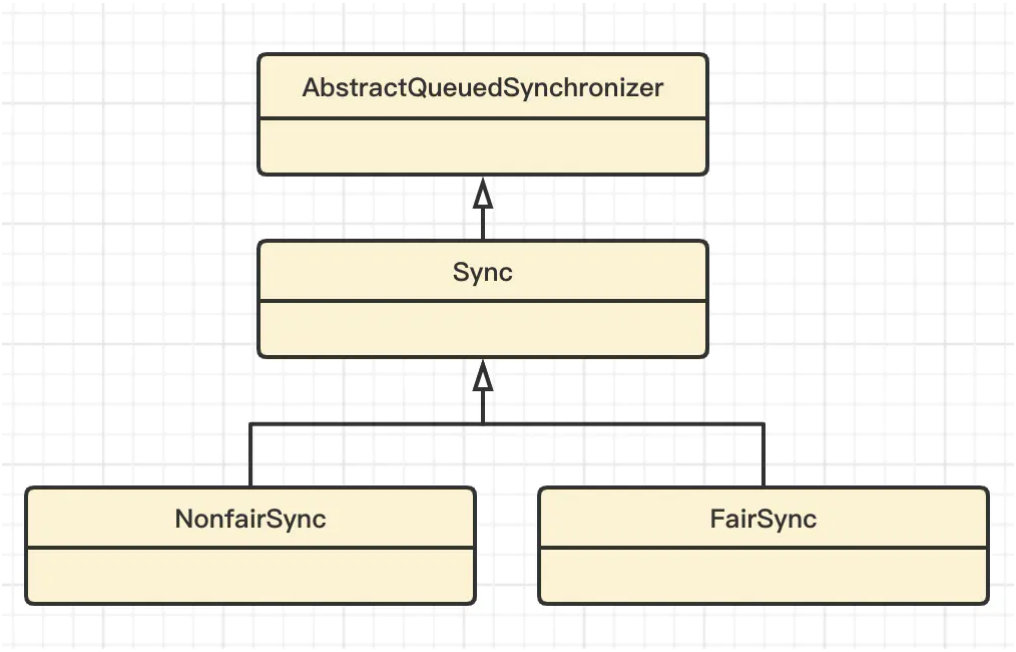
ReentrantLock(重入锁)



ReentrantLock继承了AQS实现了非公平锁NonfairSync和公平锁FairSync两种功能，并且支持可重入锁(同一个线程多次加锁，例如方法递归调用场景，避免死锁)；实现了Lock接口的lock、trylock、unlock等函数；默认使用非公平锁.

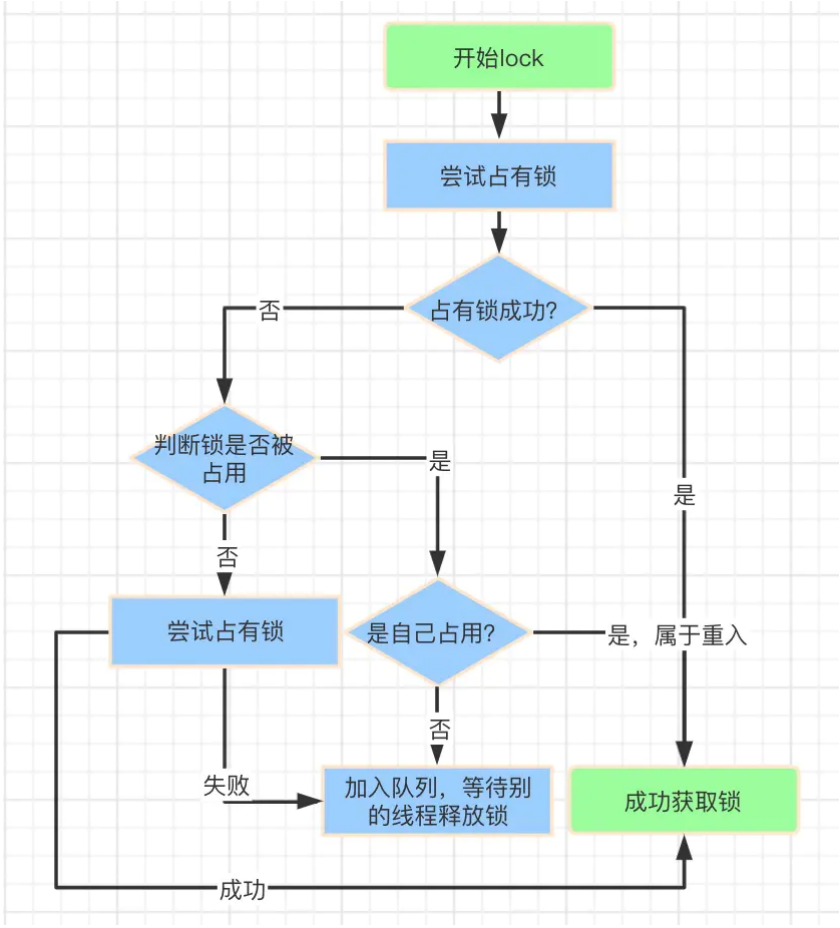
公平/非公平

公平/非公平是指线程获取锁的顺序是否按排队的顺序(等的时间越长优先级越高)，如果当前线程不排队直接去竞争锁，比同步队列中的线程先获得锁，则是非公平方式，否则是公平的方式.

非公平锁

- tryAcquire

  非公平锁中tryAcquire调用nonfairSync函数进行锁的竞争，首先获得state，然后判断state==0，如果为0则表示无线程持有，则**直接用CAS尝试将state置为1**，进行锁的争抢；不为0，则表示当前已经有线程持有，则判断当前线程是否为自己，若为自己则将state+1进行重入锁，否则返回false.



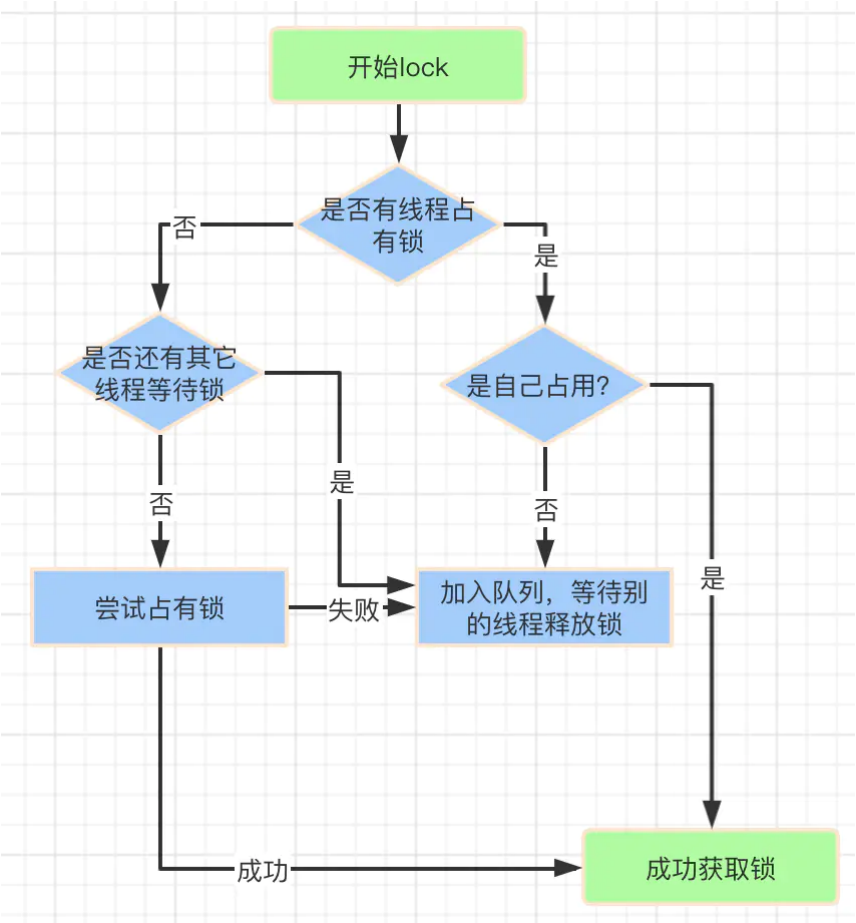
- `tryRelease`

  锁的释放`tryRelease`函数将当前`state`设置为`state-1(独占锁无序CAS)`，如果`state=0`，则表示完全释放，否则表示还被当前线程持有.

公平锁

- `tryAcquire`

  公平锁中`tryAcquire`首先获得`state`，判断`state`是否为`0`，为`0`时**先判断当前同步队列中是否有有效的节点**，有则竞争失败返回`false`，没有则进行锁的争抢；`state>0`，判断是否可重入，可重入则`+1`，否则返回`false`.



- tryRelease

  锁的释放tryRelease函数将当前state设置为state-1(独占锁无序CAS)，如果state=0，则表示完全释放，否则表示还被当前线程持有.

Lock接口实现

- lock
  - nonfairSyn

    非公平锁实现lock时，当state=0时，先去CAS抢占一次，失败再去调用acquire进行非公平锁抢占.
  - fairSyn

    公平锁实现lock时，当state=0时，直接调用acquire进行公平锁抢占.
- trylock

  调用非公平锁nonfairTryAcquire进行一次抢占，失败则退出，不加入同步队列等待；也实现了超时抢占，抢占失败后进入同步队列进行限时等待.
- unlock

  调用release进行锁释放.
- lockInterruptibly

  利用AQS实现了中断响应加锁.
- 等待/通知机制

  利用AQS实现了等待/通知机制.

- **参考资料**
  - Java 并发之 ReentrantLock 深入分析(与Synchronized区别) - 简书 (jianshu.com)