Java 多线程 实现方式

作者：yangshixin
原文链接：https://ld246.com/article/1482372026729
来源网站：链滴

Java 多线程实现方式主要有三种：继承 Thread 类、实现 Runnable 接口、使用 ExecutorServic
、Callable、Future 实现有返回结果的多线程。其中前两种方式线程执行完后都没有返回值，只有最
一种是带返回值的。
1、继承 Thread 类实现多线程
继承 Thread 类的方法尽管被我列为一种多线程实现方式，但 Thread 本质上也是实现了 Runnable
口的一个实例，它代表一个线程的实例，并且，启动线程的唯一方法就是通过 Thread 类的 start()实
方法。start()方法是一个 native 方法，它将启动一个新线程，并执行 run()方法。这种方式实现多线
很简单，通过自己的类直接 extend Thread，并复写 run()方法，就可以启动新线程并执行自己定义的
run()方法。例如：
[java] view plain copy
public class MyThread extends Thread {
public void run() {
System.out.println("MyThread.run()");
}
}
在合适的地方启动线程如下：
[java] view plain copy
MyThread myThread1 = new MyThread();
MyThread myThread2 = new MyThread();
myThread1.start();
myThread2.start();
2、实现 Runnable 接口方式实现多线程
如果自己的类已经 extends 另一个类，就无法直接 extends Thread，此时，必须实现一个 Runnable
接口，如下：
[java] view plain copy
public class MyThread extends OtherClass implements Runnable {
public void run() {
System.out.println("MyThread.run()");
}
}
为了启动 MyThread，需要首先实例化一个 Thread，并传入自己的 MyThread 实例：
[java] view plain copy
MyThread myThread = new MyThread();
Thread thread = new Thread(myThread);
thread.start();
事实上，当传入一个 Runnable target 参数给 Thread 后，Thread 的 run()方法就会调用 target.run(
，参考 JDK 源代码：
[java] view plain copy
public void run() {
if (target != null) {
target.run();
}
}
3、使用 ExecutorService、Callable、Future 实现有返回结果的多线程
ExecutorService、Callable、Future 这个对象实际上都是属于 Executor 框架中的功能类。想要详
了解 Executor 框架的可以访问 <a href="https://ld246.com/forward?goto=http%3A%2F%2Fw
w.javaeye.com%2Ftopic%2F366591" target="_blank" rel="nofollow ugc">http://www.javaeye.
om/topic/366591 ，这里面对该框架做了很详细的解释。返回结果的线程是在 JDK1.5 中引入
新特征，确实很实用，有了这种特征我就不需要再为了得到返回值而大费周折了，而且即便实现了也
能漏洞百出。
可返回值的任务必须实现 Callable 接口，类似的，无返回值的任务必须 Runnable 接口。执行 Callab
e 任务后，可以获取一个 Future 的对象，在该对象上调用 get 就可以获取到 Callable 任务返回的 Ob
ect 了，再结合线程池接口 ExecutorService 就可以实现传说中有返回结果的多线程了。下面提供了
个完整的有返回结果的多线程测试例子，在 JDK1.5 下验证过没问题可以直接使用。代码如下：

[java] view plain copy
import java.util.concurrent.*;
import java.util.Date;
import java.util.List;
import java.util.ArrayList;
/**

有返回值的线程
*/
@SuppressWarnings("unchecked")
public class Test {
public static void main(String[] args) throws ExecutionException,
InterruptedException {
System.out.println("----程序开始运行----");
Date date1 = new Date();
int taskSize = 5;
// 创建一个线程池
ExecutorService pool = Executors.newFixedThreadPool(taskSize);
// 创建多个有返回值的任务
List list = new ArrayList();
for (int i = 0; i < taskSize; i++) {
Callable c = new MyCallable(i + " ");
// 执行任务并获取 Future 对象
Future f = pool.submit(c);
// System.out.println(">>>" + f.get().toString());
list.add(f);
}
// 关闭线程池
pool.shutdown();
// 获取所有并发任务的运行结果
for (Future f : list) {
// 从 Future 对象上获取任务的返回值，并输出到控制台
System.out.println(">>>" + f.get().toString());
}
Date date2 = new Date();
System.out.println("----程序结束运行----，程序运行时间【"

(date2.getTime() - date1.getTime()) + "毫秒】");
}
}

class MyCallable implements Callable<object> {
private String taskNum;
MyCallable(String taskNum) {
this.taskNum = taskNum;
}
public Object call() throws Exception {
System.out.println(">>>" + taskNum + "任务启动");
Date dateTmp1 = new Date();
Thread.sleep(1000);
Date dateTmp2 = new Date();

long time = dateTmp2.getTime() - dateTmp1.getTime();
System.out.println(">>>" + taskNum + "任务终止");
return taskNum + "任务返回运行结果,当前任务时间【" + time + "毫秒】";
}
}
代码说明：
上述代码中 Executors 类，提供了一系列工厂方法用于创先线程池，返回的线程池都实现了 Executor ervice 接口。
public static ExecutorService newFixedThreadPool(int nThreads)
创建固定数目线程的线程池。
public static ExecutorService newCachedThreadPool()
创建一个可缓存的线程池，调用 execute 将重用以前构造的线程（如果线程可用）。如果现有线程没可用的，则创建一个新线程并添加到池中。终止并从缓存中移除那些已有 60 秒钟未被使用的线程。<r>
public static ExecutorService newSingleThreadExecutor()
创建一个单线程化的 Executor。
public static ScheduledExecutorService newScheduledThreadPool(int corePoolSize)
创建一个支持定时及周期性的任务执行的线程池，多数情况下可用来替代 Timer 类。
ExecutoreService 提供了 submit()方法，传递一个 Callable，或 Runnable，返回 Future。如果 Executor 后台线程池还没有完成 Callable 的计算，这调用返回 Future 对象的 get()方法，会阻塞直计算完成。
转载于：http://www.cnblogs.com/yezhenhan/archive/2012/01/09/2317636.html
</object>