# 概述

如果想在Spring中使用任务调度功能，除了集成调度框架Quartz这种方式，也可以使用Spring自己的调度任务框架。

使用Spring的调度框架，优点是：支持注解(@Scheduler)，可以省去大量的配置。

# 实时触发调度任务

## TaskScheduler接口

Spring3引入了TaskScheduler接口，这个接口定义了调度任务的抽象方法。

TaskScheduler接口的声明：

**public** **interface** TaskScheduler {

ScheduledFuture schedule(Runnable task, Trigger trigger);

ScheduledFuture schedule(Runnable task, Date startTime);

ScheduledFuture scheduleAtFixedRate(Runnable task, Date startTime, **long** period);

ScheduledFuture scheduleAtFixedRate(Runnable task, **long** period);

ScheduledFuture scheduleWithFixedDelay(Runnable task, Date startTime, **long** delay);

ScheduledFuture scheduleWithFixedDelay(Runnable task, **long** delay);

}

从以上方法可以看出TaskScheduler有两个重要维度：

一个是要调度的方法，即一个实现了Runnable接口的线程类的run()方法；另一个就是触发条件。

### TaskScheduler接口的实现类

它有三个实现类：DefaultManagedTaskScheduler、ThreadPoolTaskScheduler

、TimerManagerTaskScheduler。

DefaultManagedTaskScheduler：基于JNDI的调度器。

TimerManagerTaskScheduler：托管commonj.timers.TimerManager实例的调度器。

ThreadPoolTaskScheduler：提供线程池管理的调度器，它也实现了TaskExecutor接口，从而使的单一的实例可以尽可能快地异步执行。

## Trigger接口

Trigger接口抽象了触发条件的方法。

Trigger接口的声明：

**public** **interface** Trigger {

Date nextExecutionTime(TriggerContext triggerContext);

}

### Trigger接口的实现类

CronTrigger：实现了cron规则的触发器类（和Quartz的cron规则相同）。

PeriodicTrigger：实现了一个周期性规则的触发器类（例如：定义触发起始时间、间隔时间等）。

## 完整范例

实现一个调度任务的功能有以下几个关键点：

1. **定义调度器**

在spring-bean.xml中进行配置

使用task:scheduler标签定义一个大小为10的线程池调度器，spring会实例化一个ThreadPoolTaskScheduler。

***注：不要忘记引入xsd：***

http://www.springframework.org/schema/task

http://www.springframework.org/schema/task/spring-task-3.1.xsd

*<?***xml version="1.0" encoding="UTF-8"***?>*<**beans xmlns="http://www.springframework.org/schema/beans"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xmlns:mvc="http://www.springframework.org/schema/mvc"  
 xmlns:task="http://www.springframework.org/schema/task"  
 xsi:schemaLocation="http://www.springframework.org/schema/beans  
 http://www.springframework.org/schema/beans/spring-beans-3.1.xsd  
 http://www.springframework.org/schema/mvc  
 http://www.springframework.org/schema/mvc/spring-mvc-3.1.xsd  
 http://www.springframework.org/schema/task  
 http://www.springframework.org/schema/task/spring-task-3.1.xsd"**>  
  
 <**mvc:annotation-driven**/>  
 <**task:scheduler id="myScheduler" pool-size="10"**/>  
</**beans**>

1. 定义调度任务

定义实现Runnable接口的线程类。

**import** org.slf4j.Logger;  
**import** org.slf4j.LoggerFactory;  
  
**public class** DemoTask **implements** Runnable {  
 **final** Logger **logger** = LoggerFactory.*getLogger*(**this**.getClass());  
  
 @Override  
 **public void** run() {  
 **logger**.info(**"call DemoTask.run"**);  
 }  
}

1. 装配调度器，并执行调度任务

在一个Controller类中用@Autowired注解装配TaskScheduler。

然后调动TaskScheduler对象的schedule方法启动调度器，就可以执行调度任务了。

**import** org.springframework.beans.factory.annotation.Autowired;  
**import** org.springframework.scheduling.TaskScheduler;  
**import** org.springframework.scheduling.support.CronTrigger;  
**import** org.springframework.stereotype.Controller;  
**import** org.springframework.web.bind.annotation.RequestMapping;  
**import** org.springframework.web.bind.annotation.RequestMethod;  
  
@Controller  
@RequestMapping(**"/scheduler"**)  
**public class** SchedulerController {  
   
 @Autowired  
 TaskScheduler **scheduler**;  
   
 @RequestMapping(value = **"/start"**, method = RequestMethod.***POST***)  
 **public void** start() {  
 **scheduler**.schedule(**new** DemoTask(), **new** CronTrigger(**"0/5 \* \* \* \* \*"**));  
 }  
}

访问/scheduler/start接口，启动调度器，可以看到如下日志内容：

13:53:15.010 [myScheduler-1] [INFO ] o.zp.notes.spring.scheduler.DemoTask.run - call DemoTask.run

13:53:20.003 [myScheduler-1] [INFO ] o.zp.notes.spring.scheduler.DemoTask.run - call DemoTask.run

13:53:25.004 [myScheduler-2] [INFO ] o.zp.notes.spring.scheduler.DemoTask.run - call DemoTask.run

13:53:30.005 [myScheduler-1] [INFO ] o.zp.notes.spring.scheduler.DemoTask.run - call DemoTask.run

# @Scheduler的使用方法

Spring的调度器一个很大的亮点在于@Scheduler注解，这可以省去很多繁琐的配置。

## 启动注解

使用@Scheduler注解先要使用<task:annotation-driven>

启动注解开关。

例：

*<?***xml version="1.0" encoding="UTF-8"***?>*<**beans xmlns="http://www.springframework.org/schema/beans"   
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xmlns:mvc="http://www.springframework.org/schema/mvc"   
 xmlns:task="http://www.springframework.org/schema/task"  
 xsi:schemaLocation="http://www.springframework.org/schema/beans  
 http://www.springframework.org/schema/beans/spring-beans-3.1.xsd  
 http://www.springframework.org/schema/mvc  
 http://www.springframework.org/schema/mvc/spring-mvc-3.1.xsd  
 http://www.springframework.org/schema/task  
 http://www.springframework.org/schema/task/spring-task-3.1.xsd"**>  
<**mvc:annotation-driven**/>  
  
 <**task:annotation-driven executor="myExecutor" scheduler="myScheduler"**/>  
 <**task:executor id="myExecutor" pool-size="5"**/>  
 <**task:scheduler id="myScheduler" pool-size="10"**/>  
</**beans**>

## @Scheduler定义触发条件

例：使用fixedDelay指定触发条件为每5000毫秒执行一次。注意：必须在上一次调度成功后的5000秒才能执行。

*@Scheduled(fixedDelay=5000)*

**public** **void** doSomething() {

*// something that should execute periodically*

}

例：使用fixedRate指定触发条件为每5000毫秒执行一次。注意：无论上一次调度是否成功，5000秒后必然执行。

*@Scheduled(fixedRate=5000)*

**public** **void** doSomething() {

*// something that should execute periodically*

}

例：使用initialDelay指定方法在初始化1000毫秒后才开始调度。

*@Scheduled(initialDelay=1000, fixedRate=5000)*

**public** **void** doSomething() {

*// something that should execute periodically*

}

例：使用cron表达式指定触发条件为每5000毫秒执行一次。cron规则和Quartz中的cron规则一致。

*@Scheduled(cron="\*/5 \* \* \* \* MON-FRI")*

**public** **void** doSomething() {

*// something that should execute on weekdays only*

}

## 完整范例

1. 启动注解开关，并定义调度器和执行器

*<?***xml version="1.0" encoding="UTF-8"***?>*<**beans xmlns="http://www.springframework.org/schema/beans"   
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xmlns:mvc="http://www.springframework.org/schema/mvc"   
 xmlns:task="http://www.springframework.org/schema/task"  
 xsi:schemaLocation="http://www.springframework.org/schema/beans  
 http://www.springframework.org/schema/beans/spring-beans-3.1.xsd  
 http://www.springframework.org/schema/mvc  
 http://www.springframework.org/schema/mvc/spring-mvc-3.1.xsd  
 http://www.springframework.org/schema/task  
 http://www.springframework.org/schema/task/spring-task-3.1.xsd"**>  
<**mvc:annotation-driven**/>  
  
 <**task:annotation-driven executor="myExecutor" scheduler="myScheduler"**/>  
 <**task:executor id="myExecutor" pool-size="5"**/>  
 <**task:scheduler id="myScheduler" pool-size="10"**/>  
</**beans**>

1. 使用@Scheduler注解来修饰一个要调度的方法

下面的例子展示了@Scheduler注解定义触发条件的不同方式。

**import** org.slf4j.Logger;  
**import** org.slf4j.LoggerFactory;  
**import** org.springframework.scheduling.annotation.Scheduled;  
**import** org.springframework.stereotype.Component;  
  
**import** java.text.SimpleDateFormat;  
**import** java.util.Date;  
  
*/\*\*  
 \** ***@title*** *ScheduledTasks  
 \** ***@description*** *使用@Scheduler注解调度任务范例  
 \** ***@author*** *Vicotr Zhang  
 \** ***@date*** *2016年8月31日  
 \*/*@Component  
**public class** ScheduledMgr {  
 **private final** SimpleDateFormat **dateFormat** = **new** SimpleDateFormat(**"yyyy-MM-dd HH:mm:ss"**);  
  
 **final** Logger **logger** = LoggerFactory.*getLogger*(**this**.getClass());  
  
 */\*\*  
 \* 构造函数中打印初始化时间  
 \*/* **public** ScheduledMgr() {  
 **logger**.info(**"Current time: {}"**, **dateFormat**.format(**new** Date()));  
 }  
  
 */\*\*  
 \* fixedDelay属性定义调度间隔时间。调度需要等待上一次调度执行完成。  
 \*/* @Scheduled(fixedDelay = 5000)  
 **public void** testFixedDelay() **throws** Exception {  
 Thread.*sleep*(6000);  
 **logger**.info(**"Current time: {}"**, **dateFormat**.format(**new** Date()));  
 }  
  
 */\*\*  
 \* fixedRate属性定义调度间隔时间。调度不等待上一次调度执行完成。  
 \*/* @Scheduled(fixedRate = 5000)  
 **public void** testFixedRate() **throws** Exception {  
 Thread.*sleep*(6000);  
 **logger**.info(**"Current time: {}"**, **dateFormat**.format(**new** Date()));  
 }  
  
 */\*\*  
 \* initialDelay属性定义初始化后的启动延迟时间  
 \*/* @Scheduled(initialDelay = 1000, fixedRate = 5000)  
 **public void** testInitialDelay() **throws** Exception {  
 Thread.*sleep*(6000);  
 **logger**.info(**"Current time: {}"**, **dateFormat**.format(**new** Date()));  
 }  
  
 */\*\*  
 \* cron属性支持使用cron表达式定义触发条件  
 \*/* @Scheduled(cron = **"0/5 \* \* \* \* ?"**)  
 **public void** testCron() **throws** Exception {  
 Thread.*sleep*(6000);  
 **logger**.info(**"Current time: {}"**, **dateFormat**.format(**new** Date()));  
 }  
}

我刻意设置触发方式的间隔都是5s，且方法中均有Thread.sleep(6000);语句。从而确保方法在下一次调度触发时间点前无法完成执行，来看一看各种方式的表现吧。

启动spring项目后，spring会扫描@Component注解，然后初始化ScheduledMgr。

接着，spring会扫描@Scheduler注解，初始化调度器。调度器在触发条件匹配的情况下开始工作，输出日志。

截取部分打印日志来进行分析。

10:58:46.479 [localhost-startStop-1] [INFO ] o.z.n.s.scheduler.ScheduledTasks.<init> - Current time: 2016-08-31 10:58:46

10:58:52.523 [myScheduler-1] [INFO ] o.z.n.s.scheduler.ScheduledTasks.testFixedRate - Current time: 2016-08-31 10:58:52

10:58:52.523 [myScheduler-3] [INFO ] o.z.n.s.scheduler.ScheduledTasks.testFixedDelay - Current time: 2016-08-31 10:58:52

10:58:53.524 [myScheduler-2] [INFO ] o.z.n.s.scheduler.ScheduledTasks.testInitialDelay - Current time: 2016-08-31 10:58:53

10:58:55.993 [myScheduler-4] [INFO ] o.z.n.s.scheduler.ScheduledTasks.testCron - Current time: 2016-08-31 10:58:55

10:58:58.507 [myScheduler-1] [INFO ] o.z.n.s.scheduler.ScheduledTasks.testFixedRate - Current time: 2016-08-31 10:58:58

10:58:59.525 [myScheduler-5] [INFO ] o.z.n.s.scheduler.ScheduledTasks.testInitialDelay - Current time: 2016-08-31 10:58:59

10:59:03.536 [myScheduler-3] [INFO ] o.z.n.s.scheduler.ScheduledTasks.testFixedDelay - Current time: 2016-08-31 10:59:03

10:59:04.527 [myScheduler-1] [INFO ] o.z.n.s.scheduler.ScheduledTasks.testFixedRate - Current time: 2016-08-31 10:59:04

10:59:05.527 [myScheduler-4] [INFO ] o.z.n.s.scheduler.ScheduledTasks.testInitialDelay - Current time: 2016-08-31 10:59:05

10:59:06.032 [myScheduler-2] [INFO ] o.z.n.s.scheduler.ScheduledTasks.testCron - Current time: 2016-08-31 10:59:06

10:59:10.534 [myScheduler-9] [INFO ] o.z.n.s.scheduler.ScheduledTasks.testFixedRate - Current time: 2016-08-31 10:59:10

10:59:11.527 [myScheduler-10] [INFO ] o.z.n.s.scheduler.ScheduledTasks.testInitialDelay - Current time: 2016-08-31 10:59:11

10:59:14.524 [myScheduler-4] [INFO ] o.z.n.s.scheduler.ScheduledTasks.testFixedDelay - Current time: 2016-08-31 10:59:14

10:59:15.987 [myScheduler-6] [INFO ] o.z.n.s.scheduler.ScheduledTasks.testCron - Current time: 2016-08-31 10:59:15

构造方法打印一次，时间点在10:58:46。

testFixedRate打印四次，每次间隔6秒。说明，**fixedRate不等待上一次调度执行完成，在间隔时间达到时立即执行**。

testFixedDelay打印三次，每次间隔大于6秒，且时间不固定。说明，**fixedDelay等待上一次调度执行成功后，开始计算间隔时间，再执行**。

testInitialDelay第一次调度时间和构造方法调度时间相隔7秒。说明，**initialDelay在初始化后等待指定的延迟时间才开始调度**。

testCron打印三次，时间间隔并非5秒或6秒，显然，**cron等待上一次调度执行成功后，开始计算间隔时间，再执行**。

此外，可以从日志中看出，打印日志的线程最多只有10个，说明2.1中的调度器线程池配置生效。

# 参考

Spring Framework官方文档：

http://docs.spring.io/spring/docs/current/spring-framework-reference/htmlsingle/