链滴

高并发程序设计（4）——JDK 并发包（重入锁、公平锁、读写锁）

作者：wanglei0622

原文链接：https://ld246.com/article/1488796343449

来源网站：链滴

<h2>重入锁（ReentrantLock）</h2>

重入锁完全可以替代synchronized关键字，而且重入锁有着明显的显示操作过程，开发人员手动定何时加锁，何时释放锁，灵活性好于同步锁，
在同一个线程中，重入锁可以重复进入，当然获取多少次，也要释放多少次
lockInterruptibly():可响应中断，同步锁中 只有调用了对象的wait或者线程sleep方法，中断时有反应，重入锁可以直接响应。
tryLock():我们无法判断一个线程为什么迟迟拿不到锁，也许是死锁了，也可能是产生饿，tryLock()可以给定一个等待时间，到时间没有得到锁，就会返回false，也可以不带时间参数，则即申请锁，并返回申请结果，这种方式不会引起线程等待，因此不会产生死锁。

<h3>重入锁好搭档：Condition条件</h3>

和Object.wait(),notify()大致相同，只是Condition是和重入锁配合使用的
condition调用await()和signal（）时，也得先调用ReentrantLock.lock()获得锁再执行。
和wait()一样，condition调用await()后，这个线程也会释放锁。

<h2>公平锁</h2>

大多数时候锁的申请都是不公平的，公平锁会按时间顺序，保证先到先得，这样不会产生饥饿，终都会得到资源，synchronized获得的都是非公平的锁，ReentrantLock(boolean fair) 重入锁的构方法可以指定是否使用公平锁
因为实现公平锁需要维护一个有序队列，所以公平锁性能相对低下，没有特殊要求，不要使用。

<h2>读写锁（ReadWriteLock）</h2>

读写锁在只读的时候完全并行不会阻塞，读写或者写写的时候还是要先获得锁。当系读远远大于写时，使用读写锁可以发挥最大效率。

<h2>信号量（Semaphore）</h2>

信号量是对锁的扩展，无论同步锁还是重入锁，一次都只允许一个线程访问一个资源，信号量可允许多个线程，同时访问一个资源。

<h2>倒计时器（CountDownLatch）</h2>

让主线程等到计时器管理的线程都结束再继续执行，类似join的加强实现。

<h2>循环栅栏（CyclicBarrier）</h2>

可循环使用的线程计数器，当计数条件达到时，执行指定动作。

<h2>线程阻塞工具（LockSupport）</h2>

在线程任意位置让线程阻塞，比Thead.suspend相比，弥补了犹豫resume在阻塞之前执行时，成死锁。和Object()相比，不用先获得锁，也不会抛出中断异常。
LockSupport.park()不用保证 unpark()一定在它之后执行，因为它为线程准备了一个许可，就发生在unpark()之后也一样可以使park()立即返回。
线程状态会是waiting,不会和suspend那样还是runnable状态。
