66 CAS 有什么缺点?

本课时主要讲解 CAS 有什么缺点。

前面我们讲过 CAS 是有很多优点的，比如可以避免加互斥锁，可以提高程序的运行效率，但是同样 CAS 也有非常明显的缺点。所以我们在使用 CAS 的时候应该同时考虑到它的优缺点，合理地进行技术选型。

下面我们就来看一下 CAS 有哪几个主要的缺点。

ABA 问题

首先，CAS 最大的缺点就是 ABA 问题。

决定 CAS 是否进行 swap 的判断标准是“当前的值和预期的值是否一致”，如果一致，就认为在此期间这个数值没有发生过变动，这在大多数情况下是没有问题的。

但是在有的业务场景下，我们想确切知道**从上一次看到这个值以来到现在，这个值是否发生过变化。**例如，这个值假设**从 A 变成了 B，再由 B 变回了 A**，此时，我们不仅认为它发生了变化，并且会认为它变化了两次。

在这种场景下，我们使用 CAS，就看不到这两次的变化，因为仅判断“当前的值和预期的值是否一致”就是不够的了。CAS 检查的并不是值有没有发生过变化，而是去比较这当前的值和预期值是不是相等，如果变量的值从旧值 A 变成了新值 B 再变回旧值 A，由于最开始的值 A 和现在的值 A 是相等的，所以 CAS 会认为变量的值在此期间**没有发生过变化。**所以，**CAS 并不能检测出在此期间值是不是被修改过，它只能检查出现在的值和最初的值是不是一样。**

我们举一个例子：假设第一个线程拿到的初始值是 100，然后进行计算，在计算的过程中，有第二个线程把初始值改为了 200，然后紧接着又有第三个线程把 200 改回了 100。等到第一个线程计算完毕去执行 CAS 的时候，它会比较当前的值是不是等于最开始拿到的初始值 100，此时会发现确实是等于 100，所以线程一就认为在此期间值没有被修改过，就理所当然的把这个 100 改成刚刚计算出来的新值，但实际上，在此过程中已经有其他线程把这个值修改过了，这样就会发生 **ABA 问题。**

如果发生了 ABA 问题，那么线程一就根本无法知晓在计算过程中是否有其他线程把这个值修改过，由于第一个线程发现当前值和预期值是相等的，所以就会认为在此期间没有线程修改过变量的值，所以它**接下来的一些操作逻辑，是按照在此期间这个值没被修改过”的逻辑去处理的**，比如它可能会打印日志：“本次修改十分顺利”，但是它**本应触发其他的逻辑**，比如当它发现了在此期间有其他线程修改过这个值，其实本应该打印的是“本次修改过程受到了干扰”。

那么如何解决这个问题呢？添加一个**版本号**就可以解决。

我们在变量值自身之外，再添加一个版本号，那么这个值的变化路径就从 **A→B→A 变成了 1A→2B→3A**，这样一来，就可以**通过对比版本号来判断值是否变化过**，这比我们直接去对比两个值是否一致要更靠谱，所以通过这样的思路就可以解决 ABA 的问题了。

在 atomic 包中提供了 **AtomicStampedReference** 这个类，它是专门用来解决 ABA 问题的，解决思路正是利用版本号，AtomicStampedReference 会维护一种类似 <Object,int> 的数据结构，其中的 int 就是用于计数的，也就是版本号，它可以对这个对象和 int 版本号同时进行原子更新，从而也就解决了 ABA 问题。因为我们去判断它是否被修改过，不再是以值是否发生变化为标准，而是以版本号是否变化为标准，即使值一样，它们的版本号也是不同的。

以上就是对 CAS 的第一个缺点—— ABA 问题的介绍。

自旋时间过长

CAS 的第二个缺点就是自旋时间过长。

由于单次 CAS 不一定能执行成功，所以 **CAS 往往是配合着循环来实现的**，有的时候甚至是死循环，不停地进行重试，直到线程竞争不激烈的时候，才能修改成功。

可是如果我们的应用场景本身就是高并发的场景，就有可能导致 CAS 一直都操作不成功，这样的话，**循环时间就会越来越长**。而且在此期间，CPU 资源也是一直在被消耗的，这会对性能产生很大的影响。所以这就要求我们，要根据实际情况来选择是否使用 CAS，在高并发的场景下，通常 CAS 的效率是不高的。

范围不能灵活控制

CAS 的第三个缺点就是不能灵活控制线程安全的范围。

通常我们去执行 CAS 的时候，是针对某一个，而不是多个共享变量的，这个变量可能是 Integer 类型，也有可能是 Long 类型、对象类型等等，但是我们不能针对多个共享变量同时进行 CAS 操作，因为这多个变量之间是独立的，简单的把原子操作组合到一起，并不具

备原子性。因此如果我们想对多个对象同时进行 CAS 操作并想保证线程安全的话，是比较困难的。

有一个解决方案，那就是利用一个新的类，来整合刚才这一组共享变量，这个新的类中的多个成员变量就是刚才的那多个共享变量，然后再利用 atomic 包中的 AtomicReference 来把这个新对象整体进行 CAS 操作，这样就可以保证线程安全。

相比之下，如果我们使用其他的线程安全技术，那么调整线程安全的范围就可能变得非常容易，比如我们用 synchronized 关键字时，如果想把更多的代码加锁，那么**只需要把更多的代码放到同步代码块里面**就可以了。

总结

下面我们进行总结，本课时介绍了 CAS 的三个缺点，分别是 **ABA 问题、自旋时间过长以及线程安全的范围不能灵活控制。**我们了解了它的缺点之后，在进行技术选型的时候就可以有的放矢了。

上一页 下一页