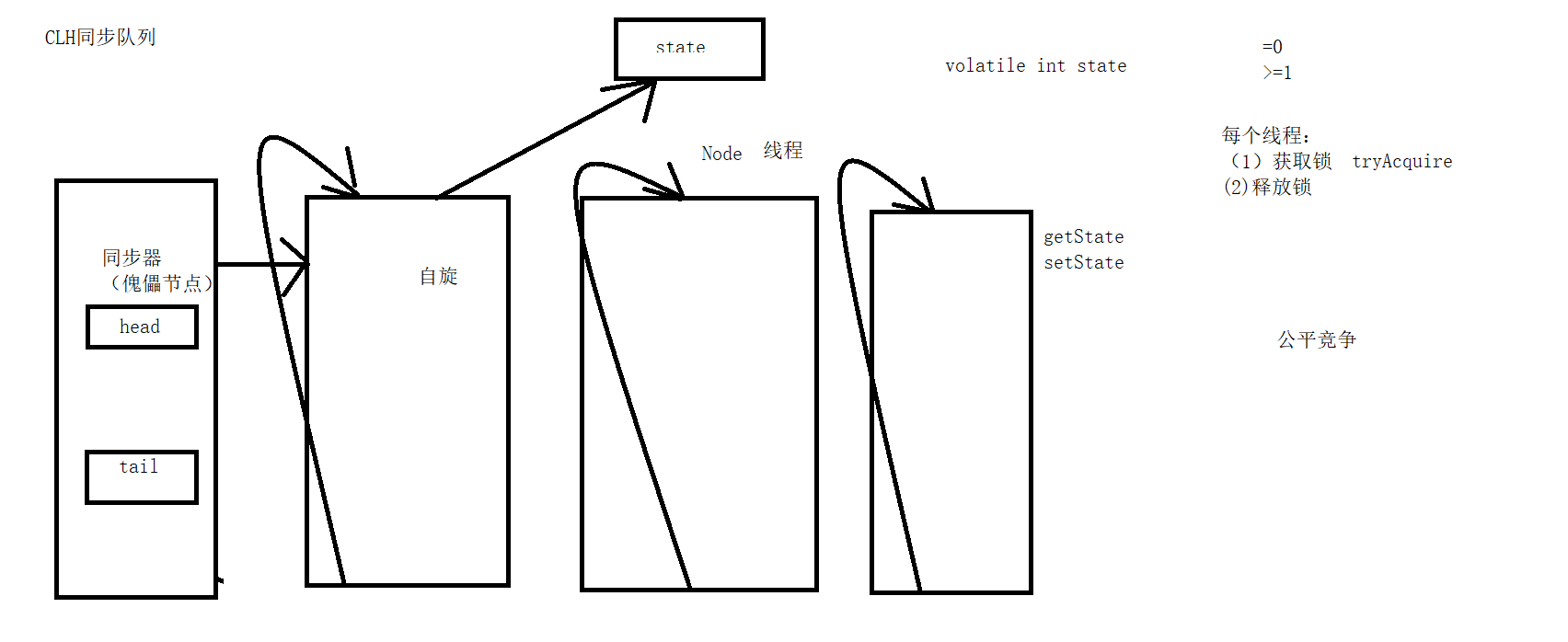
锁降级 ： 写锁 降级到 读锁

锁升级 ： 读锁 降级到 写锁 （ReenansLock内部不支持）

一、锁

1. AQS写一个锁（） CLH



Lock

Synchronized

CAS AQS子类定义为非公共内部帮助器类（私有的内部类继承AQS）,写锁的时候的一个帮助器，提供获取锁和是释放锁的功能.模板。

acquire(int arg) 以独占模式获取对象，忽略中断。

acquireShared(int arg) 以共享模式获取对象，忽略中断。

tryAcquire(arg) 试图在独占模式下获取对象状态。

tryAcquireShared(int arg) 试图在共享模式下获取对象状态

**[release](http://tool.oschina.net/uploads/apidocs/jdk-zh/java/util/concurrent/locks/AbstractQueuedSynchronizer.html" \l "release(int))**(int arg)           以独占模式释放对象。

**[releaseShared](http://tool.oschina.net/uploads/apidocs/jdk-zh/java/util/concurrent/locks/AbstractQueuedSynchronizer.html" \l "releaseShared(int))**(int arg)           以共享模式释放对象

实例：

可重入性：同一个锁多统一资源进行占有的时候，直接分配给这个线程(无可重入性 就会死锁)

1. ReentrantLock

公平锁

非公平锁

1. ReentrantReaderWriter

读取者优先或写入者优先强加给锁访问的排序。但是，它确实支持可选的公平策略。

1. 并发工具使用

APP--->三个线程到不同的公司查询--》result

1. CountdownLatch（查询航班）

原理 ： 类似 Thread的 join 方法， 待线程执行完毕，继续执行主线程。

CountdownLatch 意思是待所有线程执行完毕 ，然后串行执行 后面的逻辑。

在CountdownLatch 内部依赖AQS , 创建CountdownLatch 需要往构造器传 默认线程数 ， 当线程每执行玩一个任务时，线程数 减一 。 当线程数 = 0 的时候 countDownLatch.waite() ; 主线程执行

使用 ： CountdownLatch c = new CountdownLatch (2);

... 逻辑

c.countDown(); // state -1

c.waite() ; //唤醒主线程

1. CyclieBarries /ˈsaɪklɪk ˈbæriər/ : 循环屏障

待所有线程准备就绪后 ， 然后几乎同时 开始处理 逻辑 （类似一道屏障）

多构造器：

**public** CyclicBarrier(**int** parties , Runnable barrierAction)

**public** CyclicBarrier(**int** parties )

parties ： 线程数

barrierAction ： 当线程数 全部准备好 立即执行的 逻辑 run() {}

与CountdownLatch 不同的是 ，CyclicBarrier 的初始 int值时 0 ，当新的线程就绪后 就会 -1 。直到 线程数等于 0 然后执行逻辑

用法 ： CyclicBarrier c = new CyclicBarrier(2);

...逻辑

c.awit(); //parties -1 当默认减到0 表示所有线程准备就绪

## **CyclicBarrier与CountDownLatch比较**

1. CountDownLatch:一个线程(或者多个)，等待另外N个线程完成某个事情之后才能执行；CyclicBarrier:N个线程相互等待，任何一个线程完成之前，所有的线程都必须等待。

CountDownLatch.wait(时间，单位); 在设置时间内如果 N个线程还没完成就 直接走主线程，不在等待 ， n个线程还是继续做他们的事情

　　2）CountDownLatch:一次性的；CyclicBarrier:可以重复使用。

　　3）CountDownLatch基于AQS；CyclicBarrier基于锁和Condition。本质上都是依赖于volatile和CAS实现的。

1. Semaphore /ˈseməfɔːr/ 信号量 ****许可证管理器****

并发下 用于协调 各个线程 ，以保证合理高效的 处理资源

作业：

1. 自己写一个锁。实现生产者和消费者（集合）
2. countdownLatch、CyclicBarrier