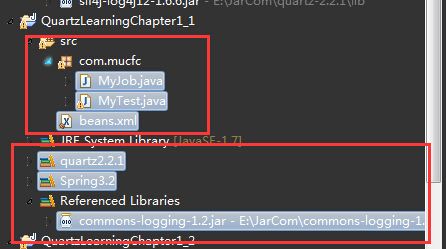
**[Spring4+Quqrtz2.2.1实现定时实例](http://www.itnose.net/detail/6259212.html)**

2015-04-29 15:07

本文示例工程下载   
一、工程创建

1、新建一个工程，导入相应的包Spring3.2.9+Quqrtz2.2.1+commons-logging



集成起来比较简单,对线程无需任何理解，Spring手册上也有清楚的说明。只需写一个POJO，其余的都是配置，而且若使用CronTriggerBean，可以配置很复杂的任务调度。

编写一个普通的JAVA类

根据业务，写一个简单的JAVA类，和普通的类没有区别。并配置到Spring配置文件上中。例：

package com.mucfc;

import java.text.SimpleDateFormat;

import java.util.Date;

/\*\*

\*事件类

\*作者 林炳文（ling20081005@126.com 博客：http://blog.csdn.net/evankaka）

\*时间 2015.4.29

\*/

public class MyJob {

public void run(){

System.out.println("Hello Spring with Quzrtz "+

new SimpleDateFormat("yyyy-MM-dd HH:mm:ss ").format(new Date()));

}

}

二、Spring中配置

1、创建job的bean实例

<!-- 配置Job的bean -->

<bean id="myJob" class="com.mucfc.MyJob" />

2.使用MethodInvokingJobDetailFactoryBean建立任务

只需要配置即可，配置信息示例如下：

<!-- 配置jobDetail -->

<bean id="myJobDetail"

class="org.springframework.scheduling.quartz.MethodInvokingJobDetailFactoryBean">

<!-- 用到的Job实例 -->

<property name="targetObject">

<ref bean="myJob"/>

</property>

<!-- Job实例中的方法 -->

<property name="targetMethod">

<value>run</value>

</property>

</bean>

说明：

|  |  |
| --- | --- |
| targetObject | 目标对象，即需要定时执行的POJO |
| targetMethod | 目标方法，即需要定时执行的POJO方法 |
| concurrent | 默认情况下，Quartz Jobs是无状态的，可能导致jobs之间互相的影响，如果你为相同的JobDetail指定两个Trigger,很可能当第一个job完成之前，第二个job就开始了。如果Jobdetail实现了Stateful接口，就不会发生这样的事情。第二个job将不会在第一个job完成之前开始。为了使得jobs不并发运行，设置concurrent标记为false |

3、使用CronTriggerBean建立规则，调度任务

<!-- 配置触发器Trigger -->

<bean id="myTrigger" class="org.springframework.scheduling.quartz.CronTriggerFactoryBean">

<property name="name" value="work\_default\_name"/>

<property name="group" value="work\_default"/>

<property name="jobDetail">

<ref bean="myJobDetail"/>

</property>

<property name="cronExpression">

<!-- 每5秒执行一次 -->

<value>0/5 \* \* \* \* ?</value>

</property>

</bean>

说明：

|  |  |
| --- | --- |
| jobDetail | 任务详情，即所需要调度的任务 |
| cronExpression | 调用规则，即什么时候调用。详细说明见附录1 |

4.使用SchedulerFactoryBean包装任务

<!-- 配置scheduler工厂 -->

<bean id="scheduler" class="org.springframework.scheduling.quartz.SchedulerFactoryBean">

<property name="triggers">

<list>

<ref bean="myTrigger"/>

</list>

</property>

</bean>

说明：

|  |  |
| --- | --- |
| triggers | 触发器，调用哪些任务 |
| autoStartup | 是否自动启动，该Bean被初始化后是否自动启动定时任务。  Set whether to automatically start the scheduler after initialization. |
| schedulerName | 给这个计划设定一个名称。  Set the name of the Scheduler to fetch from the SchedulerFactory. |

整个配置文件如下：

<?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:p="http://www.springframework.org/schema/p"

xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

<!-- 配置Job的bean -->

<bean id="myJob" class="com.mucfc.MyJob" />

<!-- 配置jobDetail -->

<bean id="myJobDetail"

class="org.springframework.scheduling.quartz.MethodInvokingJobDetailFactoryBean">

<!-- 用到的Job实例 -->

<property name="targetObject">

<ref bean="myJob"/>

</property>

<!-- Job实例中的方法 -->

<property name="targetMethod">

<value>run</value>

</property>

</bean>

<!-- 配置触发器Trigger -->

<bean id="myTrigger" class="org.springframework.scheduling.quartz.CronTriggerFactoryBean">

<property name="name" value="work\_default\_name"/>

<property name="group" value="work\_default"/>

<property name="jobDetail">

<ref bean="myJobDetail"/>

</property>

<property name="cronExpression">

<!-- 每5秒执行一次 -->

<value>0/5 \* \* \* \* ?</value>

</property>

</bean>

<!-- 配置scheduler工厂 -->

<bean id="scheduler" class="org.springframework.scheduling.quartz.SchedulerFactoryBean">

<property name="triggers">

<list>

<ref bean="myTrigger"/>

</list>

</property>

</bean>

</beans>

三、使用

package com.mucfc;

import org.quartz.Scheduler;

import org.quartz.SchedulerException;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MyTest {

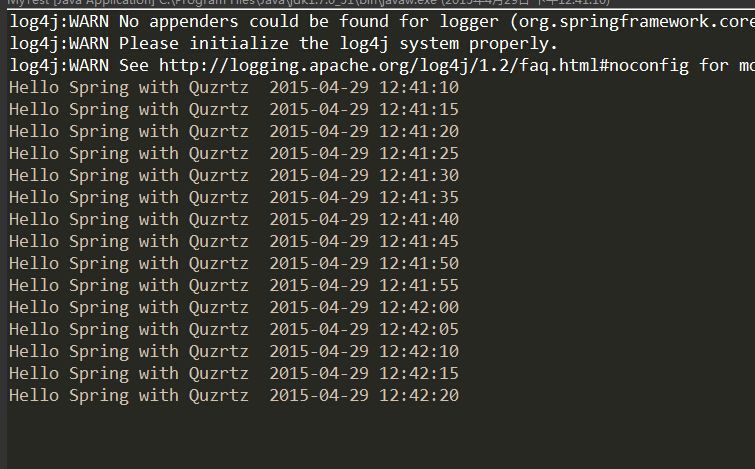
public static void main(String[] args) {

ApplicationContext context=new ClassPathXmlApplicationContext("beans.xml");

}

}

我们在学习Spring中知道，ApplicationContext在加载的时候它会自动生成bean的实例，也就是说这个运行后定时器就会开始执行了，结果如下：



若是不想让它一加载就启动，就可以用beanfactory来加载xml文件，然后再手动打开

package com.mucfc;

import org.quartz.Scheduler;

import org.quartz.SchedulerException;

import org.springframework.beans.factory.BeanFactory;

import org.springframework.beans.factory.xml.XmlBeanFactory;

import org.springframework.core.io.ClassPathResource;

import org.springframework.core.io.Resource;

public class MyTest {

public static void main(String[] args) {

//ApplicationContext context=new ClassPathXmlApplicationContext("beans.xml");

Resource res = new ClassPathResource("beans.xml");

BeanFactory bf = new XmlBeanFactory(res);

Scheduler scheduler=(Scheduler)bf.getBean("scheduler");

try {

scheduler.start();

} catch (SchedulerException e) {

e.printStackTrace();

}

}

}

**核心概念**

**1、scheduler是一个计划调度器容器，容器里面可以盛放众多的JobDetail和trigger，当容器启动后，里面的每个JobDetail都会根据trigger按部就班自动去执行。**

**2、JobDetail是一个可执行的工作，它本身可能是有状态的。**

**3、Trigger代表一个调度参数的配置，什么时候去调。**

**4、当JobDetail和Trigger在scheduler容器上注册后，形成了装配好的作业（JobDetail和Trigger所组成的一对儿），就可以伴随容器启动而调度执行了。**

**5、scheduler是个容器，容器中有一个线程池，用来并行调度执行每个作业，这样可以提高容器效率。**

**四、cron 表达式的格式**

Quartz cron 表达式的格式十分类似于 UNIX cron 格式，但还是有少许明显的区别。区别之一就是 Quartz 的格式向下支持到秒级别的计划，而 UNIX cron 计划仅支持至分钟级。许多我们的触发计划要基于秒级递增的(例如，每45秒)，因此这是一个非常好的差异。  
  
在 UNIX cron 里，要执行的作业（或者说命令）是存放在 cron 表达式中的，在第六个域位置上。Quartz 用 cron 表达式存放执行计划。引用了 cron 表达式的 CronTrigger 在计划的时间里会与 job 关联上。  
  
另一个与 UNIX cron 表达式的不同点是在表达式中支持域的数目。UNIX 给出五个域(分、时、日、月和周)，Quartz 提供七个域。表 5.1 列出了 Quartz cron 表达式支持的七个域。

**Quartz Cron 表达式支持到七个域**

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **是否必须** | **允许值** | **特殊字符** |
| 秒 | 是 | 0-59 | , - \* / |
| 分 | 是 | 0-59 | , - \* / |
| 时 | 是 | 0-23 | , - \* / |
| 日 | 是 | 1-31 | , - \* ? / L W C |
| 月 | 是 | 1-12 或 JAN-DEC | , - \* / |
| 周 | 是 | 1-7 或 SUN-SAT | , - \* ? / L C # |
| 年 | 否 | 空 或 1970-2099 | , - \* / |

月份和星期的名称是不区分大小写的。FRI 和 fri 是一样的。   
  
域之间有空格分隔，这和 UNIX cron 一样。无可争辩的，我们能写的最简单的表达式看起来就是这个了：   
  
\* \* \* ? \* \*   
  
这个表达会每秒钟(每分种的、每小时的、每天的)激发一个部署的 job。   
  
**·理解特殊字符**   
  
同 UNIX cron 一样，Quartz cron 表达式支持用特殊字符来创建更为复杂的执行计划。然而，Quartz 在特殊字符的支持上比标准 UNIX cron 表达式更丰富了。   
  
**\* 星号**   
  
使用星号(\*) 指示着你想在这个域上包含所有合法的值。例如，在月份域上使用星号意味着每个月都会触发这个 trigger。  
  
表达式样例：   
  
0 \* 17 \* \* ?   
  
意义：每天从下午5点到下午5:59中的每分钟激发一次 trigger。它停在下午 5:59 是因为值 17 在小时域上，在下午 6 点时，小时变为 18 了，也就不再理会这个 trigger，直到下一天的下午5点。   
  
在你希望 trigger 在该域的所有有效值上被激发时使用 \* 字符。   
  
**? 问号**   
  
? 号只能用在日和周域上，但是不能在这两个域上同时使用。你可以认为 ? 字符是 "我并不关心在该域上是什么值。" 这不同于星号，星号是指示着该域上的每一个值。? 是说不为该域指定值。   
  
不能同时这两个域上指定值的理由是难以解释甚至是难以理解的。基本上，假定同时指定值的话，意义就会变得含混不清了：考虑一下，如果一个表达式在日域上有值11，同时在周域上指定了 WED。那么是要 trigger 仅在每个月的11号，且正好又是星期三那天被激发？还是在每个星期三的11号被激发呢？要去除这种不明确性的办法就是不能同时在这两个域上指定值。   
  
只要记住，假如你为这两域的其中一个指定了值，那就必须在另一个字值上放一个 ?。   
  
表达式样例：   
  
0 10,44 14 ? 3 WEB   
  
意义：在三月中的每个星期三的下午 2:10 和 下午 2:44 被触发。   
  
**, 逗号**   
  
逗号 (,) 是用来在给某个域上指定一个值列表的。例如，使用值 0,15,30,45 在秒域上意味着每15秒触发一个 trigger。   
  
表达式样例：   
  
0 0,15,30,45 \* \* \* ?   
  
意义：每刻钟触发一次 trigger。   
  
**/ 斜杠**   
  
斜杠 (/) 是用于时间表的递增的。我们刚刚用了逗号来表示每15分钟的递增，但是我们也能写成这样 0/15。   
  
表达式样例：   
  
0/15 0/30 \* \* \* ?   
  
意义：在整点和半点时每15秒触发 trigger。   
  
**- 中划线**   
  
中划线 (-) 用于指定一个范围。例如，在小时域上的 3-8 意味着 "3,4,5,6,7 和 8 点。"  域的值不允许回卷，所以像 50-10 这样的值是不允许的。   
  
表达式样例：   
  
0 45 3-8 ? \* \*   
  
意义：在上午的3点至上午的8点的45分时触发 trigger。   
  
**L 字母**  
L 说明了某域上允许的最后一个值。它仅被日和周域支持。当用在日域上，表示的是在月域上指定的月份的最后一天。例如，当月域上指定了 JAN 时，在日域上的 L 会促使 trigger 在1月31号被触发。假如月域上是 SEP，那么 L 会预示着在9月30号触发。换句话说，就是不管指定了哪个月，都是在相应月份的时最后一天触发 trigger。   
  
表达式 0 0 8 L \* ? 意义是在每个月最后一天的上午 8:00 触发 trigger。在月域上的 \* 说明是 "每个月"。   
  
当 L 字母用于周域上，指示着周的最后一天，就是星期六 (或者数字7)。所以如果你需要在每个月的最后一个星期六下午的 11:59 触发 trigger，你可以用这样的表达式 0 59 23 ? \* L。   
  
当使用于周域上，你可以用一个数字与 L 连起来表示月份的最后一个星期 X。例如，表达式 0 0 12 ? \* 2L 说的是在每个月的最后一个星期一触发 trigger。

|  |
| --- |
| **不要让范围和列表值与 L 连用**  虽然你能用星期数(1-7)与 L 连用，但是不允许你用一个范围值和列表值与 L 连用。这会产生不可预知的结果。 |

**W 字母**   
  
W 字符代表着平日 (Mon-Fri)，并且仅能用于日域中。它用来指定离指定日的最近的一个平日。大部分的商业处理都是基于工作周的，所以 W 字符可能是非常重要的。例如，日域中的 15W 意味着 "离该月15号的最近一个平日。" 假如15号是星期六，那么 trigger 会在14号(星期四)触发，因为距15号最近的是星期一，这个例子中也会是17号（译者Unmi注：不会在17号触发的，如果是15W，可能会是在14号(15号是星期六)或者15号(15号是星期天)触发，也就是只能出现在邻近的一天，如果15号当天为平日直接就会当日执行）。W 只能用在指定的日域为单天，不能是范围或列表值。   
  
**# 井号**   
  
# 字符仅能用于周域中。它用于指定月份中的第几周的哪一天。例如，如果你指定周域的值为 6#3，它意思是某月的第三个周五 (6=星期五，#3意味着月份中的第三周)。另一个例子 2#1 意思是某月的第一个星期一 (2=星期一，#1意味着月份中的第一周)。注意，假如你指定 #5，然而月份中没有第 5 周，那么该月不会触发。

示例：

表达式意义

"0 0 12 \* \*?" 每天中午12点触发

"0 15 10 ? \*\*" 每天上午10:15触发

"0 15 10 \* \*?" 每天上午10:15触发

"0 15 10 \* \* ?\*" 每天上午10:15触发

"0 15 10 \* \* ?2005" 2005年的每天上午10:15触发

"0 \* 14 \* \*?" 在每天下午2点到下午2:59期间的每1分钟触发

"0 0/5 14 \* \*?" 在每天下午2点到下午2:55期间的每5分钟触发

"0 0/5 14,18 \*\* ?" 在每天下午2点到2:55期间和下午6点到6:55期间的每5分钟触发

"0 0-5 14 \* \*?" 在每天下午2点到下午2:05期间的每1分钟触发

"0 10,44 14 ? 3WED" 每年三月的星期三的下午2:10和2:44触发

"0 15 10 ? \*MON-FRI" 周一至周五的上午10:15触发

"0 15 10 15 \*?" 每月15日上午10:15触发

"0 15 10 L \*?" 每月最后一日的上午10:15触发

"0 15 10 ? \*6L" 每月的最后一个星期五上午10:15触发

"0 15 10 ? \* 6L2002-2005" 2002年至2005年的每月的最后一个星期五上午10:15触发

"0 15 10 ? \*6#3" 每月的第三个星期五上午10:15触发