

Java定时任务调度详解



张丰哲 (/u/cb569cce501b) ✓已关注

2017.09.24 10:40 字数 2035 阅读 738 评论 11 喜欢 34 赞赏 1

(/u/cb569cce501b)

前言

在实际项目开发中，除了Web应用、SOA服务外，还有一类不可缺少的，那就是定时任务调度。定时任务的场景可以说非常广泛，比如某些视频网站，购买会员后，每天会给会员送成长值，每月会给会员送一些电影券；比如在保证最终一致性的场景中，往往利用定时任务调度进行一些比对工作；比如一些定时需要生成的报表、邮件；比如一些需要定时清理数据的任务等。本篇博客将系统的介绍定时任务调度，会涵盖Timer、ScheduledExecutorService、开源工具包Quartz，以及Spring和Quartz的结合等内容。

JDK原生定时工具：Timer

定时任务调度：基于给定的时间点、给定的时间间隔、给定的执行次数自动执行的任务。

Timer位于java.util包下，其内部包含且仅包含一个后台线程（TimeThread）对多个业务任务（TimeTask）进行定时定频率的调度。

schedule的四种用法和scheduleAtFixedRate的两种用法

```
public void schedule(TimerTask task, long delay)
public void schedule(TimerTask task, Date time)
public void schedule(TimerTask task, long delay, long period)
public void schedule(TimerTask task, Date firstTime, long period)
public void scheduleAtFixedRate(TimerTask task, long delay, long period)
public void scheduleAtFixedRate(TimerTask task, Date firstTime, long period)
```

Timer的核心方法

参数说明：

task：所要执行的任务，需要extends TimeTask override run()

time/firstTime：首次执行任务的时间

period：周期性执行Task的时间间隔，单位是毫秒

delay：执行task任务前的延时时间，单位是毫秒

很显然，通过上述的描述，我们可以实现：

延迟多久后执行一次任务；指定时间执行一次任务；延迟一段时间，并周期性执行任务；指定时间，并周期性执行任务；

思考1：如果time/firstTime指定的时间，在当前时间之前，会发生什么呢？



在时间等于或者超过time/firstTime的时候，会执行task！也就是说，如果time/firstTime指定的时间在当前时间之前，就会立即得到执行。

思考2：schedule和scheduleAtFixedRate有什么区别？

scheduleAtFixedRate：每次执行时间为上一次任务开始起向后推一个period间隔，也就是说下次执行时间相对于上一次任务开始的时间点，因此执行时间不会延后，但是存在任务并发执行的问题。

schedule：每次执行时间为上一次任务结束后推一个period间隔，也就是说下次执行时间相对于上一次任务结束的时间点，因此执行时间会不断延后。

思考3：如果执行task发生异常，是否会影响其他task的定时调度？

如果TimeTask抛出RuntimeException，那么Timer会停止所有任务的运行！

思考4：Timer的一些缺陷？

前面已经提及到Timer背后是一个单线程，因此Timer存在管理并发任务的缺陷：所有任务都是由同一个线程来调度，所有任务都是串行执行，意味着同一时间只能有一个任务得到执行，而前一个任务的延迟或者异常会影响到之后的任务。

其次，Timer的一些调度方式还算比较简单，无法适应实际项目中任务定时调度的复杂度。

一个简单的Demo实例

```
public class TimerDemo {  
  
    public static void main(String[] args) {  
  
        Timer timer = new Timer();  
        Calendar calendar = Calendar.getInstance();  
        // timer.schedule(new MyTask(), calendar.getTime(), 1000);  
        timer.scheduleAtFixedRate(new MyTask(), calendar.getTime(), 1000);  
    }  
}  
  
class MyTask extends TimerTask {  
  
    @Override  
    public void run() {  
        System.out.println("execute time start is : " +  
            new SimpleDateFormat("yyyy-MM-dd hh:mm:ss").format(this.scheduledExecutionTime()));  
        try {  
            Thread.sleep(2000);  
        } catch (InterruptedException e) {  
            e.printStackTrace();  
        }  
    }  
}
```

Timer Demo



```
execute time start is : 2017-09-23 07:15:51
execute time start is : 2017-09-23 07:15:52
execute time start is : 2017-09-23 07:15:53
execute time start is : 2017-09-23 07:15:54
execute time start is : 2017-09-23 07:15:55
```

运行结果

Timer其他需要关注的方法

cancel()：终止Timer计时器，丢弃所有当前已安排的任务（TimeTask也存在cancel()方法，不过终止的是TimeTask）

purge()：从计时器的任务队列中移除已取消的任务，并返回个数

JDK对定时任务调度的线程池支持：

ScheduledExecutorService

由于Timer存在的问题，JDK5之后便提供了基于线程池的定时任务调度：ScheduledExecutorService。

设计理念：每一个被调度的任务都会被线程池中的一个线程去执行，因此任务可以并发执行，而且相互之间不受影响。

我们直接看例子：

```
/**
 * Created by zhangfz on 2017/9/23.
 */
public class ScheduleExecutorServiceDemo implements Runnable{
    @Override
    public void run() {
        System.out.println("执行"+new Date());
    }

    public static void main(String[] args) {
        ScheduledExecutorService scheduledExecutorService = Executors.newScheduledThreadPool(10);
        scheduledExecutorService.scheduleAtFixedRate(new ScheduleExecutorServiceDemo(), 1000, 2000,
            TimeUnit.MILLISECONDS);
    }
}
```

基于线程池的定时任务调度

运行结果：

```
执行Sat Sep 23 19:46:27 CST 2017
执行Sat Sep 23 19:46:29 CST 2017
执行Sat Sep 23 19:46:31 CST 2017
执行Sat Sep 23 19:46:33 CST 2017
执行Sat Sep 23 19:46:35 CST 2017
```

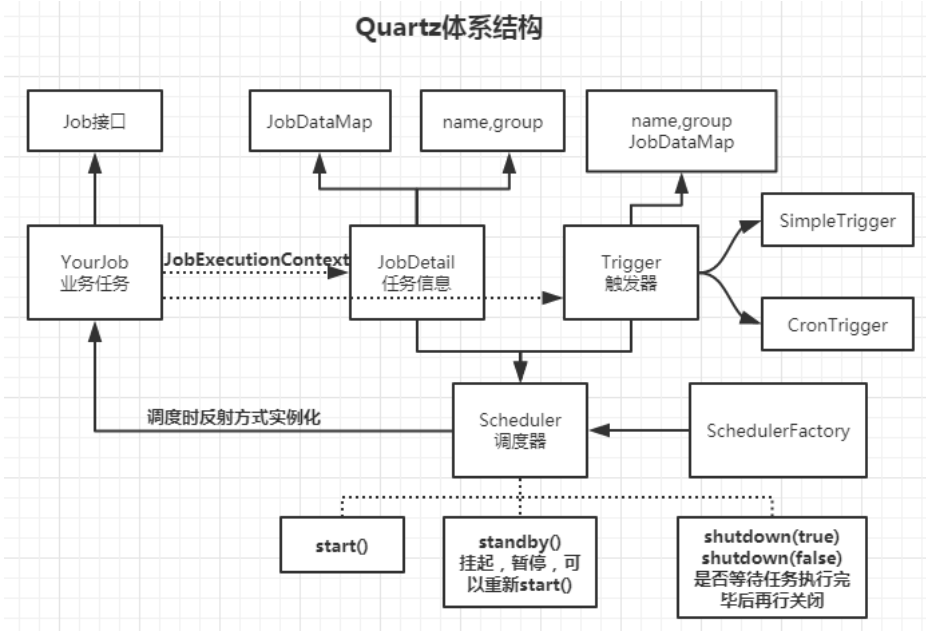
result

定时任务大哥：Quartz



虽然ScheduledExecutorService对Timer进行了线程池的改进，但是依然无法满足复杂的定时任务调度场景。因此OpenSymphony提供了强大的开源任务调度框架：Quartz。Quartz是纯Java实现，而且作为Spring的默认调度框架，由于Quartz的强大的调度功能、灵活的使用方式、还具有分布式集群能力，可以说Quartz出马，可以搞定一切定时任务调度！

Quartz的体系结构

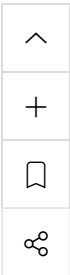


Quartz体系结构

先来看一个Demo：

```
public class MyJob implements Job {
    @Override
    public void execute(JobExecutionContext jobExecutionContext) throws JobExecutionException {
        System.out.println("执行" + new Date() + ", " +
            jobExecutionContext.getJobDetail().getJobDataMap().get("name"));
    }
}
```

业务Job



```
public class QuartzDemo {  
    public static void main(String[] args) throws SchedulerException {  
  
        JobDetail jobDetail = JobBuilder.newJob(MyJob.class)  
            .withIdentity("myJob", "myGroup")  
            .usingJobData("name", "zhangfengzhe")  
            .build();  
        //每两秒执行一次,直到永远  
        Trigger trigger = TriggerBuilder.newTrigger()  
            .withIdentity("triggerName", "triggerGroup")  
            .withSchedule(SimpleScheduleBuilder.simpleSchedule()  
                .withIntervalInSeconds(2).repeatForever())  
            .startNow()  
            .build();  
  
        SchedulerFactory schedulerFactory = new StdSchedulerFactory();  
        Scheduler scheduler = schedulerFactory.getScheduler();  
        scheduler.scheduleJob(jobDetail, trigger);  
        scheduler.start();  
    }  
}
```

Quartz

说明：

- 1、从代码上来看，有XxxBuilder、XxxFactory，说明Quartz用到了Builder、Factory模式，还有非常易懂的链式编程风格。
- 2、Quartz有3个核心概念：调度器（Scheduler）、任务（Job&JobDetail）、触发器（Trigger）。（一个任务可以被多个触发器触发，一个触发器只能触发一个任务）
- 3、注意当Scheduler调度Job时，实际上会通过反射newInstance一个新的Job实例（待调度完毕后销毁掉），同时会把JobExecutionContext传递给Job的execute方法，Job实例通过JobExecutionContext访问到Quartz运行时的环境以及Job本身的明细数据。
- 4、JobDataMap可以装载任何可以序列化的数据，存取很方便。需要注意的是JobDetail和Trigger都可以各自关联上JobDataMap。JobDataMap除了可以通过上述代码获取外，还可以在YourJob实现类中，添加相应setter方法获取。
- 5、Trigger用来告诉Quartz调度程序什么时候执行，常用的触发器有2种：SimpleTrigger（类似于Timer）、CronTrigger（类似于Linux的Crontab）。
- 6、实际上，Quartz在进行调度器初始化的时候，会加载quartz.properties文件进行一些属性的设置，比如Quartz后台线程池的属性（threadCount）、作业存储设置等。它会先从工程中找，如果找不到那么就是用quartz.jar中的默认的quartz.properties文件。
- 7、Quartz存在监听器的概念，比如任务执行前后、任务的添加等，可以方便实现任务的监控。

CronTrigger示例



```
//每两秒执行一次,直到永远
//
//      Trigger trigger = TriggerBuilder.newTrigger()
//          .withIdentity("triggerName", "triggerGroup")
//          .withSchedule(SimpleScheduleBuilder.simpleSchedule()
//              .withIntervalInSeconds(2).repeatForever())
//          .startNow()
//          .build();
//
//      Trigger trigger = TriggerBuilder.newTrigger()
//          .withIdentity("triggerName", "triggerGroup")
//          .withSchedule(CronScheduleBuilder.cronSchedule("0/2 * * ? * *"))
//          .build();
```

CronTrigger

Cron表达式的写法

特殊字符	含义
*	表示所有值。例如在分的字段上设置'',表示每一分钟都会触发。
?	表示不指定值。使用的场景为不需要关心当前设置这个字段的值。例如要在每月的10号触发一个操作,但不关心是周几,所以需要周位置的那个字段设置为'?'.具体设置为0 0 10 * ?
-	表示区间。例如在小时上设置'10-12',表示10,11,12点都会触发。
,	表示指定多个值,例如在周字段上设置'MON,WED,FRI'表示周一,周三和周五触发
/	用于递增触发。如在秒上面设置'5/15'表示从5秒开始,每增15秒触发(5,20,35,50)。在月字段上设置'1/3'所示每月1号开始,每隔三天触发一次。
L	表示最后的意思。在日字段设置上,表示当月的最后一天(依据当前月份,如果是二月还会依据是否是闰年[leap]),在周字段上表示星期六,相当于'7'或'SAT'。如果在'L'前加上数字,则表示该数据的最后一个。例如在周字段上设置'6L'这样的格式则表示'本月最后一个星期五'
W	表示离指定日期的最近那个工作日(周一至周五)。例如在日字段上设置'15W',表示离每月15号最近的那个工作日触发。如果15号正好是周六,则找最近的周五(14号)触发,如果15号是周末,则找最近的下周一(16号)触发。如果15号正好在工作日(周一至周五),则就在该天触发。如果指定格式为'1W',它则表示每月1号往后最近的工作日触发。如果1号正是周六,则将在3号下周一触发。(注,'W'前只能设置具体的数字,不允许区间'-')。
#	序号(表示每月的第几个周几),例如在周字段上设置'6#3'表示在每月的第三个周六(注意如果指定'#5',正好第五周没有周六,则不会触发该配置(用在母亲节和父亲节再合适不过了))

特殊符号说明

字段	是否必填	允许值	允许的特殊字符
秒	是	0~59	, - * /
分	是	0~59	, - * /
小时	是	0~23	, - * /
日	是	1~31	, - * ? / L W C
月	是	1~12 或者 JAN-DEC	, - * /
周	是	1~7 或者 SUN-SAT	, - * ? / L C #
年	否	empty, 1970~2099	, - * /

cron表达式

这里给出一些常用的示例：

0 15 10 * * ? * 每天10点15分触发

0 15 10 * * ? 2017 2017年每天10点15分触发

0 * 14 * * ? 每天下午的 2点到2点59分每分触发

0 0/5 14 * * ? 每天下午的 2点到2点59分(整点开始,每隔5分触发)

0 0/5 14,18 * * ? 每天下午的 2点到2点59分、18点到18点59分(整点开始,每隔5分触发)

0 0-5 14 * * ? 每天下午的 2点到2点05分每分触发

0 15 10 ? * 6L 每月最后一周的星期五的10点15分触发

0 15 10 ? * 6#3 每月的第三周的星期五开始触发

我们可以通过一些Cron在线工具非常方便的生成，比如<http://www.pppet.net/>等。



Spring和Quartz的整合

实际上，Quartz和Spring结合是很方便的，无非就是进行一些配置。大概基于2种方式：

第一，普通的类，普通的方法，直接在配置中指定（`MethodInvokingJobDetailFactoryBean`）。

第二，需要继承`QuartzJobBean`，复写指定方法（`executeInternal`）即可。

然后，就是一些触发器、调度器的配置了，这里不再展开介绍了，只要弄懂了原生的Quartz的使用，那么和Spring的结合使用就会很简单。

好了，到这里，定时任务调度就结束了，周末愉快！

2017-09-24 张丰哲

📖 日记本 (/nb/10261827)

举报文章 © 著作权归作者所有



张丰哲 (/u/cb569cce501b) ♂
写了 63893 字，被 2168 人关注，获得了 1683 个喜欢
(/u/cb569cce501b)

✓ 已关注




资深Java工程师 51CTO博客【2014-2016】：<http://zhangfengzhe.blog.51cto.com/>

好好学习，天天赞赏~

赞赏支持




👍 喜欢 | 34




更多分享

(<http://cwb.assets.jianshu.io/notes/images/1741866>)

 写下你的评论...

11条评论 只看作者

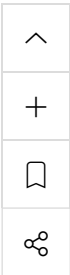
按喜欢排序 按时间正序 按时间倒序




张丰哲 (/u/cb569cce501b) 作者
2楼 · 2017.09.25 17:33
(/u/cb569cce501b)
补充下：Spring在3.0后推出了Spring自主研发的定时任务工具Spring Task，只需要Spring的依赖，一个轻量级的Quartz，可以采用注解的方式，使用起来非常方便~


👍 赞


💬 回复







似浅 (/u/0dee9f928c0e)： 请问spring的定时任务有手动触发执行和停止的机制么

2017.10.27 17:10  回复





张丰哲 (/u/cb569cce501b)： @似浅 (/users/0dee9f928c0e) 这个好像有点矛盾，既然是定时任务，就是定时触发。你说的“手动触发执行和停止”是指？

2017.10.28 20:39  回复




似浅 (/u/0dee9f928c0e)： @张丰哲 (/users/cb569cce501b) 比如用户登陆进入系统时，隔5分钟调用一次另一系统的接口，直到用户退出时，停止这个任务。

2017.10.30 14:36  回复

 添加新评论

还有1条评论， 展开查看






阿韦爱Android (/u/f408bdadacce)

3楼 · 2017.09.26 09:10


(/u/f408bdadacce)

不知道rxjava（ 安卓线程调度有关框架 ）是不是源于Spring的思想编写出来的。。。求博主分析一下


 赞  回复




张丰哲 (/u/cb569cce501b)： 这个可以留给你分享哈，Android程序员，哈哈~

2017.10.07 20:31  回复

举报

 添加新评论






最近胖了的dxx (/u/2cb177ed02a8)

4楼 · 2017.10.12 20:39


(/u/2cb177ed02a8)


Its没有讲到。目前公司在用。

 赞  回复



张丰哲 (/u/cb569cce501b)： 恩恩，light-task-scheduler轻量级分布式任务调度框架，这个博主还没有涉及到过， 😊

2017.10.12 21:58  回复

 添加新评论



你的小李 (/u/4f448009f2d9)

5楼 · 2017.10.19 16:35

(/u/4f448009f2d9)

很详细☐

 赞  回复






张丰哲 (/u/cb569cce501b)： @你的小李 (/users/4f448009f2d9) 常来看看，多交流， ☐

2017.10.19 20:41  回复

 添加新评论


被以下专题收入，发现更多相似内容


-  收入我的专题
- 


程序员 (/c/NEt52a?utm_source=desktop&utm_medium=notes-included-collection)
- 




首页投稿 (/c/bDHhpK?utm_source=desktop&utm_medium=notes-included-collection)









-  代码改变世界 (/c/0f5e015fc36c?utm_source=desktop&utm_medium=notes-included-collection)
-  web前端学习 (/c/8e68b6aea750?utm_source=desktop&utm_medium=notes-included-collection)
-  抓包 (/c/35b39ab856c1?utm_source=desktop&utm_medium=notes-included-collection)

- 推荐阅读

更多精彩内容 > (/)
- 工程化专题之Maven（下） (/p/34740cd1fb58?utm_c...** (/p/34740cd1fb58?
utm_campaign=maleskine&utm_content=note&utm

前言 《工程化专题之Maven（上）》 本文是工程化专题之Maven的下篇，主要涵盖的是Maven的Profile/Filter特性，多模块开发以及私服等内容。 不同环境...

张丰哲 (/u/cb569cce501b?
utm_campaign=maleskine&utm_content=user&utm_medium=pc_all_hots&utm_source=recommendation)
- Spring归纳小结 (/p/e0bf42b6850b?utm_campaign=...** (/p/e0bf42b6850b?
utm_campaign=maleskine&utm_content=note&utm

前言 如果说有什么框架是Java程序员必然会学习、使用到的，那么Spring肯定是其中之一。本篇博客，将根据博主在日常工作中对Spring的使用做一个系统...

张丰哲 (/u/cb569cce501b?
utm_campaign=maleskine&utm_content=user&utm_medium=pc_all_hots&utm_source=recommendation)
- 职场新人，如何快速学习并做好PPT？ (/p/c4d3d88c5...** (/p/c4d3d88c5c4d?
utm_campaign=maleskine&utm_content=note&utm

在职场混的人都知道，要想解决并做好一件事情。最好的技巧就是寻找问题背后的规律，最后再总结和复盘自己的经验。那么对于做PPT其实也是一样的...

郑少PPT (/u/47b9764d2127?
utm_campaign=maleskine&utm_content=user&utm_medium=pc_all_hots&utm_source=recommendation)
- 那个在同学聚会上消失的人后来怎么样了 (/p/00d370cf...** (/p/00d370cfe81e?
utm_campaign=maleskine&utm_content=note&utm

01 我就是在同学聚会上消失的那个人，生硬中略带些尴尬。六年前的一天，曾经的高中二班照例周末召开同学聚会，班长给我发了信息包括时间地点，我没...

非凡的希瑞 (/u/4f6fc465fd83?
utm_campaign=maleskine&utm_content=user&utm_medium=pc_all_hots&utm_source=recommendation)
- 大学毕业，多少人活成了廉价劳动力 (/p/3aa858ff133f...** (/p/3aa858ff133f?
utm_campaign=maleskine&utm_content=note&utm

文/怀左同学 01 学弟今年六月毕业，晃来晃去，到现在还没找下一份合适的工作。原本自信满满的他，目前很苦恼：“我在大学时做了几十种兼职，积累了...

怀左同学 (/u/62478ec15b74?
utm_campaign=maleskine&utm_content=user&utm_medium=pc_all_hots&utm_source=recommendation)

