5.JDK并发包1

1. 各种同步控制工具的使用

1.1. ReentrantLock

1.1.1. 可重入

单线程可以重复进入，但要重复退出

1.1.2. 可中断

lockInterruptibly()

1.1.3. 可限时

超时不能获得锁，就返回false，不会永久等待构成死锁

1.1.4. 公平锁

先来先得
public ReentrantLock(boolean fair)
public static ReentrantLock fairLock = new ReentrantLock(true);

1.2. Condition

1.2.1. 概述

类似于 Object.wait()和Object.notify()
与ReentrantLock结合使用

1.2.2. 主要接口

void await() throws InterruptedException;
void awaitUninterruptibly();
long awaitNanos(long nanosTimeout) throws InterruptedException;
boolean await(long time, TimeUnit unit) throws InterruptedException;
boolean awaitUntil(Date deadline) throws InterruptedException;
void signal();
void signalAll();

1.2.3. API详解

await()方法会使当前线程等待，同时释放当前锁，当其他线程中使用signal()时或者signalAll()方法时，线程会重新获得锁并继续执行。或者当线程被中断时，也能跳出等待。这和Object.wait()方法很相似。

awaitUninterruptibly()方法与await()方法基本相同，但是它并不会再等待过程中响应中断。

singal()方法用于唤醒一个在等待中的线程。相对的singalAll()方法会唤醒所有在等待中的线程。这和Object.notify()方法很类似。

1.3. Semaphore

1.3.1. 概述

共享锁

运行多个线程同时临界区

1.3.2. 主要接口

public void acquire()
public void acquireUninterruptibly()
public boolean tryAcquire()
public boolean tryAcquire(long timeout, TimeUnit unit)
public void release()

1.4. ReadWriteLock

1.4.1. 概述

ReadWriteLock是JDK5中提供的读写分离锁

1.4.2. 访问情况

读-读不互斥：读读之间不阻塞。

读-写互斥：读阻塞写，写也会阻塞读。

写-写互斥：写写阻塞。

	读	写
读	非阻塞	阻塞
写	阻塞	阻塞

1.4.3. 主要接口

private static ReentrantReadWriteLock readWriteLock=new ReentrantReadWriteLock();
private static Lock readLock = readWriteLock.readLock();
private static Lock writeLock = readWriteLock.writeLock();

1.5. CountDownLatch

1.5.1. 概述

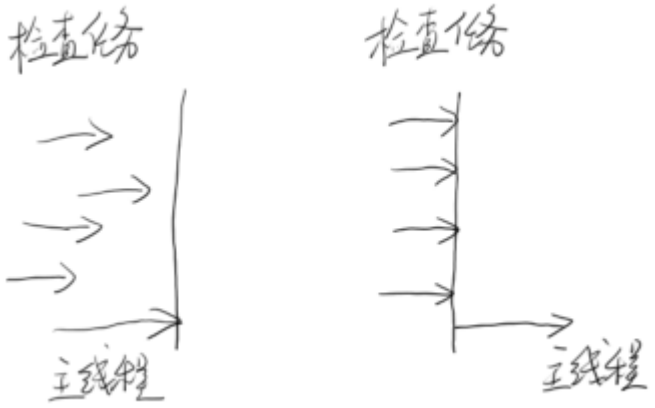
倒数计时器

一种典型的场景就是火箭发射。在火箭发射前，为了保证万无一失，往往还要进行各项设备、仪器的检查。只有等所有检查完毕后，引擎才能点火。这种场景就非常适合使用CountDownLatch。它可以使得点火线程，等待所有检查线程全部完工后，再执行

1.5.2. 主要接口

static final CountDownLatch end = new CountDownLatch(10);
end.countDown();
end.await();

1.5.3. 示意图



1.6. CyclicBarrier

1.6.1. 概述

循环栅栏

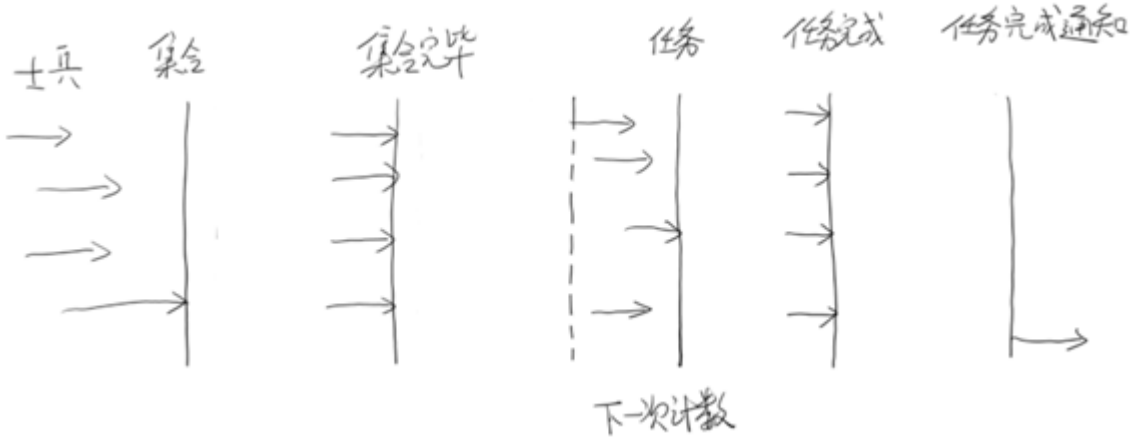
Cyclic意为循环，也就是说这个计数器可以反复使用。比如，假设我们将计数器设置为10。那么凑齐第一批10个线程后，计数器就会归零，然后接着凑齐下一批10个线程

1.6.2. 主要接口

public CyclicBarrier(int parties, Runnable barrierAction)
barrierAction就是当计数器一次计数完成后，系统会执行的动作

await()

1.6.3. 示意图



1.7. LockSupport

1.7.1. 概述

提供线程阻塞原语

1.7.2. 主要接口

LockSupport.park();
LockSupport.unpark(t1);

1.7.3. 与suspend()比较

不容易引起线程冻结

1.7.4. 中断响应

能够响应中断，但不抛出异常。

中断响应的结果是，park()函数的返回，可以从Thread.interrupted()得到中断标志

1.8. ReentrantLock 的实现

1.8.1. CAS状态

1.8.2. 等待队列

1.8.3. park()

2. 并发容器及典型源码分析

2.1. 集合包装

2.1.1. HashMap

Collections.synchronizedMap

```
public static Map m=Collections.synchronizedMap(new HashMap());
```

2.1.2. List

synchronizedList

2.1.3. Set

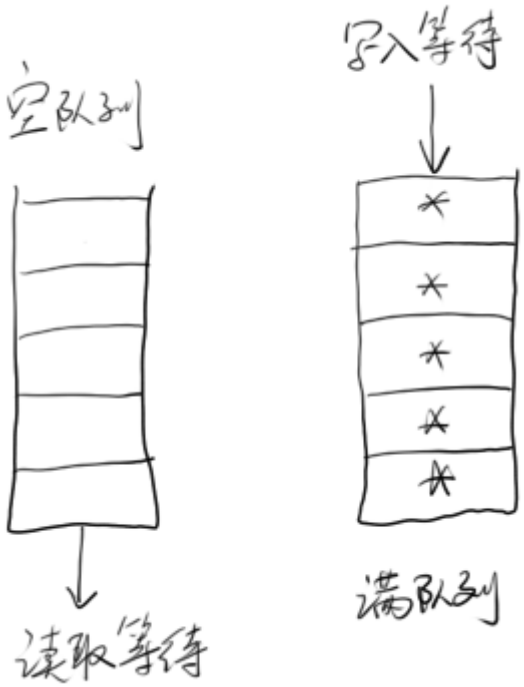
synchronizedSet

2.2. ConcurrentHashMap

高性能HashMap

2.3. BlockingQueue

阻塞队列



2.4. ConcurrentLinkedQueue