**[spring 定时器](http://binfox7777.javaeye.com/blog/199775)**

**关键字: spring 定时器**

本文章取自Spring in Action   
在Spring中有两种流行配置：Java的Timer类和OpenSymphony的Quartz。   
**1.Java Timer定时**   
首先继承java.util.TimerTask类实现run方法   
import java.util.TimerTask;   
public class EmailReportTask extends TimerTask{   
    @Override   
    public void run() {   
        ...   
    }     
}   
**在Spring定义**   
<bean id="reportTimerTask" class="EmailReportTask">...</bean>   
**配置定时器**   
<bean id="scheduleReportTask" class="org.springframework.scheduling.timer.ScheduledTimerTask">   
<property name="timerTask" ref="reportTimerTask" />   
<property name="period">   
<value>86400000</value>   
</property>   
</bean>   
timerTask属性告诉ScheduledTimerTask运行哪个。86400000代表24个小时   
**启动定时器**   
Spring的TimerFactoryBean负责启动定时任务   
<bean class="org.springframework.scheduling.timer.TimerFactoryBean">   
<property name="scheduledTimerTasks">   
   <list><ref bean="scheduleReportTask"/></list>   
</property>   
</bean>   
scheduledTimerTasks里显示一个需要启动的定时器任务的列表。   
可以通过设置delay属性延迟启动   
<bean id="scheduleReportTask" class="org.springframework.scheduling.timer.ScheduledTimerTask">   
<property name="timerTask" ref="reportTimerTask" />   
<property name="period">   
<value>86400000</value>   
</property>   
<property name="delay">   
<value>3600000</value>   
</property>   
</bean>   
这个任务我们只能规定每隔24小时运行一次，无法精确到某时启动   
**2.Quartz定时器**   
首先继承QuartzJobBean类实现executeInternal方法   
import org.quartz.JobExecutionContext;   
import org.quartz.JobExecutionException;   
import org.springframework.scheduling.quartz.QuartzJobBean;   
  
public class **EmailReportJob** extends QuartzJobBean{   
protected void executeInternal(JobExecutionContext arg0)   
throws JobExecutionException {   
...   
}   
}   
**在Spring中定义**   
<bean id="reportJob" class="org.springframework.scheduling.quartz.**JobDetailBean**">   
<property name="jobClass">   
<value>EmailReportJob</value>   
</property>   
<property name="jobDataAsMap">   
    <map>   
        <entry key="courseService">   
            <ref bean="courseService"/>   
            </entry>   
    </map>   
</property>   
</bean>   
在这里我们并没有直接声明一个EmailReportJob Bean，而是声明了一个JobDetailBean。这个是Quartz的特点。JobDetailBean是Quartz的 org.quartz.JobDetail的子类，它要求通过jobClass属性来设置一个Job对象。   
使用Quartz的JobDetail中的另一个特别之处是EmailReportJob的courseService属性是间接设置的。 JobDetail的jobDataAsMap属性接受一个Map，包括设置给jobClass的各种属性，当。JobDetailBean实例化时，它 会将courseService Bean注入到EmailReportJob 的courseService 属性中。   
**启动定时器**   
Quartz的org.quartz.Trigger类描述了何时及以怎样的频度运行一个Quartz工作。Spring提供了两个触发器SimpleTriggerBean和CronTriggerBean。   
**SimpleTriggerBean**与scheduledTimerTasks类似。指定工作的执行频度，模仿scheduledTimerTasks配置   
<bean id="simpleReportTrigger" class="org.springframework.scheduling.quartz.SimpleTriggerBean">   
<property name="jobDetail" ref="reprotJob" />   
<property name="startDelay">   
<value>360000</value>   
</property>   
<property name="repeatInterval">   
    <value>86400000</value>   
</property>   
</bean>   
startDelay也是延迟1个小时启动   
**CronTriggerBean**指定工作的准确运行时间   
<bean id="cronReportTrigger" class="org.springframework.scheduling.quartz.CronTriggerBean">   
<property name="jobDetail" ref="reprotJob" />   
<property name="cronExpression">   
<value>0 0 6 \* \* ?</value>   
</property>   
</bean>   
属性cronExpression告诉何时触发。最神秘就是cron表达式：   
Linux系统的计划任务通常有cron来承担。一个cron表达式有至少6个（也可能7个）有**空格分隔**的时间元素。从左到右：   
1.秒2.分3.小时4.月份中的日期（1-31）5.月份（1-12或JAN-DEC)6.星期中的日期（1-7或SUN-SAT）7.年份（1970-2099）   
每个元素都显示的规定一个值（如6），一个区间（9-12），一个列表（9，11，13）或一个通配符（\*）。因为4和6这两个元素是互斥的，因此应该通过设置一个问号（？）来表明不想设置的那个字段，“/”如果值组合就表示重复次数（10/6表示每10秒重复6次）。   
**启动定时器**   
<bean class="org.springframework.scheduling.quartz.SchedulerFactoryBean">   
    <property name="triggers">   
       <list><ref bean="cronReportTrigger"/></list>   
    </property>   
</bean>   
triggers属性接受一组触发器。

# Spring中正确使用Quartz和CronExpression

Quartz作为企业级任务调度框架以其灵活的使用方式、强大的功能已经得到广泛应用，作为一向喜欢将业内流行的工具纳入支持的Spring自然已经内置 了对Quartz的支持，使得Quartz中最常使用的SimpleTrigger和CronTrigger的使用得到了最大简化，分别对应Spring 的org.springframework.scheduling.quartz.SimpleTriggerBean和 org.springframework.scheduling.quartz.CronTriggerBean，这两个类用起来非常方便，其中 SimpleTrigger更类似于JDK中的Timer，它只是简单的以某个时间间隔来执行某个任务而已，比较简陋，而CronTrigger功能则十 分强大，可以设定制定任务在任意指定时刻内调用，其使用Unix中的Cron Expression来制定调度策略，十分灵活，不过Cron Expression可能需要用点时间来学习，不过一旦掌握会觉得真的很不错，掌握了这两种Trigger基本上就可以应付实现大多数J2EE应用中的时 间任务调度服务了。

     下面举一个简单例子说明一下在Spring1.2.5中使用Quartz1.5的方式（所需要的包在Spring1.2.5的发行版中的lib目录下可以找到，将其拷贝到工程的类路径中）。

     首先建立两个要调度的任务，他们都必须继承自 org.springframework.scheduling.quartz.QuartzJobBean，而业务逻辑都是放在 executeInternal方法中，指定的任务逻辑实现之后，需要将其注入到一个JobDetailBean中，JobDetailBean可以看作 为一个具体任务的设置它指定了要执行的任务和执行任务的时间策略，JobDetailBean不用实现，只需要在Spring的配置文件中设置便可：

package com.peuo.albumSys.scheduling;

import java.util.Iterator;  
import java.util.List;  
import java.util.Set;

import org.quartz.JobExecutionContext;  
import org.quartz.JobExecutionException;  
import org.springframework.scheduling.quartz.QuartzJobBean;

import org.apache.log4j.Logger;

import com.peuo.albumSys.business.IAlbumSysService;  
import com.peuo.albumSys.domainobj.User;  
import com.peuo.albumSys.domainobj.Album;

/\*\*  
\* 在指定的时刻查询当前所有的用户，并显示其姓名。  
\* @author Shippo Field  
\*/  
public class ScheduledQueryUsers extends QuartzJobBean {  
     private static Logger log = Logger.getLogger(ScheduledQueryUsers.class);  
      
     private IAlbumSysService service = null;

     /\*\*  
      \* @param service 要设置的业务逻辑Bean实例，可以是已被纳入事务管理的某个Bean，这样以来QuartzJobBean内部逻辑也可以纳入事务管理。  
      \*/  
     public void setService(IAlbumSysService service) {  
         this.service = service;  
     }  
      
     /\* （非 Javadoc）  
      \* @see org.springframework.scheduling.quartz.QuartzJobBean#executeInternal(org.quartz.JobExecutionContext)  
      \*/  
     protected void executeInternal(JobExecutionContext ctx)  
             throws JobExecutionException {  
         log.debug("Now entering method executeInternal()...");  
         List users = service.findAllUser();  
         StringBuffer str = new StringBuffer();  
         for(int i = 0; i < users.size(); i++){  
             User user = (User)users.get(i);  
             str.append("\nUser[" + i + "] = " + user.getUserName());  
             Set albums = user.getAlbumSet();  
             Iterator it = albums.iterator();  
             str.append("(Albums:");  
             while(it.hasNext()){  
                 str.append(((Album)it.next()).getAlbumName()).append(";");  
             }  
         }  
         log.debug(str.toString());  
     }

}

package com.peuo.albumSys.scheduling;

import java.util.Date;

import org.quartz.JobExecutionContext;  
import org.quartz.JobExecutionException;

import org.springframework.scheduling.quartz.QuartzJobBean;

import org.apache.log4j.Logger;

import com.peuo.albumSys.business.IAlbumSysService;  
import com.peuo.albumSys.domainobj.Album;  
import com.peuo.albumSys.domainobj.Photo;

/\*\*  
\* 本程序会在指定的时刻插入照片  
\* @author Field  
\*/  
public class ScheduledAddPhoto extends QuartzJobBean {  
     private static Logger log = Logger.getLogger(ScheduledAddPhoto.class);  
      
     private IAlbumSysService service = null;  
     private static int count = 0;  
     /\* （非 Javadoc）  
      \* @see org.springframework.scheduling.quartz.QuartzJobBean#executeInternal(org.quartz.JobExecutionContext)  
      \*/  
     protected void executeInternal(JobExecutionContext arg0)  
             throws JobExecutionException {  
         Date now = new Date(System.currentTimeMillis());  
         Photo newPhoto = new Photo();  
         newPhoto.setAlbum(new Album(new Integer(1)));  
         newPhoto.setPhotoName( now.toString());  
         newPhoto.setPhotoUrl("Photo url" + ++count);  
         service.addPhoto(newPhoto);  
         log.debug("New photo added : name =" + newPhoto.getPhotoName() +"; url = " + newPhoto.getPhotoUrl());  
     }

     /\*\*  
      \* @return 当前业务逻辑实例。  
      \*/  
     public IAlbumSysService getService() {  
         return service;  
     }  
     /\*\*  
      \* @param service 要设置的业务逻辑实例。  
      \*/  
     public void setService(IAlbumSysService service) {  
         this.service = service;  
     }  
}  
然后在Spring的配置文件中分别设置一个SimpleTriggerBean和CronTriggerBean来调用上面两个任务，配置如下：

<?xml version="1.0" encoding="UTF-8"?>  
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN" "<http://www.springframework.org/dtd/spring-beans.dtd>">

<beans>

……………………………………………………………………………………………………

    <bean id="queryUserJob" class="org.springframework.scheduling.quartz.JobDetailBean">  
     <property name="jobClass">  
      <value>com.peuo.albumSys.scheduling.ScheduledQueryUsers</value>  
     </property>  
     <property name="jobDataAsMap">  
      <map>  
       <entry key="service">  
        <ref local="albumSysService"/>  
     </entry>  
      </map>  
     </property>  
    </bean>  
     
    <bean id="simpleTrigger" class="org.springframework.scheduling.quartz.SimpleTriggerBean">  
     <property name="jobDetail">  
      <ref local="queryUserJob"/>  
     </property>  
     <property name="startDelay">  
      <value>15000</value>  
     </property>  
     <property name="repeatInterval">  
      <value>60000</value>  
     </property>  
     <property name="repeatCount">  
      <value>-1</value>  
     </property>  
    </bean>  
     
<bean id="addPhotoJob" class="org.springframework.scheduling.quartz.JobDetailBean">  
   <property name="jobClass">  
    <value>com.peuo.albumSys.scheduling.ScheduledAddPhoto</value>  
   </property>  
<property name="jobDataAsMap">  
    <map>  
     <entry key="service">  
      <ref local="albumSysService" />  
     </entry>  
    </map>  
   </property>  
</bean>  
  
<bean id="cronTrigger" class="org.springframework.scheduling.quartz.CronTriggerBean">  
   <property name="jobDetail">  
    <ref local="addPhotoJob"/>  
   </property>  
   <property name="cronExpression">  
    <value>0 20 19 \* \* ?</value>  
   </property>  
</bean>  
  
    <bean id="sfb" class="org.springframework.scheduling.quartz.SchedulerFactoryBean">  
     <property name="triggers">  
      <list>  
       <ref local="simpleTrigger"/>  
     <ref local="cronTrigger"/>  
      </list>  
     </property>  
    </bean>

上面红色标出的部分可以看出，Spring对于将QuartzJob注入JobDetailBean的方式有点特殊，你需要将要注入给QuartzJob的某个属性注入到JobDetailBean的jobDataAsMap中， 这不同于Spring平时直接对某个Bean的属性直接注入的方式，这主要是因为Quartz执行一个任务（JOB或者叫作业）的时候会提供给任务一个 JobExecutionContext的任务执行上下文，从这个上下文中可以获得Job的环境设置，具体的可以看一下Quartz的API文档就明白 了，反正只要记住这里的注入方式比较特殊就行了。

以上配置文件分别设置了一个SimpleTriggerBean和CronTriggerBean，前者将会在程序执行15秒之后开始不断以间隔一 分钟的方式不断查询当前用户信息并显示出来，后者则指定每天的19：20分执行向ID为1的用户的相册内添加一张照片的任务，其Cron Expression为0 20 19 \* \* ?。

从以上可以看出，Spring大大简化了对Quartz任务时间调度框架的使用，将其完美的融合入了Spring提供的IoC容器，只是其注入方式需要注意，还有就是Cron Expression一定要掌握，这个是Quartz真正的强大之处。

网上似乎对Cron表达式的中文介绍相当少，我干脆就把Quartz中的doc翻译一下，各位需要的朋友可以快速了解一下大致用法：

一个Cron-表达式是一个由六至七个字段组成由空格分隔的字符串，其中6个字段是必须的而一个是可选的，如下：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **字段名** |  | **允许的值** |  | **允许的特殊字符** |
| 秒 |  | 0-59 |  | , - \* / |
| 分 |  | 0-59 |  | , - \* / |
| 小时 |  | 0-23 |  | , - \* / |
| 日 |  | 1-31 |  | , - \* ? / L W C |
| 月 |  | 1-12 or JAN-DEC |  | , - \* / |
| 周几 |  | 1-7 or SUN-SAT |  | , - \* ? / L C # |
| 年 (可选字段) |  | empty, 1970-2099 |  | , - \* / |

'\*' 字符可以用于所有字段，在“分”字段中设为"\*"表示"每一分钟"的含义。

'?' 字符可以用在“日”和“周几”字段. 它用来指定 '不明确的值'. 这在你需要指定这两个字段中的某一个值而不是另外一个的时候会被用到。在后面的例子中可以看到其含义。

'-' 字符被用来指定一个值的范围，比如在“小时”字段中设为"10-12"表示"10点到12点".

',' 字符指定数个值。比如在“周几”字段中设为"MON,WED,FRI"表示"the days Monday, Wednesday, and Friday".

'/' 字符用来指定一个值的的增加幅度. 比如在“秒”字段中设置为"0/15"表示"第0, 15, 30, 和 45秒"。而 "5/15"则表示"第5, 20, 35, 和 50". 在'/'前加"\*"字符相当于指定从0秒开始. 每个字段都有一系列可以开始或结束的数值。对于“秒”和“分”字段来说，其数值范围为0到59，对于“小时”字段来说其为0到23, 对于“日”字段来说为0到31, 而对于“月”字段来说为1到12。"/"字段仅仅只是帮助你在允许的数值范围内从开始"第n"的值。 因此 对于“月”字段来说"7/6"只是表示7月被开启而不是“每六个月”, 请注意其中微妙的差别。

'L'字符可用在“日”和“周几”这两个字段。它是"last"的缩写, 但是在这两个字段中有不同的含义。例如,“日”字段中的"L"表示"一个月中的最后一天" —— 对于一月就是31号对于二月来说就是28号（非闰年）。而在“周几”字段中, 它简单的表示"7" or "SAT"，但是如果在“周几”字段中使用时跟在某个数字之后, 它表示"该月最后一个星期×" —— 比如"6L"表示"该月最后一个周五"。当使用'L'选项时,指定确定的列表或者范围非常重要，否则你会被结果搞糊涂的。

'W' 可用于“日”字段。用来指定历给定日期最近的工作日(周一到周五) 。比如你将“日”字段设为"15W"，意为: "离该月15号最近的工作日"。因此如果15号为周六，触发器会在14号即周五调用。如果15号为周日, 触发器会在16号也就是周一触发。如果15号为周二,那么当天就会触发。然而如果你将“日”字段设为"1W", 而一号又是周六, 触发器会于下周一也就是当月的3号触发,因为它不会越过当月的值的范围边界。'W'字符只能用于“日”字段的值为单独的一天而不是一系列值的时候。

'L'和'W'可以组合用于“日”字段表示为'LW'，意为"该月最后一个工作日"。

'#' 字符可用于“周几”字段。该字符表示“该月第几个周×”，比如"6#3"表示该月第三个周五( 6表示周五而"#3"该月第三个)。再比如: "2#1" = 表示该月第一个周一而 "4#5" = 该月第五个周三。注意如果你指定"#5"该月没有第五个“周×”，该月是不会触发的。

'C' 字符可用于“日”和“周几”字段，它是"calendar"的缩写。 它表示为基于相关的日历所计算出的值（如果有的话）。如果没有关联的日历, 那它等同于包含全部日历。“日”字段值为"5C"表示"日历中的第一天或者5号以后"，“周几”字段值为"1C"则表示"日历中的第一天或者周日以后"。

对于“月份”字段和“周几”字段来说合法的字符都不是大小写敏感的。

下面是一些完整的例子:

|  |  |  |
| --- | --- | --- |
| **表达式** |  | **含义** |
| "0 0 12 \* \* ?" |  | 每天中午十二点触发 |
| "0 15 10 ? \* \*" |  | 每天早上10：15触发 |
| "0 15 10 \* \* ?" |  | 每天早上10：15触发 |
| "0 15 10 \* \* ? \*" |  | 每天早上10：15触发 |
| "0 15 10 \* \* ? 2005" |  | 2005年的每天早上10：15触发 |
| "0 \* 14 \* \* ?" |  | 每天从下午2点开始到2点59分每分钟一次触发 |
| "0 0/5 14 \* \* ?" |  | 每天从下午2点开始到2：55分结束每5分钟一次触发 |
| "0 0/5 14,18 \* \* ?" |  | 每天的下午2点至2：55和6点至6点55分两个时间段内每5分钟一次触发 |
| "0 0-5 14 \* \* ?" |  | 每天14:00至14:05每分钟一次触发 |
| "0 10,44 14 ? 3 WED" |  | 三月的每周三的14：10和14：44触发 |
| "0 15 10 ? \* MON-FRI" |  | 每个周一、周二、周三、周四、周五的10：15触发 |
| "0 15 10 15 \* ?" |  | 每月15号的10：15触发 |
| "0 15 10 L \* ?" |  | 每月的最后一天的10：15触发 |
| "0 15 10 ? \* 6L" |  | 每月最后一个周五的10：15触发 |
| "0 15 10 ? \* 6L" |  | 每月最后一个周五的10：15触发 |
| "0 15 10 ? \* 6L 2002-2005" |  | 2002年至2005年的每月最后一个周五的10：15触发 |
| "0 15 10 ? \* 6#3" |  | 每月的第三个周五的10：15触发 |

请注意'?' 和'\*' 在“日”和“周几”字段中的用法!

# [spring里quartz调度的连续触发](http://scnujie.javaeye.com/blog/262777)

   作法是通过spring在执行的触发器的监听器的里面注入下一个要执行的监听器实例，这   
  
样就可以达到连续地触发，而至于具体怎样的触发：比如说什么时候触发，下一个相隔多久才   
  
触发这些则可以通过配置文件进行配置。   
    
    这是用于连续触发的Listener:

Java代码

1. package com.sww.quartz.listeners;
3. import org.apache.log4j.Logger;
4. import org.quartz.JobDetail;
5. import org.quartz.JobExecutionContext;
6. import org.quartz.Scheduler;
7. import org.quartz.Trigger;
8. import org.quartz.listeners.TriggerListenerSupport;
9. import org.springframework.scheduling.quartz.QuartzJobBean;
10. import org.springframework.scheduling.quartz.SimpleTriggerBean;
12. public class NextJobTriggerListener extends TriggerListenerSupport {
13. private  Logger logger = Logger.getLogger(getClass());
14. private String name;
15. public String getName() {
16. return this.name;
17. }
18. public void setName(String name)
19. {
20. this.name=name;
21. }
22. private SimpleTriggerBean nextTrigger;
23. public void setNextTrigger(SimpleTriggerBean nextTrigger) {
24. this.nextTrigger = nextTrigger;
25. }
26. @Override
27. public void triggerComplete(Trigger trigger, JobExecutionContext context, int code) {
28. try{
29. logger.debug("Trigger :"+trigger.getGroup()+"."+trigger.getName()+" completed with code:"+code+",next trigger is "+nextTrigger.getGroup()+"."+nextTrigger.getName());
30. if(code!=3)return;
31. Scheduler schduler=context.getScheduler();
32. JobDetail nextJob=nextTrigger.getJobDetail();
33. //查找名称和即将加入的任务一样的任务
34. JobDetail oldJob=schduler.getJobDetail(nextJob.getName(),nextJob.getGroup());
35. //查找名称和即将加入的触发器一样的触发器
36. Trigger oldTrigger=schduler.getTrigger( nextTrigger.getName(),nextTrigger.getGroup());
38. if(oldJob==null&&oldTrigger==null)//同名的任务和触发器都不存在
39. {
40. logger.debug("inside scheduleJob."+code);
41. schduler.scheduleJob(nextJob,nextTrigger);
42. }else//同名的任务或触发器
43. {
45. logger.debug("oldJob==null:"+(oldJob==null));
46. logger.debug("oldTrigger==null:"+(oldTrigger==null));
47. }
48. super.triggerComplete(trigger, context, code);
49. }catch(Exception e)
50. {
51. e.printStackTrace();
52. }
53. }

56. }

package com.sww.quartz.listeners;

import org.apache.log4j.Logger;

import org.quartz.JobDetail;

import org.quartz.JobExecutionContext;

import org.quartz.Scheduler;

import org.quartz.Trigger;

import org.quartz.listeners.TriggerListenerSupport;

import org.springframework.scheduling.quartz.QuartzJobBean;

import org.springframework.scheduling.quartz.SimpleTriggerBean;

public class NextJobTriggerListener extends TriggerListenerSupport {

private Logger logger = Logger.getLogger(getClass());

private String name;

public String getName() {

return this.name;

}

public void setName(String name)

{

this.name=name;

}

private SimpleTriggerBean nextTrigger;

public void setNextTrigger(SimpleTriggerBean nextTrigger) {

this.nextTrigger = nextTrigger;

}

@Override

public void triggerComplete(Trigger trigger, JobExecutionContext context, int code) {

try{

logger.debug("Trigger :"+trigger.getGroup()+"."+trigger.getName()+" completed with code:"+code+",next trigger is "+nextTrigger.getGroup()+"."+nextTrigger.getName());

if(code!=3)return;

Scheduler schduler=context.getScheduler();

JobDetail nextJob=nextTrigger.getJobDetail();

//查找名称和即将加入的任务一样的任务

JobDetail oldJob=schduler.getJobDetail(nextJob.getName(),nextJob.getGroup());

//查找名称和即将加入的触发器一样的触发器

Trigger oldTrigger=schduler.getTrigger( nextTrigger.getName(),nextTrigger.getGroup());

if(oldJob==null&&oldTrigger==null)//同名的任务和触发器都不存在

{

logger.debug("inside scheduleJob."+code);

schduler.scheduleJob(nextJob,nextTrigger);

}else//同名的任务或触发器

{

logger.debug("oldJob==null:"+(oldJob==null));

logger.debug("oldTrigger==null:"+(oldTrigger==null));

}

super.triggerComplete(trigger, context, code);

}catch(Exception e)

{

e.printStackTrace();

}

}

}

  这是具体怎样触发的spring的配置文件：

Xml代码

1. <?xml version="1.0" encoding="UTF-8"?>
2. <beans xmlns="http://www.springframework.org/schema/beans"
3. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4. xmlns:jee="http://www.springframework.org/schema/jee"
5. xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
6. http://www.springframework.org/schema/jee http://www.springframework.org/schema/jee/spring-jee-2.0.xsd">
8. <bean
9. class="org.springframework.scheduling.quartz.SchedulerFactoryBean">
10. <property name="quartzProperties">
11. <props>
12. <prop key="org.quartz.threadPool.class">
13. org.quartz.simpl.SimpleThreadPool
14. </prop>
15. <prop key="org.quartz.threadPool.threadCount">10</prop>
16. <prop key="org.quartz.threadPool.threadPriority">
17. 5
18. </prop>
19. <prop
20. key="org.quartz.threadPool.threadsInheritContextClassLoaderOfInitializingThread">
21. true
22. </prop>
23. </props>
24. </property>
25. <property name="triggers">
26. <list>
27. <ref bean="provinceTrigger"/>
28. </list>
29. </property>
30. <property name="triggerListeners">
31. <list>
32. <ref bean="provinceTriggerListener"/>
33. <ref bean="cityTriggerListener"/>
34. <ref bean="linkTriggerListener"/>
35. </list>
36. </property>
37. </bean>
39. <bean id="provinceJob"
40. class="org.springframework.scheduling.quartz.JobDetailBean">
41. <property name="jobClass">
42. <value>com.tms.quartz.jobs.ProvinceJob</value>
43. </property>
44. </bean>
45. <bean id="provinceTrigger" class="org.springframework.scheduling.quartz.CronTriggerBean">
46. <property name="jobDetail">
47. <ref bean="provinceJob" />
48. </property>
49. <property name="cronExpression">
50. <value>0 30 0 \* \* ?</value>
51. </property>
52. <property name="triggerListenerNames">
53. <list>
54. <value>provinceTriggerListener</value>
55. </list>
56. </property>
57. </bean>
58. <bean id="provinceTriggerListener" class="com.tms.quartz.listeners.NextJobTriggerListener">
59. <property name="nextTrigger" ref="cityTrigger"></property>
60. <property name="name" value="provinceTriggerListener"></property>
61. </bean>

64. <bean id="cityJob"
65. class="org.springframework.scheduling.quartz.JobDetailBean">
66. <property name="jobClass">
67. <value>com.tms.quartz.jobs.CityJob</value>
68. </property>
69. </bean>
70. <bean id="cityTrigger"
71. class="org.springframework.scheduling.quartz.SimpleTriggerBean">
72. <property name="jobDetail">
73. <ref bean="cityJob" />
74. </property>
75. <property name="startDelay">
76. <value>60000</value>
77. </property>
78. <property name="repeatCount">
79. <value>0</value>
80. </property>
81. <property name="repeatInterval">
82. <value>60000</value>
83. </property>
84. <property name="triggerListenerNames">
85. <list>
86. <value>cityTriggerListener</value>
87. </list>
88. </property>
89. </bean>
90. <bean id="cityTriggerListener" class="com.tms.quartz.listeners.NextJobTriggerListener">
91. <property name="nextTrigger" ref="linkTrigger"></property>
92. <property name="name" value="cityTriggerListener"></property>
93. </bean>
95. <!--
96. <bean id="linkJob"
97. class="org.springframework.scheduling.quartz.JobDetailBean">
98. <property name="jobClass">
99. <value>com.tms.quartz.jobs.CityLinkJob</value>
100. </property>
101. </bean>
102. <bean id="linkTrigger"
103. class="org.springframework.scheduling.quartz.SimpleTriggerBean">
104. <property name="jobDetail">
105. <ref bean="linkJob" />
106. </property>
107. <property name="startDelay">
108. <value>60000</value>
109. </property>
110. <property name="repeatCount">
111. <value>0</value>
112. </property>
113. <property name="repeatInterval">
114. <value>60000</value>
115. </property>
116. <property name="triggerListenerNames">
117. <list>
118. <value>linkTriggerListener</value>
119. </list>
120. </property>
121. </bean>
122. <bean id="linkTriggerListener" class="com.tms.quartz.listeners.NextJobTriggerListener">
123. <property name="nextTrigger" ref="sightTrigger"></property>
124. <property name="name" value="linkTriggerListener"></property>
125. </bean>
126. -->
128. <bean id="sightJob"
129. class="org.springframework.scheduling.quartz.JobDetailBean">
130. <property name="jobClass">
131. <value>com.tms.quartz.jobs.SightJob</value>
132. </property>
133. </bean>
134. <bean id="sightTrigger"
135. class="org.springframework.scheduling.quartz.SimpleTriggerBean">
136. <property name="jobDetail">
137. <ref bean="sightJob" />
138. </property>
139. <property name="startDelay">
140. <value>60000</value>
141. </property>
142. <property name="repeatCount">
143. <value>0</value>
144. </property>
145. <property name="repeatInterval">
146. <value>60000</value>
147. </property>
148. </bean>
149. </beans>