**Spring Task**

**快速入门**

传智播客.黑马程序员

# 课程目标

目标1: 了解定时任务和Spring Task

目标2: 会进行开发环境的搭建(Jar包的导入和配置文件)

目标3: 会使用纯XML配置的方式实现定时任务

目标4: 会使用全注解的方式实现定时任务

目标5: 会在Spring Boot中实现定时任务

# Spring Task简介

在实际开发中, 根据业务需求, 我们经常需要处理一些定时任务, 例如:每天凌晨备份一下数据, 每月清空一次日志, 页面广告一周之后过期, 每三个月清空一下cookie等等。目前最常见的实现定时任务的方式有三种:

1. Java自带的java.util.Timer类，这个类允许你调度一个java.util.TimerTask任务。使用这种方式可以让程序按照某一个频度或间隔运行，但不能在指定时间运行, 一般用的较少。

2. 使用Quartz，这是一个功能比较强大的的开源工具，可以让程序在指定时间运行，也可以按照某一个频度或间隔运行，配置起来稍显复杂。

3. Spring 3.0以后自带task调度工具，可以将它看成一个轻量级的Quartz，而且使用起来比Quartz更加简单方便, 我们这里重点讲解这种方式。

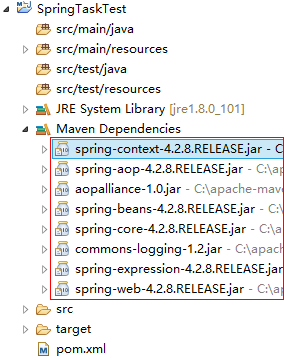
# 2. 开发环境准备

## 2.1 创建项目并导入jar包

创建一个Java Web项目SpringTaskTest1, 然后导入jar包, 如果你是Maven项目, 就在pom文件中引入spring-context和spring-web的坐标即可, 如下图所示:



Spring Task的主要API存在于spring-context包中, 引入后的效果如下图所示:



这是进行Spring Task开发所需的最少,最基本的jar包, 注意: spring必须是3.0以上的版本。

## 2.2 配置文件

要使用spring进行开发需要在web.xml中进行spring监听器的配置, 如下图所示:



该配置可以保证启动服务器时去解析spring配置文件,并加载spring容器

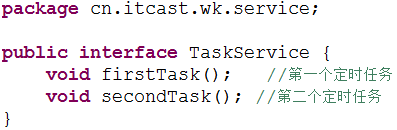
# 3. 如何使用Spring Task

使用spring框架进行定时任务开发, 无非就是两种方式: 纯XML配置方式和全注解方式, 接下来我们会分别介绍这两种方式。

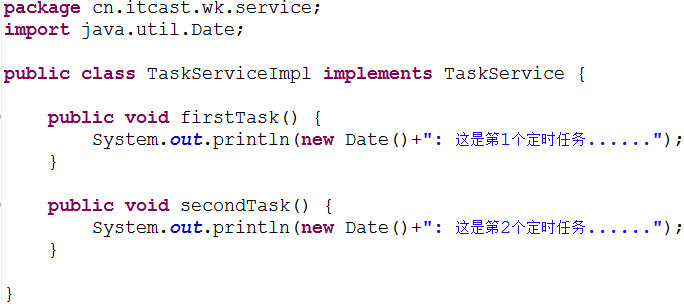
## 3.1 纯XML配置方式

### 3.1.1 简单定时任务

定时任务中的业务功能我们按照常规的处理方式放到业务层实现, 所以首先编写一个业务层接口TaskService, 如下图所示:



接下来编写该接口的实现类TaskServiceImpl, 在该类中实现两个定时任务, 业务功能上做了简化处理, 只是往控制台输出一句话, 如下图所示:

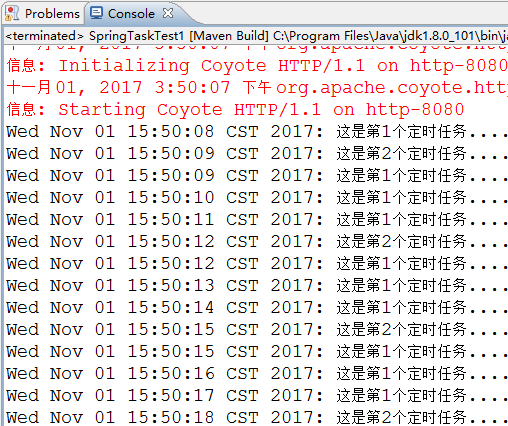


两个定时任务中的业务功能已经实现, 接下来如何运行这两个定时任务呢? 我们需要在applicationContext.xml中进行配置, 如下图所示:



* 由于我们需要使用task标签,所以需要在配置文件的头部引入task的命名空间和xsd文件
* 每个定时任务用<task:scheduled>标签定义,<task:scheduled-tasks>中可以包含多个定时任务, 这些定时任务不分先后顺序
* <task:scheduled>标签的ref属性用来设置一个bean对象, method属性用来设置bean对象中的某个方法名, initial-delay属性用来设置当tomcat启动完毕后延迟多少毫秒开始运行定时任务, fixed-delay属性用来设置每隔多少毫秒运行一次定时任务,即定时任务的频率或间隔, 注意：initial-delay和fixed-delay属性只有在spring 3.2以上版本才有效

我们在spring配置文件中定义了两个任务, 第一个任务在tomcat启动完毕后延迟1秒钟开始运行,之后每隔1秒运行一次; 第二个任务在tomcat启动完毕后延迟2秒钟开始运行,之后每隔3秒运行一次, 具体效果如下图所示:



### 3.1.2 复杂定时任务

上面我们演示的是最简单的定时任务,这种任务有着固定的频率或间隔, 当然在实际开发中我们也会碰到定时规则比较复杂的情况, 这时候我们就需要用到强大的cron表达式了。

cron表达式是一个字符串, 用来定义复杂的定时规则, 由七部分组成, 每部分中间用空格隔开, 每部分的含义如下表所示:

|  |  |  |
| --- | --- | --- |
| **组成部分** | **含义** | **取值范围** |
| 第一部分 | Seconds (秒) | 0－59 |
| 第二部分 | Minutes(分) | 0－59 |
| 第三部分 | Hours(时) | 0-23 |
| 第四部分 | Day-of-Month(天) | 1-31 |
| 第五部分 | Month(月) | 0-11或JAN-DEC |
| 第六部分 | Day-of-Week(星期) | 1-7(1表示星期日)或SUN-SAT |
| 第七部分 | Year(年) 可选 | 1970-2099 |

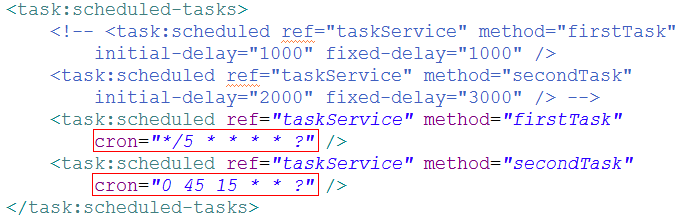
另外, cron表达式还可以包含一些特殊符号来定义更加灵活的定时规则, 如下表所示:

|  |  |
| --- | --- |
| **符号** | **含义** |
| ? | 表示不确定的值,任意的一天 |
| \* | 表示整个时间段 |
| , | 设置多个值,例如”26,29,33”表示在26分,29分和33分各自运行一次任务 |
| - | 设置取值范围,例如”5-20”，表示从5分到20分钟每分钟运行一次任务 |
| / | 设置频率或间隔,如"1/15"表示从1分开始,每隔15分钟运行一次任务 |
| L | 用于每月，或每周，表示每月的最后一天，或每个月的最后星期几,例如"6L"表示"每月的最后一个星期五" |
| W | 表示离给定日期最近的工作日,例如"15W"放在每月（day-of-month）上表示"离本月15日最近的工作日" |
| # | 表示该月第几个周X。例如”6#3”表示该月第3个周五 |

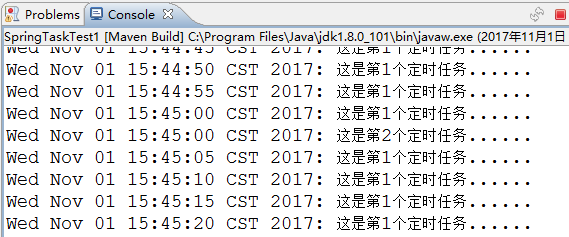
为了让大家更熟悉cron表达式的用法, 接下来我们给大家列举了大量的例子, 如下表所示:

|  |  |
| --- | --- |
| **cron表达式** | **含义** |
| \*/5 \* \* \* \* ? | 每隔5秒运行一次任务 |
| 0 0 23 \* \* ? | 每天23点运行一次任务 |
| 0 0 1 1 \* ? | 每月1号凌晨1点运行一次任务 |
| 0 0 23 L \* ? | 每月最后一天23点运行一次任务 |
| 0 26,29,33 \* \* \* ? | 在26分、29分、33分运行一次任务 |
| 0 0 0,13,18,21 \* \* ? | 每天的0点、13点、18点、21点都运行一次任务 |
| 0 0/30 9-17 \* \* ? | 朝九晚五工作时间内每半小时运行一次任务 |
| 0 15 10 ? \* 6#3 | 每月的第三个星期五上午10:15运行一次任务 |

接下来我们给上面例子中的两个任务分别设置两个不同的cron表达式, 第一个任务每隔5秒钟运行一次, 第二个任务是在每天的15:45运行一次, 如下图所示:



其他地方的代码不做任何修改, 运行效果如下图所示:



## 3.2 全注解方式

除了采用XML配置的方式可以实现定时任务之外, spring也支持全注解的方式来实现定时任务。 为了跟上面的例子区分开, 接下来我们单独再创建一个项目SpringTaskTest2来演示。 创建项目并导入Jar包, 以及配置web.xml的过程跟上面的例子一模一样, 这里不再演示。

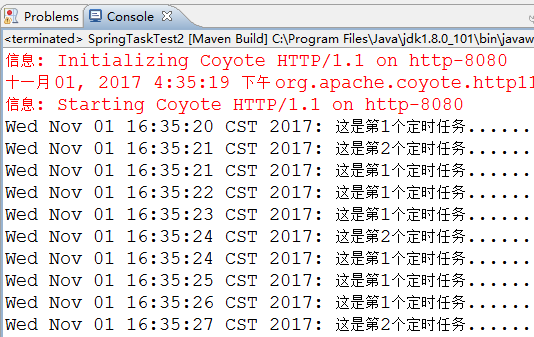
我们可以直接在业务层的实现类中使用@Service注解, 该注解用到类上表示该类的对象由spring负责创建和管理, 等价于applicationContext.xml中的<bean>配置, @Scheduled注解用到方法上用于设置定时规则, 等价于applicationContext.xml中的<task:scheduled>配置, 具体代码如下图所示:



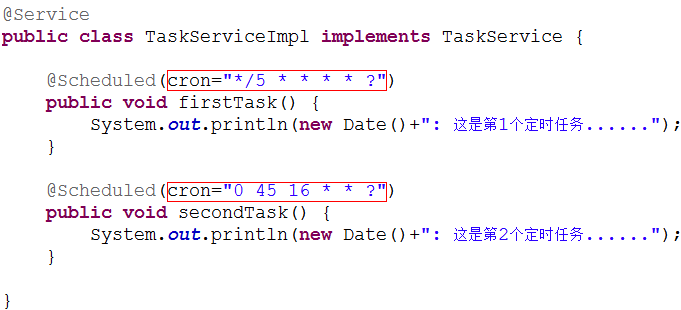
当然, 要想让这些注解起作用, 我们还必须在applicationContext.xml中做如下配置:



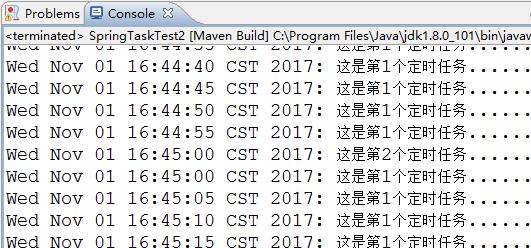
项目运行效果如下图所示:



@Scheduled注解也支持cron表达式, 接下来我们修改一下实现类中的代码, 让第一个任务每隔5秒钟运行一次, 第二个任务在每天的16:45运行一次, 如下图所示:



项目运行效果如下图所示:



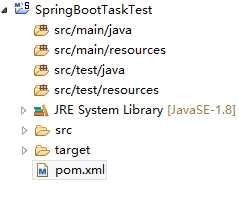
# 4. 在Spring Boot中实现定时任务

随着项目功能以及业务逻辑的日益复杂，Spring的配置文件以及Bean的依赖关系也日益复杂。从Spring 3.0开始，Spring团队就在开发过程中大量使用“约定优先配置”的思想来摆脱Spring框架中各类繁复纷杂的配置。

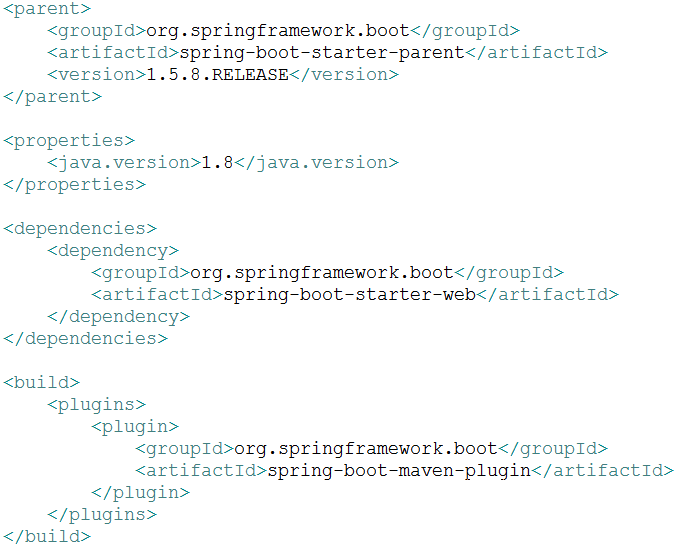
Spring Boot就是在这样的一个背景下产生的，它是为了简化Spring应用的创建、运行、调试、部署而出现的，使用它可以做到专注于Spring应用的开发，而无需过多关注XML的配置。同时它集成了大量常用的第三方库（例如Jackson, JDBC, Mongo, Redis, Mail等等），并且在Spring Boot中几乎可以零配置的使用这些第三方库。

目前有越来越多的企业去使用Spring Boot, 所以接下来回到我们的本次主题上(定时任务), 开始介绍如何在Spring Boot中实现定时任务。

* 新建一个Maven Project, 打包方式为war, 如下图所示:

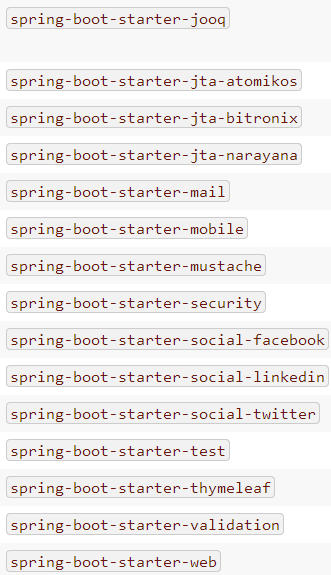
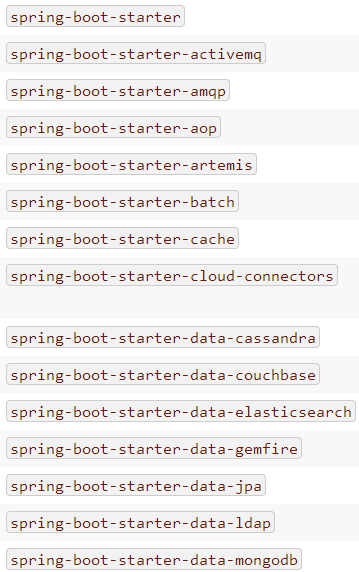


* 在pom.xml中引入Spring Boot, 如下图所示:

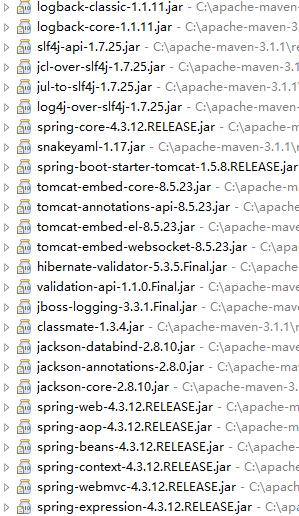


在项目的pom文件中一般建议继承spring-boot-starter-parent, 因为spring-boot-starter-parent中已经把spring的核心包,日志以及编码格式等通用配置设置好了。

另外, spring boot根据不同的开发环境把jar包进行了分类分模块管理, 如下图所示:

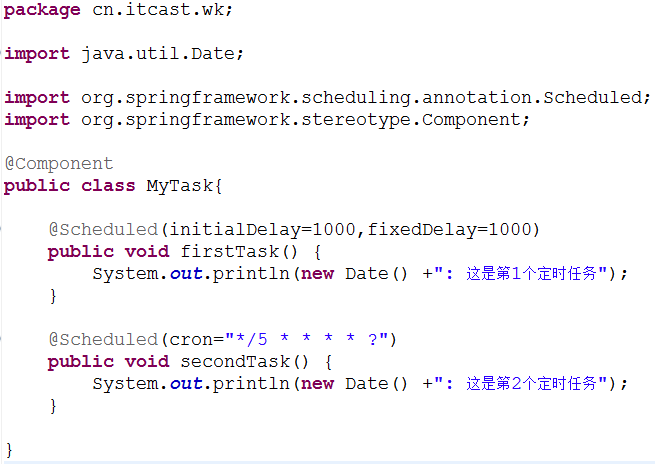


我们在项目中只需要实现定时任务, 因此只需要具备最基本的web开发的环境即可, 所以我们在项目的pom文件中引入了spring-boot-strater-web的依赖, 该依赖具体会在项目中引入如下jar包:



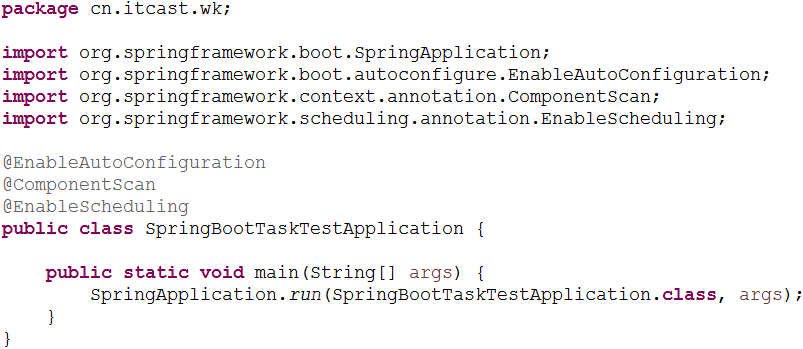
这些Jar包中包含了spring的基本包,还有tomcat服务器,能够满足基本的web开发需求

* 在项目的cn.itcast.wk包中新建一个任务类MyTask, 并编写两个定时任务方法, 具体代码如下图所示:



我们给MyTask类配置了@Component注解, 以便交给spring去管理, 在该类的两个方法上使用了@Scheduled注解来配置定时规则, 其中第二个方法使用了cron表达式

* Spring boot能以普通Java程序的形式来启动, 我们编写一个启动类, 如下图所示:

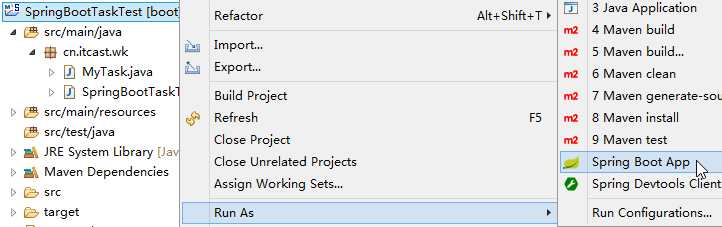


Spring boot之所以能以普通Java程序的形式来启动并运行, 主要是依靠@EnableAutoConfiguration和 @ComponentScan这两个注解:

@EnableAutoConfiguration: 开启自动配置, 让 Spring Boot 根据应用所声明的依赖来对 Spring 框架进行自动配置

@ComponentScan: 开启自动扫描默认包下的全部标有 @Component、@Service、@Repository、@Controller注解的类，并注册成bean交由spring管理

* 在右键菜单中选择[Run As]-[Spring Boot App]即可运行, 如下图所示:



具体运行效果如下图所示:

