19 你知道哪几种锁？分别有什么特点？

本课时我们首先会对锁的分类有一个整体的概念，了解锁究竟有哪些分类标准。然后在后续的课程中，会对其中重要的锁进行详细讲解。

锁的 7 大分类

需要首先指出的是，这些多种多样的分类，是评价一个事物的多种标准，比如评价一个城市，标准有人口多少、经济发达与否、城市面积大小等。而一个城市可能同时占据多个标准，以北京而言，人口多，经济发达，同时城市面积还很大。

同理，对于 Java 中的锁而言，一把锁也有可能同时占有多个标准，符合多种分类，比如 ReentrantLock 既是可中断锁，又是可重入锁。

根据分类标准我们把锁分为以下 7 大类别，分别是：

- 偏向锁/轻量级锁/重量级锁；
- 可重入锁/非可重入锁；
- 共享锁/独占锁；
- 公平锁/非公平锁；
- 悲观锁/乐观锁；
- 自旋锁/非自旋锁；
- 可中断锁/不可中断锁。

以上是常见的分类标准，下面我们来逐一介绍它们的含义。

偏向锁/轻量级锁/重量级锁

第一种分类是偏向锁/轻量级锁/重量级锁，这三种锁特指 synchronized 锁的状态，通过在对象头中的 mark word 来表明锁的状态。

- 偏向锁

如果自始至终，对于这把锁都不存在竞争，那么其实就没必要上锁，只需要打个标记就行了，这就是偏向锁的思想。一个对象被初始化后，还没有任何线程来获取它的锁时，那么它就是可偏向的，当有第一个线程来访问它并尝试获取锁的时候，它就将这个线程记录下来，以后如果尝试获取锁的线程正是偏向锁的拥有者，就可以直接获得锁，开销很小，性能最好。

- 轻量级锁

JVM 开发者发现在很多情况下，synchronized 中的代码是被多个线程交替执行的，而不是同时执行的，也就是说并不存在实际的竞争，或者是只有短时间的锁竞争，用 CAS 就可以解决，这种情况下，用完全互斥的重量级锁是没必要的。轻量级锁是指当锁原来是偏向锁的时候，被另一个线程访问，说明存在竞争，那么偏向锁就会升级为轻量级锁，线程会通过自旋的形式尝试获取锁，而不会陷入阻塞。

- 重量级锁

重量级锁是互斥锁，它是利用操作系统的同步机制实现的，所以开销相对比较大。当多个线程直接有实际竞争，且锁竞争时间长的时候，轻量级锁不能满足需求，锁就会膨胀为重量级锁。重量级锁会让其他申请却拿不到锁的线程进入阻塞状态。

你可以发现锁升级的路径：无锁→偏向锁→轻量级锁→重量级锁。

综上所述，偏向锁性能最好，可以避免执行 CAS 操作。而轻量级锁利用自旋和 CAS 避免了重量级锁带来的线程阻塞和唤醒，性能中等。重量级锁则会把获取不到锁的线程阻塞，性能最差。

可重入锁/非可重入锁

第 2 个分类是可重入锁和非可重入锁。可重入锁指的是线程当前已经持有这把锁了，能在不释放这把锁的情况下，再次获取这把锁。同理，不可重入锁指的是虽然线程当前持有了这把锁，但是如果想再次获取这把锁，也必须要先释放锁后才能再次尝试获取。

对于可重入锁而言，最典型的就是 ReentrantLock 了，正如它的名字一样，reentrant 的意思就是可重入，它也是 Lock 接口最主要的一个实现类。

共享锁/独占锁

第 3 种分类标准是共享锁和独占锁。共享锁指的是我们同一把锁可以被多个线程同时获得，而独占锁指的就是，这把锁只能同时被一个线程获得。我们的读写锁，就最好地诠释了共享锁和独占锁的理念。读写锁中的读锁，是共享锁，而写锁是独占锁。读锁可以被同时读，可以同时被多个线程持有，而写锁最多只能同时被一个线程持有。

公平锁/非公平锁

第 4 种分类是公平锁和非公平锁。公平锁的公平的含义在于如果线程现在拿不到这把锁，那么线程就都会进入等待，开始排队，在等待队列里等待时间长的线程会优先拿到这把锁，有先来先得的意思。而非公平锁就不那么“完美”了，它会在一定情况下，忽略掉已经在排队的线程，发生插队现象。

悲观锁/乐观锁

第 5 种分类是悲观锁，以及与它对应的乐观锁。悲观锁的概念是在获取资源之前，必须先拿到锁，以便达到“独占”的状态，当前线程在操作资源的时候，其他线程由于不能拿到锁，所以其他线程不能来影响我。而乐观锁恰恰相反，它并不要求在获取资源前拿到锁，也不会锁住资源；相反，乐观锁利用 CAS 理念，在不独占资源的情况下，完成了对资源的修改。

自旋锁/非自旋锁

第 6 种分类是自旋锁与非自旋锁。自旋锁的理念是如果线程现在拿不到锁，并不直接陷入阻塞或者释放 CPU 资源，而是开始利用循环，不停地尝试获取锁，这个循环过程被形象地比喻为“自旋”，就像是线程在“自我旋转”。相反，非自旋锁的理念就是没有自旋的过程，如果拿不到锁就直接放弃，或者进行其他的处理逻辑，例如去排队、陷入阻塞等。

可中断锁/不可中断锁

第 7 种分类是可中断锁和不可中断锁。在 Java 中，synchronized 关键字修饰的锁代表的是不可中断锁，一旦线程申请了锁，就没有回头路了，只能等到拿到锁以后才能进行其他的逻辑处理。而我们的 ReentrantLock 是一种典型的可中断锁，例如使用 lockInterruptibly 方法在获取锁的过程中，突然不想获取了，那么也可以在中断之后去做其他的事情，不需要一直傻等到获取到锁才离开。

上一页 下一页