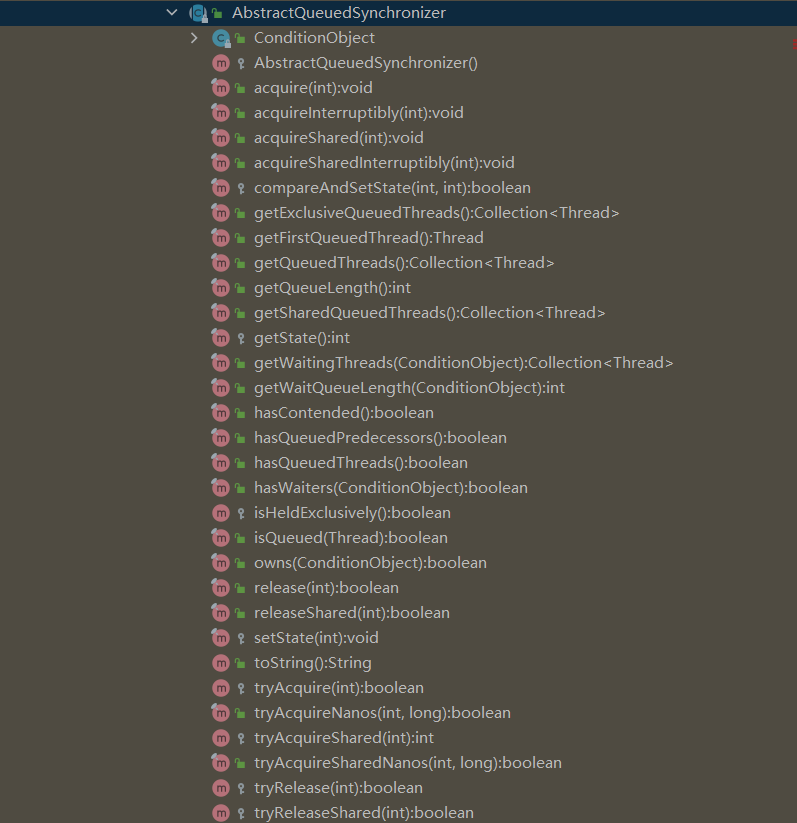
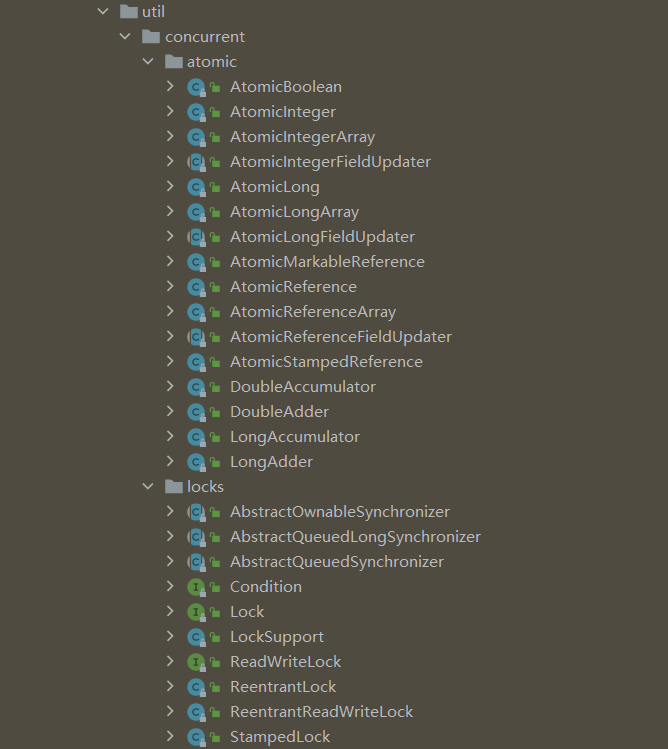
AQS 是什么？ AQS 的全称为 AbstractQueuedSynchronizer ，翻译过来的意思就是抽象队列同步器。这个类在 java.util.concurrent.locks 包下面。



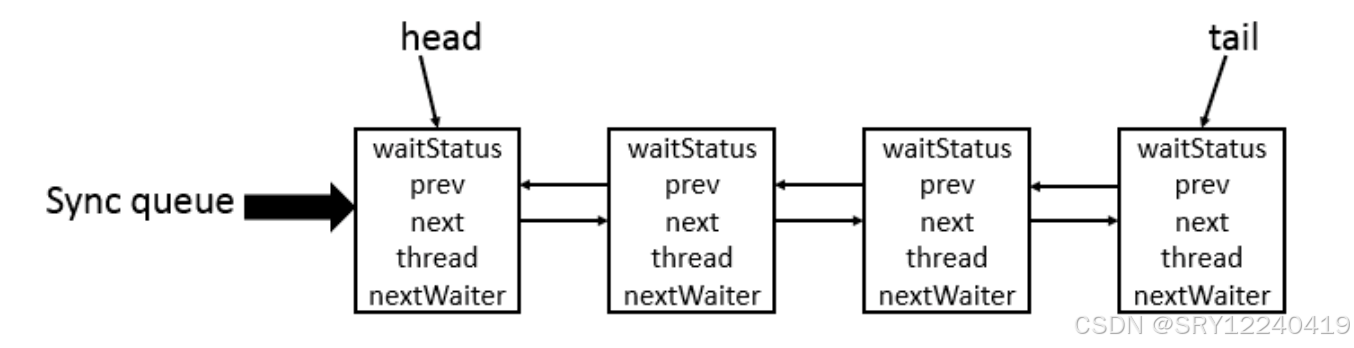


AQS是一个用来构建锁和同步器的框架，使用AQS能简单且高效地构造出应用广泛的大量的同步器，比如我们提到的ReentrantLock，Semaphore，其他的诸如ReentrantReadWriteLock，SynchronousQueue，FutureTask等等皆是基于AQS的。

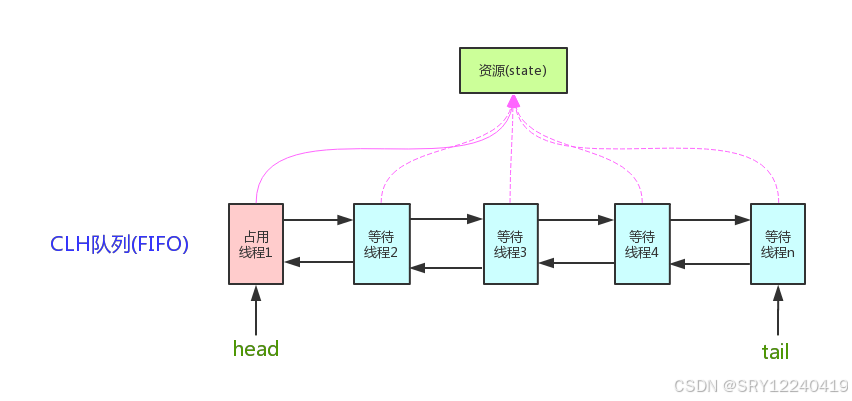
AQS 的原理是什么？AQS核心思想是,如果被请求的共享资源空闲，则将当前请求资源的线程设置为有效的工作线程，并且将共享资源设置为锁定状态。

如果被请求的共享资源被占用，那么就需要一套线程阻塞等待以及被唤醒时锁分配的机制,这个机制AQS是用CLH队列锁实现的,即将暂时获取不到锁的线程加入到队列中。CLH(Craig,Landin,and Hagersten) 队列是一个虚拟的双向队列(虚拟的双向队列即不存在队列实例，仅存在结点之间的关联关系)。AQS是将每条请求共享资源的线程封装成一个 CLH锁队列的一个结点(Node)来实现锁的分配。在 CLH 同步队列中，一个节点表示一个线程，它保存着线程的引用(thread)、 当前节点在队列中的状态(waitStatus)、 前驱节点(prev)

、后继节点(next)。 CLH 队列结构如下图所示:

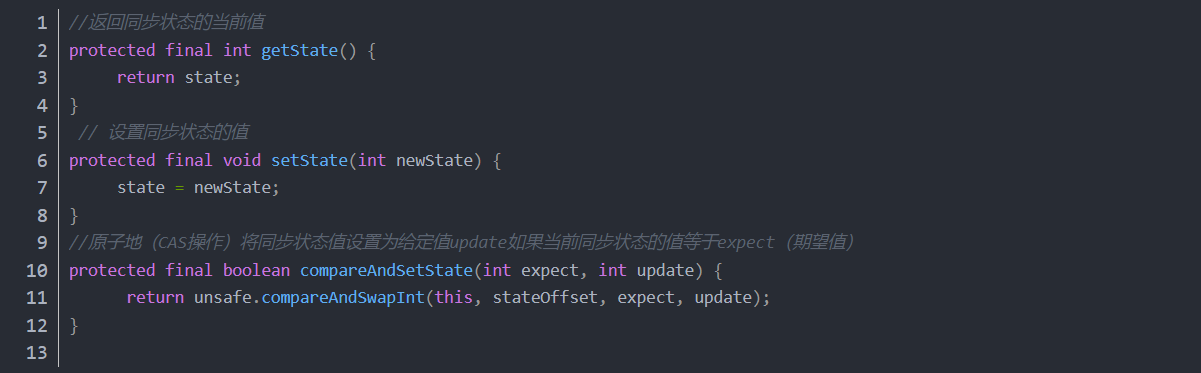


AQS(AbstractQueuedSynchronizer)的核心原理图如下:



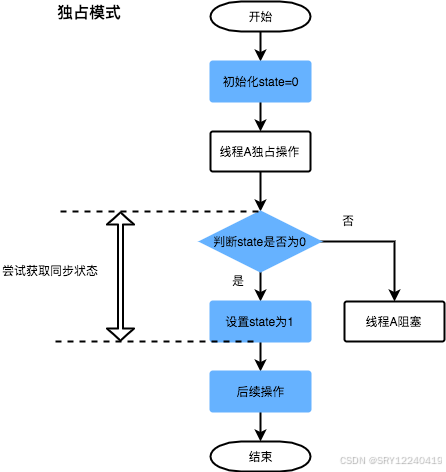
AQS 提供了一个由volatile修饰,并且采用CAS方式修改的int类型的成员变量state表示同步状态，通过内置的线程等待队列来完成获取资源线程的排队工作。

另外，状态信息 state 可以通过 protected 类型的getState()、setState()和compareAndSetState()进行操作。并且，这几个方法都是final修饰的，在子类中无法被重写。



以可重入的互斥锁 ReentrantLock 为例，它的内部维护了一个 state 变量，用来表示锁的占用状态。state 的初始值为 0，表示锁处于未锁定状态。当线程 A 调用 lock() 方法时，会尝试通过 tryAcquire() 方法独占该锁，并让 state 的值加 1。如果成功了，那么线程 A 就获取到了锁。如果失败了，那么线程 A 就会被加入到一个等待队列（CLH 队列）中，直到其他线程释放该锁。假设线程 A 获取锁成功了，释放锁之前，A 线程自己是可以重复获取此锁的（state 会累加）。这就是可重入性的体现：一个线程可以多次获取同一个锁而不会被阻塞。但是，这也意味着，一个线程必须释放与获取的次数相同的锁，才能让 state 的值回到 0，也就是让锁恢复到未锁定状态。

只有这样，其他等待的线程才能有机会获取该锁。



AQS 资源共享方式

AQS 定义两种资源共享方式：Exclusive（独占，只有一个线程能执行，如ReentrantLock）和Share（共享，多个线程可同时执行，如Semaphore/CountDownLatch）。一般来说，自定义同步器的共享方式要么是独占，要么是共享，他们也只需实现tryAcquire-tryRelease、tryAcquireShared-tryReleaseShared中的一种即可。但AQS也支持自定义同步器同时实现独占和共享两种方式，如ReentrantReadWriteLock。

总结

在AQS中维护者一个FIFO的同步队列,当线程获取同步状态失败后,则会加入到这个CLH同步队列的队尾并一直保持着自旋.在CLH同步队列中的线程在自旋时会判断其前驱节点是否为首节点,如果为首节点则不断尝试获取同步获取状态(CAS机制),获取成功后则退出CLH同步队列.当线程执行完逻辑后,释放同步状态,并唤醒其后继节点。