ReentrantLock实现Lock接口，在ReentrantLock中引用了AbstractQueuedSynchronizer的子类，所有的同步操作都是依靠AbstractQueuedSynchronizer（队列同步器）实现。

ReentrantLock支持两种锁模式，公平锁和非公平锁。默认的实现是非公平的

1. reentrantLock重入锁，是实现Lock接口的一个类，也是在实际编程中使用频率很高的一个锁，**支持重入性，表示能够对共享资源能够重复加锁，即当前线程获取该锁再次获取不会被阻塞**。在java关键字synchronized隐式支持重入性（关于synchronized可以[看这篇文章](https://juejin.im/post/5ae6dc04f265da0ba351d3ff)），synchronized通过获取自增，释放自减的方式实现重入。与此同时，ReentrantLock还支持**公平锁和非公平锁**两种方式。那么，要想完完全全的弄懂ReentrantLock的话，**主要也就是ReentrantLock同步语义的学习：1. 重入性的实现原理；2. 公平锁和非公平锁。**
2. ReentrantLock支持两种锁：**公平锁**和**非公平锁**。**何谓公平性，是针对获取锁而言的，如果一个锁是公平的，那么锁的获取顺序就应该符合请求上的绝对时间顺序，满足FIFO**。ReentrantLock的构造方法无参时是构造非公平锁，源码为：**公平锁每次都是从同步队列中的第一个节点获取到锁，而非公平性锁则不一定，有可能刚释放锁的线程能再次获取到锁**。
3. **公平锁 VS 非公平锁**
4. 1,公平锁每次获取到锁为同步队列中的第一个节点，**保证请求资源时间上的绝对顺序**，而非公平锁有可能刚释放锁的线程下次继续获取该锁，则有可能导致其他线程永远无法获取到锁，**造成“饥饿”现象**。2,公平锁为了保证时间上的绝对顺序，需要频繁的上下文切换，而非公平锁会降低一定的上下文切换，降低性能开销。因此，ReentrantLock默认选择的是非公平锁，则是为了减少一部分上下文切换，保证了系统更大的吞吐量。

可以看出要想理解ReentrantLock关键核心在于对队列**同步器AbstractQueuedSynchronizer（简称同步器）**的理解。

同步器是用来构建锁和其他同步组件的基础框架，它的实现主要依赖一个int成员变量来表示同步状态以及通过一个FIFO队列构成等待队列。它的**子类必须重写AQS的几个protected修饰的用来改变同步状态的方法**，其他方法主要是实现了排队和阻塞机制。**状态的更新使用getState,setState以及compareAndSetState这三个方法**。

AQS的设计是使用模板方法设计模式，它将**一些方法开放给子类进行重写，而同步器给同步组件所提供模板方法又会重新调用被子类所重写的方法**。举个例子，AQS中需要重写的方法tryAcquire：

AQS提供的模板方法可以分为3类：1,独占式获取与释放同步状态；2,共享式获取与释放同步状态；3,查询同步队列中等待线程情况；