1、何为延时双删

延迟双删（Delay Double Delete）是一种在数据更新或删除时**为了保证数据一致性**而采取的策略。这种策略通常**用于解决数据在缓存和数据库中不一致的问题**。

具体来说，在某些场景下，我们需要先更新或删除数据库中的数据，然后再更新或删除缓存中的数据，以保证数据的一致性。但在某些情况下，由于网络延迟、服务器故障或其他原因，**可能导致缓存中的数据更新或删除失败，从而导致数据库和缓存中的数据不一致**。

> 🌈
>
> 值得注意的是，不管哪种方案，都避免不了Redis存在脏数据的问题，**只能减轻**这个问题，要想彻底解决，得要用到同步锁和对应的业务逻辑层面解决。

2、常用缓存策略

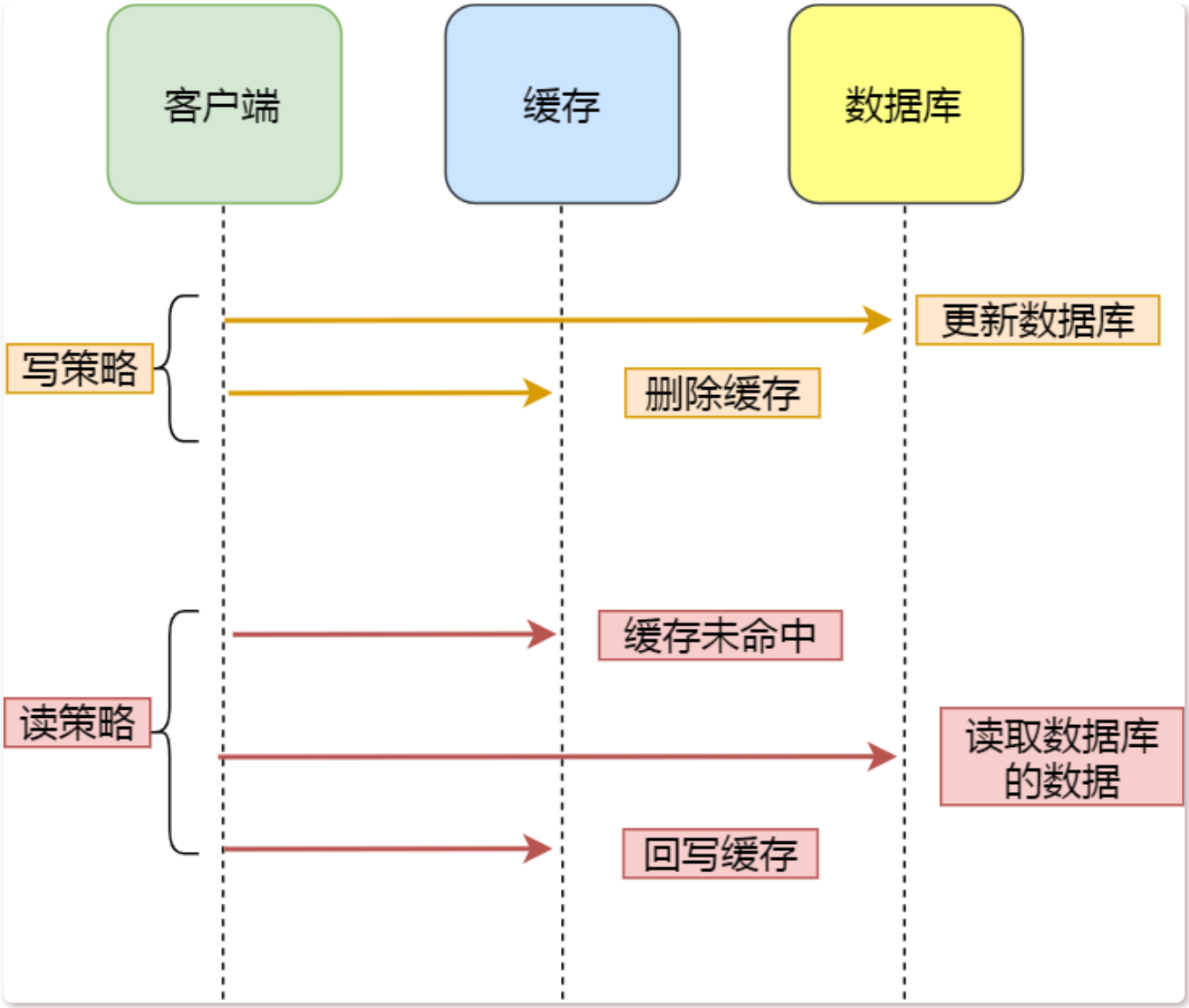
2.1、介绍

这里只提及Cache Aside（旁路缓存）策略，这是我们操作Redis时最常用的一个策略，在该策略中**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。

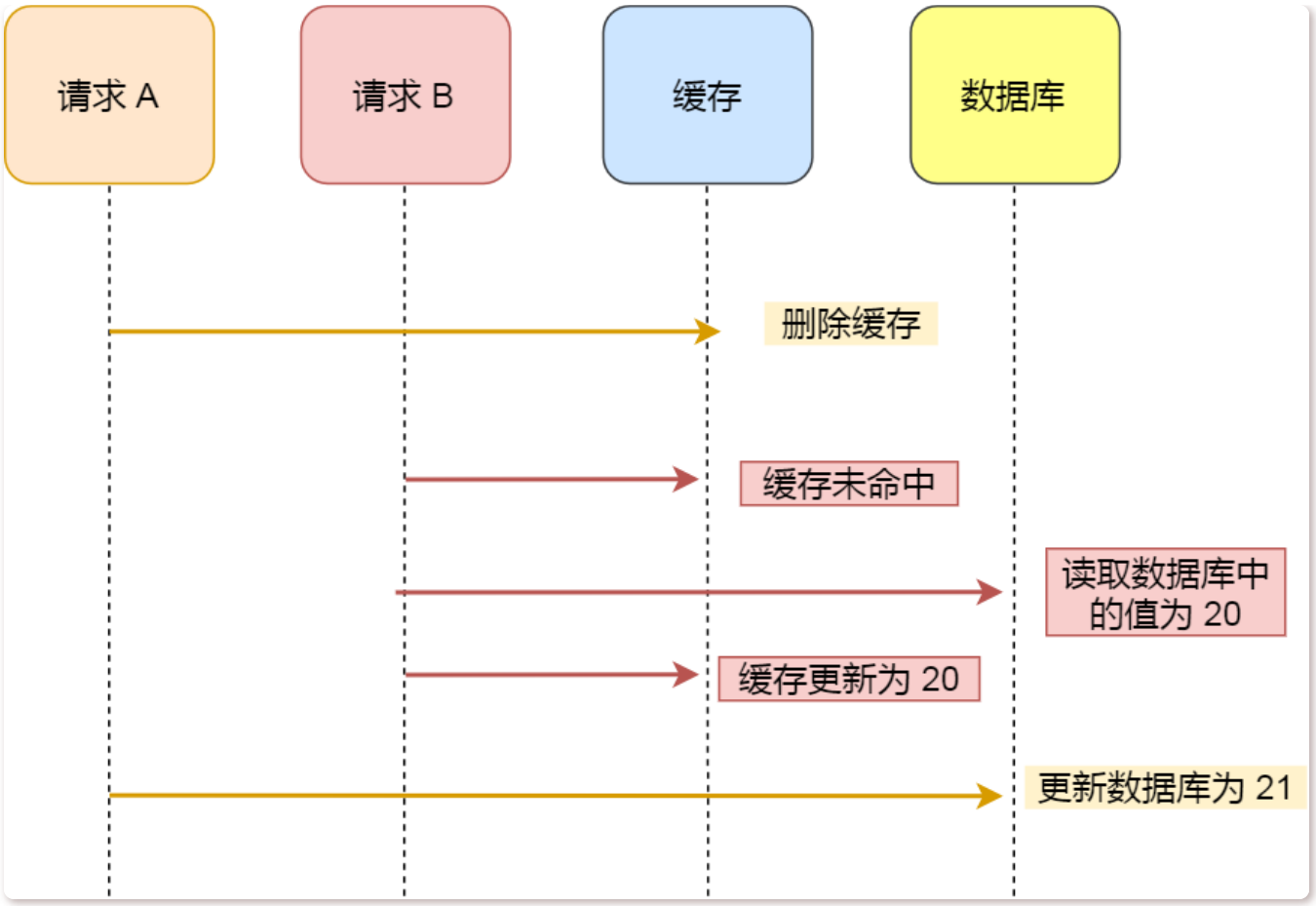


> 🌈
>
> 注意，写策略的步骤的顺序不能倒过来，即**不能先删除缓存再更新数据库**，原因是在「读+写」并发的时候，会出现缓存和数据库的数据不一致性的问题。

2.2、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：

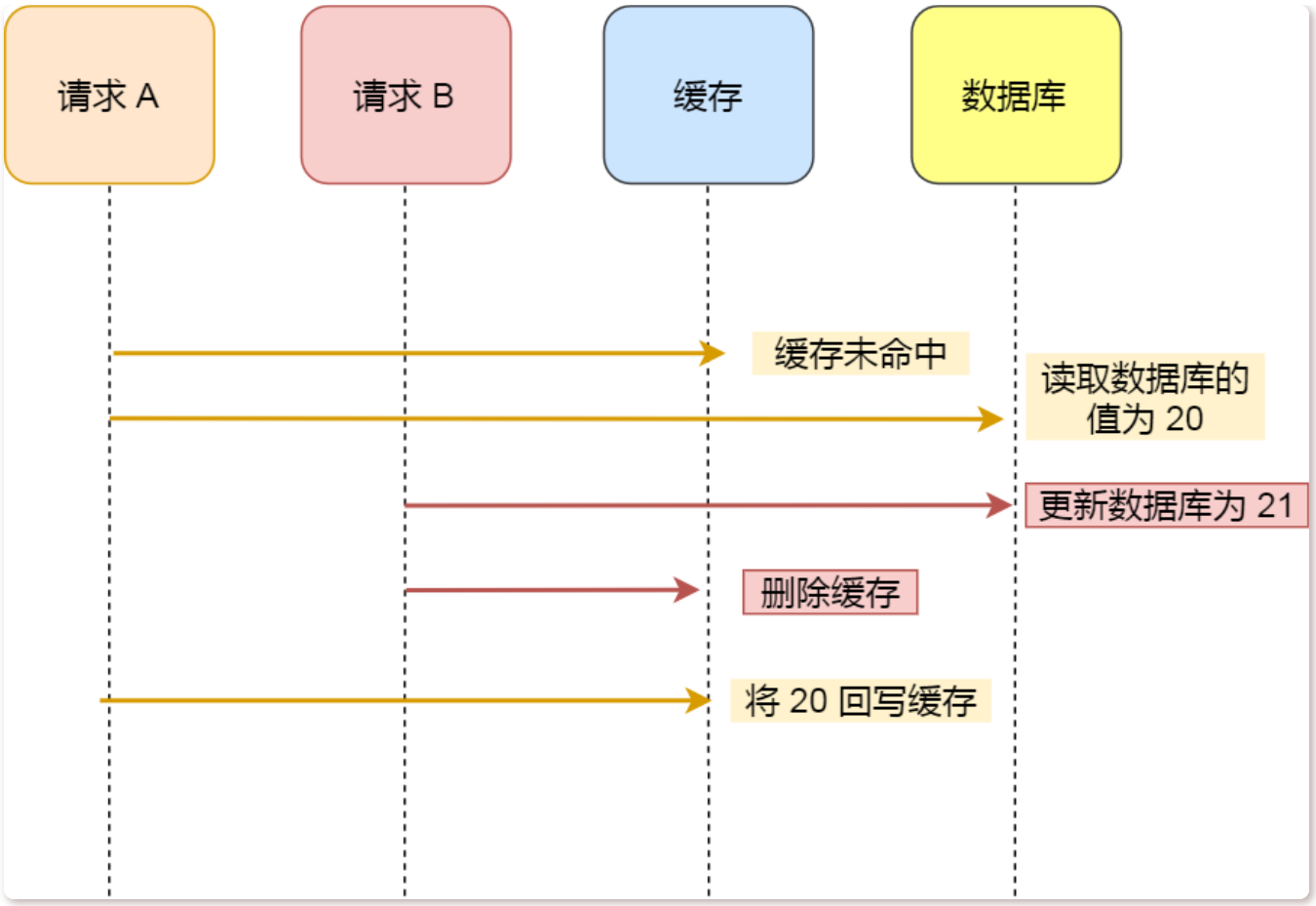
- 假设某个用户的年龄是20，**请求A**要**更新**用户年龄为21，所以它会**删除缓存**中的内容。
- 这时，另一个**请求B**要**读取**这个用户的年龄，它查询**缓存发现未命中**后，会从数据库中读取到年龄为20，并且**写回到缓存**中。
- **请求A**继续**更改数据库**，将用户的年龄更新为21。
- 最终，该用户年龄在**缓存**中是20（**旧值**），在**数据库**中是21（**新值**），缓存和数据库的**数据不一致**。



2.3、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。

2.4、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。

3、延时双删实现

在前面介绍到，先更新数据库后删Redis缓存是一致性相对最高的。这是就有人举手了：我就想要先删缓存怎么办？这时**延时双删**就出现了，针对「先删除缓存，再更新数据库」方案在「读 + 写」并发请求而造成缓存不一致的解决办法是「**延迟双删**」。

延迟双删实现的伪代码如下：

```
#删除缓存
redis.delKey(X)
#更新数据库
db.update(X)
#睡眠
Thread.sleep(N)
#再删除缓存
redis.delKey(X)
```

这里做一个详细介绍：

1. 首先，代码**先删除了 Redis 中的缓存数据**，以确保接下来的读取操作会从数据库中读取最新的数据。
2. 接着，代码**更新了数据库中的数据**，将数据更新为最新的值。
3. 在此之后，代码**让当前线程休眠一段时间N**，这个时间段是为了给数据库操作足够的时间来完成，确保数据已经持久化到数据库中。
4. 最后，代码**再次删除 Redis 中的缓存数据**。这里是延迟双删的**关键步骤**。由于之前已经删除了缓存数据，再次删除的目的是为了防止在 `Thread.sleep(N)` 的时间内有其他线程读取到旧的缓存数据。因为在这段时间内，缓存数据已经被清空，所以其他线程在读取数据时会发现缓存中不存在，然后从数据库中读取最新的数据并写入缓存，从而保证了数据的一致性。

需要注意的是，这种延迟双删策略并不能完全保证数据的一致性。

如果在 `Thread.sleep(N)` 的时间内发生了其他线程的写入操作，并且将新数据写入了缓存中，那么在第二次删除缓存时，会将这个新数据从缓存中删除，可能导致缓存和数据库中的数据不一致。

因此，**延迟双删策略只能在一定程度上提高数据一致性的概率，但不能完全解决数据一致性的问题**。更加严格的数据一致性保证需要使用更复杂的机制，比如使用消息队列等。

4、为什么要使用延时双删

在延时双删策略中，当需要更新数据库中的数据时，首先会先删除缓存，然后再进行数据库的更新操作。这样做的目的是为了**避免在数据库更新的过程中，有其他请求读取了已经失效的缓存数据**。

通过延时双删策略，可以保证在数据库更新期间，其他读取请求在缓存不命中的情况下，会直接读取数据库的最新数据，而不会读取到已经失效的缓存数据。这样就保证了数据的一致性和缓存的即时更新。

延时双删策略虽然会增加一次缓存删除的开销，但是可以有效地提高数据的一致性，并且在高并发读取的场景下，减轻数据库的读取压力，提高读取性能和响应速度。

5、方案选择

对于先删除缓存后更新数据库这种方案，由于出现数据不一致性的可能性偏高，数据库读写压力偏大以及性能偏低，因此这一方案一般不予与考虑，这里主要对**延时双删方案**和**先更新数据库后删除缓存方案**进行分析。

针对于前面的介绍，可以分析出以下结论：

- **延时双删适用于对数据一致性要求较高的场景**。它能够保证在数据库更新期间，读取请求不会读取到已经失效的缓存数据，从而保证数据的一致性。但是它需要进行两次缓存删除操作，可能会增加一定的资源开销；
- **先更新数据库后删除缓存适用于对一致性要求较低，对性能要求较高的场景**。它能够减少一次缓存删除的开销，但是在数据库更新期间，读取请求可能会读取到已经失效的缓存数据，从而导致数据不一致。

同时，还可以根据实际情况做一些权衡和优化。比如可以使用读写锁来减少数据库更新期间的并发读取请求，从而降低数据不一致的可能性。或者可以考虑使用更高效的缓存淘汰算法，来降低缓存的过期时间，减少缓存失效的影响。

方案	优点	缺点	实现复杂度	适用场景
先更新数据库后删除缓存	减少了一次缓存删除的开销	在数据库更新期间，读取请求可能读取到失效的缓存数据	简单	数据一致性要求较低、对性能要求较高的场景
延时双删	保证了数据一致性，读取请求不会读取到失效的缓存数据	需要进行两次缓存删除操作，增加了一定的资源开销	复杂	数据一致性要求较高的场景，同时对性能影响有一定容忍度的场景

6、延时双删真的完美吗

在认识到这个方案的时候，我就冒出了这么一种疑问不知道大家有没有：

> 🌈
>
> 为什么要执行第一次删除缓存的操作呢？留着缓存不是也能缓解数据库并发读取的压力吗？执行第一次删除缓存的操作还会多花费一定的资源去执行删除操作。

为了解决这一个问题我也去查询了许多资料和博文吸取经验，这个问题也是得到了一定的解决，如果有错误希望大伙热情提出。

首先，**为什么要执行第一次删除缓存的操作？**这是因为在并发环境下，如果直接更新数据库而不删除缓存，会导致脏数据问题。考虑以下场景：

1. **线程A**读取缓存中的旧数据。
2. **线程B**更新数据库中的数据，并删除缓存。
3. **线程A**继续使用缓存中的旧数据，因为此时它不知道缓存已经被删除。

为了避免这种脏数据问题，需要在更新数据库之前，先删除缓存，这样其他读取请求会从数据库中读取最新数据。

接着说**为什么需要延迟再次删除缓存**。延迟再次删除缓存的目的是为了在数据库更新期间，保留旧数据的缓存，以缓解数据库并发读取的压力。在延迟时间内，其他读取请求会从缓存中读取旧数据，而不会直接读取数据库。

虽然执行第一次删除缓存的操作会带来一定的资源开销，但通过合理设置延迟时间和优化缓存策略，可以在高并发读取场景下，有效降低对数据库的直接读取次数，从而提高读取性能和并发性能。这样在一段时间内，仍能从缓存中获取数据，减少数据库压力，而在数据库更新完成后，再次删除缓存以确保最终的数据一致性。

> 🌈
>
> 这么看来是有那么一点点脱裤子放屁的感觉哈，我刚开始也是有这么一种感觉的，但是一切都要以实际场景来决定的。

第一次删除缓存的操作是为了以一定的资源开销为代价，让缓存中的旧数据在一定时间内**相对较新**，以便在数据库更新期间，其他读取请求可以从缓存中获取旧数据，从而减轻对数据库的直接读取压力。这有些**类似于写锁**，在更新数据库时，尽可能的保证写之前的数据是最新的，但只是尽可能，虽然大部分保证了，但是还是会有一定的可能会出现脏数据问题。

这样做的目的是**为了在高并发读取场景下提高性能，通过缓存中的旧数据，避免大量读取请求直接访问数据库，降低数据库的并发读取压力**。同时，因为缓存的更新是延迟进行的，所以在一定时间内，读取请求会持续从缓存中获取数据，而不会频繁访问数据库，从而提高了读取性能和响应速度。

在第一次删除缓存到更新数据库期间，请求压力其实是由数据库服务和Redis服务两者一起承担的。当请求缓存不命中时，请求会打到数据库查询数据并写回缓存，之后请求压力将会由Redis服务承担。

7、如何确定延时的时间

确定延时双删中延时的时间是一个需要根据实际场景和需求来进行权衡的过程。延时的时间需要**根据数据库的更新操作耗时、缓存的过期时间以及应用的实际负载情况**，通过**不断的测试**来确定。

如果数据库的更新操作通常很快，可以选择较短的延时时间，比如几百毫秒或一秒钟。这样可以尽快地更新缓存，减少读取请求的直接访问数据库的次数，提高缓存的读取性能。

如果数据库的更新操作较为耗时，可能需要选择较长的延时时间，比如几秒钟或更长。这样可以保证数据库的更新操作完成后再删除缓存，避免读取请求获取到过期的缓存数据，保证数据一致性。

另外，**延时的时间还需要考虑缓存的过期时间**。如果缓存的过期时间较长，可以适当缩短延时的时间；如果缓存的过期时间较短，可以适当延长延时的时间，以免过早地删除缓存导致数据不一致。

> 
>
> 对于还需要考虑缓存的过期时间原因如下：

假设在延时双删策略中，第一次删除缓存后，会有一段时间的延时，然后再进行第二次删除缓存。如果此时缓存的过期时间设置得很短，比如只有几秒钟，那么在第二次删除缓存之前，缓存可能已经过期，而应用程序在读取缓存时会发现缓存已失效，从而不得不去数据库中查询最新数据。

为了避免这种情况，延时双删的延时时长应该要大于缓存的过期时间，确保在第二次删除缓存之前，缓存还是有效的，这样可以保证应用程序读取到的数据是一致的。

同时还需要考虑数据更新的频率和缓存的使用情况。如果数据更新较为频繁，那么延时双删的延时时长应该要适当缩短，以便及时更新缓存；如果缓存的使用率很低，可以适当延长延时时长，以减少对缓存服务的压力。