链滴

实例带你搞懂 Java 多线程 && 线程池之（贰）：简单的线程池应用

作者：adlered

原文链接：https://ld246.com/article/1552543536651

来源网站：链滴

前言

这篇文章的代码**非常简单**，代码后的**实验要求**一定**不要忽略**，你一定能理解线程池的算法。

如没看过第一章，请先点我跳转

原文链接：实例带你搞懂 Java 多线程 && 线程池之（贰）：简单的线程池应用

套用代码

打开你的IDE，并新建一个类，将下方代码拷贝：

```
import java.lang.reflect.Executable;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

public class TestThreadPool {
    ExecutorService executorService = Executors.newFixedThreadPool(2);
    public static void main(String[] args) {
        //实例化类
        TestThreadPool testThreadPool = new TestThreadPool();
        //调用动态方法
        testThreadPool.threadPool();
    }

    public void threadPool() {
        Thread1 thread1 = new Thread1();
        Thread2 thread2 = new Thread2();
        for (int i = 0; i < 3; i++) {
            System.out.println("线程池已提交：" + i);
            executorService.execute(thread1);
            executorService.execute(thread2);
        }
        //executorService.shutdown();
        //executorService.shutdownNow();
    }
}

/**
 * 线程1
 */
class Thread1 implements Runnable {
    @Override
    public void run() {
        System.out.println("WORKING ON THREAD 1");
        try {
            Thread.sleep(500);
        } catch (Exception e) {}
    }
}

/**
 * 线程2
 */
class Thread2 implements Runnable {
    @Override
    public void run() {
        System.out.println("WORKING ON THREAD 2");
        try {
            Thread.sleep(500);
        } catch (Exception e) {}
    }
```

```
}
```

此时有三个类被定义，其中Thread1和Thread2为两个线程的定义。

运行项目

现在运行你的项目，运行结果如下：

```
线程池已提交：0
WORKING ON THREAD 1
线程池已提交：1
WORKING ON THREAD 2
线程池已提交：2
WORKING ON THREAD 1
WORKING ON THREAD 2
WORKING ON THREAD 1
WORKING ON THREAD 2
```

阅读代码，在threadPool中我们将线程以execute()提交到了线程池，而通过for循环可以知道共循环交了三次线程，其中每次分别提交了两个线程。

关闭线程池

你可能发现了，线程池在执行完毕后程序并没有结束，需要手动结束。

现在，将注释掉的executorService.shutdown()取消注释，再次运行，你会发现程序在执行线程池后程池自动进行了自销毁且程序自动关闭。

```
线程池已提交：0
WORKING ON THREAD 1
线程池已提交：1
WORKING ON THREAD 2
线程池已提交：2
WORKING ON THREAD 1
WORKING ON THREAD 2
WORKING ON THREAD 1
WORKING ON THREAD 2
```

现在，重新注释掉executorService.shutdown()，并**取消注释**executorService.shutdownNow()，行程序：

```
线程池已提交：0
线程池已提交：1
WORKING ON THREAD 1
线程池已提交：2
WORKING ON THREAD 2
```

可以看到的是，线程池在没有执行完毕的情况下就终止了程序的运行。这是因为：

```
    executorService.shutdown();
    executorService.shutdownNow();
```

shutdown()方法就像你点击了电脑的“关机”按钮一样，它不会立即关闭，而是等待未执行完毕的务全部执行完毕之后才会关闭线程池。

shutdownNow()方法是无视线程池的状态，强制关闭线程池。

调整线程池大小

在代码的最上方，你可以看到：

```
ExecutorService executorService = Executors.newFixedThreadPool(2);
```

其中，我们将newFixedThreadPool(2)拆分来看：

线程池集合

newFixedThreadPool()是Executors（线程池集合）中的一种线程池，在Executors类中共有**4种**线池可供选择：

newCachedThreadPool 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，若无可回收，则新建线程。
newFixedThreadPool 创建一个定长线程池，可控制线程最大并发数，超出的线程会在队列中等待。
newScheduledThreadPool 创建一个定长线程池，支持定时及周期性任务执行。
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，保所有任务按照指定顺序(FIFO, LIFO, 优先级)执行。

而我们使用的newFixedThreadPool线程池便是一种定长线程池，它在运行时有如下特性：

每次只同时运行指定数量的线程，多出的线程会排队等候，直至前面的线程执行完毕再执行。

定义线程池

再来看**2**，正如你所料的一样，它表示该线程池每次同时只能运行2个线程，其它超出的线程必须排队候执行完毕。

后语

现在，相信你已经对线程池有了大概的了解了。下一章我们将细说线程池的定义方式。