## Synchronized

1. **Synchronized使用场景**

Synchronized一般用在对象方法，类方法，和同步块中。但是在方法上使用Synchronized和在同步块中使用是不一样的，方法上采用的是字节码中标志位ACC\_SYNCHRONIZED来进行同步。而同步代码块则采用对象头中的锁指针指向一个monitor来完成同步。

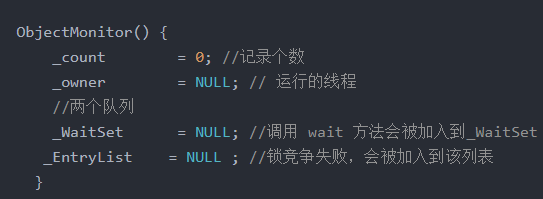
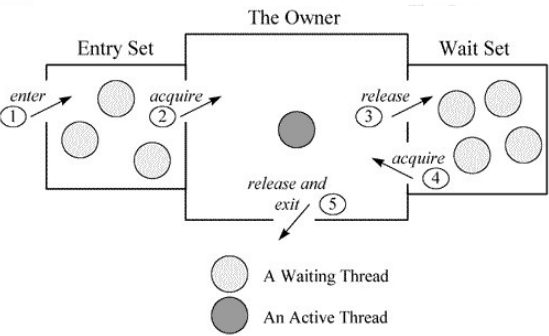
1. **对象与monitor对象**

对象在内存中存储的布局可以分为三块区域：对象头，实例数据，对齐填充。对象头可以分为“Mark Word”和类型指针。“Mark Word”中存储对象自身运行时的数据，如哈希码，GC分代年龄，锁的标志位，线程持有的锁，偏向锁的ID等，为了提高利用率，“Mark Word”是非固定的数据结构：

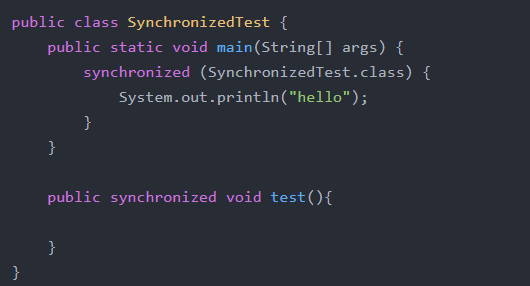


Monitor对象是用来实现重量级锁的关键对象，锁的标志位是10，存放着指向monitor对象的指针。每个对象都有一个monitor对象与之关联。Monitor对象可以和对象一起创建销毁，或者当线程尝试获取对象锁的时候，自动生成。

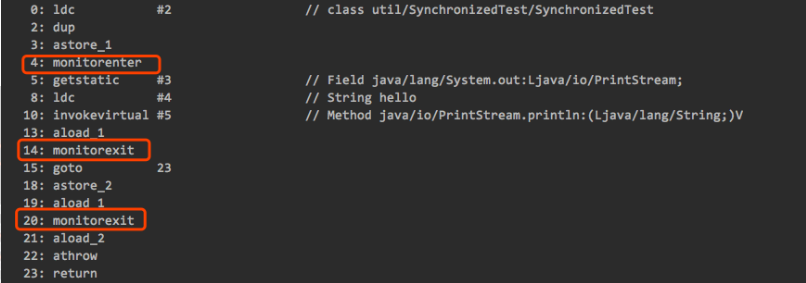
Monitor对象的结构如下：



1. **Synchronized代码块原理**



反编译上述代码得到如下字节码：



当执行monitorenter指令的时候：

* 当前线程尝试去获取monitor的持有权，当monitor的计数器为0，那线程可以成功获取，并将计数器设为1，取锁成功。
* 如果当前线程已经持有了monitor，那么可以重入这个monitor，计数器也会+1。
* 倘若其他线程已经拥有monitor的所有权，那么当前线程就会阻塞，直到别的线程执行monitorexit方法，并将count置为0，其他线程才可以争取该monitor。编译器会确保每个monitorenter都有一个monitorexit与之对应。意味着当synchronized方法里面出现了异常，当前线程会自动放弃持有的锁。

1. **偏向锁：**

当锁对象第一次被线程获取的时候，虚拟机将会把对象头中的标志位设为“01”，即偏向模式。同时使用CAS操作把获取到这个锁的线程ID记录在对象的Mark Word之中，同时会在栈帧中记录偏向的锁的threadId。如果CAS成功，持有偏向锁的线程每次进入这个锁相关的同步块，虚拟机可以不做任何同步。

当另一个线程尝试获取这个锁的时候，偏向模式就宣告结束。根据偏向对象目前是否处于被锁定的状态，撤销偏向锁后恢复到未锁定（01）或轻量级锁定（00）状态。

1. **轻量级锁：**

虚拟机首先将在当前线程的栈帧中建立一个名为锁的记录（Lock Record）的空间，用于存储锁对象目前的Mark Word的拷贝，然后虚拟机使用CAS操作尝试将对象的Mark Word更新为指向Lock Record的指针。如果成功，那么这个线程就拥有了该对象锁，并且对象Mard Word的标志位转变为00，即表示此对象处于轻量级锁定状态。如果这个操作失败，虚拟机首先检查对象的Mark Word是否指向当前线程的栈帧，如果指向，说明当前线程已经拥有了这个对象的锁，那就可以直接进入同步块，否则说明这个锁对象已经被其他线程抢占。如果有两条以上的线程用同一个锁，那么轻量级锁就不再生效，要膨胀为重量级锁。

1. **自旋锁与适应性自旋锁**

当线程尝试获取同步资源失败的时候，自旋锁会让线程进入自旋，意味着不放弃CPU片段。如果在自旋完成后，前面锁定同步资源的线程已经释放了锁，那么当前线程可以不必要阻塞，而是直接获取同步资源，从而避免了线程切换的开销。自旋锁虽然避开了线程切换，但是要占用处理器时间，如果被锁住的时间比较长，那么自旋锁就会白白浪费处理器资源。所以自旋等待的时间必须要有限度，如果超过了一定值，那么就挂起该线程。

适应性自旋锁：适应性意味着自旋的次数不再固定，而是由前面的自旋成功的几率来决定。

## AQS