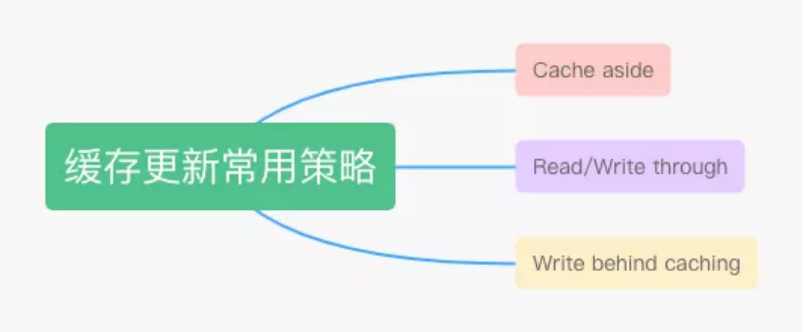
高并发场景下，到底先更新缓存还是先更新数据库？

原创 雷架 爱笑的架构师 1月12日

收录于话题
#玩转Redis面试 11 #图解系列 13

在大型系统中，为了减少数据库压力通常会引入缓存机制，一旦引入缓存又很容易造成缓存和数据库数据不一致，导致用户看到的是旧数据。

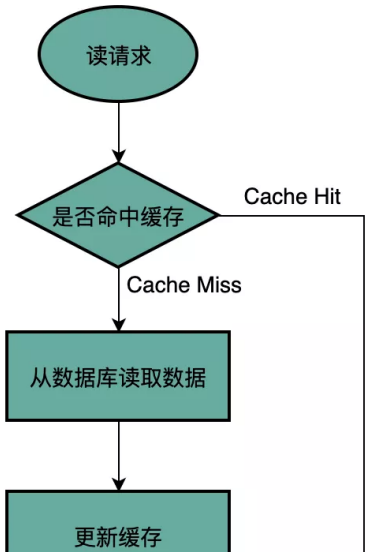
为了减少数据不一致的情况，更新缓存和数据库的机制显得尤为重要，接下来带领大家踩踩坑。

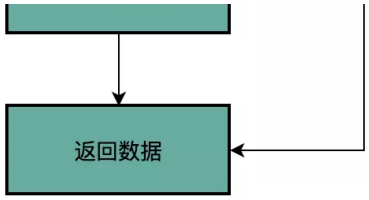


Cache aside

`Cache aside` 也就是 旁路缓存 ，是比较常用的缓存策略。

(1) 读请求 常见流程

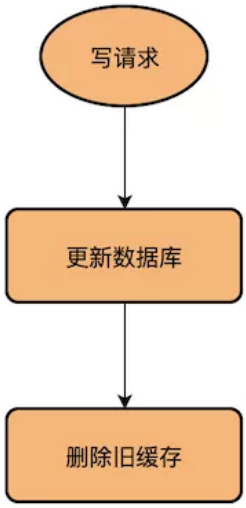




Cache aside 读请求

应用首先会判断缓存是否有该数据，缓存命中直接返回数据，缓存未命中即缓存穿透到数据库，从数据库查询数据然后回写到缓存中，最后返回数据给客户端。

（2） 写请求 常见流程



Cache aside 写请求

首先更新数据库，然后从缓存中删除该数据。

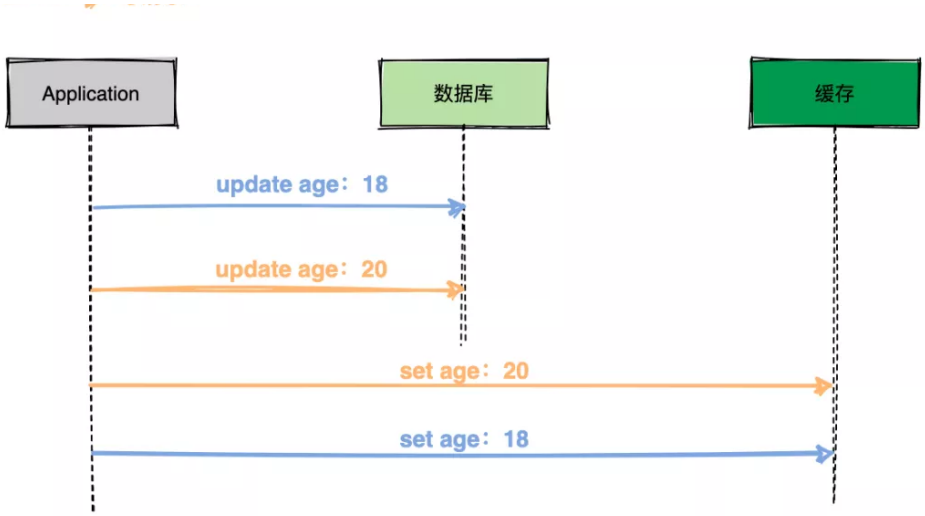
看了写请求的图之后，有些同学可能要问了：为什么要删除缓存，直接更新不就行了？这里涉及到几个坑，我们一步一步踩下去。

Cache aside踩坑

Cache aside策略如果用错就会遇到深坑，下面我们来逐个踩。

踩坑一：先更新数据库，再更新缓存

如果同时有两个 写请求 需要更新数据，每个写请求都先更新数据库再更新缓存，在并发场景可能会出现数据不一致的情况。



先更新数据库，再更新缓存

如上图的执行过程：

（1） 写请求1 更新数据库，将 age 字段更新为18；

（2） 写请求2 更新数据库，将 age 字段更新为20；

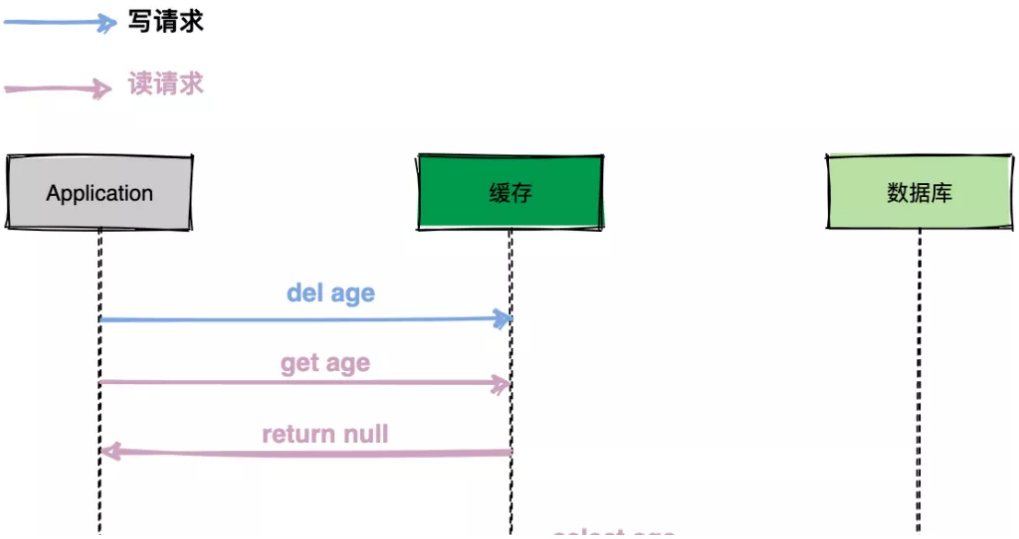
（3） 写请求2 更新缓存，缓存 age 设置为20；

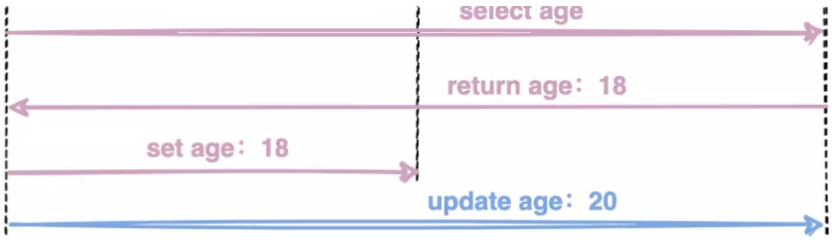
（4） 写请求1 更新缓存，缓存 age 设置为18；

执行完预期结果是数据库 age 为20，缓存 age 为20，结果缓存 age为18，这就造成了缓存数据不是最新的，出现了脏数据。

踩坑二：先删缓存，再更新数据库

如果 写请求 的处理流程是 先删缓存再更新数据库 ，在一个 读请求 和一个 写请求 并发场景下可能会出现数据不一致情况。





先删缓存，再更新数据库

如上图的执行过程：

（1） 写请求 删除缓存数据；

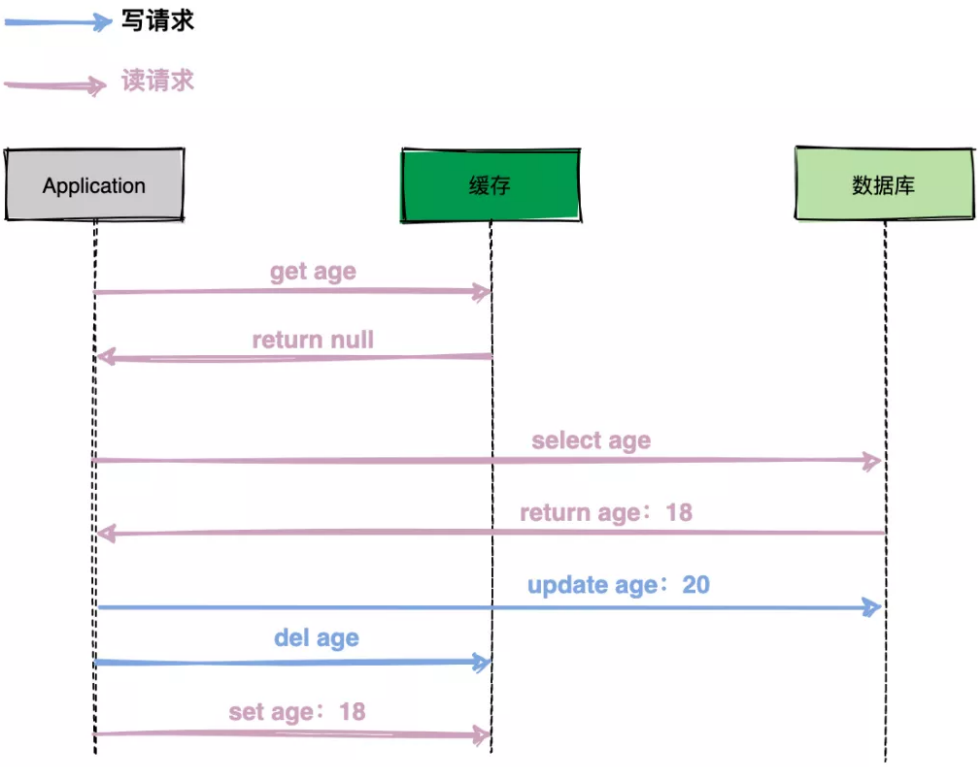
（2） 读请求 查询缓存未击中(Hit Miss)，紧接着查询数据库，将返回的数据回写到缓存中；

（3） 写请求 更新数据库。

整个流程下来发现 数据库 中age为20， 缓存 中age为18，缓存和数据库数据不一致，缓存出现了脏数据。

踩坑三：先更新数据库，再删除缓存

在实际的系统中针对 写请求 还是推荐 先更新数据库再删除缓存 ，但是在理论上还是存在问题，以下面这个例子说明。



先更新数据库，再删除缓存

如上图的执行过程：

（1） 读请求 先查询缓存，缓存未击中，查询数据库返回数据；

（2） 写请求 更新数据库，删除缓存；

（3） 读请求 回写缓存；

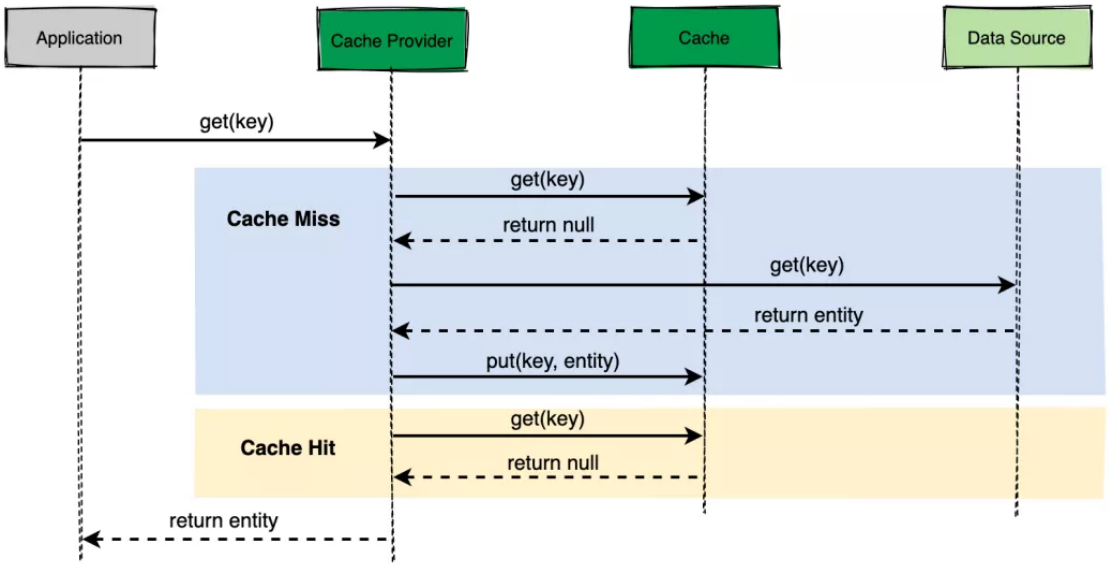
整个流程操作下来发现 数据库age为20 ， 缓存age为18 ，即数据库与缓存不一致，导致应用程序从缓存中读到的数据都为旧数据。

但我们仔细想一下，上述问题发生的概率其实非常低，因为通常数据库更新操作比内存操作耗时多出几个数量级，上图中最后一步回写缓存（set age 18）速度非常快，通常会在更新数据库之前完成。

如果这种极端场景出现了怎么办？我们得想一个兜底的办法： 缓存数据设置过期时间 。通常在系统中是可以允许少量的数据短时间不一致的场景出现。

Read through

在 Cache Aside 更新模式中，应用代码需要维护两个数据源头：一个是缓存，一个是数据库。而在 `Read-Through` 策略下，应用程序无需管理缓存和数据库，只需要将数据库的同步委托给缓存提供程序 `Cache Provider` 即可。所有数据交互都是通过 抽象缓存层 完成的。



Read-Through流程

如上图，应用程序只需要与 `Cache Provider` 交互，不用关心是从缓存取还是数据库。

在进行大量读取时， `Read-Through` 可以减少数据源上的负载，也对缓存服务的故障具备一定的弹性。如果缓存服务挂了，则缓存提供程序仍然可以通过直接转到数据源来进行操作。

`Read-Through 适用于多次请求相同数据的场景` ，这与 Cache-Aside 策略非常相似，但是二者还是存在一些差别，这里再次强调一下：

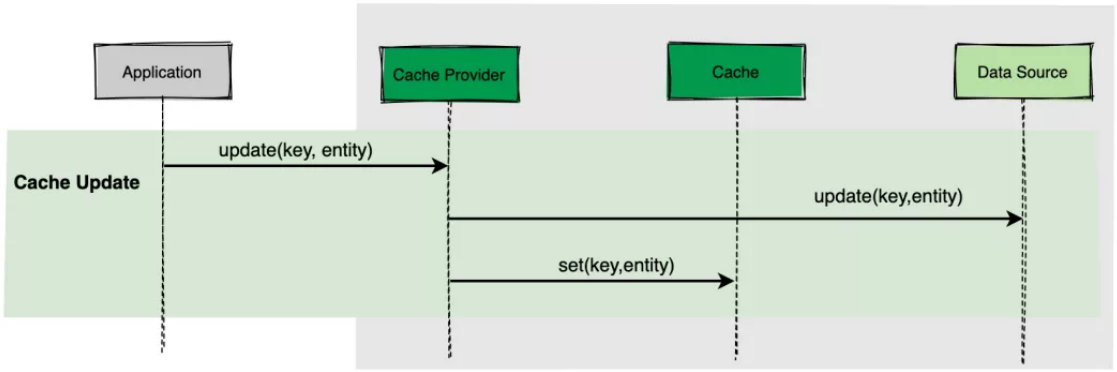
- 在 Cache-Aside 中，应用程序负责从数据源中获取数据并更新到缓存。
- 在 Read-Through 中，此逻辑通常是由独立的缓存提供程序（Cache Provider）支持。

Write through

`Write-Through` 策略下，当发生数据更新(Write)时，缓存提供程序 `Cache Provider` 负责更新底层数据源和缓存。

缓存与数据源保持一致，并且写入时始终通过 抽象缓存层 到达数据源。

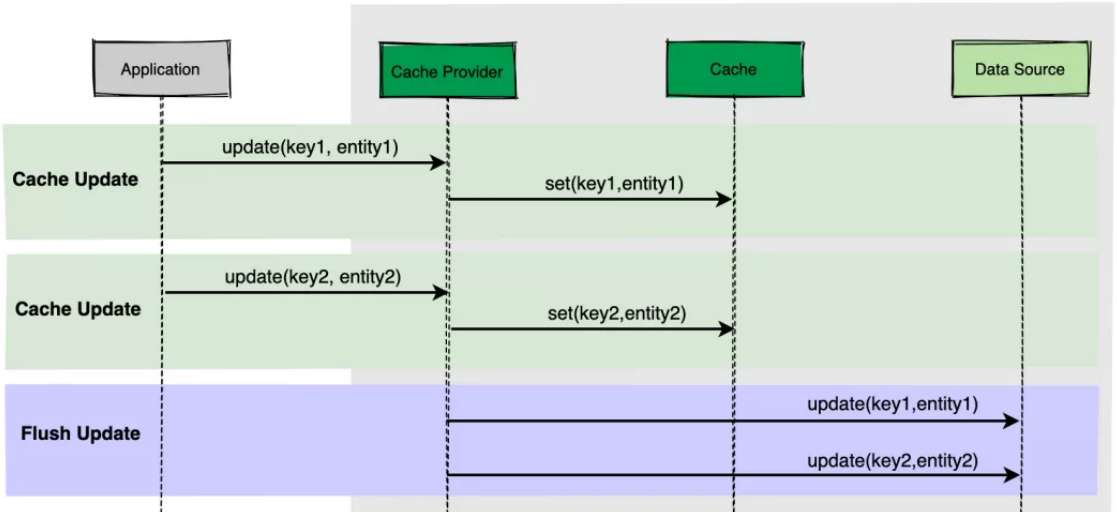
`Cache Provider` 类似一个代理的作用。



Write-Through流程

Write behind

`Write behind` 在一些地方也被成为 `Write back` ， 简单理解就是：应用程序更新数据时只更新缓存， `Cache Provider` 每隔一段时间将数据刷新到数据库中。说白了就是 延迟写入 。



Write behind流程

如上图，应用程序更新两个数据，Cache Provider 会立即写入缓存中，但是隔一段时间才会批量写入数据库中。

这种方式有优点也有缺点：

- 优点 是数据写入速度非常快，适用于频繁写的场景。
- 缺点 是缓存和数据库不是强一致性，对一致性要求高的系统慎用。

总结一下

学了这么多，相信大家对缓存更新的策略都已经有了清晰的认识。最后稍稍总结一下。

缓存更新的策略主要分为三种：

- Cache aside
- Read/Write through
- Write behind

Cache aside 通常会先更新数据库，然后再删除缓存，为了兜底通常还会将数据设置缓存时间。

Read/Write through 一般是由一个 Cache Provider 对外提供读写操作，应用程序不用感知操作的是缓存还是数据库。

Write behind简单理解就是延迟写入，Cache Provider 每隔一段时间会批量输入数据库，优点是应用程序写入速度非常快。

好了，今天先到这里了，大家学会了吗？

- END -

日常求赞求关注环节：

这篇文章图画的真可爱，你们觉得呢？扫描二维码添加我的个人微信，可以与我一对一深度交流。需要进群的备注一下【学技术】，群里有 BAT 等一线大厂大佬哦，不说话光看就能学东西。

原创不易，点个【分享】【在看】鼓励支持一下，爱你们哦~

收录于话题 #玩转Redis面试·11个

上一篇
『玩转Redis系列』雷架整理一套组合拳包通关

下一篇
一张图搞懂Redis缓存雪崩、缓存穿透、缓存击穿

喜欢此内容的人还喜欢

Redis八股文背诵版v0.2
后端技术小牛说

大厂动态规划面试汇总，重量级干货，彻夜整理
盼盼编程