在多线程的使用场景中，synchronized加锁是一个很关键的机制，一起了解一下吧

1、synchronized执行过程中什么情况下是偏向锁、轻量级锁、重量级锁？

2、synchronized加锁过程中发生了什么？

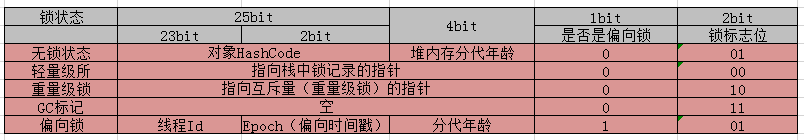
3、什么条件下会触发锁膨胀？

一、基础概念

开始之前先一起了解几个基本概念

1、Mark Word

java对象头的Mark Word中存储了HashCode、分代年龄、锁状态等信息，来看下Mark Word的结构，如下图



2、栈帧（stack frame）

方法执行时，在jvm的栈中会创建一个栈帧用来存储局部变量、操作数栈、动态链接、方法出口等信息。方法从调用到执行完成，就是栈帧在虚拟机栈中入栈到出栈的过程。（所以代码块中的局部变量可以实现入栈创建，出栈销毁）

线程中的许多方法同时处于执行状态，对执行引擎来说，活动线程中，栈顶的栈帧才是有效的，称为当前栈帧，与这个栈帧关联的方法称为当前方法。

二、了解synchronized

Synchronized是一直在java的加锁中扮演重要角色，经常有人说它是重量级锁，不过如果真这么想，那么对它的了解还停留在JDK5上，JDK6以后做了了优化，根据不同情况分别使用向锁、轻量级锁和重量级锁了。

JVM通过Monoitor来实现的synchronized的加锁，monitorenter在编译后会加到同步代码块的开始位置，monitorexit加到结束和异常的位置。

让用javap反编译一下接下来的代码代码

Object m\_lock = **new** Object();

**public** **void** demo()

{

**synchronized**(m\_lock)

{

**int** i=0;

}

}

执行javap –c Demo后如下图



在第6行插入了monitorenter，15行插入了monitorexit，来保证代码块同步。

接下来一起了解下Synchronized在什么情况下是偏向锁、轻量级锁和重量级锁，以及锁的膨胀

三、偏向锁、轻量级锁和重量级锁的使用和锁膨胀

加锁是因为多个线程竞争临界资源，只有一个线程竞争、两个线程去竞争、n多个线程竞争的激烈程度是不同的。竞争越激烈的情况下，获取锁的代价越大，所以为了减少性能消耗，jvm根据不同竞争情况，将锁分为偏向锁、轻量级锁、重量级锁。

偏向锁、轻量级锁和重量级锁是怎么加锁的呢 ？

1、偏向锁

偏向锁是三种锁中加锁消耗最小的。

HotSpot作者认为大多数情况下，锁不存在竞争关系，总是会被一个线程持有，为了减少互斥锁的代价

偏向锁是怎么加锁成功呢，将Mark Word中的线程Id标记为当前线程id，就加锁成功了，具体的流程如下：

先读取对象的Mark Word 判断是否处于可偏向的状态，即检查Mark Word中的 是偏向状态和锁标志位。

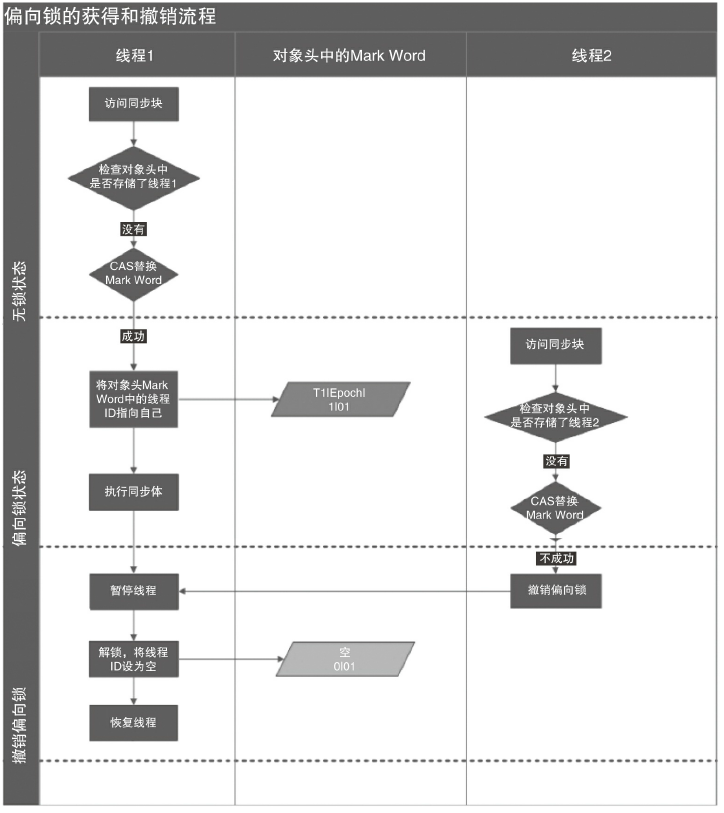
如果可偏向的，说明当前线程可加锁，那么就用CAS操作去将线程id写入到Mark Word中，如果获取锁成功，执行同步代码。如果CAS操作失败，说明其他线程在竞争，并取到了偏向锁，那么等待全局安全点（GC运行之前所有线程需要在安全点阻塞，这就GC过程中常说的Stop The World），将偏向状态改为0，验证已获取锁的线程是否存活，如果死亡，将锁标志位恢复到无锁状态，重新加锁。如果存活，将锁标志位升级为轻量级锁（01）。

如果不是可偏向的，首先验证Mark Word中线程id是否为当前线程，如果是继续执行代码。如果不是，说明锁对象偏向其他线程，等待安全点，验证是否需要升级为轻量级锁。

这就是偏向锁的整个加锁过程了。

偏向锁的锁状态在线程结束后，不会被置为无锁状态，只有在新的线程来获取锁的时候，在安全点设置为无锁状态或者升级为轻量级锁。

这个过程可以参考下图（借用高并发编程艺术的图）：



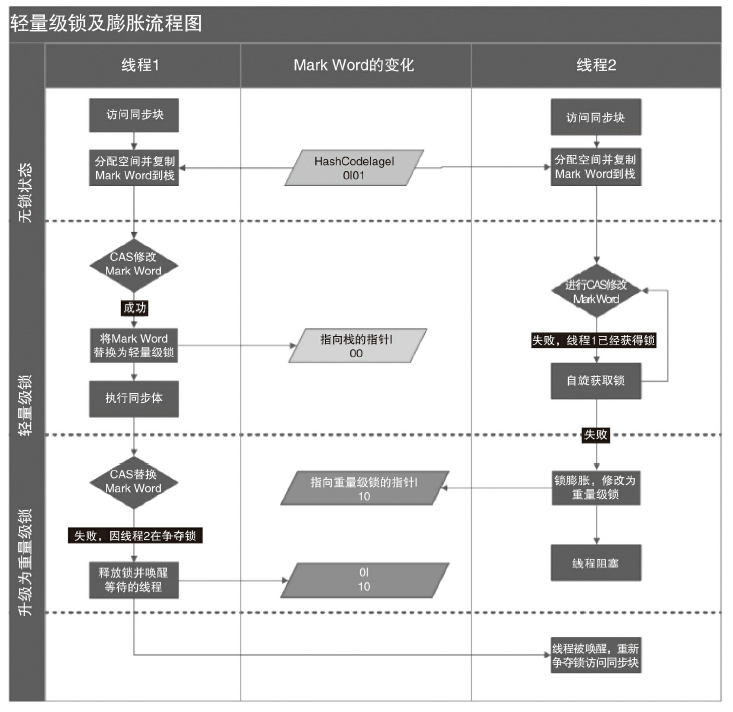
2、轻量级锁

偏向锁升级为轻量级锁，轻量级锁采用CAS自旋锁的方式来完成加锁，相对于重量级锁加锁的代价相对小一些，如果一直获取不到锁状态，自旋占用的资源会超过重量级锁，所以轻量级锁膨胀为重量级锁的条件就是自旋达超过一定次数（默认为10，可以修改PreBlockSpin参数调整）。

轻量级锁的加锁流程如下：

执行同步代码块之前，JVM会在线程的栈帧中创建一个存储锁记录的空间（Lock Record），并将Mark Word拷贝复制到锁记录中（因为已经脱离了原始的Mark Word，官方以displaced  作为前缀，即Displaced Mark Word（置换标记字））。然后尝试通过CAS将Mark Word中的锁记录的指针，指向创建的Lock Record。如果成功表示获取锁状态成功，如果失败，则进入自旋获取锁状态。

可以参考下图（借用高并发编程艺术的图）：



3、重量级锁

重量级锁的加锁需要通过mutex和condition variable（个人理解condition variable提供了wait和notify来阻塞线程）来实现的。

重量级锁会让抢占锁的线程从用户态转变为内核态，开销很大。

注意 ：锁膨胀这个过程，只允许升级，不允许降级，即只能偏向锁升级为轻量级锁，轻量级锁升级为重量级锁，不能反过来重量级锁降级为轻量级锁。

文章到这里就结束了，有问题可以留言。