链滴

java--Executor

作者：hymn

原文链接：https://ld246.com/article/1591066007443

来源网站：链滴

```
public interface Executor(){
    void ecextor(Runnable command);
}
```

主要方法：

创建线程池：

```
// 1、
// newFixedThreadPool();
// 创建一个固定线程数量的线程池，每当提交一个任务时就创建一个线程，
// 直到达到线程池数量，这时线程池的规模将不会变化
Executor exec = Executors.newFixedThreadPool(100);

Runnable task = new Runnable(){
    public void run(){
        //do something
    }
};
exec.execute(task);
// 2、
// newCacheThreadPool();
// 创建一个可缓存的线程池，如果线程池规模超过了处理需求时，那么将回收空闲的线程，
// 当需求增加时，则添加新的线程，线程池规模没有限制。

// 3、
// newSingleThreadPool();
// 这是一个单线程的Executor,他创建单个工作线程来执行任务，如果这个线程异常结束
// 则会创建另一个来代替，他能确保任务在队列中串行执行（FIFO,LIFO,优先级）。内部提供了大量
步机制，
// 确保任务执行的操作对后续线程都是可见的。
```

```
// 4、
// newScheduledThreadPool()
// 创建一个固定线程的线程池，可以延迟或者定时执行
```

用法：

```
// 多线程执行
public class ThreadTask implements Executor{
    public void execute(Runnable r){
        new Thread(r).start();
    };
}

// 串行
public class Task implements Executor{
    public void execute(Runnable r){
        r.run();
    };
}
```

和Future Callable一起使用

```
// ExecutorService 继承了 Executor
ExecutorService exec =  ...;

Callable task = new Callable(){
    public Result call(){
        // do something
        return new Result();
    }
};

Future future = exec.submit(task);

future.get();
```

invokeAll() (ExecutorService)

```
    /**
     * Executes the given tasks, returning a list of Futures holding
     * their status and results when all complete.
     * {@link Future#isDone} is {@code true} for each
     * element of the returned list.
     * Note that a <em>completed</em> task could have
     * terminated either normally or by throwing an exception.
     * The results of this method are undefined if the given
     * collection is modified while this operation is in progress.
     *
     * @param tasks the collection of tasks
     * @param <T> the type of the values returned from the tasks
     * @return a list of Futures representing the tasks, in the same
     *         sequential order as produced by the iterator for the
     *         given task list, each of which has completed
```

```
     * @throws InterruptedException if interrupted while waiting, in
     *         which case unfinished tasks are cancelled
     * @throws NullPointerException if tasks or any of its elements are {@code null}
     * @throws RejectedExecutionException if any task cannot be
     *         scheduled for execution
     */
    <T> List<Future<T>> invokeAll(Collection<? extends Callable<T>> tasks)
        throws InterruptedException;
```

CompletionService

如果向 Executor 提交了一组计算任务，并且希望在计算完成后获得结果，那么可以保留与每个任务关联的 Future，然后反复使用 get 方法，同时将参数 timeout 指定为 0，从而通过轮询来判断任务是否完成。这种方法虽然可行，但却有些繁琐。幸运的是，还有一种更好的方法：完成服务 (CompletionService)。

CompletionService 将 Executor 和 BlockingQueue 的功能融合在一起。你可以将 Callable 任务提交给它来执行，然后使用类似于队列操作的 take 和 poll 等方法来获得已完成的结果，而这些结果会在完成时将被封装为 Future。ExecutorCompletionService 实现了 CompletionService，并将计算部分委托给一个 Executor。

ExecutorCompletionService 的实现非常简单。在构造函数中创建一个 BlockingQueue 来保存计算完成的结果。当计算完成时，调用 Future-Task 中的 done 方法。当提交某个任务时，该任务将首先包装为一个 QueueingFuture，这是 FutureTask 的一个子类，然后再改写子类的 done 方法，并将结果放入 BlockingQueue 中，如程序清单 6-14 所示。take 和 poll 方法委托给了 BlockingQueue，这些方法会在得出结果之前阻塞。

```
private class QueueingFuture<V> extends FutureTask<V> {
    QueueingFuture(Callable<V> c) { super(c); }
    QueueingFuture(Runnable t, V r) { super(t, r); }

    protected void done() {
        completionQueue.add(this);
    }
}
```

程序清单 6-15　使用 CompletionService，使页面元素在下载完成后立即显示出来

```
public class Renderer {
    private final ExecutorService executor;

    Renderer(ExecutorService executor) { this.executor = executor; }

    void renderPage(CharSequence source) {
        List<ImageInfo> info = scanForImageInfo(source);
        CompletionService<ImageData> completionService =
            new ExecutorCompletionService<ImageData>(executor);
        for (final ImageInfo imageInfo : info)
            completionService.submit(new Callable<ImageData>() {
                public ImageData call() {
                    return imageInfo.downloadImage();
                }
            });

        renderText(source);

        try {
            for (int t = 0, n = info.size(); t < n; t++) {
                Future<ImageData> f = completionService.take();
                ImageData imageData = f.get();
                renderImage(imageData);
            }
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
        } catch (ExecutionException e) {
            throw launderThrowable(e.getCause());
        }
    }
}
```