## synchronized的优化和实现原理

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”。

无锁 < 偏向锁 < 轻量级锁 < 重量级锁

锁只能单向升级，不能降级

synchronized的优化：

锁粗化：减少不必要的lock和unlock，将临近的同步代码块整合成一个同步代码块，扩大锁的范围。

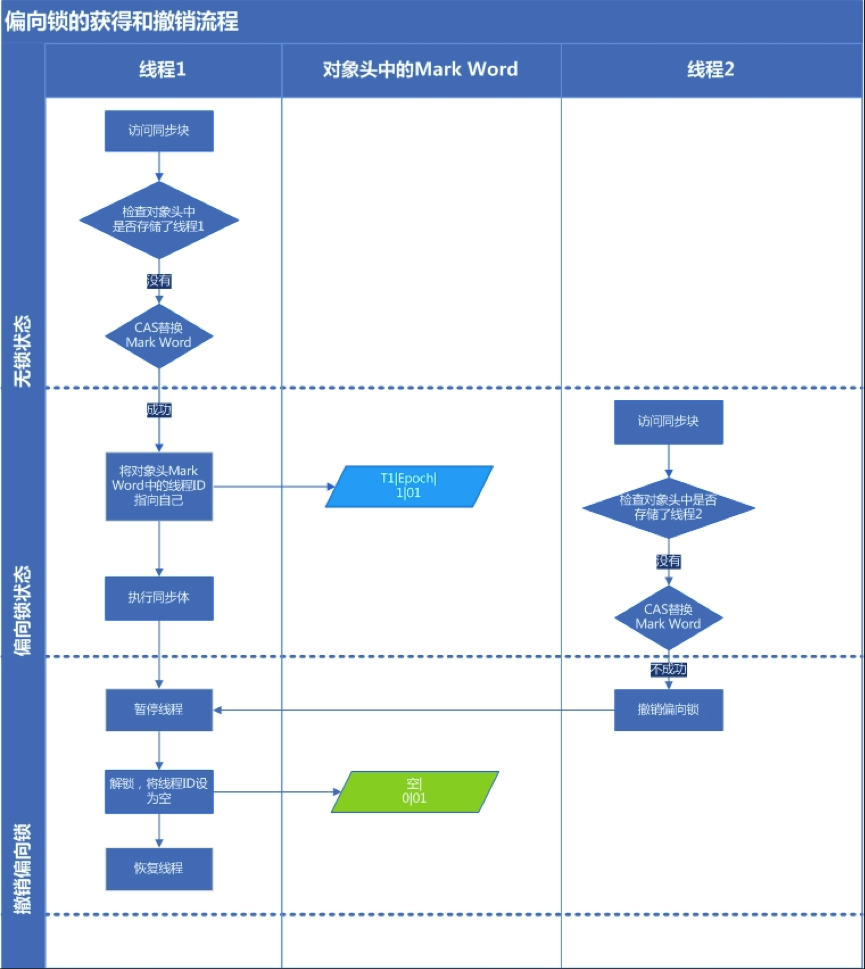
锁消除：通过运行时JIT编译器，将同步代码块中不需要保护的变量，减小锁的粒度。

轻量化锁：默认大部分代码处于无竞争状态，synchronized关键字采用了CAS来获取和释放锁，两次CAS。CAS比较的是如果遇到竞争则先进行适应性自旋，如果自旋仍不能获取锁则进入阻塞状态（重量级锁），等待释放锁时被唤醒。与偏向锁不同的是，每次退出同步代码块都需要释放锁。

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，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），如果不是当前线程，则是执行CAS替换Mark Word，如果第一次还没人获取锁，则替换成功。如果发生竞争即第二个线程产生替换Mark Word，该对象会撤销偏向锁，升级为轻量化锁。

适应性自旋：当获取轻量级锁失败后，会尝试一定次数的自旋等待，如果仍然不能获取锁，则阻塞（变为重量级锁）。

偏向锁 -> 轻量级锁：



轻量级锁 -> 重量级锁：  
