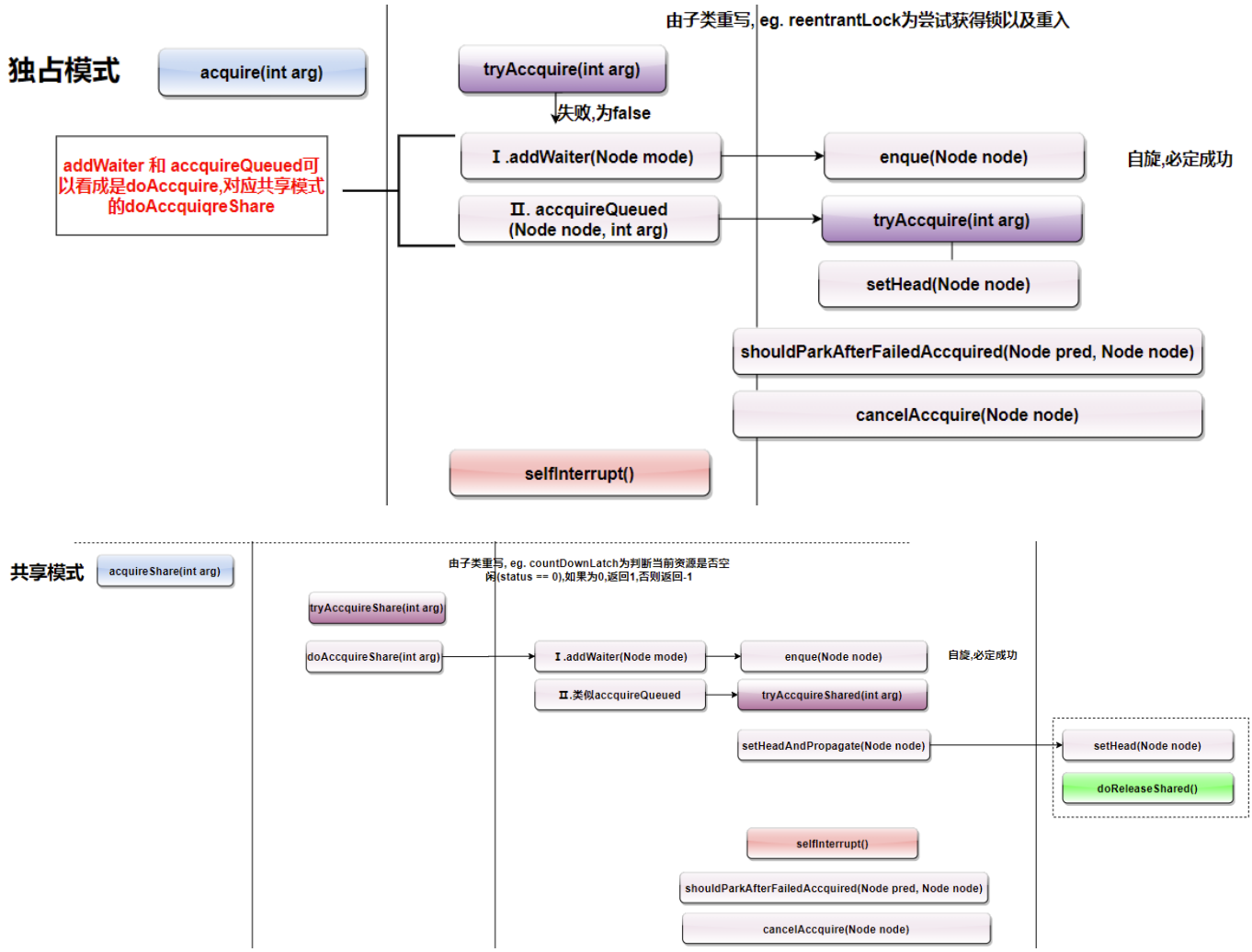
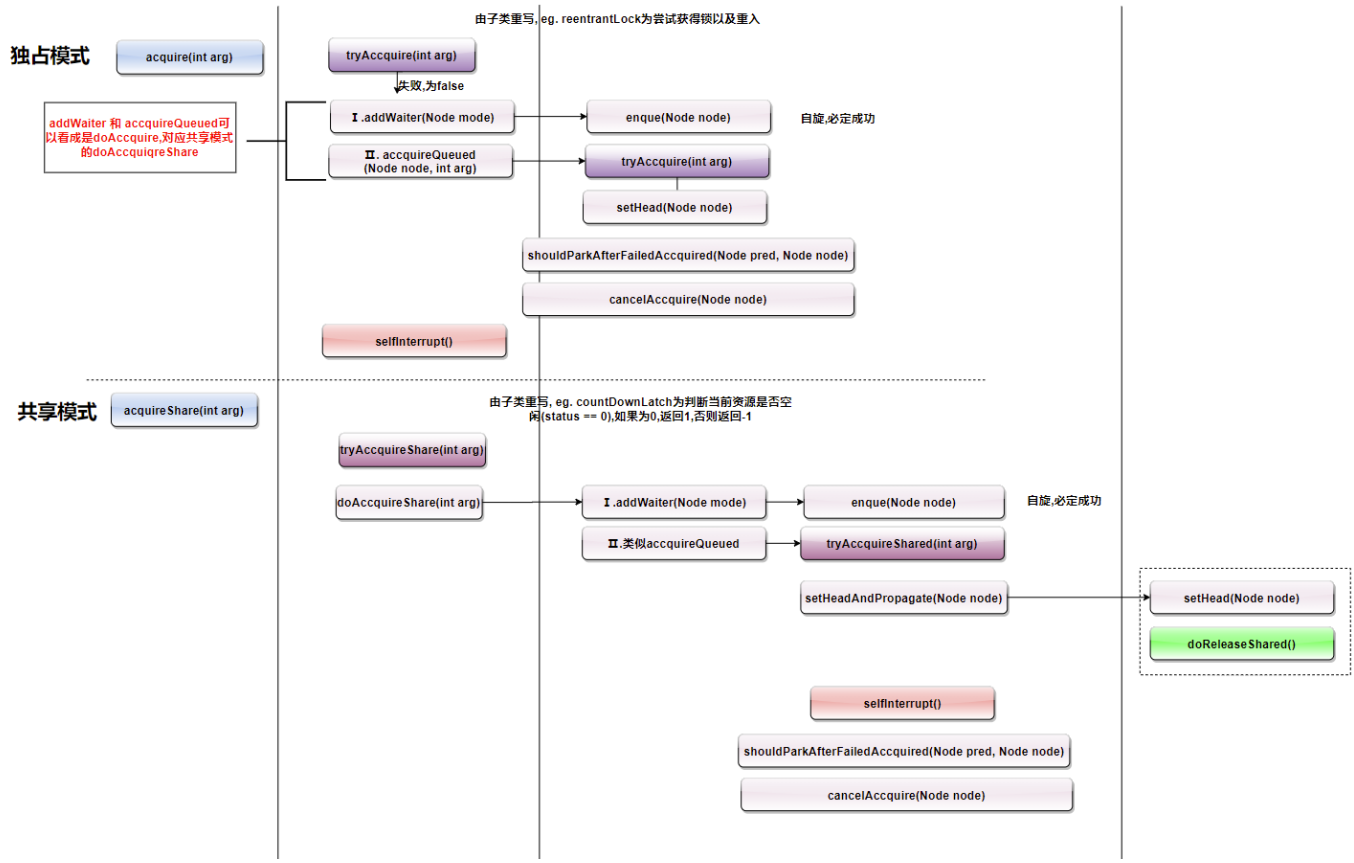
AQS

设计模式: **模板方法模式**

- 独占模式/共享模式





accquire和accquireShare区别

- accquire
  - tryAcquire尝试获得资源失败,并且将当前节点封装成node,阻塞队列入队成功,会调用**selfInterrupt**将当前线程挂起
- accquireShare
  - tryAcquireShare尝试获得arg个资源失败,会**doAcquireShare** 独占模式

```
public final void acquire(int arg) {
        if (!tryAcquire(arg) &&
            acquireQueued(addWaiter(Node.EXCLUSIVE), arg))
            selfInterrupt();
    }
```

共享模式

```
public final void acquireShared(int arg) {
        if (tryAcquireShared(arg) < 0)
            doAcquireShared(arg);
    }
```

acquireQueued和doAccquireShared

独占模式

```
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);
        }
    }
```

共享模式

```
private void doAcquireShared(int arg) {
        final Node node = addWaiter(Node.SHARED);
        // 下面类似于accquireQueued
        boolean failed = true;
        try {
            boolean interrupted = false;
            for (;;) {
                final Node p = node.predecessor();
                if (p == head) {
                    int r = tryAcquireShared(arg);
                    if (r >= 0) {
                        setHeadAndPropagate(node, r);
                        p.next = null; // help GC
                        if (interrupted)
                            selfInterrupt();
                        failed = false;
                        return;
                    }
                }
                if (shouldParkAfterFailedAcquire(p, node) &&
                    parkAndCheckInterrupt())
                    interrupted = true;
            }
        } finally {
            if (failed)
                cancelAcquire(node);
```

```
        }
    }
```

区别

- 区别1
  - 独占模式:先通过addWaiter封装线程为node节点,再调用acquireQueue方法
  - 共享模式:doAccuireShared中将线程封装为节点
- 区别2
  - 独占模式:采用tryAccquire(由子类重写)获得指定量的资源
  - 共享模式:采用tryAccquireShared(由子类重写)获得指定量的资源
    - r ← tryAccquireShared()
- 区别3
  - 独占模式:使用setHead(node)将当前线程设置为头节点
    - 只有解锁,tryRelease之后,猜用调用**doRelease**尝试释放资源
  - 共享模式:使用setHeadAndPropagate(node, r)将当前线程设置为头节点
    - 设置头节点之后,会调用**doReleanseShared**释放资源
- 区别4:
  - 独占模式:在设置完头节点之后,返回中断标志
    - 即:设置完头节点之后,不会响应中断
  - 共享模式:设置完头节点后,如果
- 区别5:
  - 在shouldParkAfterFailAccuire中的处理方式不同

区别2:tryAccuire和tryAccquireShared区别

独占模式(ReentrantLock实现)

- 当资源可用(==0),尝试抢锁
- 当资源不可用,比对当前线程是否为持锁线程,如果是,重入
- 方法返回结果是**boolean**,表示抢占锁或重入成功/失败

```
protected final boolean tryAcquire(int acquires) {
            final Thread current = Thread.currentThread();
            int c = getState();
            if (c == 0) {
                if (!hasQueuedPredecessors() &&
                    compareAndSetState(0, acquires)) {
                    setExclusiveOwnerThread(current);
                    return true;
                }
            }
            else if (current == getExclusiveOwnerThread()) {
                int nextc = c + acquires;
                if (nextc < 0)
                    throw new Error("Maximum lock count exceeded");
                setState(nextc);
                return true;
            }
```

```
            return false;
        }
```

共享模式(CountDownLatch实现)

- 只判断,不修改
  - 当资源可用, 返回 1
  - 否则, 返回 -1
- 方法返回结果为**int**

```
protected int tryAcquireShared(int acquires) {
            return (getState() == 0) ? 1 : -1;
        }
```

区别3:setHead和setHeadAndPropagate区别

setHead

- 其实就是将当前线程设置为头节点

```
private void setHead(Node node) {
        head = node;
        node.thread = null;
        node.prev = null;
    }
```

setHeadAndPropagate(node, r)

- r为tryAccquireShare的结果,表示剩余可共享的资源
- h ← 保留原head的引用
- 调用set head,将当前node设置为头节点
- 若还存在可共享的资源(r > 0)
  - 或者 (无锁, r == 0) 原来队列不存在
  - 或者 (无锁, r == 0) 原来队列存在,原头节点值为SIGNAL, CONDITION, 或 PROPAGATE
  - 或者 (无锁, r == 0) 现在队列不存在
  - 或者 (无锁, r == 0) 现在队列存在,现头节点值为SIGNAL, CONDITION, 或 PROPAGATE
  - s ← **当node的下一个节点**
    - 如果s为null或者s也处于共享模式,**doReleaseShare()**

```
private void setHeadAndPropagate(Node node, int propagate) {
        Node h = head; // Record old head for check below
        setHead(node);
        /*
         * Try to signal next queued node if:
         *   Propagation was indicated by caller,
```

```
             *     or was recorded (as h.waitStatus either before
             *     or after setHead) by a previous operation
             *     (note: this uses sign-check of waitStatus because
             *      PROPAGATE status may transition to SIGNAL.)
             * and
             *   The next node is waiting in shared mode,
             *     or we don't know, because it appears null
             *
             * The conservatism in both of these checks may cause
             * unnecessary wake-ups, but only when there are multiple
             * racing acquires/releases, so most need signals now or soon
             * anyway.
             */
            if (propagate > 0 || h == null || h.waitStatus < 0 ||
                (h = head) == null || h.waitStatus < 0) {
                Node s = node.next;
                if (s == null || s.isShared())
                    doReleaseShared();
            }
        }
```

区别5:shouldParkAfterFailAccuire中对共享模式和独占模式的区别

- 前提:
  - 进入到shouldParkAfterFailAccquire表明此时线程已经tryAccquire试图获取锁失败
- 如果节点处于SIGNAL(-1)状态
  - `/** waitStatus value to indicate successor's thread needs unparking */`
  - 暗示后继节点需要被唤醒
  - 返回true
- 否则,如果节点 > 0
  - 表明当前节点处于CANCEL状态
  - node.prev = pred = pred.prev;,可以看成是
    - pred = pred.prev
    - node.prev = prev
  - 即从node出发,一直往前,一直找到第一个waitStatus <= 0的节点
- 否则,如果节点不为-1,不>0
  - /* * waitStatus must be 0 or PROPAGATE. Indicate that we * need a signal, but don't park yet. Caller will need to * retry to make sure it cannot acquire before parking. */
  - 当前节点的状态可能为0或**-3,共享模式**,表明需要一个中断信号,但是还没有挂起
  - 调用方重试,需要确保ta在挂起前不能被获得

```
private static boolean shouldParkAfterFailedAcquire(Node pred, Node node) {
        int ws = pred.waitStatus;
        if (ws == Node.SIGNAL)
```

```
            /*
             * This node has already set status asking a release
             * to signal it, so it can safely park.
             */
            return true;
        if (ws > 0) {
            /*
             * Predecessor was cancelled. Skip over predecessors and
             * indicate retry.
             */
            do {
                node.prev = pred = pred.prev;
            } while (pred.waitStatus > 0);
            pred.next = node;
        } else {
            /*
             * waitStatus must be 0 or PROPAGATE.  Indicate that we
             * need a signal, but don't park yet.  Caller will need to
             * retry to make sure it cannot acquire before parking.
             */
            compareAndSetWaitStatus(pred, ws, Node.SIGNAL);
        }
        return false;
    }
```