多线程同步--Lock

笔记本:	java-多线程		
创建时间:	2019/5/4 16:39	**更新时间:**	2019/5/4 17:01
作者:	ANNER		
URL:	https://juejin.im/post/5ab9a5b46fb9a028ce7b9b7e		

Lock接口

在Lock接口出现之前，Java程序是靠synchronized关键字实现锁功能的。JDK1.5之后并发包中新增了Lock接口以及相关实现类来实现锁功能。

Lock接口的实现类：

- ReentrantLock
- ReentrantReadWriteLock.ReadLock
- ReentrantReadWriteLock.WriteLock

最好不要把获取锁的过程写在try语句块中，因为如果在获取锁时发生了异常，异常抛出的同时也会导致锁无法被释放

lock接口提供了synchronized关键字不具备的主要特性：

特性	描述
尝试非阻塞地获取锁	当前线程尝试获取锁，如果这一时刻锁没有被其他线程获取到，则成功获取并持有锁
能被中断地获取锁	获取到锁的线程能够响应中断，当获取到锁的线程被中断时，中断异常将会被抛出，同时锁会被释放
超时获取锁	在指定的截止时间之前获取锁， 超过截止时间后仍旧无法获取则返回

Lock接口基本的方法：

方法名称	描述
void lock()	获得锁。如果锁不可用，则当前线程将被禁用以进行线程调度，并处于休眠状态，直到获取锁。
void lockInterruptibly()	获取锁，如果可用并立即返回。如果锁不可用，那么当前线程将被禁用以进行线程调度，并且处于休眠状态，和lock()方法不同的是在锁的获取中可以中断当前线程（相应中断）。
Condition newCondition()	获取等待通知组件，该组件和当前的锁绑定，当前线程只有获得了锁，才能调用该组件的wait()方法，而调用后，当前线程将释放锁。
boolean tryLock()	只有在调用时才可以获得锁。如果可用，则获取锁定，并立即返回值为true；如果锁不可用，则此方法将立即返回值为false 。
boolean tryLock(long time, TimeUnit unit)	超时获取锁，当前线程在一下三种情况下会返回： 1. 当前线程在超时时间内获得了锁；2.当前线程在超时时间内被中断；3.超时时间结束，返回false.
void unlock()	释放锁。

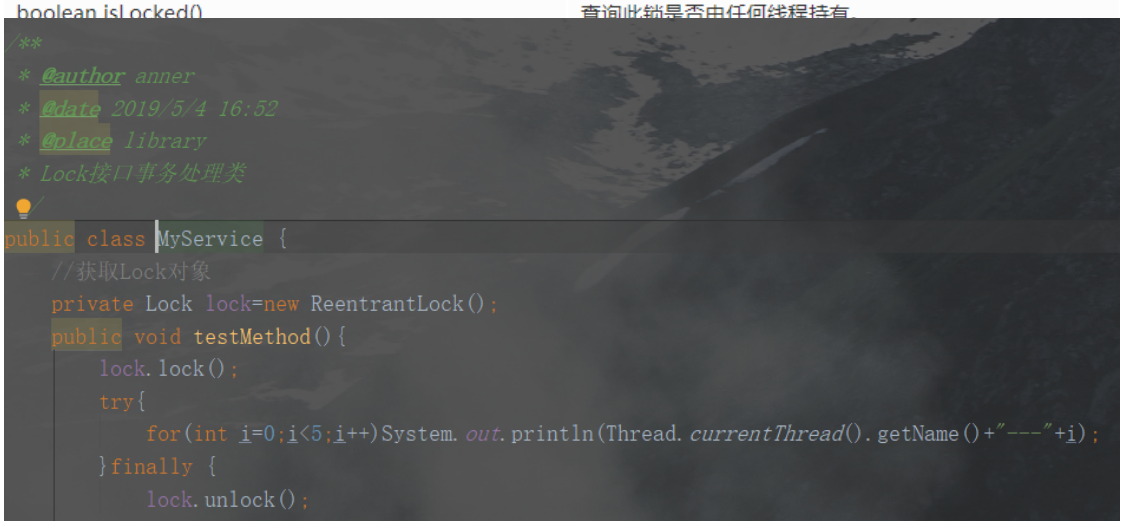
Lock接口的实现类：ReentrantLock

构造方法：

方法名称	描述
ReentrantLock()	创建一个 ReentrantLock的实例。
ReentrantLock(boolean fair)	创建一个特定锁类型（公平锁/非公平锁）的ReentrantLock的实例

ReentrantLock类常见方法(Lock接口已有方法这里没加上)：

方法名称	描述
int getHoldCount()	查询当前线程保持此锁定的个数，也就是调用lock()方法的次数。
protected Thread getOwner()	返回当前拥有此锁的线程，如果不拥有，则返回 null
protected Collection getQueuedThreads()	返回包含可能正在等待获取此锁的线程的集合
int getQueueLength()	返回等待获取此锁的线程数的估计。
protected Collection getWaitingThreads(Condition condition)	返回包含可能在与此锁相关联的给定条件下等待的线程的集合。
int getWaitQueueLength(Condition condition)	返回与此锁相关联的给定条件等待的线程数的估计。
boolean hasQueuedThread(Thread thread)	查询给定线程是否等待获取此锁。
boolean hasQueuedThreads()	查询是否有线程正在等待获取此锁。
boolean hasWaiters(Condition condition)	查询任何线程是否等待与此锁相关联的给定条件
boolean isFair()	如果此锁的公平设置为true，则返回 true 。
boolean isHeldByCurrentThread()	查询此锁是否由当前线程持有。
boolean isLocked()	查询此锁是否由任何线程持有。



```
/**
 * @author anner
 * @date 2019/5/4 16:52
 * @place library
 * Lock接口事务处理类
 */
public class MyService {
    //获取Lock对象
    private Lock lock=new ReentrantLock();
    public void testMethod(){
        lock.lock();
        try{
            for(int i=0;i<5;i++)System.out.println(Thread.currentThread().getName()+"---"+i);
        }finally {
            lock.unlock();
```

Condition接口简介

我们通过之前的学习知道了：synchronized关键字与wait()和notify/notifyAll()方法相结合可以实现等待/通知机制，ReentrantLock类当然也可以实现，但是需要借助于

Condition接口与newCondition() 方法。Condition是JDK1.5之后才有的，它具有很好的灵活性，比如可以实现多路通知功能也就是在**一个Lock对象中可以创建多个Condition实例（即对象监视器），线程对象可以注册在指定的Condition中，从而可以有选择性的进行线程通知，在调度线程上更加灵活**

在使用notify/notifyAll()方法进行通知时，被通知的线程是有JVM选择的，**使用ReentrantLock类结合Condition实例可以实现“选择性通知”**，这个功能非常重要，而且是Condition接口默认提供的。

而**synchronized关键字就相当于整个Lock对象中只有一个Condition实例**，所有的线程都注册在它一个身上。如果执行notifyAll()方法的话就会通知所有处于等待状态的线程这样会造成很大的效率问题，而Condition实例的signalAll()方法 只会唤醒注册在该Condition实例中的所有等待线程

Condition接口的常见方法：

方法名称	描述
void await()	相当于Object类的wait方法
boolean await(long time, TimeUnit unit)	相当于Object类的wait(long timeout)方法
signal()	相当于Object类的notify方法
signalAll()	相当于Object类的notifyAll方法