**ScheduledExecutorService执行周期性或定时任务**

2011-10-14 15:23:21

标签：[休闲](http://blog.51cto.com/tag-%E4%BC%91%E9%97%B2.html) [ScheduledExecutorService](http://blog.51cto.com/tag-ScheduledExecutorService.html) [职场](http://blog.51cto.com/tag-%E8%81%8C%E5%9C%BA.html) [定时任务](http://blog.51cto.com/tag-%E5%AE%9A%E6%97%B6%E4%BB%BB%E5%8A%A1.html) [周期性](http://blog.51cto.com/tag-%E5%91%A8%E6%9C%9F%E6%80%A7.html)

原创作品，允许转载，转载时请务必以超链接形式标明文章 [原始出处](http://ketqi.blog.51cto.com/1130608/687681) 、作者信息和本声明。否则将追究法律责任。<http://ketqi.blog.51cto.com/1130608/687681>

 ScheduledExecutorService扩展了ExecutorService接口，提供时间排程的功能。

|  |
| --- |
| [**schedule**](http://www.google.com/url?q=http%3A%2F%2Fwww.oschina.net%2Fuploads%2Fdoc%2Fjavase-6-doc-api-zh_CN%2Fjava%2Futil%2Fconcurrent%2FScheduledExecutorService.html%23schedule(java.util.concurrent.Callable%2C%2520long%2C%2520java.util.concurrent.TimeUnit)&sa=D&sntz=1&usg=AFQjCNEVNnGnBdJG3G7L23LvgvoSFsX7bQ)([Callable](http://www.google.com/url?q=http%3A%2F%2Fwww.oschina.net%2Fuploads%2Fdoc%2Fjavase-6-doc-api-zh_CN%2Fjava%2Futil%2Fconcurrent%2FCallable.html&sa=D&sntz=1&usg=AFQjCNGLHZdGVER_qh0cfweckPE-GQhEHA)<V> callable, long delay, [TimeUnit](http://www.google.com/url?q=http%3A%2F%2Fwww.oschina.net%2Fuploads%2Fdoc%2Fjavase-6-doc-api-zh_CN%2Fjava%2Futil%2Fconcurrent%2FTimeUnit.html&sa=D&sntz=1&usg=AFQjCNEfES1diXYadR4EFVIw_os_ITDOxQ) unit)           创建并执行在给定延迟后启用的 ScheduledFuture。 |
| [**schedule**](http://www.google.com/url?q=http%3A%2F%2Fwww.oschina.net%2Fuploads%2Fdoc%2Fjavase-6-doc-api-zh_CN%2Fjava%2Futil%2Fconcurrent%2FScheduledExecutorService.html%23schedule(java.lang.Runnable%2C%2520long%2C%2520java.util.concurrent.TimeUnit)&sa=D&sntz=1&usg=AFQjCNHjw0lKqBtzV4Y0ca2C1wXUHRhLGA)([Runnable](http://www.google.com/url?q=http%3A%2F%2Fwww.oschina.net%2Fuploads%2Fdoc%2Fjavase-6-doc-api-zh_CN%2Fjava%2Flang%2FRunnable.html&sa=D&sntz=1&usg=AFQjCNEEpl7LQHmakIPDkF72pQOBHeka5g) command, long delay, [TimeUnit](http://www.google.com/url?q=http%3A%2F%2Fwww.oschina.net%2Fuploads%2Fdoc%2Fjavase-6-doc-api-zh_CN%2Fjava%2Futil%2Fconcurrent%2FTimeUnit.html&sa=D&sntz=1&usg=AFQjCNEfES1diXYadR4EFVIw_os_ITDOxQ) unit)           创建并执行在给定延迟后启用的一次性操作。 |
| [**scheduleAtFixedRate**](http://www.google.com/url?q=http%3A%2F%2Fwww.oschina.net%2Fuploads%2Fdoc%2Fjavase-6-doc-api-zh_CN%2Fjava%2Futil%2Fconcurrent%2FScheduledExecutorService.html%23scheduleAtFixedRate(java.lang.Runnable%2C%2520long%2C%2520long%2C%2520java.util.concurrent.TimeUnit)&sa=D&sntz=1&usg=AFQjCNE6UA_4r3EAbucW71l4x9uUwwbyvA)([Runnable](http://www.google.com/url?q=http%3A%2F%2Fwww.oschina.net%2Fuploads%2Fdoc%2Fjavase-6-doc-api-zh_CN%2Fjava%2Flang%2FRunnable.html&sa=D&sntz=1&usg=AFQjCNEEpl7LQHmakIPDkF72pQOBHeka5g) command, long initialDelay, long period, [TimeUnit](http://www.google.com/url?q=http%3A%2F%2Fwww.oschina.net%2Fuploads%2Fdoc%2Fjavase-6-doc-api-zh_CN%2Fjava%2Futil%2Fconcurrent%2FTimeUnit.html&sa=D&sntz=1&usg=AFQjCNEfES1diXYadR4EFVIw_os_ITDOxQ)unit)           创建并执行一个在给定初始延迟后首次启用的定期操作，后续操作具有给定的周期；也就是将在 initialDelay 后开始执行，然后在initialDelay+period 后执行，接着在 initialDelay + 2 \* period 后执行，依此类推。 |
| [**scheduleWithFixedDelay**](http://www.google.com/url?q=http%3A%2F%2Fwww.oschina.net%2Fuploads%2Fdoc%2Fjavase-6-doc-api-zh_CN%2Fjava%2Futil%2Fconcurrent%2FScheduledExecutorService.html%23scheduleWithFixedDelay(java.lang.Runnable%2C%2520long%2C%2520long%2C%2520java.util.concurrent.TimeUnit)&sa=D&sntz=1&usg=AFQjCNGlNFn791ZXM6FSug43i50M3c-l-g)([Runnable](http://www.google.com/url?q=http%3A%2F%2Fwww.oschina.net%2Fuploads%2Fdoc%2Fjavase-6-doc-api-zh_CN%2Fjava%2Flang%2FRunnable.html&sa=D&sntz=1&usg=AFQjCNEEpl7LQHmakIPDkF72pQOBHeka5g) command, long initialDelay, long delay,[TimeUnit](http://www.google.com/url?q=http%3A%2F%2Fwww.oschina.net%2Fuploads%2Fdoc%2Fjavase-6-doc-api-zh_CN%2Fjava%2Futil%2Fconcurrent%2FTimeUnit.html&sa=D&sntz=1&usg=AFQjCNEfES1diXYadR4EFVIw_os_ITDOxQ) unit)           创建并执行一个在给定初始延迟后首次启用的定期操作，随后，在每一次执行终止和下一次执行开始之间都存在给定的延迟。 |

schedule方法被用来延迟指定时间来执行某个指定任务。如果你需要周期性重复执行定时任务可以使用scheduleAtFixedRate或者scheduleWithFixedDelay方法，它们不同的是前者以固定频率执行，后者以相对固定频率执行。

不管任务执行耗时是否大于间隔时间，scheduleAtFixedRate和scheduleWithFixedDelay都不会导致同一个任务并发地被执行。唯一不同的是scheduleWithFixedDelay是当前一个任务结束的时刻，开始结算间隔时间，如0秒开始执行第一次任务，任务耗时5秒，任务间隔时间3秒，那么第二次任务执行的时间是在第8秒开始。

ScheduledExecutorService的实现类，是ScheduledThreadPoolExecutor。ScheduledThreadPoolExecutor对象包含的线程数量是没有可伸缩性的，只会有固定数量的线程。不过你可以通过其构造函数来设定线程的优先级，来降低定时任务线程的系统占用。

特别提示：通过ScheduledExecutorService执行的周期任务，如果任务执行过程中抛出了异常，那么过ScheduledExecutorService就会停止执行任务，且也不会再周期地执行该任务了。所以你如果想保住任务都一直被周期执行，那么catch一切可能的异常。

package test;  
import java.text.SimpleDateFormat;  
import java.util.Date;  
import java.util.concurrent.ScheduledThreadPoolExecutor;  
import java.util.concurrent.TimeUnit;  
  
*/\*\**  
*\* create at 11-10-14*  
*\* @author KETQI*  
*\* @category*  
*\*/*  
**public** **class** TestScheduledThreadPoolExecutor {  
    **private** **static** SimpleDateFormat format=**new** SimpleDateFormat("yyyy-MM-dd HH:mm:ss");  
    **public** **static** **void** main(String[] args) {

*//ScheduledExecutorService exec=Executors.newScheduledThreadPool(1);*

        ScheduledThreadPoolExecutor exec = **new** ScheduledThreadPoolExecutor(1);  
        */\*\**  
*\*每隔一段时间打印系统时间，互不影响的<br/>*  
*\* 创建并执行一个在给定初始延迟后首次启用的定期操作，后续操作具有给定的周期；<br/>*  
*\* 也就是将在 initialDelay 后开始执行，然后在initialDelay+period 后执行，<br/>*  
*\* 接着在 initialDelay + 2 \* period 后执行，依此类推。*  
*\*/*  
        exec.scheduleAtFixedRate(**new** Runnable() {  
            **public** **void** run() {  
                System.out.println(format.format(**new** Date()));  
            }  
        }, 1000, 5000, TimeUnit.MILLISECONDS);  
  
        *//开始执行后就触发异常,next周期将不会运行*  
        exec.scheduleAtFixedRate(**new** Runnable() {  
            **public** **void** run() {  
                System.out.println("RuntimeException no catch,next time can't run");  
                **throw** **new** RuntimeException();  
            }  
        }, 1000, 5000, TimeUnit.MILLISECONDS);  
  
        *//虽然抛出了运行异常,当被拦截了,next周期继续运行*  
        exec.scheduleAtFixedRate(**new** Runnable() {  
            **public** **void** run() {  
                **try**{  
                    **throw** **new** RuntimeException();  
                }**catch** (Exception e){  
                    System.out.println("RuntimeException catched,can run next");  
                }  
            }  
        }, 1000, 5000, TimeUnit.MILLISECONDS);  
  
        */\*\**  
*\* 创建并执行一个在给定初始延迟后首次启用的定期操作，<br/>*  
*\* 随后，在每一次执行终止和下一次执行开始之间都存在给定的延迟。*  
*\*/*  
        exec.scheduleWithFixedDelay(**new** Runnable() {  
            **public** **void** run() {  
                System.out.println("scheduleWithFixedDelay:begin,"+format.format(**new** Date()));  
                **try** {  
                    Thread.sleep(2000);  
                } **catch** (InterruptedException e) {  
                    e.printStackTrace();  
                }  
                System.out.println("scheduleWithFixedDelay:end,"+format.format(**new** Date()));  
            }  
        },1000,5000,TimeUnit.MILLISECONDS);  
  
        */\*\**  
*\* 创建并执行在给定延迟后启用的一次性操作。*  
*\*/*  
        exec.schedule(**new** Runnable() {  
            **public** **void** run() {  
                System.out.println("The thread can only run once!");  
            }  
        },5000,TimeUnit.MILLISECONDS);  
    }  
}

本文出自 “[ketqi](http://ketqi.blog.51cto.com/)” 博客，请务必保留此出处<http://ketqi.blog.51cto.com/1130608/687681>