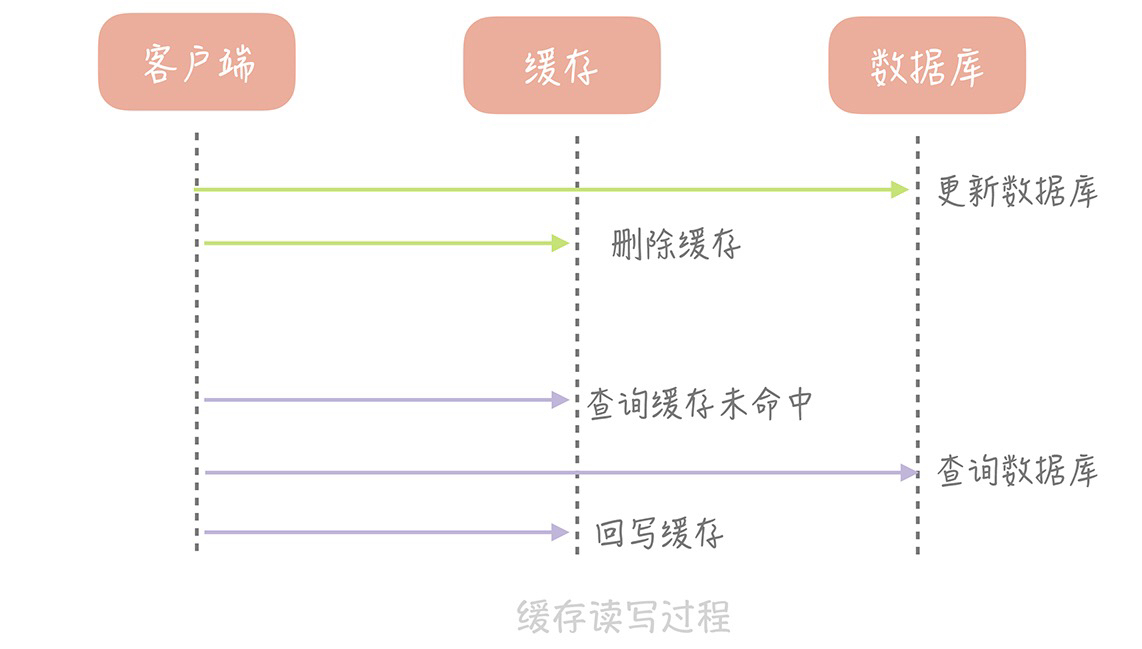
读写策略

Cache Aside（旁路缓存）策略



读策略：

1. 从缓存中读取数据；

2. 如果缓存命中，则直接返回数据；

3. 如果缓存不命中，则从数据库中查询数据；

4. 查询到数据后，将数据写入到缓存中，并且返回给用户。

写策略：

1. 更新数据库中的记录；

2. 删除缓存记录。

Cache Aside 存在的最大的问题是当写入比较频繁时，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：

1. 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；

2. 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。

Read/Write Through（读穿 / 写穿）策略

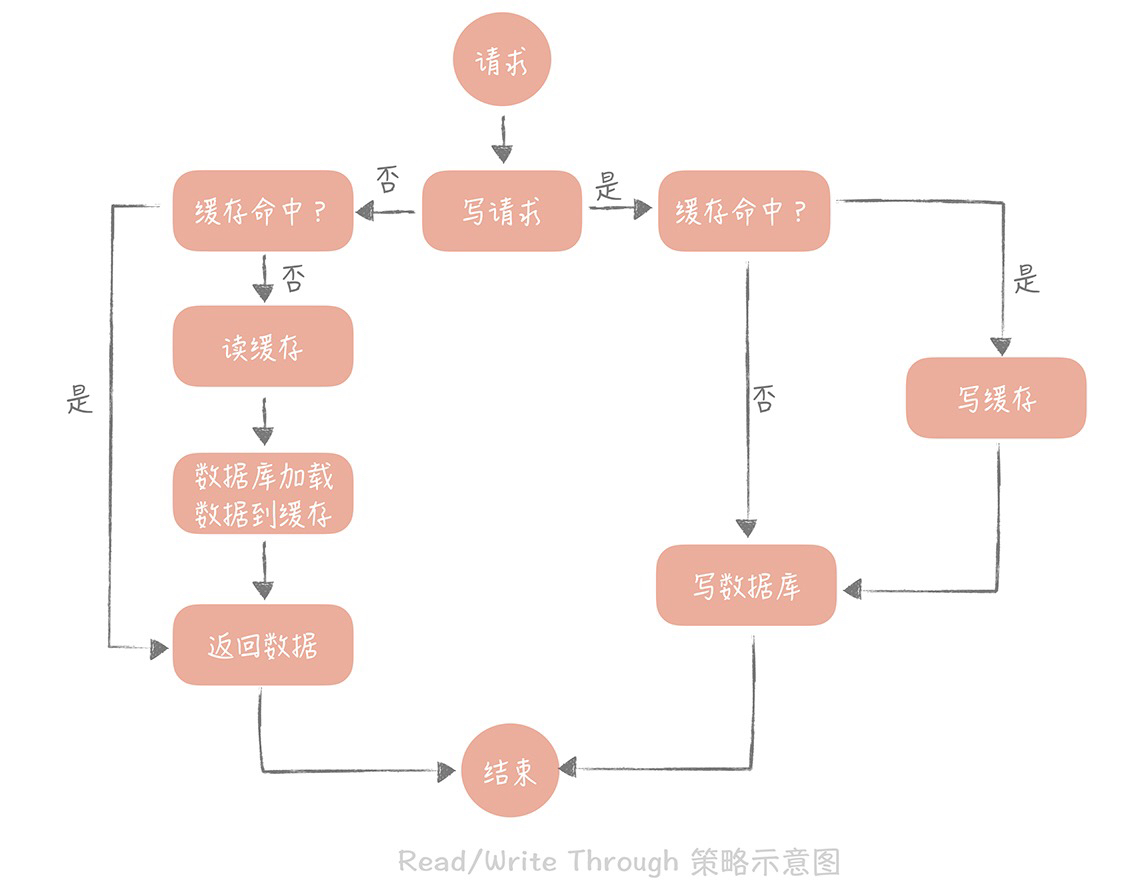
核心原则是用户只与缓存打交道，由缓存和数据库通信，写入或者读取数据。这就好比你在汇报工作的时候只对你的直接上级汇报，再由你的直接上级汇报给他的上级，你是不能越级汇报的。

Write Through 的策略是这样的：先查询要写入的数据在缓存中是否已经存在，如果已经存在，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，如果缓存中数据不存在，这种情况叫做“Write Miss（写失效）”。

一般来说，可以选择两种“Write Miss”方式：一个是“Write Allocate（按写分配）”，做法是写入缓存相应位置，再由缓存组件同步更新到数据库中；另一个是“No-write allocate（不按写分配）”，做法是不写入缓存中，而是直接更新到数据库中。

在 Write Through 策略中，一般选择“No-write allocate”方式，原因是无论采用哪种“Write Miss”方式，都需要同步将数据更新到数据库中，而“No-write allocate”方式相比“Write Allocate”还减少了一次缓存的写入，能够提升写入的性能。

Read Through 策略就简单一些，它的步骤是这样的：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中同步加载数据。

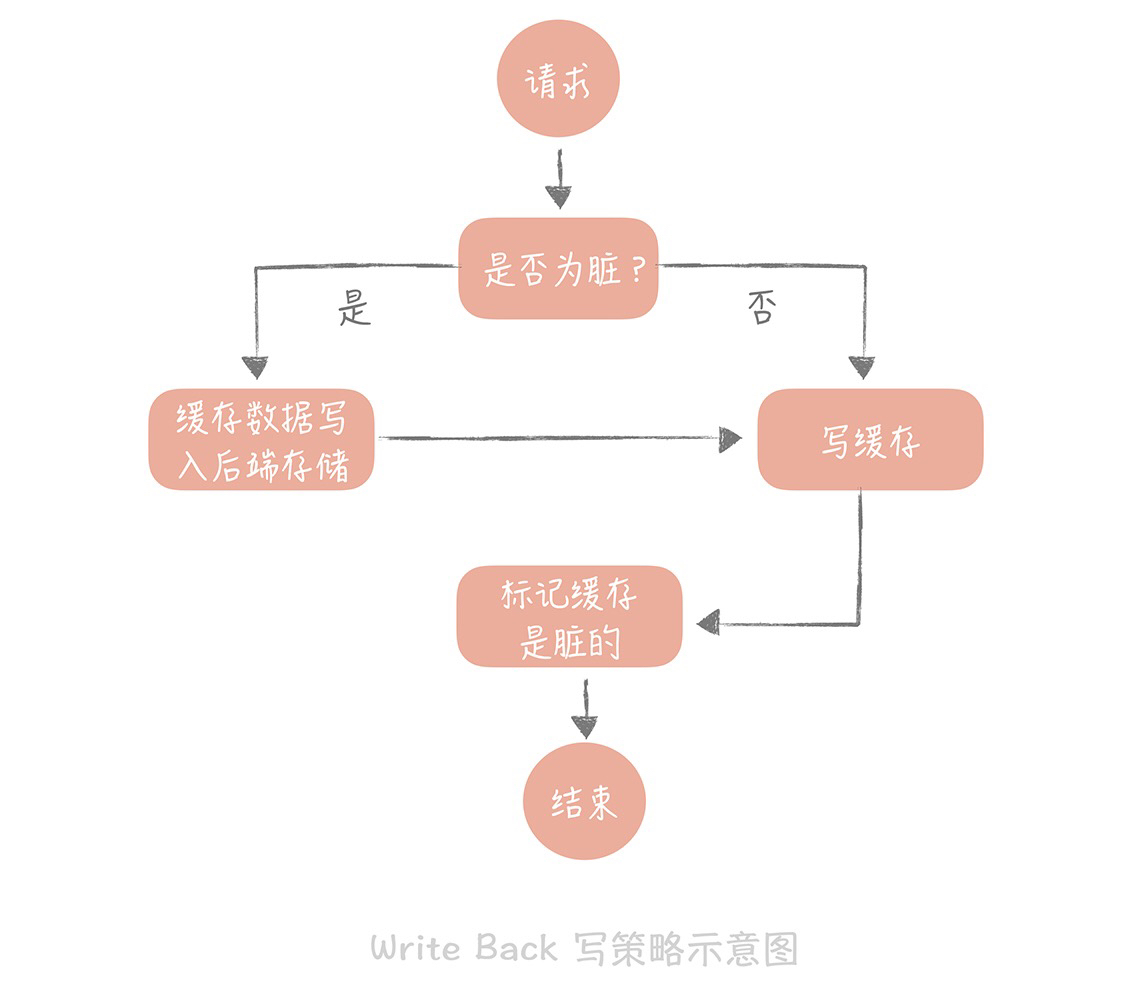


Write Through 策略中写数据库是同步的，这对于性能来说会有比较大的影响，因为相比于写缓存，同步写数据库的延迟就要高很多了。

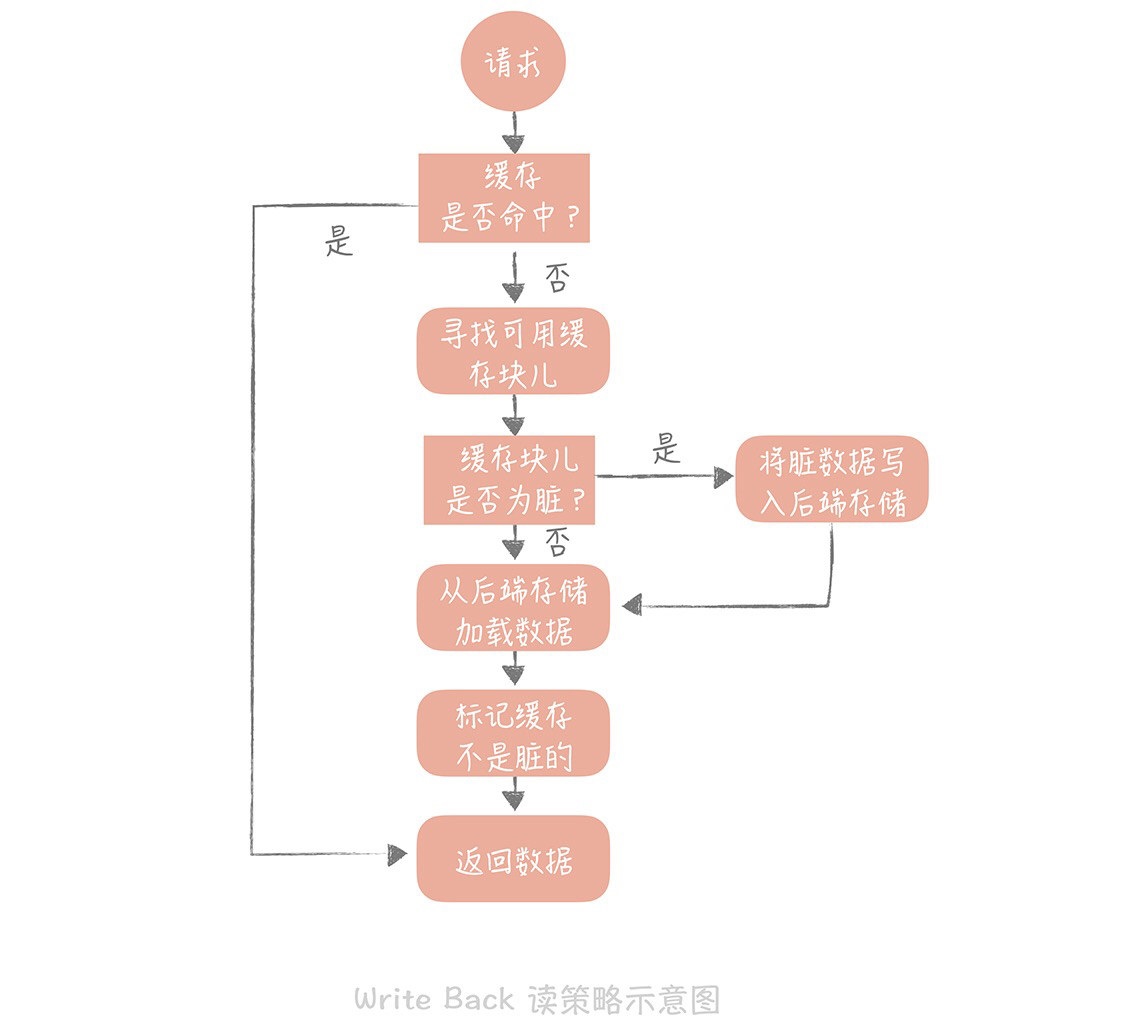
Write Back（写回）策略

核心思想是在写入数据时只写入缓存，并且把缓存块儿标记为“脏”的。而脏块儿只有被再次使用时才会将其中的数据写入到后端存储中。

需要注意的是，在“Write Miss”的情况下，采用的是“Write Allocate”的方式，也就是在写入后端存储的同时要写入缓存，这样在之后的写请求中都只需要更新缓存即可，而无需更新后端存储了，将 Write back 策略的示意图放在了下面：



如果使用 Write Back 策略的话，读的策略也有一些变化了。在读取缓存时如果发现缓存命中则直接返回缓存数据。如果缓存不命中则寻找一个可用的缓存块儿，如果这个缓存块儿是“脏”的，就把缓存块儿中之前的数据写入到后端存储中，并且从后端存储加载数据到缓存块儿，如果不是脏的，则由缓存组件将后端存储中的数据加载到缓存中，最后将缓存设置为不是脏的，返回数据就好了。



小结

1.Cache Aside 是在使用分布式缓存时最常用的策略，可以在实际工作中直接拿来使用。

2.Read/Write Through 和 Write Back 策略需要缓存组件的支持，所以比较适合在实现本地缓存组件的时候使用；

3.Write Back 策略是计算机体系结构中的策略，不过写入策略中的只写缓存，异步写入后端存储的策略倒是有很多的应用场景。