11 | MySQL如何应对高并发（一）：使用缓存保护MySQL

2020-03-21 李玥

讲述：李玥
时长 14:17　大小 13.09M

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。

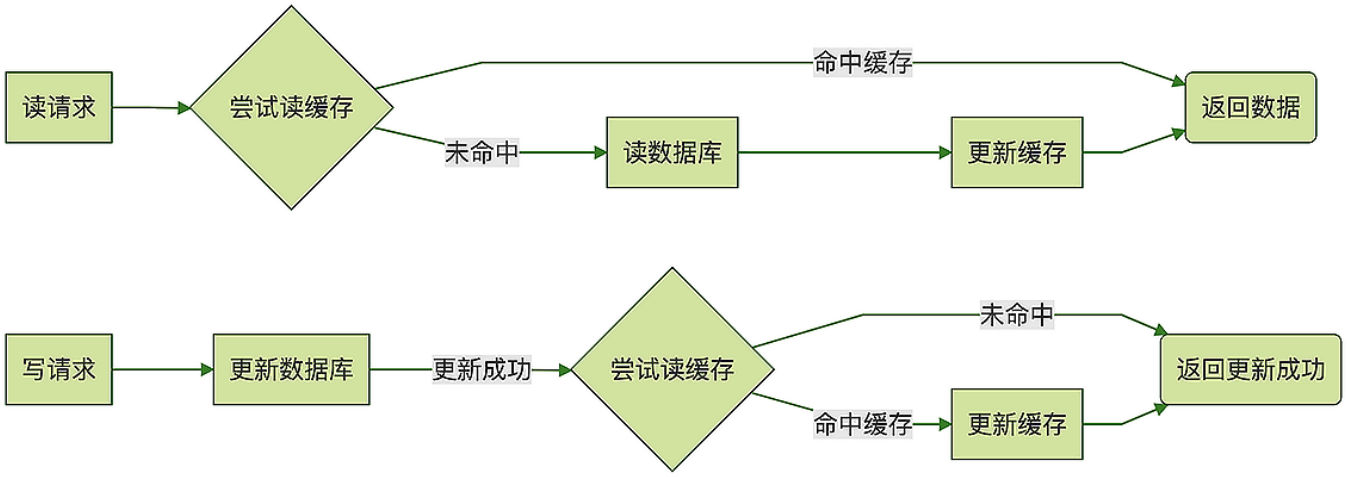
虽然说，Redis 支持将数据持久化到磁盘中，并且还支持主从复制，但你需要知道，**Redis 仍然是一个不可靠的存储，它在设计上天然就不保证数据的可靠性**，所以一般我们都使用 Redis 做缓存，很少使用它作为唯一的数据存储。

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 (18)

写留言

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...
展开

4

公号-云原生程序员
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.
展开

1 7

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:

2 5

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；...

展开

作者回复: 👍👍👍

2 3

0x12FD16B

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。

展开

2 1

Mq

2020-03-21

老师好，写数据跟删缓存不是一致的，写完数据到删缓存这段时间内其他并发访问都是脏数据，这种思维方式感觉不解决一致性问题，都会有可能出现脏读，只是时间长短问题

1

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...

展开

2 1

AAAAAAres

2020-03-23

如果读缓存不存在，然后去从库读数据来写缓存的话，主从延迟也会导致缓存中有脏数据

1

2020-03-23

是不是model的话使用缓存，列表的话是不是不适合用缓存？列表应该怎么去缓存？

作者回复: 这个还是得看业务，很多列表也可以缓存的，比如说一些排行榜数据。


刘楠
2020-03-22

一直用的Cache Aside,

展开


陶金
2020-03-22

package main

import (
"fmt"
"sync"...

展开


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...

展开


Aliliin
2020-03-21

a读到老数据的同时并没来得及写入缓存，然后b正好更新了db清空缓存，a写入之前读到的数据写入缓存。
只想到这种情况，不知道靠谱吗。

展开


正在减肥的胖籽。

2020-03-21

任何一种方式缓存使用方式和数据库之间都会有脏数据的产生，我现在的解决方案就是看业务方能接受多长时间的脏数据，然后缓存就设置多久的过期时间。2.或者数据库更新成功后，用MQ去通知刷新缓存。


小美
2020-03-21

Cache Aside解决的只是并发写请求导致的缓存数据不一致问题。对于读写这种场景并没有彻底解决。
A：读，缓存穿透，查库。
B：写，更新数据库。
B：写，删除缓存。...
展开

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。


往事随风，顺其自然
2020-03-21

老师有个问题请教你，我这边有个业务，合同编号，存在redis 中和数据库中，每次先查redis 获取合同编号后面虚寒，然好加1，保存回去，外去更新数据库，做了数据库合同编号重复，检验，但是每次还是有合通编号重复的，请问这个怎么解决？市并发导致？
展开

作者回复: 使用Redis命令INCR是可以保存原子性的。

如果是GET出来，在程序内加一，在SET回去，确实会存在并发问题。


肥low
2020-03-21

我觉得Cache Aside还是没有解决主从延迟带来的问题😱


南山
2020-03-21

只有一个体会: 清楚各种缓存策略的缺陷，想用缓存只能是结合当前业务来是否用，用什么策略