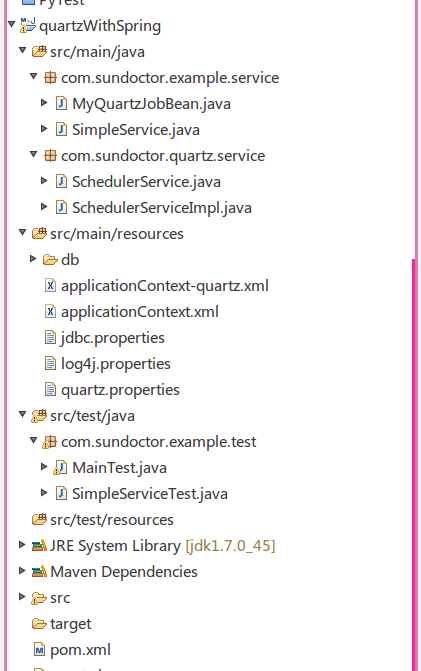
在项目中有一个需求，需要灵活配置调度任务时间，并能自由启动或停止调度。  
有关调度的实现我就第一就想到了Quartz这个开源调度组件，因为很多项目使用过，Spring结合Quartz静态配置调度任务时间，非常easy。比如：每天凌晨几点定时运行一个程序，这只要在工程中的spring配置文件中配置好spring整合quartz的几个属性就好。  
  
Spring配置文件

**Xml代码  [收藏代码](javascript:void())**

1. **<bean** id="jobDetail" class="org.springframework.scheduling.quartz.MethodInvokingJobDetailFactoryBean"**>**
2. **<property** name="targetObject" ref="simpleService" **/>**
3. **<property** name="targetMethod" value="test" **/>**
4. **</bean>**
5. **<bean** id="cronTrigger" class="org.springframework.scheduling.quartz.CronTriggerFactoryBean"**>**
6. **<property** name="jobDetail" ref="jobDetail" **/>**
7. **<property** name="cronExpression" value="0 0/50 \* ? \* \* \*" **/>**
8. **</bean>**
9. **<bean**  id="schedulerTrigger" class="org.springframework.scheduling.quartz.SchedulerFactoryBean"**>**
10. **<property** name="triggers"**>**
11. **<list>**
12. **<ref** bean="cronTrigger"**/>**
13. **</list>**
14. **</property>**
15. **</bean>**

 这种配置就是对quartz的一种简单的使用了，调度任务会在spring启动的时候加载到内存中，按照cronTrigger中定义的 cronExpression定义的时间按时触发调度任务。但是这是quartz使用“内存”方式的一种配置，也比较常见，当然对于不使用spring的项目，也可以单独整合quartz。方法也比较简单，可以从quartz的doc中找到配置方式，或者看一下《Quartz Job Scheduling Framework 》。  
  
但是对于想持久化调度任务的状态，并且灵活调整调度时间的方式来说，上面的内存方式就不能满足要求了，正如本文开始我遇到的情况，需要采用数据库方式集成 Quartz，这部分集成其实在《Quartz Job Scheduling Framework 》中也有较为详细的介绍，当然doc文档中也有，但是缺乏和spring集成的实例。  
  
一、需要构建Quartz数据库表，建表脚本在Quartz发行包的docs\dbTables目录，里面有各种数据库建表脚本，例子中采用的Quartz 2.2.1版本，使用H2内存数据库，执行了Quartz发行包的docs\dbTables\tables\_h2.sql建表。  
  
二、建立Maven project，完成后目录如下  
  
  
   
  
三、配置applicationContext.xml文件

**Xml代码  [收藏代码](javascript:void())**

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" xmlns:jdbc="http://www.springframework.org/schema/jdbc"
4. xmlns:context="http://www.springframework.org/schema/context"
5. xsi:schemaLocation="
6. http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd
7. http://www.springframework.org/schema/jdbc http://www.springframework.org/schema/jdbc/spring-jdbc.xsd
8. http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd
9. "**>**
10. **<context:component-scan** base-package="com.sundoctor" **/>**
11. <!-- 使用H2内存数据库并创建quartz数据库表 -->
12. **<jdbc:embedded-database** id="dataSource" type="H2"**>**
13. **<jdbc:script** location="classpath:db/tables\_h2.sql"**/>**
14. **</jdbc:embedded-database>**
15. **</beans>**

这里只是配置了数据库连接dataSource，为了简单方便，我使用H2内存数据库，执行tables\_h2.sql创建quartz数据库表，还没有涉及到quartz有关配置，下面且听我慢慢道来。  
  
四、实现动态定时任务  
  什么是动态定时任务：是由客户制定生成的，服务端只知道该去执行什么任务，但任务的定时是不确定的（是由客户制定）。  
这样总不能修改配置文件每定制个定时任务就增加一个trigger吧，即便允许客户修改配置文件，但总需要重新启动web服务啊，研究了下Quartz在Spring中的动态定时，发现

**Xml代码  [收藏代码](javascript:void())**

1. **<bean** id="cronTrigger" class="org.springframework.scheduling.quartz.CronTriggerFactoryBean"**>**
2. **<property** name="jobDetail" ref="jobDetail" **/>**
3. **<property** name="cronExpression" value="0 0/50 \* ? \* \* \*" **/>**
4. **</bean>**

中cronExpression是关键，如果可以动态设置cronExpression的值，就可以顺利解决问题了。这样我们就不能直接使用org.springframework.scheduling.quartz.CronTriggerFactoryBean，需要自己实现一个动态调度服务类，在其中构建CronTrigger或SimpleTrigger，动态配置时间。  
动态调度服务接口：

**Java代码  [收藏代码](javascript:void())**

1. **package** com.sundoctor.quartz.service;
3. **import** java.util.Date;
5. **import** org.quartz.CronExpression;
7. **public** **interface** SchedulerService {
8. /\*\*
9. \* 根据 Quartz Cron Expression 调试任务
10. \*
11. \* @param cronExpression
12. \*            Quartz Cron 表达式，如 "0/10 \* \* ? \* \* \*"等
13. \*/
14. **void** schedule(String cronExpression);
16. /\*\*
17. \* 根据 Quartz Cron Expression 调试任务
18. \*
19. \* @param name
20. \*            Quartz CronTrigger名称
21. \* @param cronExpression
22. \*            Quartz Cron 表达式，如 "0/10 \* \* ? \* \* \*"等
23. \*/
24. **void** schedule(String name, String cronExpression);
26. /\*\*
27. \* 根据 Quartz Cron Expression 调试任务
28. \*
29. \* @param name
30. \*            Quartz CronTrigger名称
31. \* @param group
32. \*            Quartz CronTrigger组
33. \* @param cronExpression
34. \*            Quartz Cron 表达式，如 "0/10 \* \* ? \* \* \*"等
35. \*/
36. **void** schedule(String name, String group, String cronExpression);
38. /\*\*
39. \* 根据 Quartz Cron Expression 调试任务
40. \*
41. \* @param cronExpression
42. \*            Quartz CronExpression
43. \*/
44. **void** schedule(CronExpression cronExpression);
46. /\*\*
47. \* 根据 Quartz Cron Expression 调试任务
48. \*
49. \* @param name
50. \*            Quartz CronTrigger名称
51. \* @param cronExpression
52. \*            Quartz CronExpression
53. \*/
54. **void** schedule(String name, CronExpression cronExpression);
56. /\*\*
57. \* 根据 Quartz Cron Expression 调试任务
58. \*
59. \* @param name
60. \*            Quartz CronTrigger名称
61. \* @param group
62. \*            Quartz CronTrigger组
63. \* @param cronExpression
64. \*            Quartz CronExpression
65. \*/
66. **void** schedule(String name, String group, CronExpression cronExpression);
68. /\*\*
69. \* 在startTime时执行调试一次
70. \*
71. \* @param startTime
72. \*            调度开始时间
73. \*/
74. **void** schedule(Date startTime);
76. **void** schedule(Date startTime, String group);
78. /\*\*
79. \* 在startTime时执行调试一次
80. \*
81. \* @param name
82. \*            Quartz SimpleTrigger 名称
83. \* @param startTime
84. \*            调度开始时间
85. \*/
86. **void** schedule(String name, Date startTime);
88. **void** schedule(String name, Date startTime, String group);
90. /\*\*
91. \* 在startTime时执行调试，endTime结束执行调度
92. \*
93. \* @param startTime
94. \*            调度开始时间
95. \* @param endTime
96. \*            调度结束时间
97. \*/
98. **void** schedule(Date startTime, Date endTime);
100. **void** schedule(Date startTime, Date endTime, String group);
102. /\*\*
103. \* 在startTime时执行调试，endTime结束执行调度
104. \*
105. \* @param name
106. \*            Quartz SimpleTrigger 名称
107. \* @param startTime
108. \*            调度开始时间
109. \* @param endTime
110. \*            调度结束时间
111. \*/
112. **void** schedule(String name, Date startTime, Date endTime);
114. **void** schedule(String name, Date startTime, Date endTime, String group);
116. /\*\*
117. \* 在startTime时执行调试，endTime结束执行调度，重复执行repeatCount次
118. \*
119. \* @param startTime
120. \*            调度开始时间
121. \* @param repeatCount
122. \*            重复执行次数
123. \*/
124. **void** schedule(Date startTime, **int** repeatCount);
126. /\*\*
127. \* 在startTime时执行调试，endTime结束执行调度，重复执行repeatCount次
128. \*
129. \* @param startTime
130. \*            调度开始时间
131. \* @param endTime
132. \*            调度结束时间
133. \* @param repeatCount
134. \*            重复执行次数
135. \*/
136. **void** schedule(Date startTime, Date endTime, **int** repeatCount);
138. **void** schedule(Date startTime, Date endTime, **int** repeatCount, String group);
140. /\*\*
141. \* 在startTime时执行调试，endTime结束执行调度，重复执行repeatCount次
142. \*
143. \* @param name
144. \*            Quartz SimpleTrigger 名称
145. \* @param startTime
146. \*            调度开始时间
147. \* @param endTime
148. \*            调度结束时间
149. \* @param repeatCount
150. \*            重复执行次数
151. \*/
152. **void** schedule(String name, Date startTime, Date endTime, **int** repeatCount);
154. **void** schedule(String name, Date startTime, Date endTime, **int** repeatCount, String group);
156. /\*\*
157. \* 在startTime时执行调试，endTime结束执行调度，重复执行repeatCount次，每隔repeatInterval秒执行一次
158. \*
159. \* @param startTime
160. \*            调度开始时间
161. \*
162. \* @param repeatCount
163. \*            重复执行次数
164. \* @param repeatInterval
165. \*            执行时间隔间
166. \*/
167. **void** schedule(Date startTime, **int** repeatCount, **long** repeatInterval);
169. /\*\*
170. \* 在startTime时执行调试，endTime结束执行调度，重复执行repeatCount次，每隔repeatInterval秒执行一次
171. \*
172. \* @param startTime
173. \*            调度开始时间
174. \* @param endTime
175. \*            调度结束时间
176. \* @param repeatCount
177. \*            重复执行次数
178. \* @param repeatInterval
179. \*            执行时间隔间
180. \*/
181. **void** schedule(Date startTime, Date endTime, **int** repeatCount, **long** repeatInterval);
183. **void** schedule(Date startTime, Date endTime, **int** repeatCount, **long** repeatInterval, String group);
185. /\*\*
186. \* 在startTime时执行调试，endTime结束执行调度，重复执行repeatCount次，每隔repeatInterval秒执行一次
187. \*
188. \* @param name
189. \*            Quartz SimpleTrigger 名称
190. \* @param startTime
191. \*            调度开始时间
192. \* @param endTime
193. \*            调度结束时间
194. \* @param repeatCount
195. \*            重复执行次数
196. \* @param repeatInterval
197. \*            执行时间隔间
198. \*/
199. **void** schedule(String name, Date startTime, Date endTime, **int** repeatCount, **long** repeatInterval);
201. **void** schedule(String name, Date startTime, Date endTime, **int** repeatCount, **long** repeatInterval, String group);
203. /\*\*
204. \* 暂停触发器
205. \*
206. \* @param triggerName
207. \*            触发器名称
208. \*/
209. **void** pauseTrigger(String triggerName);
211. /\*\*
212. \* 暂停触发器
213. \*
214. \* @param triggerName
215. \*            触发器名称
216. \* @param group
217. \*            触发器组
218. \*/
219. **void** pauseTrigger(String triggerName, String group);
221. /\*\*
222. \* 恢复触发器
223. \*
224. \* @param triggerName
225. \*            触发器名称
226. \*/
227. **void** resumeTrigger(String triggerName);
229. /\*\*
230. \* 恢复触发器
231. \*
232. \* @param triggerName
233. \*            触发器名称
234. \* @param group
235. \*            触发器组
236. \*/
237. **void** resumeTrigger(String triggerName, String group);
239. /\*\*
240. \* 删除触发器
241. \*
242. \* @param triggerName
243. \*            触发器名称
244. \* @return
245. \*/
246. **boolean** removeTrigdger(String triggerName);
248. /\*\*
249. \* 删除触发器
250. \*
251. \* @param triggerName
252. \*            触发器名称
253. \* @param group
254. \*            触发器组
255. \* @return
256. \*/
257. **boolean** removeTrigdger(String triggerName, String group);
258. }

动态调度服务实现类：

**Java代码  [收藏代码](javascript:void())**

1. **package** com.sundoctor.quartz.service;
3. **import** java.text.ParseException;
4. **import** java.util.Date;
5. **import** java.util.UUID;
7. **import** org.quartz.CronExpression;
8. **import** org.quartz.JobDetail;
9. **import** org.quartz.Scheduler;
10. **import** org.quartz.SchedulerException;
11. **import** org.quartz.TriggerKey;
12. **import** org.quartz.impl.triggers.CronTriggerImpl;
13. **import** org.quartz.impl.triggers.SimpleTriggerImpl;
14. **import** org.springframework.beans.factory.annotation.Autowired;
15. **import** org.springframework.stereotype.Service;
17. @Service("schedulerService")
18. **public** **class** SchedulerServiceImpl **implements** SchedulerService {
20. **private** **static** **final** String NULLSTRING = **null**;
21. **private** **static** **final** Date NULLDATE = **null**;
23. @Autowired
24. **private** Scheduler scheduler;
25. @Autowired
26. **private** JobDetail jobDetail;
28. @Override
29. **public** **void** schedule(String cronExpression) {
30. schedule(NULLSTRING, cronExpression);
31. }
33. @Override
34. **public** **void** schedule(String name, String cronExpression) {
35. schedule(name, NULLSTRING, cronExpression);
36. }
38. @Override
39. **public** **void** schedule(String name, String group, String cronExpression) {
40. **try** {
41. schedule(name, group, **new** CronExpression(cronExpression));
42. } **catch** (ParseException e) {
43. **throw** **new** IllegalArgumentException(e);
44. }
45. }
47. @Override
48. **public** **void** schedule(CronExpression cronExpression) {
49. schedule(NULLSTRING, cronExpression);
50. }
52. @Override
53. **public** **void** schedule(String name, CronExpression cronExpression) {
54. schedule(name, NULLSTRING, cronExpression);
55. }
57. @Override
58. **public** **void** schedule(String name, String group, CronExpression cronExpression) {
60. **if** (isValidExpression(cronExpression)) {
62. **if** (name == **null** || name.trim().equals("")) {
63. name = UUID.randomUUID().toString();
64. }
66. CronTriggerImpl trigger = **new** CronTriggerImpl();
67. trigger.setCronExpression(cronExpression);
69. TriggerKey triggerKey = **new** TriggerKey(name, group);
71. trigger.setJobName(jobDetail.getKey().getName());
72. trigger.setKey(triggerKey);
74. **try** {
75. scheduler.addJob(jobDetail, **true**);
76. **if** (scheduler.checkExists(triggerKey)) {
77. scheduler.rescheduleJob(triggerKey, trigger);
78. } **else** {
79. scheduler.scheduleJob(trigger);
80. }
81. } **catch** (SchedulerException e) {
82. **throw** **new** IllegalArgumentException(e);
83. }
84. }
85. }
87. @Override
88. **public** **void** schedule(Date startTime) {
89. schedule(startTime, NULLDATE);
90. }
92. @Override
93. **public** **void** schedule(Date startTime, String group) {
94. schedule(startTime, NULLDATE, group);
95. }
97. @Override
98. **public** **void** schedule(String name, Date startTime) {
99. schedule(name, startTime, NULLDATE);
100. }
102. @Override
103. **public** **void** schedule(String name, Date startTime, String group) {
104. schedule(name, startTime, NULLDATE, group);
105. }
107. @Override
108. **public** **void** schedule(Date startTime, Date endTime) {
109. schedule(startTime, endTime, 0);
110. }
112. @Override
113. **public** **void** schedule(Date startTime, Date endTime, String group) {
114. schedule(startTime, endTime, 0, group);
115. }
117. @Override
118. **public** **void** schedule(String name, Date startTime, Date endTime) {
119. schedule(name, startTime, endTime, 0);
120. }
122. @Override
123. **public** **void** schedule(String name, Date startTime, Date endTime, String group) {
124. schedule(name, startTime, endTime, 0, group);
125. }
127. @Override
128. **public** **void** schedule(Date startTime, **int** repeatCount) {
129. schedule(**null**, startTime, NULLDATE, 0);
130. }
132. @Override
133. **public** **void** schedule(Date startTime, Date endTime, **int** repeatCount) {
134. schedule(**null**, startTime, endTime, 0);
135. }
137. @Override
138. **public** **void** schedule(Date startTime, Date endTime, **int** repeatCount, String group) {
139. schedule(**null**, startTime, endTime, 0, group);
140. }
142. @Override
143. **public** **void** schedule(String name, Date startTime, Date endTime, **int** repeatCount) {
144. schedule(name, startTime, endTime, 0, 0L);
145. }
147. @Override
148. **public** **void** schedule(String name, Date startTime, Date endTime, **int** repeatCount, String group) {
149. schedule(name, startTime, endTime, 0, 0L, group);
150. }
152. @Override
153. **public** **void** schedule(Date startTime, **int** repeatCount, **long** repeatInterval) {
154. schedule(**null**, startTime, NULLDATE, repeatCount, repeatInterval);
155. }
157. @Override
158. **public** **void** schedule(Date startTime, Date endTime, **int** repeatCount, **long** repeatInterval) {
159. schedule(**null**, startTime, endTime, repeatCount, repeatInterval);
160. }
162. @Override
163. **public** **void** schedule(Date startTime, Date endTime, **int** repeatCount, **long** repeatInterval, String group) {
164. schedule(**null**, startTime, endTime, repeatCount, repeatInterval, group);
165. }
167. @Override
168. **public** **void** schedule(String name, Date startTime, Date endTime, **int** repeatCount, **long** repeatInterval) {
169. schedule(name, startTime, endTime, repeatCount, repeatInterval, NULLSTRING);
170. }
172. @Override
173. **public** **void** schedule(String name, Date startTime, Date endTime, **int** repeatCount, **long** repeatInterval, String group) {
175. **if** (**this**.isValidExpression(startTime)) {
177. **if** (name == **null** || name.trim().equals("")) {
178. name = UUID.randomUUID().toString();
179. }
181. TriggerKey triggerKey = **new** TriggerKey(name, group);
183. SimpleTriggerImpl trigger = **new** SimpleTriggerImpl();
184. trigger.setKey(triggerKey);
185. trigger.setJobName(jobDetail.getKey().getName());
187. trigger.setStartTime(startTime);
188. trigger.setEndTime(endTime);
189. trigger.setRepeatCount(repeatCount);
190. trigger.setRepeatInterval(repeatInterval);
192. **try** {
193. scheduler.addJob(jobDetail, **true**);
194. **if** (scheduler.checkExists(triggerKey)) {
195. scheduler.rescheduleJob(triggerKey, trigger);
196. } **else** {
197. scheduler.scheduleJob(trigger);
198. }
199. } **catch** (SchedulerException e) {
200. **throw** **new** IllegalArgumentException(e);
201. }
202. }
203. }
205. @Override
206. **public** **void** pauseTrigger(String triggerName) {
207. pauseTrigger(triggerName, NULLSTRING);
208. }
210. @Override
211. **public** **void** pauseTrigger(String triggerName, String group) {
212. **try** {
213. scheduler.pauseTrigger(**new** TriggerKey(triggerName, group));// 停止触发器
214. } **catch** (SchedulerException e) {
215. **throw** **new** RuntimeException(e);
216. }
217. }
219. @Override
220. **public** **void** resumeTrigger(String triggerName) {
221. resumeTrigger(triggerName, NULLSTRING);
222. }
224. @Override
225. **public** **void** resumeTrigger(String triggerName, String group) {
226. **try** {
227. scheduler.resumeTrigger(**new** TriggerKey(triggerName, group));// 重启触发器
228. } **catch** (SchedulerException e) {
229. **throw** **new** RuntimeException(e);
230. }
231. }
233. @Override
234. **public** **boolean** removeTrigdger(String triggerName) {
235. **return** removeTrigdger(triggerName, NULLSTRING);
236. }
238. @Override
239. **public** **boolean** removeTrigdger(String triggerName, String group) {
240. TriggerKey triggerKey = **new** TriggerKey(triggerName, group);
241. **try** {
242. scheduler.pauseTrigger(triggerKey);// 停止触发器
243. **return** scheduler.unscheduleJob(triggerKey);// 移除触发器
244. } **catch** (SchedulerException e) {
245. **throw** **new** RuntimeException(e);
246. }
247. }
249. **private** **boolean** isValidExpression(**final** CronExpression cronExpression) {
251. CronTriggerImpl trigger = **new** CronTriggerImpl();
252. trigger.setCronExpression(cronExpression);
254. Date date = trigger.computeFirstFireTime(**null**);
256. **return** date != **null** && date.after(**new** Date());
257. }
259. **private** **boolean** isValidExpression(**final** Date startTime) {
261. SimpleTriggerImpl trigger = **new** SimpleTriggerImpl();
262. trigger.setStartTime(startTime);
264. Date date = trigger.computeFirstFireTime(**null**);
266. **return** date != **null** && date.after(**new** Date());
267. }
268. }

SchedulerService 只有一个多态方法schedule，SchedulerServiceImpl实现SchedulerService接口，注入org.quartz.Schedulert和org.quartz.JobDetail，schedule方法可以动态配置org.quartz.CronExpression或org.quartz.SimpleTrigger调度时间。  
  
五、实现自己的org.quartz.JobDetail

**Java代码  [收藏代码](javascript:void())**

1. **package** com.sundoctor.example.service;
3. **import** org.quartz.JobExecutionContext;
4. **import** org.quartz.JobExecutionException;
5. **import** org.quartz.SchedulerException;
6. **import** org.quartz.Trigger;
7. **import** org.slf4j.Logger;
8. **import** org.slf4j.LoggerFactory;
9. **import** org.springframework.context.ApplicationContext;
10. **import** org.springframework.scheduling.quartz.QuartzJobBean;
12. **public** **class** MyQuartzJobBean **extends** QuartzJobBean {
14. **private** **static** **final** Logger logger = LoggerFactory.getLogger(MyQuartzJobBean.**class**);
16. @Override
17. **protected** **void** executeInternal(JobExecutionContext jobexecutioncontext) **throws** JobExecutionException {
18. Trigger trigger = jobexecutioncontext.getTrigger();
19. String triggerName = trigger.getKey().getName();
21. SimpleService simpleService = getApplicationContext(jobexecutioncontext).getBean("simpleService",
22. SimpleService.**class**);
23. simpleService.testMethod(triggerName);
25. }
27. **private** ApplicationContext getApplicationContext(**final** JobExecutionContext jobexecutioncontext) {
28. **try** {
29. **return** (ApplicationContext) jobexecutioncontext.getScheduler().getContext().get("applicationContextKey");
30. } **catch** (SchedulerException e) {
31. logger.error("jobexecutioncontext.getScheduler().getContext() error!", e);
32. **throw** **new** RuntimeException(e);
33. }
34. }
36. }

MyQuartzJobBean继承org.springframework.scheduling.quartz.QuartzJobBean，SimpleService如下：

**Java代码  [收藏代码](javascript:void())**

1. **package** com.sundoctor.example.service;
3. **import** org.slf4j.Logger;
4. **import** org.slf4j.LoggerFactory;
5. **import** org.springframework.stereotype.Service;
7. @Service("simpleService")
8. **public** **class** SimpleService {
10. **private** **static** **final** Logger logger = LoggerFactory.getLogger(SimpleService.**class**);
12. **public** **void** testMethod(String triggerName) {
13. // 这里执行定时调度业务
14. logger.info(triggerName);
15. }
17. }

SimpleService主要执行定时调度业务，在这里我只是简单打印一下log日志。  
配置applicationContext-quartz.xml文件：

**Xml代码  [收藏代码](javascript:void())**

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. xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd"**>**
6. **<bean** name="quartzScheduler"
7. class="org.springframework.scheduling.quartz.SchedulerFactoryBean"**>**
8. **<property** name="dataSource"**>**
9. **<ref** bean="dataSource" **/>**
10. **</property>**
11. **<property** name="applicationContextSchedulerContextKey" value="applicationContextKey" **/>**
12. **<property** name="configLocation" value="classpath:quartz.properties" **/>**
13. **</bean>**
15. **<bean** id="jobDetail" class="org.springframework.scheduling.quartz.JobDetailFactoryBean"**>**
16. **<property** name="jobClass"**>**
17. **<value>**com.sundoctor.example.service.MyQuartzJobBean**</value>**
18. **</property>**
19. **<property** name="durability" value="true" **/>**
20. **</bean>**
22. **</beans>**

dataSource：项目中用到的数据源，里面包含了quartz用到的数据库表；  
applicationContextSchedulerContextKey： 是org.springframework.scheduling.quartz.SchedulerFactoryBean这个类中把spring上下文以key/value的方式存放在了quartz的SchedulerContext，可以用applicationContextSchedulerContextKey所定义的key得到spring的ApplicationContext，然后就可使用ApplicationContext取得spring beans，使用方法参见MyQuartzJobBean；  
configLocation：用于指明quartz的配置文件的位置，如果不用spring配置quartz的话，本身quartz是通过一个配置文件进行配置的，默认名称是quartz.properties，里面配置的参数在quartz的doc文档中都有介绍，可以调整quartz，我在项目中也用这个文件部分的配置了一些属性，代码如下：

**Xml代码  [收藏代码](javascript:void())**

1. org.quartz.scheduler.instanceName = DefaultQuartzScheduler
2. org.quartz.scheduler.rmi.export = false
3. org.quartz.scheduler.rmi.proxy = false
4. org.quartz.scheduler.wrapJobExecutionInUserTransaction = false
6. orgorg.quartz.threadPool.class = org.quartz.simpl.SimpleThreadPool
7. org.quartz.threadPool.threadCount = 10
8. org.quartz.threadPool.threadPriority = 5
9. org.quartz.threadPool.threadsInheritContextClassLoaderOfInitializingThread = true
11. org.quartz.jobStore.misfireThreshold = 60000
13. #orgorg.quartz.jobStore.class = org.quartz.simpl.RAMJobStore
15. orgorg.quartz.jobStore.class = org.quartz.impl.jdbcjobstore.JobStoreTX
16. #orgorg.quartz.jobStore.driverDelegateClass=org.quartz.impl.jdbcjobstore.HSQLDBDelegate
17. orgorg.quartz.jobStore.driverDelegateClass=org.quartz.impl.jdbcjobstore.StdJDBCDelegate
18. #org.quartz.jobStore.useProperties = true
19. org.quartz.jobStore.tablePrefix = QRTZ\_
20. org.quartz.jobStore.isClustered = false
21. org.quartz.jobStore.maxMisfiresToHandleAtATime=1

这里面没有数据源相关的配置部分，采用spring注入datasource的方式已经进行了配置。  
  
六、测试  
运行如下测试类

**Java代码  [收藏代码](javascript:void())**

1. **package** com.sundoctor.example.test;
3. **import** java.text.ParseException;
4. **import** java.text.SimpleDateFormat;
5. **import** java.util.Date;
7. **import** org.quartz.SchedulerException;
8. **import** org.springframework.context.ApplicationContext;
9. **import** org.springframework.context.support.ClassPathXmlApplicationContext;
11. **import** com.sundoctor.quartz.service.SchedulerService;
13. **public** **class** MainTest {
15. /\*\*
16. \* @param args
17. \* @throws SchedulerException
18. \*/
19. **public** **static** **void** main(String[] args) **throws** SchedulerException {
21. ApplicationContext springContext = **new** ClassPathXmlApplicationContext(**new** String[] {
22. "classpath:applicationContext.xml", "classpath:applicationContext-quartz.xml" });
23. SchedulerService schedulerService = springContext.getBean("schedulerService", SchedulerService.**class**);
25. // 执行业务逻辑...
27. // 设置高度任务
28. // 每10秒中执行调试一次
29. schedulerService.schedule("0/10 \* \* ? \* \* \*");
31. Date startTime = parse("2014-08-19 16:33:00");
32. Date endTime = parse("2014-08-22 21:10:00");
34. // 2014-08-19 16:33:00开始执行调度
35. schedulerService.schedule(startTime);
37. // 2014-08-19 16:33:00开始执行调度，2014-08-22 21:10:00结束执行调试
38. schedulerService.schedule(startTime, endTime);
40. // 2014-08-19 16:33:00开始执行调度，执行5次结束
41. schedulerService.schedule(startTime, 5);
43. // 2014-08-19 16:33:00开始执行调度，每隔20秒执行一次，执行5次结束
44. schedulerService.schedule(startTime, 5, 20);
46. // 等等，查看com.sundoctor.quartz.service.SchedulerService
48. }
50. **private** **static** Date parse(String dateStr) {
51. SimpleDateFormat format = **new** SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
52. **try** {
53. **return** format.parse(dateStr);
54. } **catch** (ParseException e) {
55. **throw** **new** RuntimeException(e);
56. }
57. }
59. }

输出

**Xml代码  [收藏代码](javascript:void())**

1. [2014-08-19 22:31:50]INFO  com.sundoctor.example.service.SimpleService(line:14) -7e132a81-6d24-44e1-8d6c-15cdaeefb2ce
2. [2014-08-19 22:32:00]INFO  com.sundoctor.example.service.SimpleService(line:14) -7e132a81-6d24-44e1-8d6c-15cdaeefb2ce
3. [2014-08-19 22:32:10]INFO  com.sundoctor.example.service.SimpleService(line:14) -7e132a81-6d24-44e1-8d6c-15cdaeefb2ce
4. [2014-08-19 22:32:20]INFO  com.sundoctor.example.service.SimpleService(line:14) -7e132a81-6d24-44e1-8d6c-15cdaeefb2ce
5. [2014-08-19 22:32:30]INFO  com.sundoctor.example.service.SimpleService(line:14) -7e132a81-6d24-44e1-8d6c-15cdaeefb2ce
6. [2014-08-19 22:32:40]INFO  com.sundoctor.example.service.SimpleService(line:14) -7e132a81-6d24-44e1-8d6c-15cdaeefb2ce

这样只是简单的将quartz trigger名称打印出来。  
  
这样通过SchedulerService就可以动态配置调度时间。其实SchedulerService 还可扩展，比如可以注入多个JobDetail，调度不同的JobDetail。