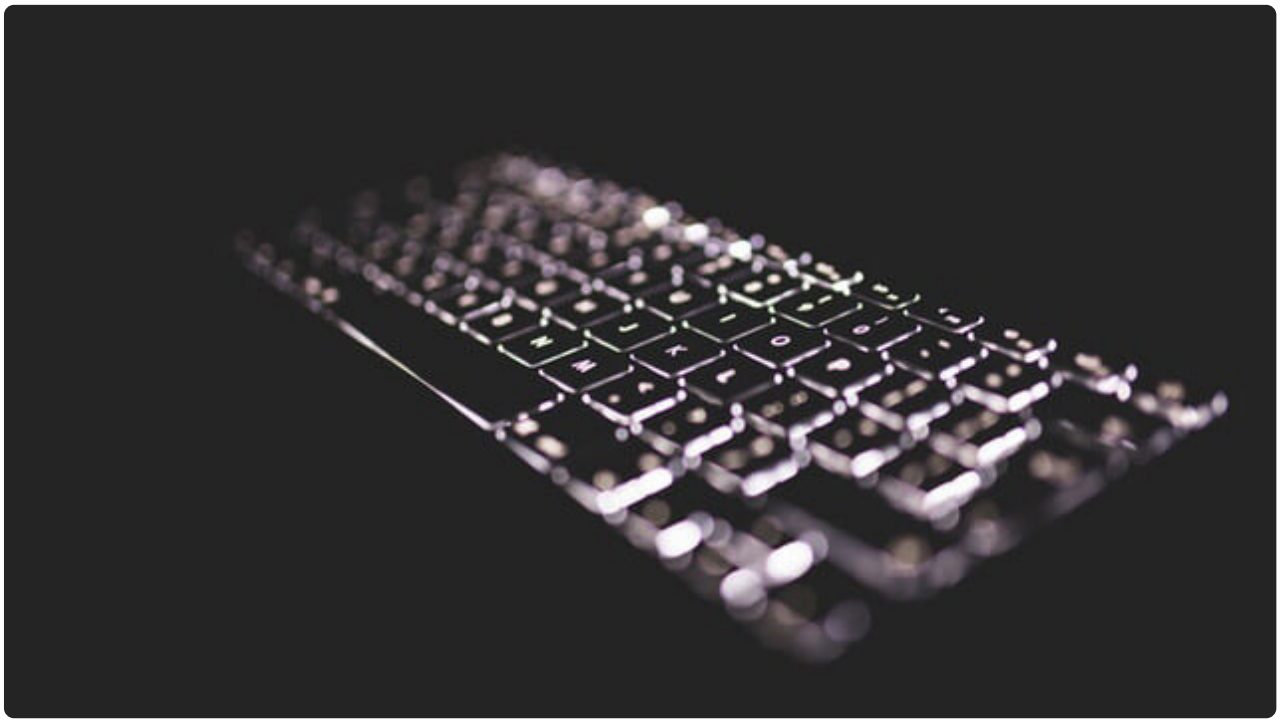
22 到底哪把锁更适合你？—synchronized与ReentrantLock对比

更新时间：2019-11-12 09:41:56

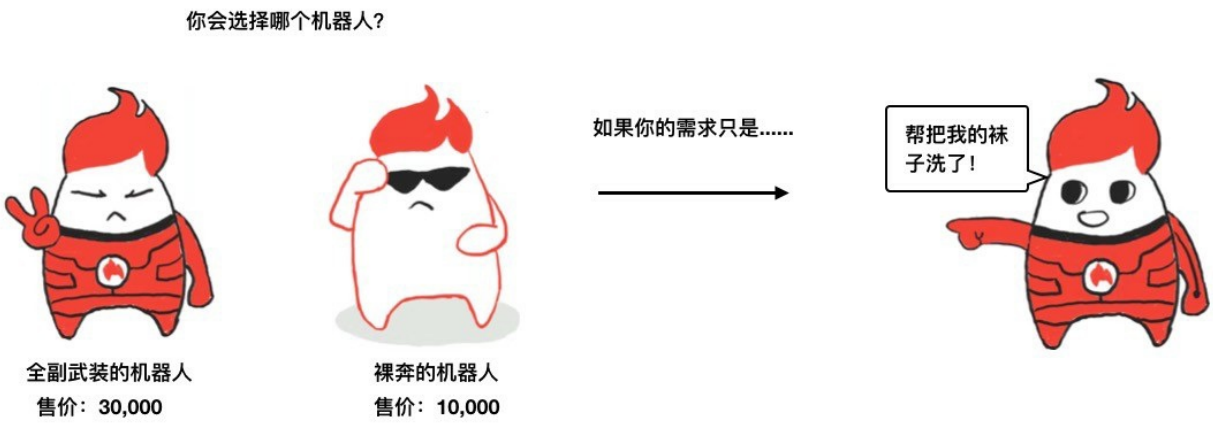


> 横眉冷对千夫指，俯首甘为孺子牛。
>
> ——鲁迅

前一节我们学习了ReentrantLock显式锁，在这之前我们还学习过 synchronized 内置锁。那么这两个锁究竟有什么区别呢？synchronized是不是就足够用了？什么时候使用ReentrantLock？本小节，我们就来一块看看这两种锁，分析一下他们的不同之处，以及应用的最佳场景。

1、synchronized与ReentrantLock比较

从功能特性上来看，ReentrantLock其实具备synchronized所有特性，可以完全取代synchronized。不过ReentrantLock设计之初并不是为了替换掉synchronized，而是当synchronized不能满足需求时，才考虑使用ReentrantLock。这是因为ReentrantLock使用起来需要更为小心，必须要显式的释放锁。一旦忘记或者执行不到释放锁的代码，那么其它线程无法获取锁，一直陷入等待之中。另外由于synchronized更被开发人员所熟知，并且编写起来，代码更为紧凑，非常的简洁。只需要把同步代码放入花括号中。执行完同步代码块中的代码，锁自动被释放。因此，一般情况下，如果synchronized能够满足我们的需求，我们还是应该尽量使用synchronized。除非是需求确实需要显式锁Lock的相关特性，我们才会选择使用显式锁。这就像我们挑选物品，当然功能多的最好，但你同时会面临着高昂的价格、复杂的使用方法、更容易出现故障等问题。所以很简单，我们应该按需选择，如果不需要那么多功能，那么就选择最简单的易用的。



下面我们从几个不同维度对synchronized和ReentrantLock做个对比。

2、性能对比

ReentrantLock 在 Java 5.0时被添加进来。那个时候，它有着比内置锁更好的竞争性。竞争性是锁的性能重点，有着好的竞争性，代表线程在锁的竞争上消耗更低，整个并发程序的性能就会更好。不过在 Java 6 开始，内置锁改进了算法，从而限制提高了内置锁的性能。

5.0 时，随着线程的增加，内置锁的性能急剧下降，而 ReentrantLock 的下降并不明显。线程增加到一定数量后，ReentrantLock 性能会达到内置锁的 4-5 倍。而在 6.0 中，两者差距并不明显，ReentrantLock 略占一点点优势。

所以结论是我们并不需要过多考虑性能因素，而采用 ReentrantLock。

3、特性对比

可以说 ReentrantLock 在特性上完胜内置锁。ReentrantLock 提供了公平和非公平锁、可定时、可轮询和可中断的锁获取方式、非块状锁结构。如果我们真的需要使用这些特性，那么不要犹豫，去使用 ReentrantLock 就好，因为 synchronized 根本就不支持。

4、公平性的选择

这个比较简单直接，ReentrantLock 支持公平锁，而内置锁不能支持公平锁。ReentrantLock 内部有一个线程排队的队列，如果 ReentrantLock 选择了公平的方式，那么队列中的线程会按照顺序去 tryLock。非公平的方式，在锁释放后，如果有新的线程来竞争锁，那么就可能插队，在等待队列中的线程被恢复并获取锁之前，新的线程获取了锁。

公平性的选择，意味着需要放弃一部分性能。大多数情况下，公平锁的性能都要低于非公平锁。这是因为挂起和恢复线程都有很大开销。选择公平锁时，从释放锁到等待队列中最前面线程被唤醒能够去 tryLock，中间有很大的时间延迟，那么这就造成了公平锁的性能会更差。

如果线程获取锁到释放锁之间的程序执行时间较长，那么公平锁的性能不会那么差。因为不会有很多的线程唤醒操作，也就是说不会有过多的时间间隙被浪费点。那么公平锁有能带来更好的公平性，所以此时我们优先选择公平锁。

如果线程持有锁执行逻辑的时间很短，而多线程并发量又很大。这造成了获取和释放锁频繁发生，从而大量时间浪费在从锁被释放到排队线程被唤醒工作的过程上。因此，此时我们更好的选择是非公平锁。

5、使用上的差异

两者在使用上的差异前文已经有所对比。synchronized 使用简单，只需要把同步代码放入 synchronized 代码块中即可，程序执行完同步代码块自动解锁。而 Lock 需要显式获取锁，然后需要配合 try 和 finally 来使用。尤其注意一定要在 finally 中释放锁。从使用角度看，我们应该优先使用 synchronized，因为更为方便，也不容易出错。

6、总结

本节我们从几个不同纬度对比了 synchronized 和 lock。首先在性能上，Java 6 以后两者并没有显著的区别。在功能上，Lock 显然更为丰富，适合更多的场景。不过在绝大多数场景中，synchronized 提供的功能完全能够满足。此外，synchronized的 使用更为简单和安全。因此，如果我们并不需要lock提供的额外功能，那么请优先使用 synchronized 方法。只有在真的需要 lock 所提供的特性时，才应选择 lock。下一节我们来看看 Lock 家族的另外一员大将，更为灵活的读写锁——ReadWriteLock。

}

21 更高级的锁—深入解析Lock

23 按需上锁—ReadWriteLock详解