看到很多小伙伴简历上写了“**熟练使用缓存**”，但是被我问到“**缓存常用的3种读写策略**”的时候却一脸懵逼。

在我看来，造成这个问题的原因是我们在学习 Redis 的时候，可能只是简单了写一些 Demo，并没有去关注缓存的读写策略，或者说压根不知道这回事。

但是，搞懂3种常见的缓存读写策略对于实际工作中使用缓存以及面试中被问到缓存都是非常有帮助的！

下面我会简单介绍一下自己对于这 3 种缓存读写策略的理解。

另外，**这3 种缓存读写策略各有优劣，不存在最佳，需要我们根据具体的业务场景选择更适合的。**

个人能力有限。如果文章有任何需要补充/完善/修改的地方，欢迎在评论区指出，共同进步！——爱你们的 Guide 哥

Cache Aside Pattern（旁路缓存模式）

Cache Aside Pattern 是我们平时使用比较多的一个缓存读写模式，比较适合读请求比较多的场景。

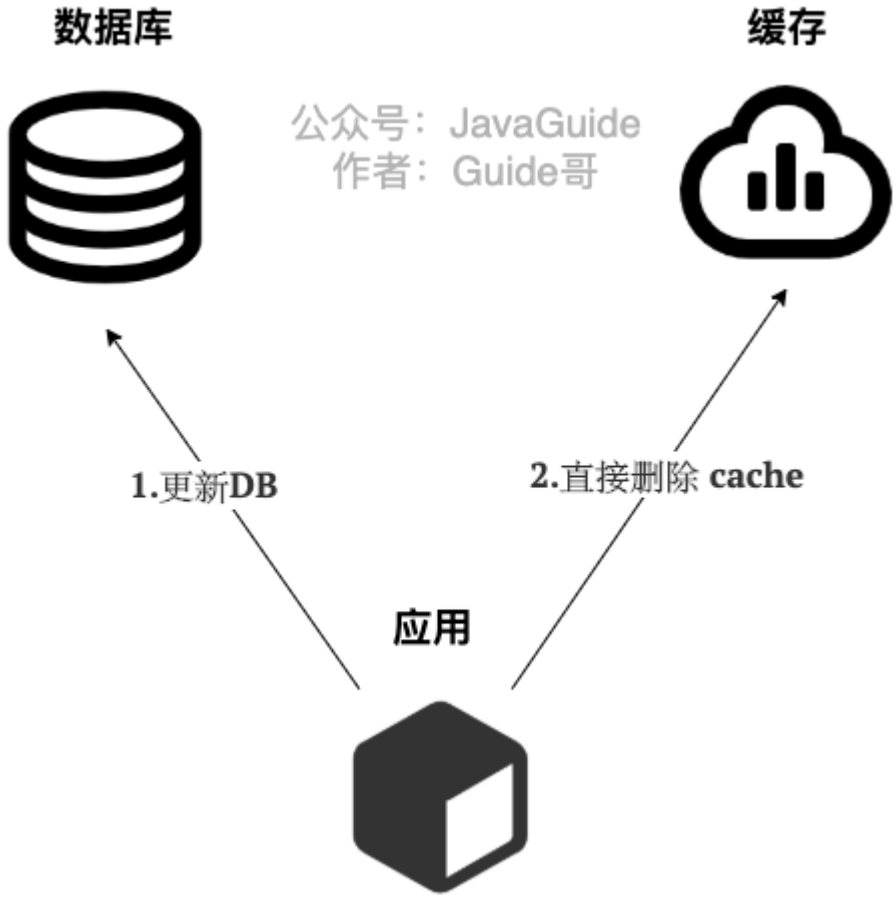
Cache Aside Pattern 中服务端需要同时维系 DB 和 cache，并且是以 DB 的结果为准。

下面我们来看一下这个策略模式下的缓存读写步骤。

写：

- 先更新 DB
- 然后直接删除 cache 。

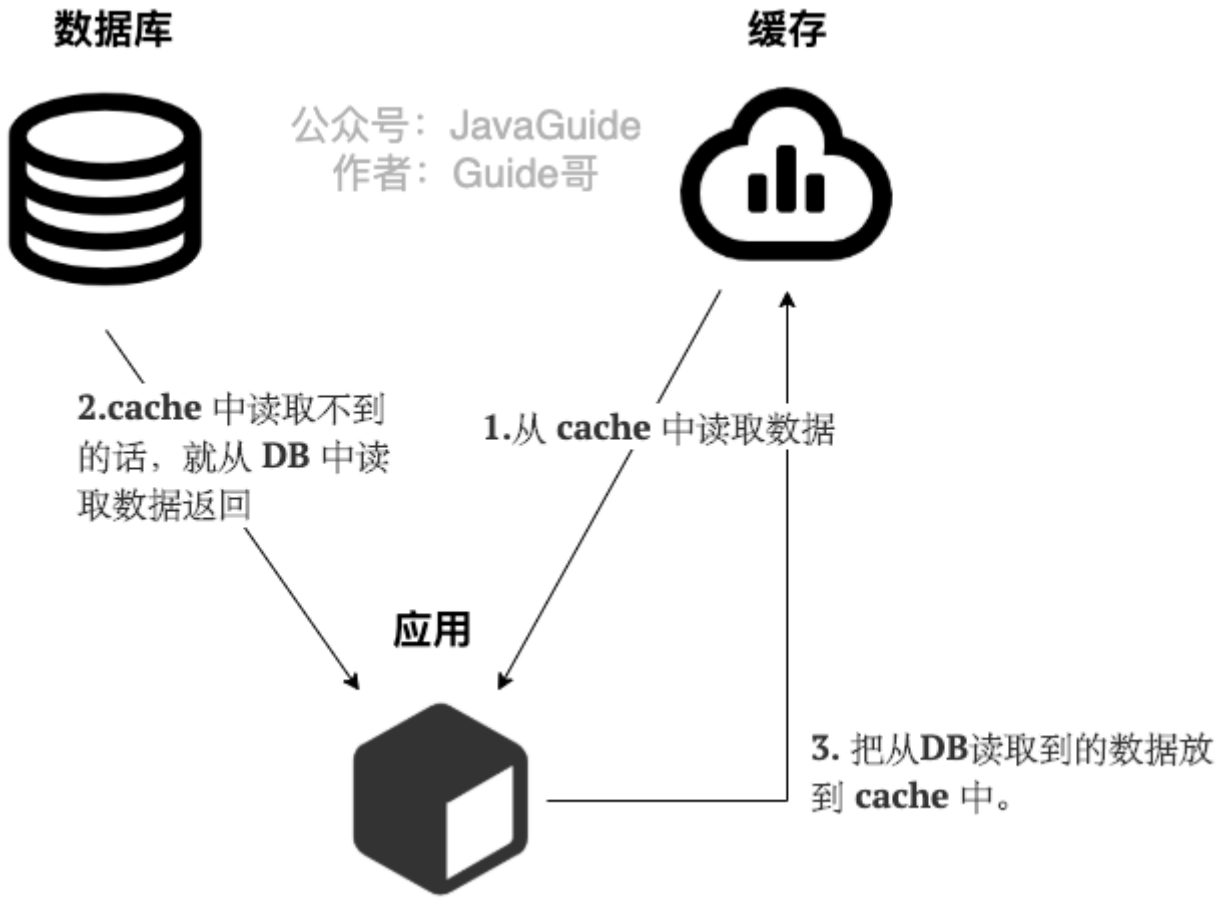
简单画了一张图帮助大家理解写的步骤。



读：

- 从 cache 中读取数据，读取到就直接返回
- cache中读取不到的话，就从 DB 中读取数据返回
- 再把数据放到 cache 中。

简单画了一张图帮助大家理解读的步骤。



你仅仅了解了上面这些内容的话是远远不够的，我们还要搞懂其中的原理。

比如说面试官很可能会追问：“**在写数据的过程中，可以先删除 cache ，后更新 DB 么?**”

答案： 那肯定是不行的！因为这样可能会造成**数据库（DB）和缓存（Cache）数据不一致**的问题。为什么呢？比如说请求1 先写数据A，请求2随后读数据A的话就很有可能产生数据不一致性的问题。这个过程可以简单描述为：

> 请求1先把cache中的A数据删除 -> 请求2从DB中读取数据->请求1再把DB中的A数据更新。

当你这样回答之后，面试官可能会紧接着就追问：“**在写数据的过程中，先更新DB，后删除cache就没有问题了么?**”

答案： 理论上来说还是可能会出现数据不一致性的问题，不过概率非常小，因为缓存的写入速度是比数据库的写入速度快很多！

比如请求1先读数据 A，请求2随后写数据A，并且数据A不在缓存中的话也有可能产生数据不一致性的问题。这个过程可以简单描述为：

> 请求1从DB读数据A->请求2写更新数据 A 到数据库并把删除cache中的A数据->请求1将数据A写入cache。

现在我们再来分析一下 **Cache Aside Pattern 的缺陷。**

缺陷1：首次请求数据一定不在 cache 的问题

解决办法：可以将热点数据可以提前放入cache 中。

缺陷2：写操作比较频繁的话导致cache中的数据会被频繁被删除，这样会影响缓存命中率 。

解决办法：

- 数据库和缓存数据强一致场景： 更新DB的时候同样更新cache，不过我们需要加一个锁/分布式锁来保证更新cache的时候不存在线程安全问题。 - 可以短暂地允许数据库和缓存数据不一致的场景： 更新DB的时候同样更新cache，但是给缓存加一个比较短的过期时间，这样的话就可以保证即使数据不一致的话影响也比较小。

Read/Write Through Pattern（读写穿透）

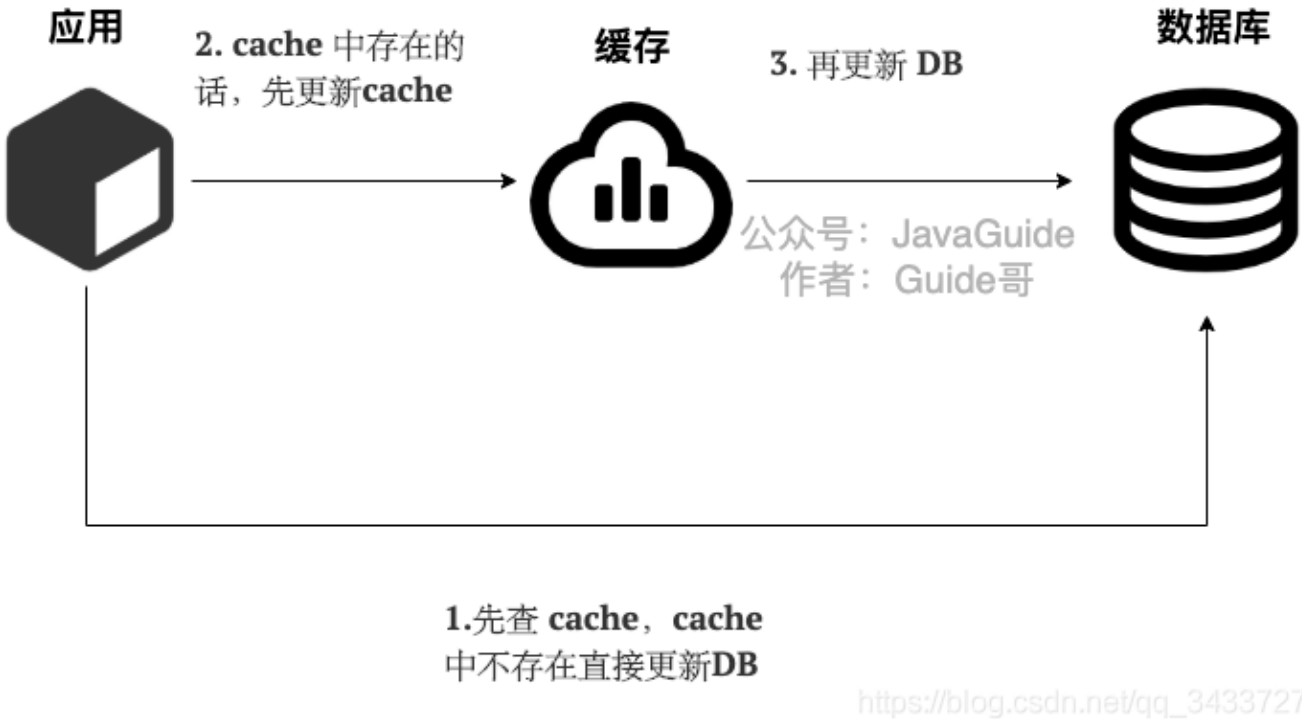
Read/Write Through Pattern 中服务端把 cache 视为主要数据存储，从中读取数据并将数据写入其中。cache 服务负责将此数据读取和写入 DB，从而减轻了应用程序的职责。

这种缓存读写策略小伙伴们应该也发现了在平时在开发过程中非常少见。抛去性能方面的影响，大概率是因为我们经常使用的分布式缓存 Redis 并没有提供 cache 将数据写入DB的功能。

写（Write Through）：

- 先查 cache，cache 中不存在，直接更新 DB。
- cache 中存在，则先更新 cache，然后 cache 服务自己更新 DB（**同步更新 cache 和 DB**）。

简单画了一张图帮助大家理解写的步骤。



读(Read Through)：

- 从 cache 中读取数据，读取到就直接返回 。
- 读取不到的话，先从 DB 加载，写入到 cache 后返回响应。

简单画了一张图帮助大家理解读的步骤。

Read-Through Pattern 实际只是在 Cache-Aside Pattern 之上进行了封装。在 Cache-Aside Pattern 下，发生读请求的时候，如果 cache 中不存在对应的数据，是由客户端自己负责把数据写入 cache，而 Read Through Pattern 则是 cache 服务自己来写入缓存的，这对客户端是透明的。

和 Cache Aside Pattern 一样， Read-Through Pattern 也有首次请求数据一定不再 cache 的问题，对于热点数据可以提前放入缓存中。

Write Behind Pattern（异步缓存写入）

Write Behind Pattern 和 Read/Write Through Pattern 很相似，两者都是由 cache 服务来负责 cache 和 DB 的读写。

但是，两个又有很大的不同：**Read/Write Through 是同步更新 cache 和 DB，而 Write Behind Caching 则是只更新缓存，不直接更新 DB，而是改为异步批量的方式来更新 DB。**

很明显，这种方式对数据一致性带来了更大的挑战，比如cache数据可能还没异步更新DB的话，cache服务可能就就挂掉了。

这种策略在我们平时开发过程中也非常非常少见，但是不代表它的应用场景少，比如消息队列中消息的异步写入磁盘、MySQL 的 InnoDB Buffer Pool 机制都用到了这种策略。

Write Behind Pattern 下 DB 的写性能非常高，非常适合一些数据经常变化又对数据一致性要求没那么高的场景，比如浏览量、点赞量。