## 锁优化

**synchronized效率低的原因：**

synchronized通过对象内部的对象监视器实现，而对象监视器依赖了操作系统的Mutex Lock。因此每次加锁和解锁都会发生切换到内核态，效率低。因此依赖Mutex Lock被称为“重量级锁”。

**自旋锁和适应性自旋：**

当资源锁定时间很短时，如果使用阻塞式的锁定，加锁解锁的开销远高于代码逻辑。在锁（轻量级锁）定时间很短的前提下，可以适当的执行自旋，如果超过一定测试仍不能获得锁，则进入阻塞状态。自旋次数可以用-XX:+UseSpinning开启，当前JDK默认是开启的。对于synchronize，获取不到锁会先执行自旋。

所谓适应性自旋是指自旋次数不固定，如果一个锁经常在自旋期间获得锁，则JVM会更倾向于对这个对象使用自旋，适当增加自旋此时；反之虚拟机可能跳过自旋过程，减少对CPU的消耗。

**锁消除：**

如果虚拟机运行期间发现对某些代码的同步不存在数据共享或竞争，则会消除锁定。例如JVM会优化String的拼接，转化为StringBuffer，StringBuffer是同步方法，但大部分情况下字符串拼接不会产生竞争，JIT会消除StringBuffer的锁。

**锁粗化：**

JVM会将临近的解锁、加锁合并为一个大的同步代码块，例如连续调用StringBuffer.append方法，JVM优化后只会在第一次append加锁，在最后一个append结束时解锁。

**轻量级锁：**

轻量是相对于使用操作系统互斥量（上下文切换），因此也称为重量级锁。目的是在竞争不强烈时，避免使用重量级锁，转而使用轻量锁。

对象头中有一段64位数据称为Mark Word，存储对象hash值（25位）、对象年龄分代（4位）和锁标志位（2位）。

锁标志位01表示未锁定。因此当一个线程试图获取锁，发现锁标记为01，则会现在栈中拷贝一份Mark Word，然后用CAS操作，将这个对象的指针覆盖原对象头的Mark Word。如果CAS成功代表这个线程已经获取了锁。

如果CAS失败原因1，可能是当前线程已经获取了锁，指针是否指向当前线程的栈（栈是线程私有的）。如果不是则说明其他线程已经获取了锁，此时发生了竞争。轻量级锁会失效，对象Mark Word会被修改为指向重量级锁对象的指针（为了被唤醒），标记位也会变为10（重量级锁）。膨胀为重量级锁。竞争线程会阻塞。

轻量级锁释放是也会CAS判断，如果Mark Word没变，则释放轻量锁。如果失败则说明锁已经膨胀为重量级锁。此时会唤醒竞争的线程。

**偏向锁：**

所谓偏向就是指偏向某一个线程，比轻量化锁更轻量，如果是第一次获取锁，会将锁标记变为01，表示可偏向，且用当前线程ID替换Mark Word。获取锁的线程首先比较对象头中线程号是否为当前线程，如果是则无需执行CAS，因为在没有竞争的条件下不会释放锁（线程号一直是第一个线程）。如果第二个线程尝试获取锁，对比发现线程ID不同，该对象会撤销偏向锁，修改锁标记为轻量级锁，升级为轻量化锁。