链滴

AQS 中 acquire(int) 方法调用 selfInterrupt 的理解

作者：wangduidui

原文链接：https://ld246.com/article/1618976505586

来源网站：链滴

```
/**
 * Acquires in exclusive mode, ignoring interrupts.  Implemented
 * by invoking at least once {@link #tryAcquire},
 * returning on success.  Otherwise the thread is queued, possibly
 * repeatedly blocking and unblocking, invoking {@link
 * #tryAcquire} until success.  This method can be used
 * to implement method {@link Lock#lock}.
 *
 * @param arg the acquire argument.  This value is conveyed to
 *        {@link #tryAcquire} but is otherwise uninterpreted and
 *        can represent anything you like.
 */
public final void acquire(int arg) {
    if (!tryAcquire(arg) &&
        acquireQueued(addWaiter(Node.EXCLUSIVE), arg))
        selfInterrupt();
}
```

selfInterrupt执行的前提是acquireQueued(addWaiter(Node.EXCLUSIVE), arg)方法返回true。这方法返回的是线程在获取锁的过程中是否发生过中断，返回true则证明发生过中断。所以acquire 中的elfInterrupt其实是对获取锁的过程中发生过的中断的补充。

为什么不直接用isInterrupt()判断，是因为在获取锁的过程中，是通过park+死循环实现的。每次par被唤醒之后都会重置中断状态，所以拿到锁的时候中断状态都是被重置后的。

---

acquireQueued(addWaiter(Node.EXCLUSIVE), arg)方法

```
/**
 * Acquires in exclusive uninterruptible mode for thread already in
 * queue. Used by condition wait methods as well as acquire.
 *
 * @param node the node
 * @param arg the acquire argument
 * @return {@code true} if interrupted while waiting
 */
final boolean acquireQueued(final Node node, int arg) {
    boolean failed = true;
    try {
        boolean interrupted = false;
        for (;;) {
            final Node p = node.predecessor();
            if (p == head && tryAcquire(arg)) {
                setHead(node);
                p.next = null; // help GC
                failed = false;
                return interrupted;
            }
            if (shouldParkAfterFailedAcquire(p, node) &&
                parkAndCheckInterrupt())
                interrupted = true;
```

```
            }
        } finally {
            if (failed)
                cancelAcquire(node);
        }
    }
```

先初始化是否发生过中断的标识为false。然后尝试获取锁，如果获取锁失败则会调用parkAndCheckInterrupt())方法，如果parkAndCheckInterrupt())返回了true则证明发生过中断，

将中断标记置为true，最后会返回这个中断标记。

---

parkAndCheckInterrupt())方法

```
    /**
     * Convenience method to park and then check if interrupted
     *
     * @return {@code true} if interrupted
     */
    private final boolean parkAndCheckInterrupt() {
        LockSupport.park(this);
        return Thread.interrupted();
    }
```

park当前线程，并且调用Thread.interrupted()方法返回中断状态，并且重置中断状态。

```
    /**
     * Tests whether the current thread has been interrupted.  The
     * <i>interrupted status</i> of the thread is cleared by this method.  In
     * other words, if this method were to be called twice in succession, the
     * second call would return false (unless the current thread were
     * interrupted again, after the first call had cleared its interrupted
     * status and before the second call had examined it).
     *
     * <p>A thread interruption ignored because a thread was not alive
     * at the time of the interrupt will be reflected by this method
     * returning false.
     *
     * @return  <code>true</code> if the current thread has been interrupted;
     *          <code>false</code> otherwise.
     * @see #isInterrupted()
     * @revised 6.0
     */
    public static boolean interrupted() {
        return currentThread().isInterrupted(true);
    }
```

---

park unpark原理简单理解

park和unpark 类似于信号量， 。park的时候会先判断变量是否大于0，如果大于0，会将变量置为0直接返回.

unpark的时候会将变量置为1，并判断之前这个变量是否等于0.等于0时要唤醒一个park的线程

所以park方法的注释中写了三种唤醒方式，unpark/interrupt/直接返回（如果先调用了unpark）

```
/**
 * Disables the current thread for thread scheduling purposes unless the
 * permit is available.
 *
 * <p>If the permit is available then it is consumed and the call returns
 * immediately; otherwise
 * the current thread becomes disabled for thread scheduling
 * purposes and lies dormant until one of three things happens:
 *
 * <ul>
 * <li>Some other thread invokes {@link #unpark unpark} with the
 * current thread as the target; or
 *
 * <li>Some other thread {@linkplain Thread#interrupt interrupts}
 * the current thread; or
 *
 * <li>The call spuriously (that is, for no reason) returns.
 * </ul>
 *
 * <p>This method does <em>not</em> report which of these caused the
 * method to return. Callers should re-check the conditions which caused
 * the thread to park in the first place. Callers may also determine,
 * for example, the interrupt status of the thread upon return.
 *
 * @param blocker the synchronization object responsible for this
 *        thread parking
 * @since 1.6
 */
public static void park(Object blocker) {
    Thread t = Thread.currentThread();
    setBlocker(t, blocker);
    UNSAFE.park(false, 0L);
    setBlocker(t, null);
}
```