1. Lock的使用
2. ReentrantLock类

之前我们使用了synchronized关键字实现线程之间的同步互斥，在JDK1.5中添加了ReentrantLock类也能够达到同样的效果，并且扩展了新的功能。

1. lock() 与 unlock()

创建一个新的ReentrantLock锁对象，调用该锁对象的lock()方法获得锁，线程调用了该方法，就持有了对象监视器，其他线程只有等待锁被释放时再次争抢。同样的，调用该所对象的unlock()方法释放锁。

private Lock lock = new ReentrantLock();

lock.lock();

lock.unlock();

1. Condition类

我们可以通过Condition对象实现等待/通知模式，这是在JDK5中出现的技术，有较好的灵活性，可以实现多路通知，即在一个Lock对象内创建多个Condition(对象监视器)实例，线程对象可以注册在指定的Condition中，从而可以有选择性地进行线程通知。而synchronized相当于整个Lock对象中只有一个单一的Condition对象，所有线程都注册在一个对象上，通知是随机的。

private Lock lock = new ReentrantLock();

private Condition condition = lock.newCondition();

contion.await();

condition,signal();

首先在一个线程中调用Lock对象的lock()方法获取锁对象，然后调用Condition对象的await()方法，使当前执行任务的线程进入等待状态，最后调用Condition对象的signal()方法，通知该Condition对象处理等待中的线程。还有signalAll()方法是唤醒所有等待的线程。

将不同的线程注册到不同的Condition对象中，通过调用不同的Condition对象的notifyAll()方法或者notify()方法就可以控制唤醒的线程对象。

1. 生产者/消费者模式

与之前使用wait()/notify()一样，这里使用的是Condition对象的await()/signal(), signalAll()方法。

1. 公平锁与非公平锁

锁对象Lock分为公平锁和非公平锁。公平锁表示线程获取锁的顺序是按照线程加锁的顺序来分配的，即先来先得的FIFO顺序，但是只能保证大多数是按照顺序的，总会有少数不按顺序分配的。而非公平锁就是随机获取锁的机制。

在创建ReentrantLock锁对象的时候，传入一个Boolean值表示是否是公平锁：

Lock lock = new ReentrantLock(true);

1. Lock锁对象的方法
2. getHoldCount()

该方法查询当前线程保持此锁对象的个数，也就是该锁对象调用lock()方法的次数。

lock.getHoldCount();

1. getQueueLength()

返回正在等待获取此锁对象的线程估计数，如共有5个线程，当1个线程先执行了await()方法后，那么还有4个线程在等待着该锁对象的释放。

lock.getQueueLength();

1. getWaitQueueLength(Condition condition)

返回等待与此锁对象相关的给定条件Condition的线程估计数，如有5个线程，每个线程都绑定在同一个Condition对象上，并且调用了该Condition对象的await()方法，则调用getWaitQueueLength(Condition condition)方法返回的是5。关键点是那个Conditino对象绑定的线程数量。

lock.getWaitQueueLength(newCondition);

1. hasQueuedThread()

查询是否有线程正在等待获取此锁对象。

lock.hasQueuedThread();

1. hasQueuedThread(Thread thread)

查询指定的线程是否正在等待获取此锁对象。

lock.hasQueuedThread(thread);

1. hasWaiters(Condition condition)

查询是否有线程正在等待与此锁对象有关的Condition条件

lock.hasWaiters(condition);

1. isFair()

判断当前的锁对象是不是公平锁，默认的情况下，ReentrantLock类使用的是非公平锁。

lock.isFair();

1. isHeldByCurrentThread()

查询当前线程是否保持着这个锁对象，这个方法通常用在释放锁对象的时候，先判断再释放。

lock.isHeldByCurrentThread();

1. isLocked()

查询此锁对象是否由任意的线程保持着，即锁对象是否正在被使用。

lock.isLocked();

1. lockInterruptibly()

如果当前线程未被终端，则可以获取该锁对象，如果已经被中断了，则出现异常。

lock.lockInterruptibly();

1. Boolean tryLock()

当某线程调用该方法时，只有在调用时锁对象未被另一个线程持有的情况下，才能够获取该锁对象。

lock.tryLock();

1. Boolean tryLock(long timeout, TimeUnit unit)

如果锁对象在给定的等待时间内没有被另一个线程获取，且当前线程未被中断，则获取该锁对象。

lock.tryLock(3, TimeUnit.SECONDS);

1. Condition对象的方法
2. awaitUninterruptibly()

与await()方法的区别是，当前线程调用该方法后，即使线程被中断了，也不会出现异常，而await()方法被中断时，立即会出现异常。

condition.awaitUninterruptibly();

1. awaitUntil(Time)

进入等待状态Time时间，之后会自动唤醒。也可以由其他的线程来提前唤醒。

condition.awaitUntil(Time);

1. ReentrantReadWriteLock类
2. 概述

虽然ReentrantLock类能够实现完全互斥的功能，但是同一时刻只有一个线程在执行ReentrantLock.lock()后面的任务，十分低效。因此，引出了一个新的读写锁ReentrantReadWriteLock类。

1. 读锁与写锁

读写锁有两个锁，一个是与读操作相关的锁，称为共享锁；一个是与写操作相关的锁，称为排它锁。很显然，多个读锁之间不是互斥的，读锁与写锁是互斥的，多个写锁是互斥的。

1. 读锁

private ReentrantReadWriteLock lock = new ReentrantReadWriteLock();

lock.readLock().lock();

lock.readLock().unlock();

1. 写锁

private ReentrantReadWriteLock lock = new ReentrantReadWriteLock();

lock.writeLock().lock();

lock.writeLock().unlock();