Executor源码阅读

# 总览

package java.util.concurrent;

public interface Executor {

void execute(Runnable command);

}

# 阅读

## Executor接口的作用

Executor接口的主要作用为执行客户端提交的任务（包裹在Runnable实例中）

## Executor接口的优点

使用Executor接口具有以下好处：

1. 任务创建与任务执行解耦

客户端只负责提交任务，不需关心任务如何执行，任务的执行将由Executor负责。

1. 避免客户端显示创建线程

如果不使用Executor，客户端处理任务集的方式可能会是这样的：

new Thread*(*new RunnableTask1*())*.start*()*;

new Thread*(*new RunnableTask2*())*.start*()*;

new Thread*(*new RunnableTask3*())*.start*()*;

使用Executor，客户端可以这样处理任务集：

Executor executor = getExecutor();  
executor.execute(new RunnableTask1());  
executor.execute(new RunnableTask2());  
executor.execute(new RunnableTask3());

## Executor接口的实现

1）Executor接口最简单的实现方式如下：

class DirectExecutor implements Executor {  
 public void execute(Runnable r) {  
 r.run();  
 }  
}}

以上的实现方式比较特殊，客户端提交的任务会立刻在客户端线程中执行。

2）更常见的，任务会被放置在客户端之外的线程中执行：

class ThreadPerTaskExecutor implements Executor {  
 public void execute(Runnable r) {  
 new Thread(r).start();  
 }  
}}

1. 有些实现还会为任务如何执行添加一套限制规则：

class SerialExecutor implements Executor {  
 //任务队列  
 final Queue<Runnable> tasks = new ArrayDeque<Runnable>();  
 final Executor executor;  
 Runnable active;  
  
 SerialExecutor(Executor executor) {  
 this.executor = executor;  
 }  
  
 */\*\*  
 \* 任务执行  
 \*/* public synchronized void execute(final Runnable r) {  
 tasks.offer(new Runnable() {  
 public void run() {  
 try {  
 r.run();  
 } finally {  
 scheduleNext();  
 }  
 }  
 });  
 if (active == null) {  
 scheduleNext();  
 }  
 }  
  
 */\*\*  
 \* 任务调度  
 \*/* protected synchronized void scheduleNext() {  
 if ((active = tasks.poll()) != null) {  
 executor.execute(active);  
 }  
 }  
}

## Executor接口的使用

一般很少直接使用Executor接口，更多使用Executor的拓展接口ExecutorService。ThreadPoolExecutor类提供了可拓展的线程池实现。Executors类为这些Executor接口提供了方便的工厂方法。

# 理解

1. Executor接口存在的意思是使程序实现任务创建和任务执行的解耦。
2. 将任务执行划分到Executor接口，能够减小客户端接口的规模，符合接口隔离原则。