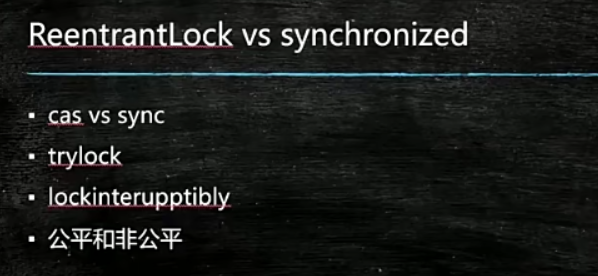
CountDownLantch

LockSupport

调用await都需要catch中断异常

底层都是AQS

1. ReentrantLock可重入锁
2. 可重入锁
3. lock.lock()、lock.unlock()：需要手动加锁、解锁
4. lock.trylock()：可以尝试加锁tryLock，根据返回值判断是否加锁成功
5. lock.lockInterruptibly()：加锁期间可以被中断，根据异常判断是否加锁成功
6. Condition，线程队列，可以精确指定唤醒signalAll哪些线程、阻塞await哪些线程 （一个condition，一个等待队列）
7. 可以设置公平锁（先进入等待队列，等待锁）、非公平锁（直接抢锁）



synchronized是非公平锁

1. CountDownLatch

（1）作用：可用于等待其他线程执行完成，类似join()

（2）步骤

* new CountDownLatch获得CountDownLatch
* countDown减减
* await()阻塞等待

1. CyclicBarrier

（1）作用：线程数达到一定数量，线程才能进行运行，否则需要等待。

（CyclicBarrier相当于公交车，满人才可以发车，即线程数达到一定数量才可以进行运行）

1. 步骤

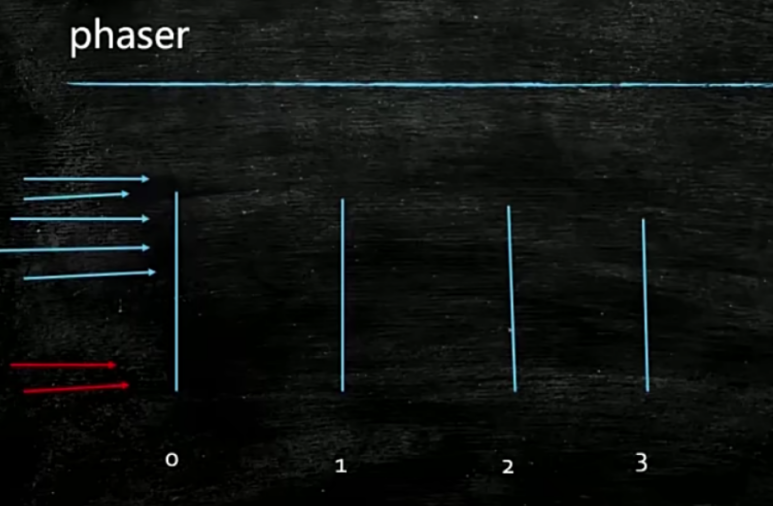
* new CyclicBarrier()，定义满足条件的线程数
* 每个线程调用cyclicBarrier.await()，等待条件满足

1. Phaser（分阶段的Barrier，每阶段线程数达到一定数量才可以进行运行）

（1）作用：分多个阶段执行，每个阶段线程数达到一定数量，才可以执行

（2）步骤

* 继承Phaser，重写onAdvance()，定义每阶段完成之后的动作，并且返回false表示结束阶段，返回true表示进行运行；并初始化线程数量。
* 定义线程，并调用phaser.arriveAndAwaitAdvance()等待条件满足，或者phaser.arriveAndDeregister结束



1. ReadWriteLock读写锁

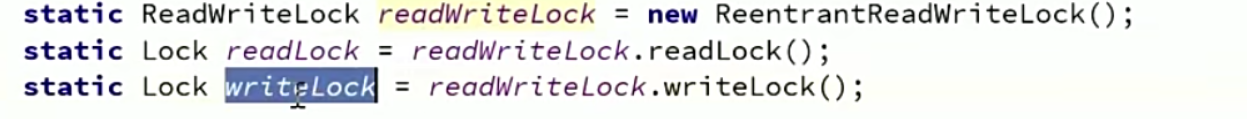
（1）作用

共享锁：读锁，读操作之间不互斥，读的时候不能写，所以得加读锁，防止脏读

排他锁：写锁，只要涉及写操作就必须同步、互斥

（2）步骤

* 声明读写锁



* 加锁解锁

lock、unlock

1. Semaphore信号量
2. 作用

可以用于限流，最多允许几个线程同时运行

1. 步骤

* 声明信号量（有公平与非公平）

20210715222917

* 尝试获取资源

s.aquire()

* 释放资源

s.release()



1. Exchanger交换器
2. 作用

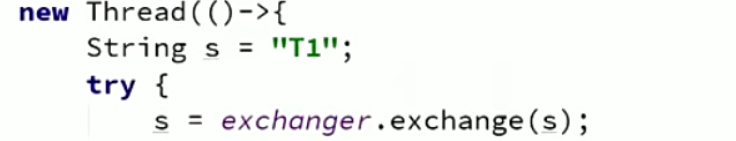
两个线程之间交换数据

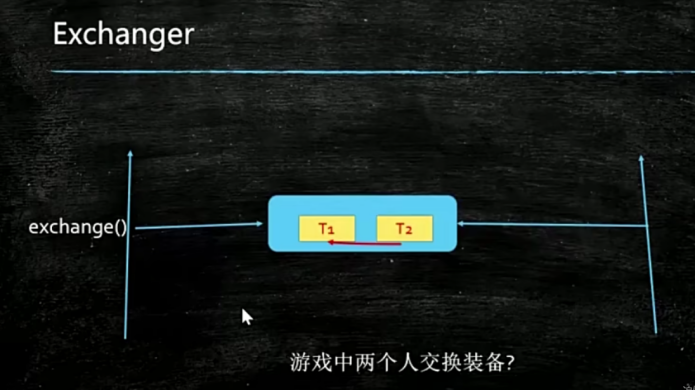
1. 步骤

* 声明Exchanger

20210715223425

* 线程阻塞交换





1. LockSupport
2. 作用

随时随地阻塞线程：不能加锁，LockSupport.park()

随时随地唤醒指定线程：不要加锁，LockSupport.unpark(thread)

unpark可以先于park调用，并且在park地方就不会阻塞。

（区别：notify不能在wait前调用）