链滴

ExecutorService 中 submit 和 execute 的区别

作者：jsy

原文链接：https://ld246.com/article/1493450244046

来源网站：链滴

一、创建任务

任务就是一个实现了Runnable接口的类。

创建的时候实run方法即可。

二、执行任务

通过java.util.concurrent.ExecutorService接口对象来执行任务，该接口对象通过工具类java.util.concurrent.Executors的静态方法来创建。

Executors此包中所定义的 Executor、ExecutorService、ScheduledExecutorService、ThreadFactory 和 Callable 类的工厂和实用方法。

ExecutorService提供了管理终止的方法，以及可为跟踪一个或多个异步任务执行状况而生成 Future 方法。 可以关闭 ExecutorService，这将导致其停止接受新任务。关闭后，执行程序将最后终止，这没有任务在执行，也没有任务在等待执行，并且无法提交新任务。

```
executorService.execute(new TestRunnable());
```

1、创建ExecutorService

通过工具类java.util.concurrent.Executors的静态方法来创建。

Executors此包中所定义的 Executor、ExecutorService、ScheduledExecutorService、ThreadFactory 和 Callable 类的工厂和实用方法。

比如，创建一个ExecutorService的实例，ExecutorService实际上是一个线程池的管理工具：

```
ExecutorService executorService = Executors.newCachedThreadPool();

ExecutorService executorService = Executors.newFixedThreadPool(3);

ExecutorService executorService = Executors.newSingleThreadExecutor();
```

2、将任务添加到线程去执行

当将一个任务添加到线程池中的时候，线程池会为每个任务创建一个线程，该线程会在之后的某个时自动执行。

三、关闭执行服务对象

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

四、获取任务的执行的返回值

在Java5之后，任务分两类：一类是实现了Runnable接口的类，一类是实现了Callable接口的类。两都可以被ExecutorService执行，但是Runnable任务没有返回值，而Callable任务有返回值。并且Callble的call()方法只能通过ExecutorService的(task) 方法来执行，并且返回一个 ，是表示任务等待完的 Future。

public interface **Callable**

返回结果并且可能抛出异常的任务。实现者定义了一个不带任何参数的叫做 call 的方法。

Callable 接口类似于，两者都是为那些其实例可能被另一个线程执行的类设计的。但是 Runnable 不返回结果，并且无法抛出经过检查的异常。

类包含一些从其他普通形式转换成 Callable 类的实用方法。

Callable中的call()方法类似Runnable的run()方法，就是前者有返回值，后者没有。

当将一个Callable的对象传递给ExecutorService的submit方法，则该call方法自动在一个线程上执行并且会返回执行结果Future对象。

同样，将Runnable的对象传递给ExecutorService的submit方法，则该run方法自动在一个线程上执，并且会返回执行结果Future对象，但是在该Future对象上调用get方法，将返回null。

五、区别

1、接收的参数不一样

2、submit有返回值，而execute没有

Method submit extends base method Executor.execute by creating and returning a Future tha can be used to cancel execution and/or wait for completion.

用到返回值的例子，比如说我有很多个做validation的task，我希望所有的task执行完，然后每个tas告诉我它的执行结果，是成功还是失败，如果是失败，原因是什么。然后我就可以把所有失败的原因合起来发给调用者。

个人觉得cancel execution这个用处不大，很少有需要去取消执行的。

而最大的用处应该是第二点。

3、submit方便Exception处理

意思就是如果你在你的task里会抛出checked或者unchecked exception，而你又希望外面的调用者够感知这些exception并做出及时的处理，那么就需要用到submit，通过捕获Future.get抛出的异常。

```
import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.*;

/**
* Callable接口测试
*
* @author leizhimin 2008-11-26 9:20:13
*/
publicclass CallableDemo {
    publicstaticvoid main(String[] args) {
        ExecutorService executorService = Executors.newCachedThreadPool();
        List<Future<String>> resultList = new ArrayList<Future<String>>();

        //创建10个任务并执行
        for (int i = 0; i < 10; i++) {
            //使用ExecutorService执行Callable类型的任务，并将结果保存在future变量中
            Future<String> future = executorService.submit(new TaskWithResult(i));
```

```
                //将任务执行结果存储到List中
                resultList.add(future);
            }

            //遍历任务的结果
            for (Future<String> fs : resultList) {
                try {
                    System.out.println(fs.get());     //打印各个线程（任务）执行的结果
                } catch (InterruptedException e) {
                    e.printStackTrace();
                } catch (ExecutionException e) {
                    e.printStackTrace();
                } finally {
                    //启动一次顺序关闭，执行以前提交的任务，但不接受新任务。如果已经关闭，
调用没有其他作用。
                    executorService.shutdown();
                }
            }
        }
}


class TaskWithResult implements Callable<String> {
        privateint id;

        public TaskWithResult(int id) {
            this.id = id;
        }

        /**
         * 任务的具体过程，一旦任务传给ExecutorService的submit方法，则该方法自动在一个线程上
行。
         *
         * @return
         * @throws Exception
         */
        public String call() throws Exception {
            System.out.println("call()方法被自动调用,干活！！！             " + Thread.currentThread(
.getName());
            //一个模拟耗时的操作
            for (int i = 999999; i > 0; i--) ;
            return"call()方法被自动调用，任务的结果是：" + id + "    " + Thread.currentThread().g
tName();
        }
}
```

运行结果：

```
call()方法被自动调用,干活！！！             pool-1-thread-1
call()方法被自动调用,干活！！！             pool-1-thread-3
call()方法被自动调用,干活！！！             pool-1-thread-4
call()方法被自动调用,干活！！！             pool-1-thread-6
call()方法被自动调用,干活！！！             pool-1-thread-2
call()方法被自动调用,干活！！！             pool-1-thread-5
call()方法被自动调用，任务的结果是：0    pool-1-thread-1
```

```
call()方法被自动调用，任务的结果是：1    pool-1-thread-2
call()方法被自动调用,干活！！！             pool-1-thread-2
call()方法被自动调用,干活！！！             pool-1-thread-6
call()方法被自动调用,干活！！！             pool-1-thread-4
call()方法被自动调用，任务的结果是：2    pool-1-thread-3
call()方法被自动调用,干活！！！             pool-1-thread-3
call()方法被自动调用，任务的结果是：3    pool-1-thread-4
call()方法被自动调用，任务的结果是：4    pool-1-thread-5
call()方法被自动调用，任务的结果是：5    pool-1-thread-6
call()方法被自动调用，任务的结果是：6    pool-1-thread-2
call()方法被自动调用，任务的结果是：7    pool-1-thread-6
call()方法被自动调用，任务的结果是：8    pool-1-thread-4
call()方法被自动调用，任务的结果是：9    pool-1-thread-3
```

```
package concurrent;

import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

/**
* Created by IntelliJ IDEA.
*
* @author leizhimin 2008-11-25 14:28:59
*/
publicclass TestCachedThreadPool {
    publicstaticvoid main(String[] args) {
//            ExecutorService executorService = Executors.newCachedThreadPool();
        ExecutorService executorService = Executors.newFixedThreadPool(5);
//      ExecutorService executorService = Executors.newSingleThreadExecutor();

        for (int i = 0; i < 5; i++) {
            executorService.execute(new TestRunnable());
            System.out.println("************* a" + i + " *************");
        }
        executorService.shutdown();
    }
}

class TestRunnable implements Runnable {
    publicvoid run() {
        System.out.println(Thread.currentThread().getName() + "线程被调用了。");
        while (true) {
            try {
                Thread.sleep(5000);
                System.out.println(Thread.currentThread().getName());
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

运行结果：

********** a0 **********
********** a1 **********
pool-1-thread-2线程被调用了。
********** a2 **********
pool-1-thread-3线程被调用了。
pool-1-thread-1线程被调用了。
********** a3 **********
********** a4 **********
pool-1-thread-4线程被调用了。
pool-1-thread-5线程被调用了。
pool-1-thread-2
pool-1-thread-1
pool-1-thread-3
pool-1-thread-5
pool-1-thread-4
pool-1-thread-2
pool-1-thread-1
pool-1-thread-3
pool-1-thread-5
pool-1-thread-4