁小明
2020-03-27

老师，经常看到说用布隆过滤来解决缓存穿透问题，这个方案有实际的案例吗?
如果是真的可以那么怎么去操作呢?
先初始化所有可能存到缓存里面数据的key到一个足够大的布隆过滤器，然后如果有新增数据就就继续往过滤器中放，删除就从过滤器里面删（又看到说不用bit的话支持累加删除）
如果发现不在过滤器中就表示一定不存在，就无需查询了。如果在过滤器中也有可能不存在，这个时候在配合null值。
这个方案靠谱么，希望老师能解答一下

作者回复: 首先这是个经典的方案，靠谱是没问题的。它可以解决问题是，不用真正去查询数据集，就可以判断，请求的数据是不是，不在数据集内。如果不在就不用去查询数据集了。

不少数据库都内置了布隆过滤器来提升查询效率，比如HBase。

布隆过滤器的缺点就是有点复杂，实现难度还是挺大的。

4

任鹏斌
2020-05-21

老师Cache Aside应该是先删缓存后更新数据库吧？先更新数据库的话一旦缓存删除失败了，就会产生脏数据

作者回复: 严格来说，在并发情况下，二种方式都有可能产生脏数据。Cache Aside Pattern建议，先操作数据库，再操作缓存。

共 4 条评论>

3

王杰
2020-04-18

作者回复: 你可以参考一下“GaGi ”同学的留言，用锁来解决并发问题。----------老师，在读线程上写锁（说独占锁比较合适），是否跟MVCC相违背，MVCC不就是为了用来解决高并发带来的读写阻塞问题吗?我这边有两种解决思路不知可否：第一用版本控制，类似MVCC，第二种用Read/Write Through，写写并发在MVCC模式下依然是阻塞的，不算违背，所以只要把更新数据库与更新缓存放入统一事务中就行。读写并发不阻塞，是因为mysql用了快照读原因，那我们可以继续写线程更新缓存，读线程采用redis的setnx方式解决覆盖

作者回复: mvcc可以很好的解决读写冲突，但是对于写写冲突，要么加锁，要么引入冲突检测机制，否则就会导致写倾斜的问题。这个在23中有详细的说明。

共 5 条评论>

3

image
2020-03-26

如果缓存时有大量命中为null如何处理？如果命中null 也进行缓存，会导致缓存增长太快，容易被攻击
如果不缓存，又容易引起大量穿透

作者回复: 这种情况理论上也没有完美的解决方案，来说说实际上的一些处理经验。

首先，避免短时间大量人为的空值攻击，这个事儿应该在上层安全或者风控层面去解决。（即使无法判断是否空值攻击，至少要拦截住短时间大量的不正常访问请求）

剩余下来的就是业务上正常的查询返回空的情况，这种可能要从业务上来设计一下，尽量避免大量可能的空值查询。

以上2点做了之后，空值查询就会少多了，这个时候可以根据实际情况选择缓存空值，或者让空值穿透。

3

程堃
2020-09-14

问题：读线程查到缓存为空后，读db数据并写入缓存，在写入之前另一个更新操作修改了db数据，会导致db与缓存数据不一致。
方案：将Cache Aside的删除操作改成设置缓存几秒后失效，或者加分布式锁

2

夏目
2020-04-08

线程1读取缓存未命中，查询当前数据库数据
线程2更新当前数据库数据，删除缓存(不存在)
线程1讲老数据更新至缓存，导致当前数据库数据与缓存不一致

陶金
2020-03-22

```
package main

import (
	"fmt"
	"sync"
	"time"
)

type BaseModel struct {
	data int
	mu sync.RWMutex
}


var cache *BaseModel
var db *BaseModel
const FIRST_DATA = 1
const SECNOD_DATA = 2
const EMPTY_DATA = 0

func init() {
	cache = new (BaseModel)
	db = new (BaseModel)
	db.setData(FIRST_DATA)
}


func main() {

	go read()
	go write(SECNOD_DATA)

	time.Sleep(3 * time.Second)
	fmt.Println("db's data is %d, cache's data is %d", db.getData(), cache.getData())
```

```
}


func read() {
	data := cache.getData()
	if data == 0 {
		data = db.getData()
		time.Sleep(2* time.Second)
		cache.setData(data)
	}
}


func write(data int) {
	time.Sleep(1*time.Second)
	db.setData(data)
	cache.setData(EMPTY_DATA)
}


func (self *BaseModel) getData() int {
	self.mu.RLock()
	defer self.mu.RUnlock()
	return self.data
}

func (self *BaseModel) setData(data int) {
	self.mu.Lock()
	defer self.mu.Unlock()
	self.data = data
}
```

大概场景如下：

1. 初始数据库中数据为“1”，缓存无数据
2. 线程A为读线程，读取缓存未果，然后读取数据库中的记录为“1”，这时候缓存阻塞住。
3. 线程B为写线程，先把数据库中的数据更新为“2”，再删除缓存，结束。
4. 此时线程A解除阻塞，然后把记录“1”更新到缓存中。

此时缓存中数据为“1”， 数据库中数据为“2”， 缓存落后于数据库中的数据。

2

R20114

2020-03-21

Cache Aside 模式在下面的场景下：
读写线程之间在执行 Cache Aside Pattern 操作的时候，写线程删除了缓存，读线程从 DB 读到老的数据，把老的数据放到了缓存中，这样就会在缓存中产生脏数据。

共 2 条评论>

2

翠羽香凝

2020-10-22

有一个问题，一直没搞明白。如果使用redis作为mysql的前置，在写数据的时候，需要同时写Mysql和redis，如果更新mysql成功后，更新redis没成功，造成了两者之间的不一致，这种情况怎么办?

1

王杰

2020-04-18

你好，使用 Cache Aside 模式更新缓存好像也会产生脏数据。就如文章里所说，当一个读线程没有从缓存中读到数据，进而去查数据库获取到数据，这时一个写线程更新了数据库并删除缓存，可能由于读线程放入旧数据到缓存时机比较晚写线程没法删到，导致缓存中可能不是最新的数据，后面再有线程过来查询自然也读不到新数据了

1