1. CountDownLatch
   1. Java.util.Concurrent类中的多线程工具类，其功能时计数器；

范例:使用多线程计数器；CountDownLatch

|  |
| --- |
| package cn.gsdx.JUC;  import java.util.concurrent.CountDownLatch;  public class ThreadCountDownLatch {  public static void main(String[] args) throws InterruptedException {  //计数器设为6  CountDownLatch count = new CountDownLatch(6) ;  for (int i = 0; i < 6; i++) {  new Thread(()->{  System.out.println(Thread.currentThread().getName()+"\t：开始执行。");  count.countDown();  },String.valueOf(i)).start();  }  count.await();  System.out.println("主线程执行完成");  }  } |

我们首先CountDownLatch count = new CountDownLatch(6) ;进行计数器的赋值，然后在每个线程中执行，count.countDown();计数器减一，如果计数器没有减完那么count.await（）就会使主线程线程等待，直到count减到0才能执行System.out.println("主线程执行完成");

总结:

CountDownLatch主要又两个方法，当一个或多个线程调用await方法时这些线程会阻塞。当其它线程调用CountDown（）方法时会将计数器减1（调用countDown方法的线程不会阻塞）当计数器的值变为0时，因await方法阻塞的线程会被唤醒。

1. CyclicBarrier
   1. CyclicBarrier的功能是当执行了指定(这里是7)个带有cyclicBarrier.await();的线程就会执行cyclicBarrier参数里面的Runnable线程；（也称为点火式）
   2. 范例:使用CyclicBarrier

|  |
| --- |
| package cn.gsdx.JUC;  import java.util.concurrent.BrokenBarrierException;  import java.util.concurrent.CyclicBarrier;  public class CyclicBarrierDemo {  public static void main(String[] args) {  CyclicBarrier cyclicBarrier = new CyclicBarrier(7,()->{  System.out.println("神龙出现");  }) ;  for (int i = 0; i < 7; i++) {  final int temp = i ;  new Thread(() ->{  System.out.println("第"+ temp + "颗龙珠");  try {  cyclicBarrier.await();  } catch (InterruptedException e) {  e.printStackTrace();  } catch (BrokenBarrierException e) {  e.printStackTrace();  }  },String.valueOf(i)).start(); ;  }  }  } |

1. Semaphore
   1. Semaphore俗称信号灯
      1. 原理: 在信号量上我们定义两种操作:

\* acquire (获取）当一个线程调用acquire操作时，它要么通过成功获取信号量（信号量减1），要么一直等下去，直到有线程释放信号量，或超时。

\* release(释放）实际上会将信号量的值加1，然后唤醒等待的线程。

\*信号量主要用于两个目的，一个是用于多个共享资源的互斥使用，另一个用于并发线程数的控制。

范例:

|  |
| --- |
| package cn.gsdx.JUC;  import java.util.concurrent.Semaphore;  import java.util.concurrent.TimeUnit;  public class SemaphoreDemo {  public static void main(String[] args) {  //模拟资源类 有三个资源  Semaphore semaphore = new Semaphore(3) ;  for (int i = 0 ;i < 6 ; i ++){  final int temp = i ;  new Thread(()->{  try {  semaphore.acquire();  System.out.println("获取到了"+ temp +"号资源");  TimeUnit.SECONDS.sleep(4);  } catch (InterruptedException e) {  e.printStackTrace();  }finally{  semaphore.release();  }  },String.valueOf(i)).start();  }  }  } |

当Semphore的参数为1时，等价于Synchornized，谁先抢到是谁的。