Executor、ExecutorService接口、Executors等

1. **Executor接口**
2. **package**

public interface Executor

Executor接口存在于java.util.concurrent包中，**具有2个子接口**，All Known Subinterfaces:**ExecutorService**, **ScheduledExecutorService；所有的实现类：**All Known Implementing Classes:**AbstractExecutorService**, ForkJoinPool, **ScheduledThreadPoolExecutor**, **ThreadPoolExecutor。**

**Executor接口只规定了一个方法:execute(Runnable command)。**

**纠正：**之前Runnable的实现类喜欢命名为Target，不过命名为Task或Command比较好。

1. **功能介绍**

**An object that executes submitted Runnable tasks.** This interface provides a way of decoupling task submission from the mechanics of how each task will be run, including details of thread use, scheduling, etc. An Executor is normally used **instead of explicitly(明确地) creating threads**. For example, rather than invoking new Thread(new(RunnableTask())).start() for each of a set of tasks, you might use:

Executor executor = anExecutor;

executor.execute(**new RunnableTask1());**

executor.execute(**new RunnableTask2());**…

1. **execute(Runnable command)**

void execute(Runnable command)

Executes the given command at some time in the future. The command may execute **in a new thread, in a pooled thread, or in the calling thread**, at the discretion of the Executor implementation. ( at the discretion of … 由…来决定)

Parameters: command - the runnable task

Throws: RejectedExecutionException - if this task cannot be accepted for execution.

NullPointerException - if command is null

1. **Executor接口的实现方式介绍**
2. 最简单的方式：直接实现

However, the Executor interface does not strictly require that execution be **asynchronous(异步的)**. In the simplest case, an executor can run the submitted task immediately in the caller's thread:

class **DirectExecutor** implements Executor {

public void execute(Runnable r) {

r.run();

}

}

1. 典型的：开启新线程

**More typically**, tasks are executed in some thread other than the caller's thread. The executor below spawns a new thread for each task.

class **ThreadPerTaskExecutor** implements Executor {

public void execute(Runnable r) {

new Thread(r).start();

}

}

1. 施加限制条件

Many Executor implementations **impose**(施加,利用) **some sort of limitation** on how and when tasks are scheduled. The executor below serializes the submission of tasks to a second executor, illustrating a **composite**(混合的) executor.

class **SerialExecutor** implements Executor {

final Queue<Runnable> tasks = new **ArrayDeque**<Runnable>();

final Executor executor;

Runnable active

SerialExecutor(Executor executor) {

this.executor = executor;

}

public **synchronized** void execute(final Runnable r) {

tasks.offer(new Runnable() {

public void run() {

try {

r.run();

} finally {

scheduleNext();

}

}

});

if (active == null) {

scheduleNext();

}

}

protected synchronized void scheduleNext() {

if ((active = tasks.poll()) != null) {

executor.execute(active);

}

}

}

The Executor implementations provided in this package implement **ExecutorService**, which is a more extensive interface. The ThreadPoolExecutor class provides an extensible thread pool implementation. The **Executors** class provides convenient factory methods for these Executors.

# Executors(提供工厂方法)

## package

public class **Executors** extends Object

Executors类存在于java.util.concurrent包中，直接继承于Object类，没有子类。

## 功能

**Executors类**中除了继承Object的方法外，其他方法都是静态static的，主要是为同包中**Executor**, **ExecutorService**, **ScheduledExecutorService**, **ThreadFactory**, and **Callable** classes 这些类提供工厂和工具方法。

Factory and utility methods for **Executor**, **ExecutorService**, **ScheduledExecutorService**, **ThreadFactory**, and **Callable** classes defined in this package. This class supports the following kinds of methods:

### Methods that create and return an **ExecutorService** set up with commonly useful configuration settings.

### Methods that create and return a **ScheduledExecutorService** set up with commonly useful configuration settings.

### Methods that create and return a "wrapped" **ExecutorService**, that disables reconfiguration by making implementation-specific methods inaccessible.

### Methods that create and return a **ThreadFactory** that sets newly created threads to a known state.

### Methods that create and return a **Callable** out of other closure-like forms, so they can be used in execution methods requiring Callable.

## 没有构造方法：

**Executors**中都是静态方法，直接使用**Executors.method()**调用即可。

## 静态方法介绍

主要用来生成常用的线程池，常用的方法有三种：

**newSingleThreadExecutor**：创建一个单线程的线程池；

**newFixedThreadPool**：创建固定大小的线程池；

**newCachedThreadPool**：创建一个可缓存的线程池。

还有： newSingleThreadScheduledExecutor：创建单线程可调度的线程池。

### **newSingleThreadExecutor**

static **ExecutorService** **newSingleThreadExecutor**()

Creates an Executor that uses a single worker thread operating off an unbounded queue.

static **ExecutorService** **newSingleThreadExecutor**(**ThreadFactory threadFactory)**

Creates an Executor that uses a single worker thread operating off an unbounded queue, and uses the provided **ThreadFactory** to create a new thread when needed.

### **newFixedThreadPool**

static **ExecutorService** **newFixedThreadPool**(int nThreads)

Creates a thread pool that reuses a fixed number of threads operating off a shared unbounded queue.

static **ExecutorService** **newFixedThreadPool**(int nThreads, ThreadFactory threadFactory)

**Creates a thread pool** that reuses a fixed number of threads operating off a shared unbounded queue, using the provided **ThreadFactory** to create new threads when needed.

### **newCachedThreadPool**

static **ExecutorService** **newCachedThreadPool**()

Creates a thread pool that creates new threads as needed, but will reuse previously constructed threads when they are available.

static **ExecutorService** **newCachedThreadPool**(ThreadFactory threadFactory)

Creates a thread pool that creates new threads as needed, but will reuse previously constructed threads when they are available, and uses the provided ThreadFactory to create new threads when needed.

### **newSingleThreadScheduledExecutor**

static **ScheduledExecutorService** **newSingleThreadScheduledExecutor**()

Creates a **single-threaded executor** that can schedule commands to run after a given delay, or to execute periodically.

static **ScheduledExecutorService** **newSingleThreadScheduledExecutor**(ThreadFactory threadFactory)

Creates a single-threaded executor that can **schedule** commands to run after a given delay, or to execute periodically.

### defaultThreadFactory():创建一个默认ThreadFactory。

static ThreadFactory defaultThreadFactory()

Returns a default thread factory used to create new threads.

### 还有其他方法，可以查看API

# ExecutorService

## package

public interface **ExecutorService** extends **Executor**

ExecutorService接口存在于**java.util.concurrent**包中，继承了Executor接口，还有一个子接口**ScheduledExecutorService；**4个实现类：AbstractExecutorService, ForkJoinPool, **ScheduledThreadPoolExecutor**, **ThreadPoolExecutor。**

## 功能

An **Executor** that provides methods to manage termination and methods that can produce a Future for tracking progress of one or more asynchronous tasks.

An **ExecutorService** can be shut down, which will cause it to reject new tasks. Two different methods are provided for shutting down an **ExecutorService**. The shutdown() method will allow previously submitted tasks to execute before terminating, while the **shutdownNow()** method prevents waiting tasks from starting and attempts to stop currently executing tasks. Upon termination, an executor has no tasks actively executing, no tasks awaiting execution, and no new tasks can be submitted. An unused ExecutorService should be shut down to allow reclamation of its resources.

shutdown方法：允许当前执行的线程执行完毕；

shutdownNow方法：试图停止当前正在进行和等待执行的线程。

Method **submit** extends base method **Executor.execute(java.lang.Runnable)** by creating and returning a **Future** that can be used to cancel execution and/or wait for completion. Methods **invokeAny** and **invokeAll** perform the most commonly useful forms of bulk execution, executing a collection of tasks and then waiting for at least one, or all, to complete. (Class ExecutorCompletionService can be used to write customized variants of these methods.)

The **Executors** class provides factory methods for the executor services provided in this package.(见Executors)

## 定义的方法

### 继承Executor接口的execute方法

void execute(Runnable command)

Executes the given command at some time in the future.

### invokeAll方法

**<T> List<Future<T>> invokeAll(Collection<? extends Callable<T>> tasks)**

Executes the given tasks, returning a list of **Futures** holding their status and results when all complete.

**<T> List<Future<T>> invokeAll(Collection<? extends Callable<T>> tasks, long timeout, TimeUnit unit)**

Executes the given tasks, returning a list of Futures holding their status and results when all complete or the timeout expires, whichever happens first.

### invokeAny方法

**<T> T invokeAny(Collection<? extends Callable<T>> tasks)**

Executes the given tasks**, returning the result of one** that has completed successfully (i.e., without throwing an exception), if any do.

**<T> T invokeAny(Collection<? extends Callable<T>> tasks, long timeout, TimeUnit unit)**

Executes the given tasks, returning the result of one that has completed successfully (i.e., without throwing an exception), if any do before the given timeout elapses.

### submit

<T> Future<T> submit(**Callable<T> task**)

Submits a value-returning task for execution and returns a **Future** representing the pending results of the task.

Future<?> submit(**Runnable task**)

Submits a Runnable task for execution and returns a Future representing that task.

<T> Future<T> submit(**Runnable task, T result**)

Submits a Runnable task for execution and returns a Future representing that task.

### shutdown方法

**void shutdown()**

Initiates an orderly shutdown in which previously submitted tasks are executed, but no new tasks will be accepted.

**List<Runnable> shutdownNow()**

Attempts to stop all actively executing tasks, halts the processing of waiting tasks, and returns a list of the tasks that were awaiting execution.

### awaitTermination方法

boolean **awaitTermination**(long timeout, TimeUnit unit)

Blocks until all tasks have completed execution after a shutdown request, or the timeout occurs, or the current thread is interrupted, whichever happens first.

### isXxx判断方法

boolean **isShutdown()**

Returns true if this executor has been shut down.

boolean **isTerminated()**

Returns true if all tasks have completed following shut down.

## 使用示例

Here is a sketch of a network service in which threads in a thread pool service incoming requests. It uses the preconfigured Executors.newFixedThreadPool(int) factory method:

class **NetworkService** implements Runnable {

private final ServerSocket serverSocket;

private final ExecutorService pool;

public NetworkService(int port, int poolSize)

throws IOException {

serverSocket = new ServerSocket(**port**);

pool = Executors.newFixedThreadPool(**poolSize**);

}

public void run() { // run the service

try {

for (;;) {

pool.execute(new Handler(serverSocket.accept()));

}

} catch (IOException ex) {

pool.shutdown();

}

}

}

class Handler implements Runnable {

private final Socket socket;

Handler(Socket socket) { this.socket = socket; }

public void run() {

// read and service request on socket

}

}

The following method shuts down an **ExecutorService** in two phases, **first** by calling shutdown to reject incoming tasks, and **then** calling shutdownNow, if necessary, to cancel any lingering tasks:

void shutdownAndAwaitTermination(ExecutorService pool) {

pool.shutdown(); // Disable new tasks from being submitted

try {

// Wait a while for existing tasks to terminate

if (!pool.awaitTermination(60, TimeUnit.SECONDS)) {

**pool.shutdownNow();** // Cancel currently executing tasks

// Wait a while for tasks to respond to being cancelled

if (!pool.awaitTermination(60, TimeUnit.SECONDS))

System.err.println("Pool did not terminate");

}

} catch (InterruptedException ie) {

// (Re-)Cancel if current thread also interrupted

pool.shutdownNow();

// Preserve interrupt status

Thread.currentThread().interrupt();

}

}

# ScheduledExecutorService

## package

public interface **ScheduledExecutorService** extends **ExecutorService**

**ScheduledExecutorService接口**存在于java.util.concurrent包中，直接继承于ExecutorService接口，间接继承于Executor，有1个实现类：**ScheduledThreadPoolExecutor**。

## 功能：

An **ExecutorService** that can schedule commands to run after a given delay, or to execute periodically.

The schedule methods create tasks with various delays and return a task object that can be used to cancel or check execution. The **scheduleAtFixedRate** and **scheduleWithFixedDelay** methods create and execute tasks that run periodically until cancelled.

Commands submitted using the Executor.execute(java.lang.Runnable) and ExecutorService submit methods are scheduled with a requested delay of zero. Zero and negative delays (but not periods) are also allowed in schedule methods, and are treated as requests for immediate execution.

All schedule methods accept relative delays and periods as arguments, not absolute times or dates. It is a simple matter to transform an absolute time represented as a Date to the required form. For example, to schedule at a certain future date, you can use: schedule(task, date.getTime() - System.currentTimeMillis(), TimeUnit.MILLISECONDS). Beware however that expiration of a relative delay need not coincide with the current Date at which the task is enabled due to network time synchronization protocols, clock drift, or other factors. The **Executors** class provides convenient factory methods for the **ScheduledExecutorService** implementations provided in this package.

## 方法介绍

### 继承Executor、ExecutorService的方法

### schedule方法

<V> **ScheduledFuture**<V> **schedule**(Callable<V> callable, long delay, TimeUnit unit)

Creates and executes a **ScheduledFuture** that becomes enabled after the given delay.

**ScheduledFuture**<?> **schedule**(Runnable command, long delay, TimeUnit unit)

Creates and executes a one-shot action that becomes enabled after the given delay.

### **scheduleAtFixedRate**

ScheduledFuture<?> **scheduleAtFixedRate**(Runnable command, long initialDelay, long period, TimeUnit unit)

Creates and executes a periodic action that becomes enabled first after the given initial delay, and subsequently with the given period; that is executions will commence after initialDelay then **initialDelay+period**, then initialDelay + 2 \* period, and so on.

### **scheduleWithFixedDelay**

ScheduledFuture<?> **scheduleWithFixedDelay**(Runnable command, long initialDelay, long delay, TimeUnit unit)

Creates and executes a periodic action that becomes enabled first after the given initial delay, and subsequently with the given delay between the termination of one execution and the commencement of the next.

### **scheduleAtFixedRate与scheduleWithFixedDelay的区别**

初始initialDelay延迟都一样，不同的是delay，scheduleAtFixedRate指的是前一个线程开始到后1个线程开始之间的时间为delay；而scheduleWithFixedDelay指的是前一个线程结束到后1个线程开始之间的时间为delay。

## 使用示例

Here is a class with a method that sets up a **ScheduledExecutorService** to beep every ten seconds for an hour:

import static java.util.concurrent.TimeUnit.\*;

class BeeperControl {

private final ScheduledExecutorService **scheduler** =

**Executors.newScheduledThreadPool(1);**

public void beepForAnHour() {

final Runnable beeper = new Runnable() {

public void run() { System.out.println("beep"); }

};

final ScheduledFuture<?> beeperHandle =

**scheduler.scheduleAtFixedRate(beeper, 10, 10, SECONDS);**

scheduler.**schedule**(new Runnable() {

public void run() { beeperHandle.cancel(true); }

}, 60 \* 60, SECONDS);

}

}

# Thread Pool 与Executor的区别

现有理解：Thread Pool中存在多个线程，这些线程可以循环接收Runnable实现任务；而Executor只是一个执行器，用来执行Runnable实现任务。对于单线程的线程池其等价与一个Executor，因为接收任务直接run即可。