此文章已于 19:00:54 2017/12/20 重新发布到 HealerJean梦想博客

JAVA线程池的使用

类别 java

# 作用：

减少创建和销毁线程的次数，每个工作线程可以多次使用，可根据系统情况调整执行的线程数量，防止消耗过多内存

# 1、newCachedThreadPool ：

## 1、解释：

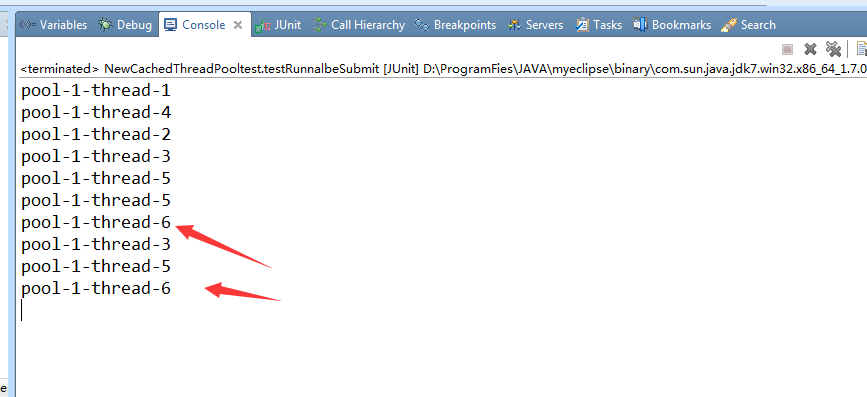
创建一个可缓存线程池，如果线程池长度超过处理需要，可灵活回收空闲线程，若无可回收，则新建线程。就是一次添加多个任务，长度会变大，会一直新建线程，当没有任务时，空闲线程会被回收，线程池为无限大，当执行第二个任务时第一个任务已经完成。会复用第一个任务的线程，而不是每次新建线程。

## 2、测试Runnable和callable使用线程池

### 1、测试Runnable 使用线程池

|  |
| --- |
| **public** **class** NewCachedThreadPooltest {  @Test  **public** **void** testRunnalbeSubmit(){  ExecutorService exec=Executors.*newCachedThreadPool*();    **for**(**int** i=0;i<10;i++) {  exec.submit(**new** MyRunnable());  }  //关闭线程池  exec.shutdown();  }    **class** MyRunnable **implements** Runnable{  @Override  **public** **void** run() {  System.*out*.println(Thread.*currentThread*().getName());  }  }  } |

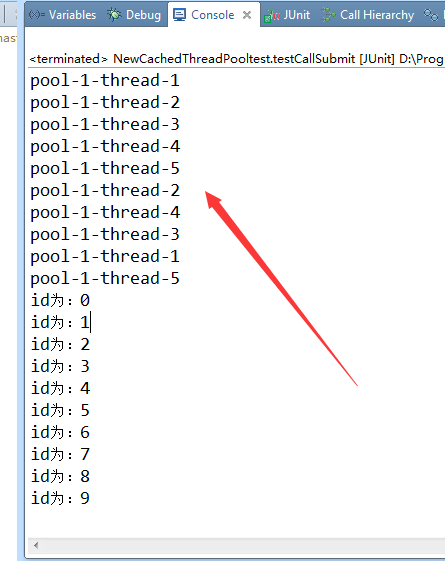
#### 1、控制台执行结果，会发现虽然上面开启了10个线程，但是线程其实是有重复使用的，这样就大大节省了资源



### 2、测试Callable，有返回值

|  |
| --- |
| **public** **class** NewCachedThreadPooltest {  @Test  **public** **void** testCallSubmit(){  ExecutorService exec=Executors.*newCachedThreadPool*();    List<Future<String>> results=**new** ArrayList<Future<String>>();    **for**(**int** i=0;i<10;i++) {    results.add(exec.submit(**new** CallableTaskWithResult(i)));  }  //关闭线程池  exec.shutdown();  **for**(Future<String> fs :results) {  **try** {  System.*out*.println(fs.get());  } **catch** (InterruptedException e) {  e.printStackTrace();  } **catch** (ExecutionException e) {  e.printStackTrace();  }  }    }    **class** CallableTaskWithResult **implements** Callable<String> {  **private** **int** id;  **public** CallableTaskWithResult(**int** id) {  **this**.id=id;  }    @Override  **public** String call() **throws** Exception {  System.*out*.println(Thread.*currentThread*().getName());  **return** "id为："+id;  }  }  } |

#### 2、控制台执行结果，发现同上面Runnable一样，还是节省了资源



# 2、newFixedThreadPool

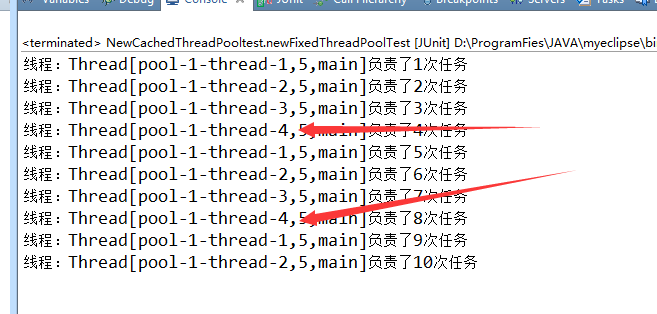
## 解释：

创建一个固定数目的、可重用的线程池。可控制现场最大并发数，超出的线程会在线程队列中等待

## 1、测试代码

|  |
| --- |
| **public** **class** FixedThreadPoolTest {  @Test  **public** **void** newFixedThreadPoolTest() **throws** InterruptedException {  // 固定线程的数量  ExecutorService m = Executors.*newFixedThreadPool*(4);    **for** (**int** i = 1; i <= 10; i++) {  **final** **int** count = i;  m.submit(**new** Runnable() {  @Override  **public** **void** run() {  System.*out*.println("线程：" + Thread.*currentThread*() + "负责了"  + count + "任务");  }  });  Thread.*sleep*(1000);  }  }  } |

## 2、控制台，会发现只有4个线程被创建了



# 3、newScheduledThreadPool:

创建一个定长线程池，支持定时及周期性任务执行。

# 4、newSingleThreadExecutor：

创建一个单线程化的线程池，它只会用唯一的工作线程来执行任务，保证所有任务按照指定顺序(FIFO, LIFO, 优先级)执行。

（5）

# 5、newSingleThreadScheduledExcutor：

创建一个单例线程池，定期或延时执行任务。

## 6、使用线程池的风险

虽然线程池是构建多线程应用程序的强大机制，但使用它并不是没有风险的。用线程池构建的应用程序容易遭受任何其它多线程应用程序容易遭受的所有并发风险，诸如同步错误和死锁，它还容易遭受特定于线程池的少数其它风险，诸如与池有关的死锁、资源不足和线程泄漏。