常见的缓存更新策略

1、旁路缓存策略

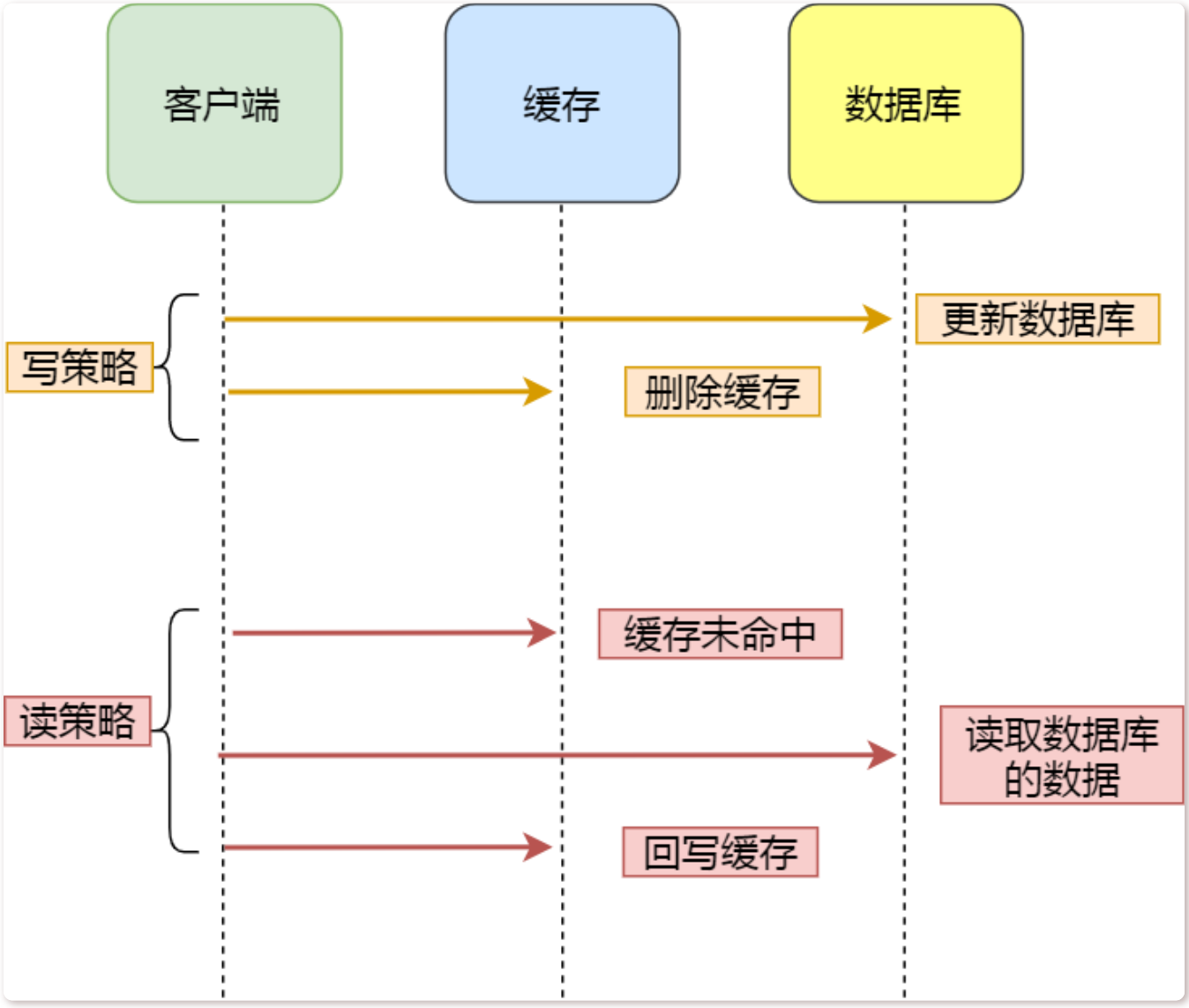
1.1、介绍

Cache Aside（旁路缓存）策略是最常用的，**应用程序直接与「数据库、缓存」交互，并负责对缓存的维护**，该策略又可以细分为「**读策略**」和「**写策略**」。

写策略：先更新数据库中的数据，再删除缓存中的数据。

读策略：

- 如果读取的数据命中了缓存，则直接返回数据；
- 如果读取的数据没有命中缓存，则从数据库中读取数据，然后将数据写入到缓存，并且返回给用户。

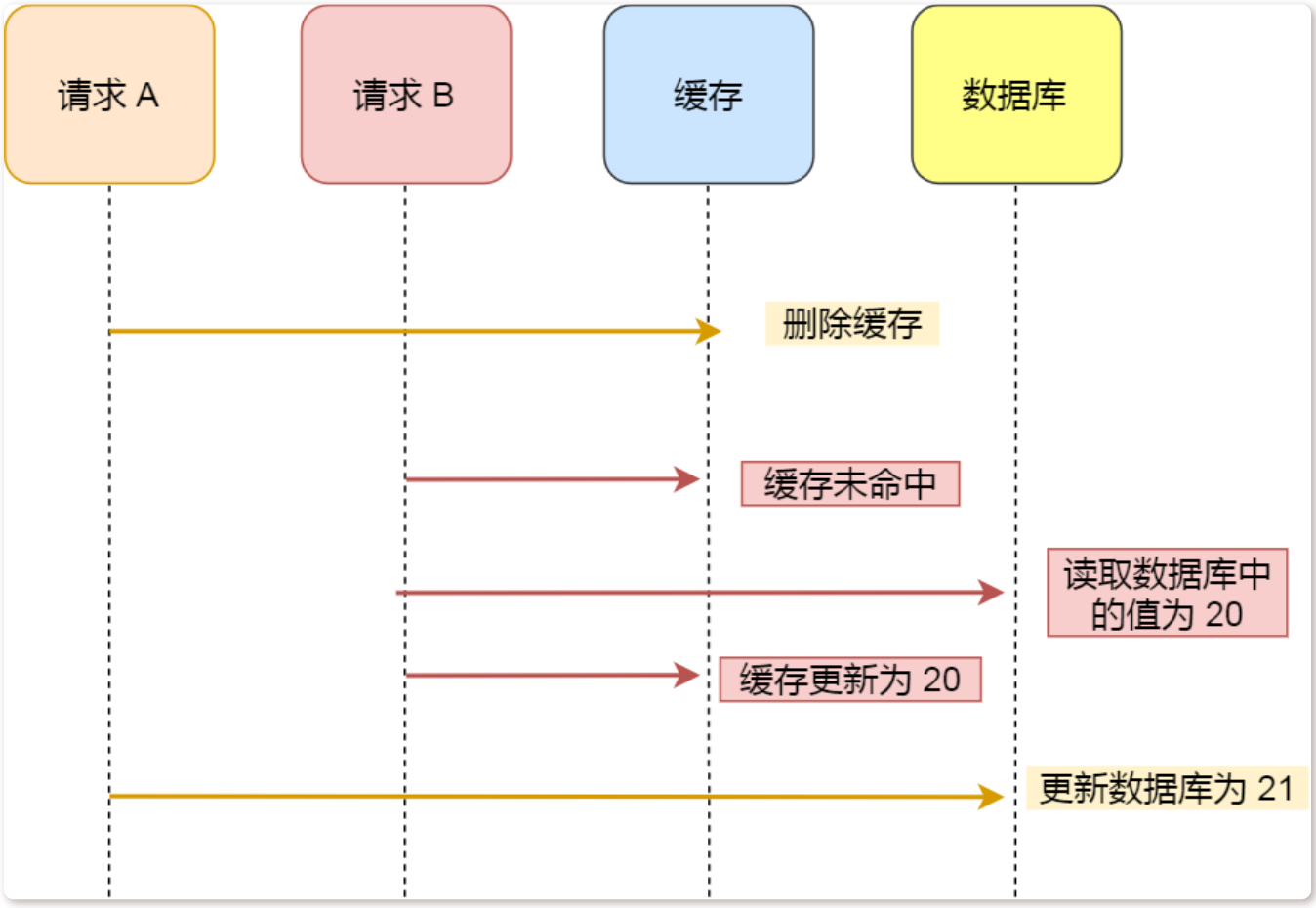


> 🌈
>
> 注意，写策略的步骤的顺序不能倒过来，即**不能先删除缓存再更新数据库**，原因是在「读+写」并发的时候，会出现缓存和数据库的数据不一致性的问题。

1.2、先删缓存后更库

前面提及到写策略的步骤的顺序不能倒过来，即**不能先删除缓存再更新数据库**，这里举一个例子演示：

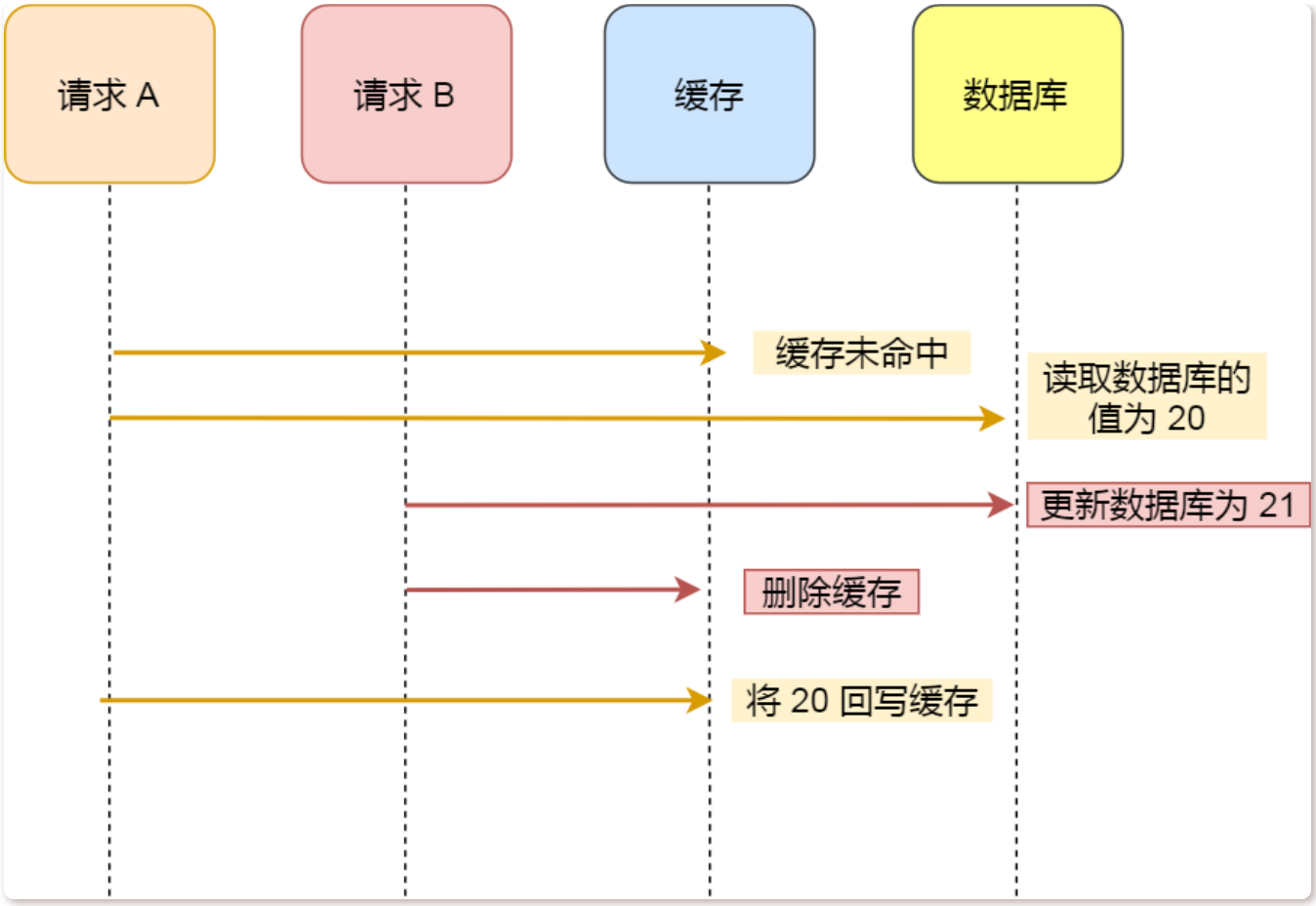
- 假设某个用户的年龄是20，**请求A**要**更新**用户年龄为21，所以它会**删除缓存**中的内容。
- 这时，另一个**请求B**要**读取**这个用户的年龄，它查询**缓存发现未命中**后，会从数据库中读取到年龄为20，并且**写回到缓存**中。
- **请求A**继续**更改数据库**，将用户的年龄更新为21。
- 最终，该用户年龄在**缓存**中是20（**旧值**），在**数据库**中是21（**新值**），缓存和数据库的**数据不一致**。



1.3、先更库后删缓存

那么**「先更新数据库再删除缓存」一定不会有数据不一致的问题吗**？继续用「读 + 写」请求的并发的场景来分析：

- 假如某个用户数据在**缓存中不存在**，**请求A读取数据**时从数据库中查询到年龄为 20。
- **在未写入缓存中时**另一个**请求B更新**数据。**请求B更新数据库**中的年龄为 21，并且**清空缓存**。
- 这时**请求A**把从数据库中读到的年龄为20的数据**写入到缓存**中。
- 最终，该用户年龄在**缓存**中是 20（**旧值**），在**数据库**中是 21（**新值**），缓存和数据库**数据不一致**。



从上面的理论上分析，先更新数据库，再删除缓存也是会出现数据不一致性的问题，**但是在实际中，这个问题出现的概率并不高**。这主要有以下原因：

- **缓存的写入通常要远远快于数据库的写入**。
- 所以在实际中很难出现**请求B**已经**更新了数据库并且删除了缓存**，**请求A**才**更新完缓存**的情况。
- 而一旦**请求A早于请求B删除缓存之前更新了缓存**，那么接下来的请求就会因为**缓存不命中**而从数据库中**重新读取数据**，所以**一般不会出现**这种不一致的情况。

1.4、使用场景

Cache Aside 策略适合读多写少的场景，不适合写多的场景，因为当写入比较频繁时，缓存中的数据会被频繁地清理，这样会对缓存的命中率有一些影响。如果业务对缓存命中率有严格的要求，那么可以考虑两种解决方案：

- 一种做法是在**更新数据时也更新缓存**，只是**在更新缓存前**先加一个**分布式锁**。因为这样在同一时间只允许一个线程更新缓存，就不会产生并发问题了。当然这么做对于写入的性能会有一些影响；
- 另一种做法同样也是在**更新数据时更新缓存**，只是**给缓存加一个较短的过期时间**。这样即使出现缓存不一致的情况，缓存的数据也会很快过期，对业务的影响也是可以接受。

2、读穿/写穿策略

2.1、介绍

Read/Write Through（读穿/写穿）策略原则是**应用程序只和缓存交互，不再和数据库交互**，而是由缓存和数据库交互，相当于更新数据库的操作由缓存自己代理了。

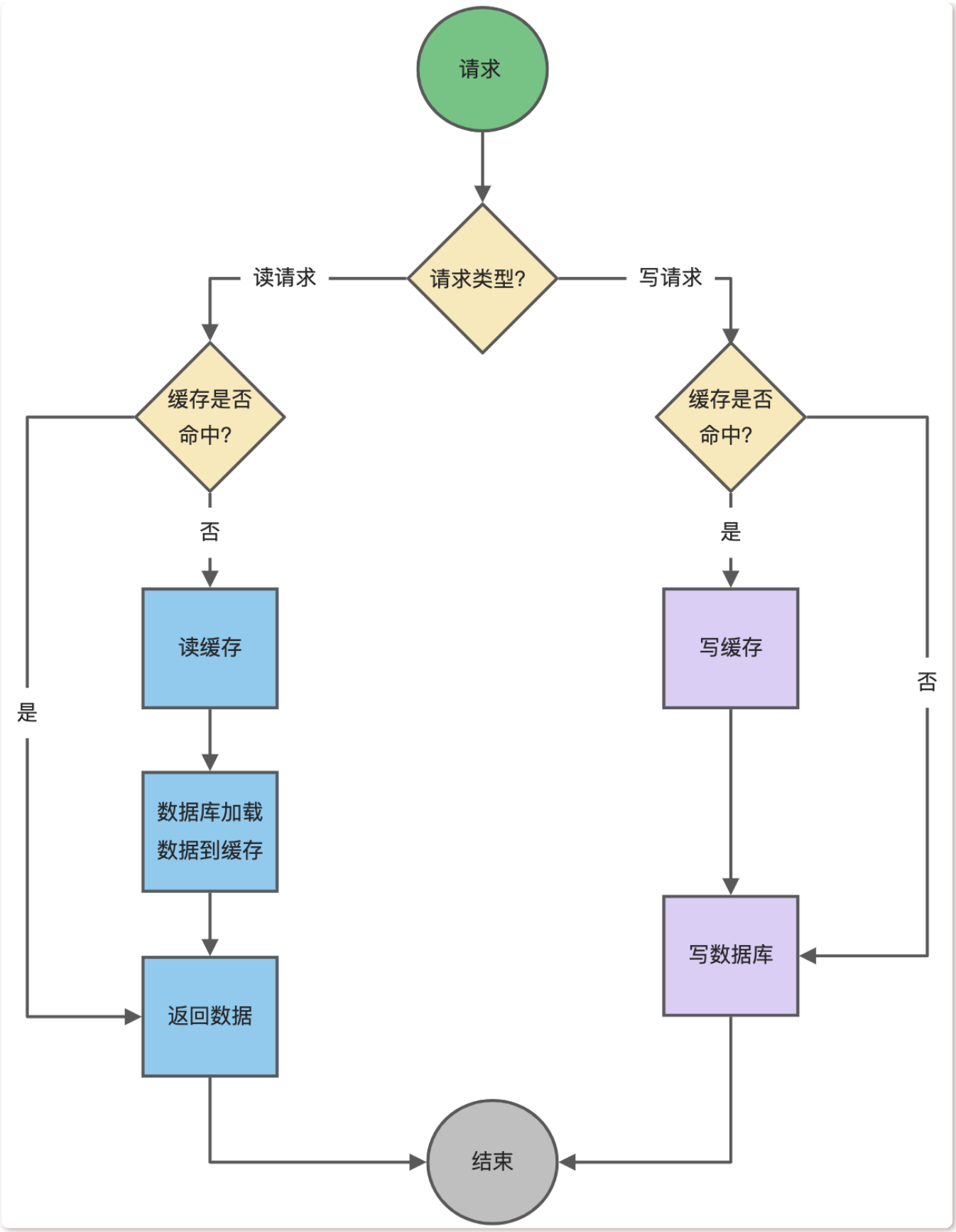
2.2、Read Through策略

先查询缓存中数据是否存在，**如果存在**则直接返回，**如果不存在**则由缓存组件负责从数据库查询数据，并将结果写入到缓存组件，**最后**缓存组件将数据返回给应用。

2.3、Write Through策略

当有**数据更新**的时候，先查询要写入的数据在**缓存中是否已经存在**：

- 如果缓存中数据**已经存在**，则更新缓存中的数据，并且由缓存组件同步更新到数据库中，然后缓存组件告知应用程序更新完成。
- 如果缓存中数据**不存在**，直接更新数据库，然后返回；



3、写回策略

Write Back（写回）策略在更新数据的时候，**只更新缓存**，同时将缓存数据设置为脏的，然后立马返回，并不会更新数据库。**对于数据库的更新**，会通过批量**异步更新**的方式进行。

实际上，Write Back（写回）策略一般不会直接应用到我们常用的数据库和缓存的场景中，因为 Redis 并没有异步更新数据库的功能，一般情况下会配合消息队列等具有**异步通信功能**的服务进行使用。

Write Back 是计算机体系结构中的设计，比如 CPU 的缓存、操作系统中文件系统的缓存都采用了 Write Back（写回）策略。

Write Back 策略特别适合写多的场景，因为发生写操作的时候， 只需要更新缓存，就立马返回了。比如，写文件的时候，实际上是写入到文件系统的缓存就返回了，并不会写磁盘。

但是带来的问题是，数据不是强一致性的，而且会有数据丢失的风险，因为缓存一般使用内存，而内存是非持久化的，所以一旦缓存机器掉电，就会造成原本缓存中的脏数据丢失。所以你会发现系统在掉电之后，之前写入的文件会有部分丢失，就是因为 Page Cache 还没有来得及刷盘造成的。