链滴

Java 原理:synchronized 机制与几类锁

作者：EricTao

原文链接：https://ld246.com/article/1532598314891

来源网站：链滴

synchronized机制

synchronized关键字是JAVA中常用的同步功能，提供了简单易用的锁功能。

synchronized有三种用法，分别为:

- 用在普通方法上，能够锁住当前对象。
- 用在静态方法上，能够锁住类
- 用在代码块上，锁住的是 synchronized()里的对象

在JDK6之前，synchronized使用的是重量级锁制，在之后synchronized加入了锁膨胀机制，显著提了synchronized关键字的效率。

基于synchronized关键字，我们来了解下几种类别的锁，同时立即synchronized的锁膨胀机制。

synchronized锁是非公平锁

一个被synchronized锁住的对象或类，就是一把锁。

另外一提，所有锁都是存储在Java对象头里的，Java对象头里的Mark Word里默认存储对象的Hash ode，分代年龄和锁标记位。也就是说Mark Word记录了锁的状态

锁膨胀机制与几类锁

锁膨胀是不可逆的

偏向锁

synchronized在JDK1.6以后默认开启偏向锁，synchronized最初都是偏向锁

表现：一个线程获取锁成功后，会在对象头里记录线程ID，以后该线程获取和释放锁都没有任何花费（因为该锁已经被绑定在该线程上了，且在膨胀前不会改变），如果其他线程尝试获取这个锁，偏向锁会膨胀为轻量锁。

优点：在只有一个线程使用锁的时候获取和退出锁没有任何花费

缺点：锁竞争激烈会很快升级为轻量锁，那么维持偏向锁的过程就是在浪费计算机资源。（因为偏向锁身就很轻量，因此浪费的资源并不多）

小结：只有一个线程使用锁的情况下，synchronized使用的锁为偏向锁。

如果锁竞争激烈，可以通过配置JDK禁用偏向锁。

轻量锁

一把锁不止一个线程使用，则偏向锁膨胀为轻量锁

表现：线程获取轻量锁时，会直接用CAS修改对象头里锁的记录，如果修改失败，代表此时锁存在多线程的竞争，轻量锁将会膨胀为重量锁。

优点：在线程之间使用锁不存在竞争时，一次CAS操作就能获取和退出锁

缺点：与偏向锁类似

小结：只要一把锁不止一个线程获取过，偏向锁就会膨胀为轻量锁。

重量锁

一把锁存在多线程竞争，则轻量锁开始自旋，自旋一定次数后仍没获取锁，则膨胀为重量锁

表现：线程获取重量锁时，如果获取失败（即锁已被其他线程获取），则使用自适应自旋锁,自旋一定数后仍没获取锁，则进入阻塞队列等待。

优点：未获取到的锁进入阻塞队列，节约CPU资源。（好吧感觉其实是没有啥优点）

缺点：重量锁是通过对象内部的监视器（monitor）实现，其中monitor的本质是依赖于底层操作系的Mutex Lock实现，操作系统实现线程之间的切换需要从用户态到内核态的切换，切换成本非常高。

小结：只要一把锁存在多线程竞争，轻量锁就会膨胀为重量锁。

自旋锁

synchronized的轻量锁，重量锁，使用了自适应自旋锁进行性能优化

首先介绍自旋锁

表现：线程获取锁失败后，不会进入阻塞等待，而是再次尝试去获取锁，如此反复，直到获取到锁，者自旋结束那么会阻塞等待。

解决问题：在某些场景下，线程持有锁的时间非常短。在线程获取锁失败后，如果线程进入阻塞将会来线程上下文的切换，上下文切换的时间可能反而高于线程反复尝试获取锁的时间。

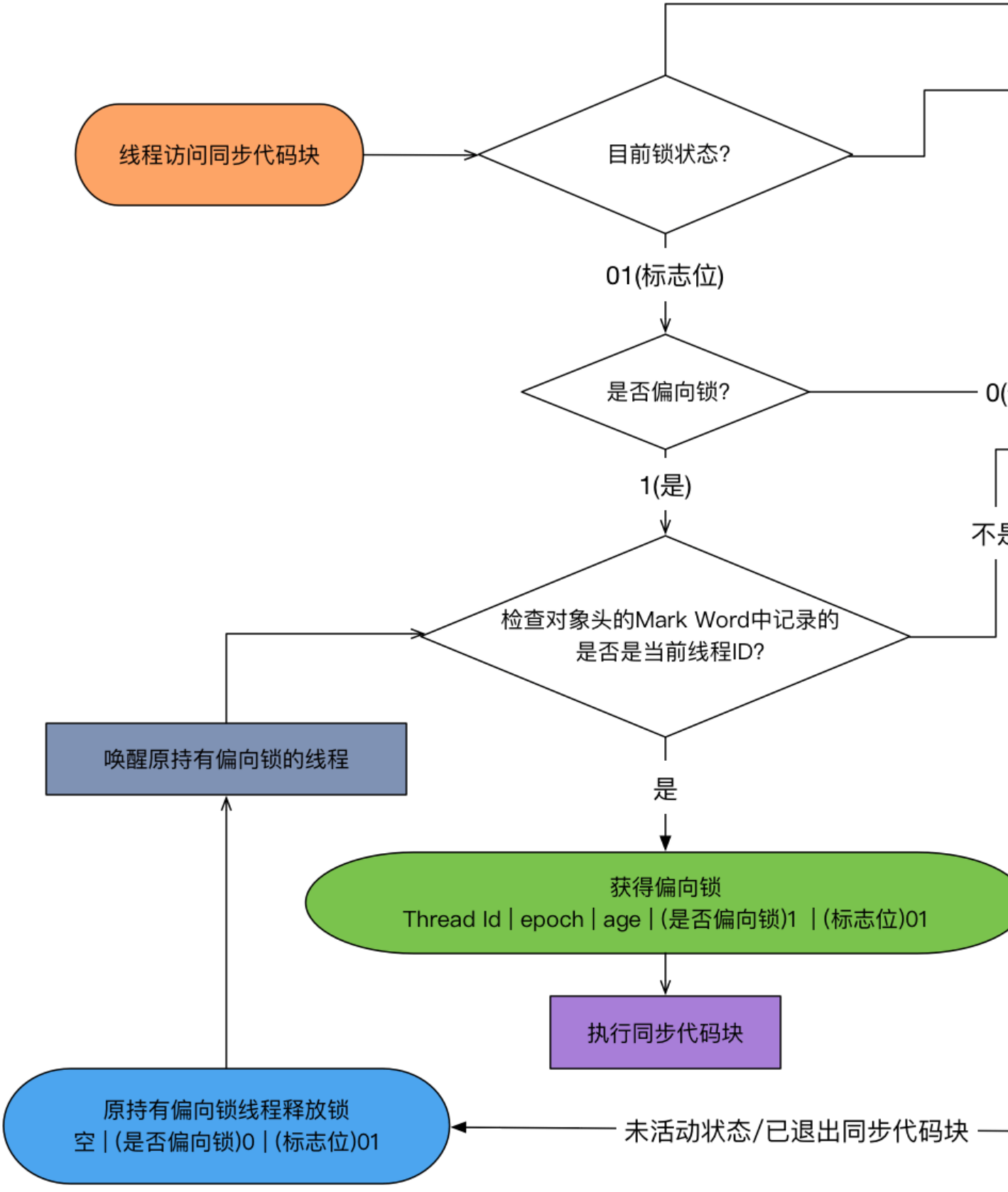
此时线程原地等待去重复获取锁。反而在性能上更有优势。

缺点：

1. 单核CPU没有线程并行，反复尝试会导致进程无法继续运行。
2. 重复尝试导致了CPU的占用，如果CPU资源紧张的话反而会性能下降
3. 如果锁的竞争时间过长，不仅没有性能提升，还浪费了大量CPU资源。

优化：使用自适应自旋锁。自适应自旋锁会根据之前的锁获取记录，优化调整自旋时间，避免造成不要的自旋。

具体synchronized流程



原文链接：Java 原理:synchronized 机制与几类锁