Semaphore（信号量）

信号量为多线程提供了更为强大的控制方法，无论是锁还是synchronize，一次都只允许一个线程访问一个资源，而信号量可以指定多少个线

通过 acquire() 获取一个许可，如果没有就等待，而 release() 释放一个许可

```
//打印类
import java.util.concurrent.Semaphore;

public class Print {

    private Semaphore semaphore = new Semaphore(5);

    public void prt(){
            try {
                    System.out.println(Thread.currentThread().getName() + " : 准备进入" );
                    semaphore.acquire();
                    Thread.sleep(3000);
                    System.out.println(Thread.currentThread().getName() + " : 进入" );
                    Thread.sleep(3000);
                    semaphore.release();
                    System.out.println(Thread.currentThread().getName() + " : 离开" );
            } catch (Exception e) {
                    e.printStackTrace();
            }
    }
}
//线程类
public class Th1 implements Runnable{

    private Print p;

    public Th1(Print p) {
            this.p = p;
    }

    @Override
    public void run() {
            p.prt();
    }
}

//Main方法类
public class M {

    public static void main(String[] args) {

            Print p = new Print();
            for(int i = 0 ; i < 14 ; i++){
                    new Thread(new Th1(p)).start();
            }
    }

```

```
}
```

倒数计时器（CountDownLatch）

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

主线程模拟裁判，八个子线程模拟运动员，裁判和八个运动员都就位以后，运动员才能开始跑步

```
import java.util.Date;
import java.util.HashMap;
import java.util.Map;
import java.util.Map.Entry;
import java.util.Random;
import java.util.concurrent.CountDownLatch;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

public class CountDownTest {
    public static void main(String[] args) {
            ExecutorService pool = Executors.newCachedThreadPool();
            final CountDownLatch cdl1 = new CountDownLatch(1);//裁判吹哨开始倒计时,归零运动员开始跑步
            final CountDownLatch cdl2 = new CountDownLatch(8);//运动员跑完了，裁判公布结果。//cdl2.count
            final Map<String , Long> map = new HashMap<String , Long>();
            for(int i =0 ; i < 8 ;i ++){//八个运动员
                    pool.execute(new Runnable() {
                            @Override
                            public void run() {
                                    System.out.println("运动员 " + Thread.currentThread().getName()
                                    try {
                                            cdl1.await();
                                    } catch (InterruptedException e) {
                                            e.printStackTrace();
                                    }
                                    System.out.println("运动员:" + Thread.currentThread().getName()
                                    try {
                                            Thread.sleep(new Random().nextInt(10) * 1000);
                                    } catch (InterruptedException e) {
                                            e.printStackTrace();
                                    }
                                    map.put(Thread.currentThread().getName(), new Date().getTime())
                                    System.out.println("运动员: " + Thread.currentThread().getName()
                                    cdl2.countDown();
                            }
                    });
            }
            long l = 0;
            try {
                    System.out.println("裁判就位");
                    Thread.sleep(3000);
                    cdl1.countDown();
                    System.out.println("裁判发了起跑消息,运动员起跑 , 裁判等待跑完");
                    l = new Date().getTime();
```

```
                    cdl2.await();
                    System.out.println("跑完了，裁判发布结果");
                    System.out.println("结果如下");
            } catch (InterruptedException e) {
                    e.printStackTrace();
            }
            pool.shutdown();

            for(Entry<String, Long> entry : map.entrySet()){
                    System.out.println(entry.getKey() + " 所用所用时间是 : "
                                    + (entry.getValue() - l));
            }
    }
}
```

CyclicBarrier(循环栅栏、回环屏障)

循环栅栏与计算器很像，但是可以反复使用，下面模拟 十个人一起去景点的场景

```
import java.util.Random;
import java.util.concurrent.CyclicBarrier;

public class Th1 implements Runnable{

    private CyclicBarrier cb;

    public Th1(CyclicBarrier cb2) {
            this.cb = cb2;
    }

    @Override
    public void run() {
            try {
                    Thread.sleep(new Random().nextInt(10) * 1000);
                    System.out.println(Thread.currentThread().getName() + "到了黄鹤楼 已经有:" + (cb.g
                    if(cb.getNumberWaiting() == 10){
                            System.out.println("全部到齐，下一站龟山");
                    }
                    cb.await();

                    Thread.sleep(new Random().nextInt(10) * 1000);
                    System.out.println(Thread.currentThread().getName() + "到了龟山 已经有:" + (cb.get
                    if(cb.getNumberWaiting() == 10){
                            System.out.println("全部到齐，下一站东湖");
                    }
                    cb.await();

                    Thread.sleep(new Random().nextInt(10) * 1000);
                    System.out.println(Thread.currentThread().getName() + "到了东湖 已经有:" + (cb.get
                    if(cb.getNumberWaiting() == 10){
                            System.out.println("全部到齐，结束");
                    }
                    cb.await();
```

```
            } catch (Exception e1) {
                    e1.printStackTrace();
            }
    }
}

import java.util.concurrent.CyclicBarrier;

public class M {

    public static void main(String[] args) {

            CyclicBarrier cb = new CyclicBarrier(10);

            for(int i = 0 ; i < 10 ; i++){
                    new Thread(new Th1(cb)).start();
            }

    }

}
```