一、线程池。多线程可以提高系统的吞吐量（cpu利用的效率）

方式一：ThreacdPooltest01{

new ThreadPlExecutor(corePoolSize ：线程池初始大小

maximumPoolSize ：线程池最大线程数

keepAliveTime ：空线程存活的最长时间

unit ：时间单位

workQueue ：等待队列

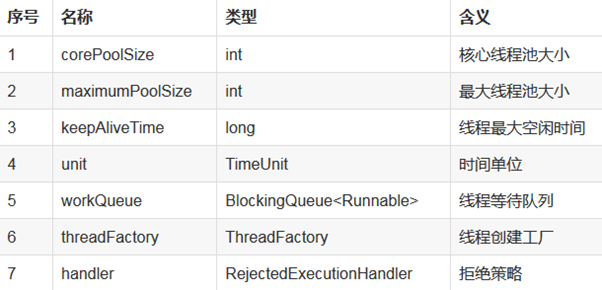
handler ：拒绝策略（当任务数大于 >

maxNumPoolSize+等待队列长度）

默认直接拒绝报异常 AbortHander

}

基本参数：



二、线程池中的线程量如果到达最大线程时，就会拒绝线程假如，并且报异常状态。

public class ThreadPoolTest01 {

    public static void main(String[] args) {

        ThreadPoolExecutor threadPool = new ThreadPoolExecutor(2,

                5,

                300,

                TimeUnit.MILLISECONDS,

               new  ArrayBlockingQueue<Runnable>(2)

                );

        for (int i = 0; i < 5; i++) {

            //String.valueOf(i) 将整数转为字符串

            MyTask myTask = new MyTask(String.valueOf(i));

            //提交任务给线程池

            threadPool.execute(myTask);

        }

    }

}

上述的代码块执行提交五个任务到线程池的操作，实际过程是这样的：

提交第1个任务到线程池：poolSize=0<corePoolSize 所以会为改任务创建一个线程对象并

                                     启动线程。

提交第2个任务到线程池：poolSize=1<corePoolSize 所以会为改任务创建一个线程对象并

                                     启动线程。

提交第3个任务到线程池：poolSize=2!<corePoolSize 此时会将任务放到等待队列（1）。

提交第4个任务到线程池：poolSize=2!<corePoolSize 没有空闲线程，此时会将任务放到等待队列（2）。

提交第5个任务到线程池：poolSize=2!<corePoolSize 没有空闲线程，判断此时

maximumPoolSize>poolSize>=corePoolSize。没有创建一个线程。

**ExecutorService（ThreadPoolExecutor的顶层接口）使用线程池中的线程执行每个提交的任务，通常我们使用Executors的工厂方法来创建ExecutorService。**

**线程池解决了两个不同的问题：**

**1. 提升性能：它们通常在执行大量异步任务时，由于减少了每个任务的调用开销，并且它们提供了一种限制和管理资源（包括线程）的方法，使得性能提升明显；**

**2. 统计信息：每个ThreadPoolExecutor保持一些基本的统计信息，例如完成的任务数量**

**  Executors.newCachedThreadPool（无界线程池，自动线程回收）**

**  Executors.newFixedThreadPool（固定大小的线程池）；**

**  Executors.newSingleThreadExecutor（单一后台线程）；**