### [spring整合quartz并持久化](http://haiziwoainixx.iteye.com/blog/1838055)

****博客分类：****

* [实用文档](http://haiziwoainixx.iteye.com/category/126161)
* [三大框架](http://haiziwoainixx.iteye.com/category/148530)

spring整合quartz有两种方式:

一.常见是使用配置文件,将定时任务保存到内存中

简单示例:

**Xml代码  [IMG_256](http://haiziwoainixx.iteye.com/blog/javascript:void())**

1. <!-- 短信催还提醒任务调度 -->
2. **<bean** id="overdueRecall"
3. class="com.sursen.souba.ddlibserve.quartz.OverdueRecallTimerTask" **/>**

6. <!--定义定时执行overdueRecallTimerTask 这个bean中的overdueRecall()方法-->
7. **<bean** id="overdueRecallTask"
8. class="org.springframework.scheduling.quartz.MethodInvokingJobDetailFactoryBean"**>**
9. **<property** name="targetObject"**>**
10. **<ref** bean="overdueRecall" **/>**
11. **</property>**
12. **<property** name="targetMethod"**>**
13. **<value>**overdueRecall**</value>**
14. **</property>**
15. **</bean>**

18. <!--触发器的bean的设置，要触发的jobDetail是overdueRecallTask-->
19. **<bean** id="overdueRecallTrigger"
20. class="org.springframework.scheduling.quartz.CronTriggerBean"**>**
21. **<property** name="jobDetail"**>**
22. **<ref** bean="overdueRecallTask" **/>**
23. **</property>**
24. **<property** name="cronExpression"**>**
25. <!-- 每天17:00运行 -->
26. **<value>**0 0 17 \* \* ?**</value>**
27. <!--<value>0 05 18 \* \* ?</value>-->
28. **</property>**
29. **</bean>**


33. <!--管理触发器的总设置,管理我们的触发器列表,可以在bean的list中放置多个触发器。-->
34. **<bean** autowire="no"
35. class="org.springframework.scheduling.quartz.SchedulerFactoryBean"**>**
36. **<property** name="triggers"**>**
37. **<list>**
38. **<ref** local="overdueRecallTrigger" **/>**
39. **</list>**
40. **</property>**
41. **</bean>**

 这中方式配置简单,但是存在问题

1.定时任务信息都报错在内存中,服务器重启会丢失信息

2.每个定时任务都是一串配置,定时任务多了不好管理

3.任务时间修改后要重新发布项目

二.quartz持久化(quartz1.6,最新版本的quartz稍有变动)

研究了一下quartz的相关文档,quartz本身就提供将任务和触发器持久化到数据库中的功能

详见[<Quartz-Job-Scheduling-Framework>](http://download.csdn.net/detail/wacde_mail/3981584)一书

要实现很简单:

 1.****安装 Quartz 数据库表****  
Quartz 包括了所有被支持的数据库平台的 SQL 脚本。你能在 <quartz\_home>/docs/dbTables 目录下找到那些 SQL 脚本，这里的 <quartz\_home> 是解压 Quartz 分发包后的目录。  
2.****配置** **JobStoreTX****  
  
要告诉 Quartz 运行环境你想使用一个别的 JobStore 而不是默认的 RAMJobStore，你必须配置几个属性。配置它们的顺序无关紧要，只要保证在第一次运行程序之前都做了设置。  
  
在你的classpath下加入一个配置文件quartz.properties参数如下:

****表 6.3. 可用于设置** **JobStoreTX** **的配置属性****

|  |  |
| --- | --- |
| ****属性**** | ****默认值**** |
| org.quartz.jobStore.driverDelegateClass |  |
| ****描述****：能理解不同数据库系统中某一特定方言的驱动代理 |  |
| org.quartz.jobStore.dataSource |  |
| ****描述****：用于 quartz.properties 中数据源的名称 |  |
| org.quartz.jobStore.tablePrefix | QRTZ\_ |
| ****描述****：指定用于 Scheduler 的一套数据库表名的前缀。假如有不同的前缀，Scheduler 就能在同一数据库中使用不同的表。 |  |
| org.quartz.jobStore.userProperties | False |
| ****描述：****"use properties" 标记指示着持久性 JobStore 所有在 JobDataMap 中的值都是字符串，因此能以 名-值 对的形式存储，而不用让更复杂的对象以序列化的形式存入 BLOB 列中。这样会更方便，因为让你避免了发生于序列化你的非字符串的类到 BLOB 时的有关类版本的问题。 |  |
| org.quartz.jobStore.misfireThreshold | 60000 |
| ****描述****：在 Trigger 被认为是错过触发之前，Scheduler 还容许 Trigger 通过它的下次触发时间的毫秒数(译者注：据原文翻译，真的不好理解，实际效果可参看：[http://www.blogjava.net/Unmi/archive/2007/10/23/153413.html](http://haiziwoainixx.iteye.com/Unmi/archive/2007/10/23/153413.html" \t "http://haiziwoainixx.iteye.com/blog/_blank) 我在评论中的实验)。默认值(假如你未在配置中存在这一属性条目) 是 60000(60 秒)。这个不仅限于JDBC-JobStore；它也可作为 RAMJobStore 的参数 |  |
| org.quartz.jobStore.isClustered | False |
| ****描述****：设置为 true 打开集群特性。如果你有多个 Quartz 实例在用同一套数据库时，这个属性就必须设置为 true。 |  |
| org.quartz.jobStore.clusterCheckinInterval | 15000 |
| ****描述****：设置一个频度(毫秒)，用于实例报告给集群中的其他实例。这会影响到侦测失败实例的敏捷度。它只用于设置了 isClustered 为 true 的时候。 |  |
| org.quartz.jobStore.maxMisfiresToHandleAtATime | 20 |
| ****描述****：这是 JobStore 能处理的错过触发的 Trigger 的最大数量。处理太多(超过两打) 很快会导致数据库表被锁定够长的时间，这样就妨碍了触发别的(还未错过触发) trigger 执行的性能。 |  |
| org.quartz.jobStore.dontSetAutoCommitFalse | False |
| ****描述****：设置这个参数为 true 会告诉 Quartz 从数据源获取的连接后不要调用它的setAutoCommit(false) 方法。这在少些情况下是有帮助的，比如假如你有这样一个驱动，它会抱怨本来就是关闭的又来调用这个方法。这个属性默认值是 false，因为大多数的驱动都要求调用 setAutoCommit(false)。 |  |
| org.quartz.jobStore.selectWithLockSQL | SELECT \* FROM {0}LOCKS WHERE LOCK\_NAME = ? FOR UPDATE |
| ****描述****：这必须是一个从 LOCKS 表查询一行并对这行记录加锁的 SQL 语句。假如未设置，默认值就是 SELECT \* FROM {0}LOCKS WHERE LOCK\_NAME = ? FOR UPDATE，这能在大部分数据库上工作。{0} 会在运行期间被前面你配置的TABLE\_PREFIX 所替换。 |  |
| org.quartz.jobStore.txIsolationLevelSerializable | False |
| ****描述****：值为 true 时告知 Quartz(当使用 JobStoreTX 或 CMT) 调用 JDBC 连接的setTransactionIsolation(Connection.TRANSACTION\_SERIALIZABLE) 方法。这有助于阻止某些数据库在高负载和长时间事物时锁的超时。 |  |

示例,使用的是sqlserver数据库:

**Xml代码  [IMG_257](http://haiziwoainixx.iteye.com/blog/javascript:void())**

1. #org.quartz.scheduler.instanceName = Mscheduler
2. org.quartz.scheduler.instanceId = AUTO
4. #============================================================================
5. # Configure ThreadPool
6. #============================================================================
8. orgorg.quartz.threadPool.class = org.quartz.simpl.SimpleThreadPool
9. org.quartz.threadPool.threadCount = 3
10. org.quartz.threadPool.threadPriority = 5
12. #============================================================================
13. # Configure JobStore
14. #============================================================================
16. #orgorg.quartz.jobStore.class = org.quartz.simpl.RAMJobStore
17. orgorg.quartz.jobStore.class = org.quartz.impl.jdbcjobstore.JobStoreTX
18. orgorg.quartz.jobStore.driverDelegateClass = org.quartz.impl.jdbcjobstore.MSSQLDelegate
19. org.quartz.jobStore.useProperties = true
20. org.quartz.jobStore.dataSource = myDS
21. org.quartz.jobStore.tablePrefix = QRTZ\_
22. org.quartz.jobStore.isClustered = false
23. org.quartz.jobStore.maxMisfiresToHandleAtATime=1
24. #============================================================================
25. # Configure Datasources
26. #============================================================================
28. org.quartz.dataSource.myDS.driver =  net.sourceforge.jtds.jdbc.Driver
29. org.quartz.dataSource.myDS.URL =  jdbc\:jtds\:sqlserver\://ip/dbk
30. org.quartz.dataSource.myDS.user = db2
31. org.quartz.dataSource.myDS.password = db2
32. org.quartz.dataSource.myDS.maxConnections = 5
34. #============================================================================
35. # Configure Plugins
36. #============================================================================
38. #orgorg.quartz.plugin.triggHistory.class = org.quartz.plugins.history.LoggingJobHistoryPlugin
40. #orgorg.quartz.plugin.jobInitializer.class = org.quartz.plugins.xml.JobInitializationPlugin
41. #org.quartz.plugin.jobInitializer.fileNames = jobs.xml
42. #org.quartz.plugin.jobInitializer.overWriteExistingJobs = true
43. #org.quartz.plugin.jobInitializer.failOnFileNotFound = true
44. #org.quartz.plugin.jobInitializer.scanInterval = 10
45. #org.quartz.plugin.jobInitializer.wrapInUserTransaction = false

然后可直接运行以下测试类:

**Java代码  [IMG_258](http://haiziwoainixx.iteye.com/blog/javascript:void())**

1. **package** com.sursen.test.service;
3. **import** java.text.ParseException;
5. **import** org.apache.commons.lang.StringUtils;
6. **import** org.quartz.CronTrigger;
7. **import** org.quartz.Job;
8. **import** org.quartz.JobDetail;
9. **import** org.quartz.Scheduler;
10. **import** org.quartz.SchedulerException;
11. **import** org.quartz.SchedulerFactory;
12. **import** org.quartz.Trigger;
13. **import** org.quartz.impl.StdSchedulerFactory;

16. **public** **class** QuartzTest  {
17. **private** **static** SchedulerFactory sf = **new** StdSchedulerFactory();
18. **private** **static** String JOB\_GROUP\_NAME = "ddlib";
19. **private** **static** String TRIGGER\_GROUP\_NAME = "ddlibTrigger";

22. /\*\*添加一个定时任务，使用默认的任务组名，触发器名，触发器组名\*/
23. **public** **static** **void** addJob(String jobName,Job job,String cronExpression)
24. **throws** SchedulerException, ParseException{
25. addJob(jobName,**null**,jobName,**null**,job,cronExpression);
26. }
28. /\*\*
29. \* 添加一个定时任务
30. \* @param jobName 任务名
31. \* @param jobGroupName 任务组名
32. \* @param triggerName 触发器名
33. \* @param triggerGroupName 触发器组名
34. \* @param job     任务
35. \* @param cronExpression    时间设置，参考quartz说明文档
36. \*/
37. **public** **static** **void** addJob(String jobName,String jobGroupName,
38. String triggerName,String triggerGroupName,Job job,String cronExpression)
39. **throws** SchedulerException, ParseException{
40. **if**(StringUtils.isBlank(jobGroupName)){
41. jobGroupName = JOB\_GROUP\_NAME;
42. }
43. **if**(StringUtils.isBlank(triggerGroupName)){
44. triggerGroupName = TRIGGER\_GROUP\_NAME;
45. }
46. Scheduler sched = sf.getScheduler();
47. JobDetail jobDetail = **new** JobDetail(jobName, jobGroupName, job.getClass());//任务名，任务组，任务执行类
48. CronTrigger  trigger = **new** CronTrigger(jobName,triggerGroupName,cronExpression);//触发器名,触发器组,cron表达式
49. sched.scheduleJob(jobDetail,trigger);
50. //启动
51. **if**(!sched.isShutdown()){
52. sched.start();
53. }
54. }
56. /\*\*
57. \* 修改一个任务的触发时间(使用默认的任务组名，触发器名，触发器组名)
58. \*/
59. **public** **static** **void** modifyJobTime(String jobName,String cronExpression)
60. **throws** SchedulerException, ParseException{
61. modifyJobTime(jobName, **null**, cronExpression);
62. }
64. /\*\*
65. \* 修改一个任务的触发时间
66. \*/
67. **public** **static** **void** modifyJobTime(String triggerName,String triggerGroupName,
68. String cronExpression)**throws** SchedulerException, ParseException{
69. **if**(StringUtils.isBlank(triggerGroupName)){
70. triggerGroupName = TRIGGER\_GROUP\_NAME;
71. }
72. Scheduler sched = sf.getScheduler();
73. Trigger trigger = sched.getTrigger(triggerName,triggerGroupName);
74. **if**(trigger != **null**){
75. CronTrigger ct = (CronTrigger)trigger;
76. //修改时间
77. ct.setCronExpression(cronExpression);
78. //重启触发器
79. sched.resumeTrigger(triggerName,triggerGroupName);
80. }
81. }
83. /\*\*移除一个任务和触发器(使用默认的任务组名，触发器名，触发器组名)\*/
84. **public** **static** **void** removeJob(String jobName,String triggerName)
85. **throws** SchedulerException{
86. removeJob(jobName, **null**, triggerName, **null**);
87. }
89. /\*\*移除一个任务和触发器 \*/
90. **public** **static** **void** removeJob(String jobName,String jobGroupName,
91. String triggerName,String triggerGroupName)
92. **throws** SchedulerException{
93. **if**(StringUtils.isBlank(jobGroupName)){
94. jobGroupName = JOB\_GROUP\_NAME;
95. }
96. **if**(StringUtils.isBlank(triggerGroupName)){
97. triggerGroupName = TRIGGER\_GROUP\_NAME;
98. }
99. Scheduler sched = sf.getScheduler();
100. sched.pauseTrigger(triggerName,triggerGroupName);//停止触发器
101. sched.unscheduleJob(triggerName,triggerGroupName);//移除触发器
102. sched.deleteJob(jobName,jobGroupName);//删除任务
103. }


107. **public** **static** **void** main(String[] args) **throws** SchedulerException, ParseException {
108. //      addJob("test", new TestJob(), "\*/5 \* \* \* \* ?");
109. //      addJob("zht", new TestJob(), "\*/10 \* \* \* \* ?");
110. //      removeJob("myJob","myJobGroup", "myTrigger","myTriggerGroup");
111. removeJob("test","test");
112. removeJob("zht","zht");
114. }
115. }

三.spring下quartz的持久化

在spring配置文件中加入以下代码

**Xml代码  [IMG_259](http://haiziwoainixx.iteye.com/blog/javascript:void())**

1. <!-- quartz持久化存储  -->
2. **<bean** id="scheduler"
3. class="org.springframework.scheduling.quartz.SchedulerFactoryBean" **>**
4. <!--   **<property** name="dataSource"**>**
5. **<ref** bean="ddlibserveDataSource" **/>**
6. **</property>**   --**>**
7. <!--  <property name="schedulerName" value="Mscheduler" />   -->
8. **<property** name="configLocation" value="classpath:quartz.properties" **/>**
9. **<property** name="applicationContextSchedulerContextKey"
10. value="applicationContextKey" **/>**
11. **<property** name="autoStartup" value="true" **/>**
12. **</bean>**

之后将

**Xml代码  [IMG_260](http://haiziwoainixx.iteye.com/blog/javascript:void())**

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

注入到所需要的service中即可.

其中

**Xml代码  [IMG_261](http://haiziwoainixx.iteye.com/blog/javascript:void())**

1. <!--   **<property** name="dataSource"**>**
2. **<ref** bean="ddlibserveDataSource" **/>**
3. **</property>**   --**>**

 为注入的数据源,

如果使用这种方式那配置文件中的相关配置要注释掉.

或者将配置文件中信息写入到spring配置文件中:

**Xml代码  [IMG_262](http://haiziwoainixx.iteye.com/blog/javascript:void())**

1. <!-- quartz持久化存储  -->
2. **<bean** id="scheduler"
3. class="org.springframework.scheduling.quartz.SchedulerFactoryBean" **>**
4. **<property** name="dataSource"**>**
5. **<ref** bean="ddlibserveDataSource" **/>**
6. **</property>**
8. <!--方式一 : quartz.properties配置文件是放在classpath下
9. **<property** name="configLocation" value="classpath:quartz.properties" **/>** --**>**
11. <!--方式二 : quartz.properties配置文件是放在WEB-INF下
12. **<property** name="configLocation" value="/WEB-INF/quartz.properties" **/>** --**>**
14. <!--方式三 : quartz.properties配置文件写入到配置XML中 -->
15. **<property** name="quartzProperties"**>**
16. **<props>**
17. **<prop** key="org.quartz.scheduler.instanceName"**>**quartzScheduler**</prop>**
18. **<prop** key="org.quartz.scheduler.instanceId"**>**AUTO**</prop>**
19. **<prop** key="org.quartz.threadPool.class"**>**org.quartz.simpl.SimpleThreadPool**</prop>**
20. **<prop** key="org.quartz.threadPool.threadCount"**>**3**</prop>**
21. **<prop** key="org.quartz.threadPool.threadPriority"**>**5**</prop>**
22. **<prop** key="org.quartz.jobStore.misfireThreshold"**>**60000**</prop>**
23. **<prop** key="org.quartz.jobStore.class"**>**org.quartz.impl.jdbcjobstore.JobStoreTX**</prop>**
24. **<prop** key="org.quartz.jobStore.driverDelegateClass"**>**org.quartz.impl.jdbcjobstore.MSSQLDelegate **</prop>**
25. **<prop** key="org.quartz.jobStore.selectWithLockSQL"**>**SELECT \* FROM {0}LOCKS UPDLOCK WHERE LOCK\_NAME = ?**</prop>**
26. **<prop** key="org.quartz.jobStore.tablePrefix"**>**QRTZ\_**</prop>**
27. **<prop** key="org.quartz.jobStore.isClustered"**>**true**</prop>**
28. **<prop** key="org.quartz.jobStore.clusterCheckinInterval"**>**20000**</prop>**
29. **</props>**
30. **</property>**
31. **<property** name="applicationContextSchedulerContextKey"  value="applicationContextKey" **/>**
32. **<property** name="autoStartup" value="true" **/>**
33. **</bean>**

期间遇到的问题留作记录:

1.持久化时遇到报字段不正确或表不存在的问题

原因: 我使用的是quartz 2.1建的表而jar包是1.6的

解决方法: 将建表语句和jar包统一版本即可

2.报dataSource name不存在的问题

原因: quartz.properties中的 org.quartz.jobStore.dataSource = myDS

忘记打开或者与数据源配置没有保持一致

解决方法: 打开保持一致即可

3.使用注入dataSource时报

Failure obtaining db row lock: 第 1 行: FOR UPDATE 子句仅允许用于 DECLARE CU...

改为数据源写在quartz.properties中解决

还不清楚为什么注入不行(解决)

|  |  |
| --- | --- |
| org.quartz.jobStore.selectWithLockSQL | SELECT \* FROM {0}LOCKS WHERE LOCK\_NAME = ? FOR UPDATE |
| ****描述****：这必须是一个从 LOCKS 表查询一行并对这行记录加锁的 SQL 语句。假如未设置，默认值就是 SELECT \* FROM {0}LOCKS WHERE LOCK\_NAME = ? FOR UPDATE，这能在大部分数据库上工作。{0} 会在运行期间被前面你配置的TABLE\_PREFIX 所替换。 |  |

 这条sql在sqlserver2000中不能运行修改为:

**Xml代码  [IMG_263](http://haiziwoainixx.iteye.com/blog/javascript:void())**

1. org.quartz.jobStore.selectWithLockSQL=SELECT \* FROM {0}LOCKS UPDLOCK WHERE LOCK\_NAME = ?

 问题解决

4. <property name="configLocation" value="classpath:quartz.properties" />

这种方式配置quartz.properties文件需要放在classpath下,而我们项目中都是放在Web-info下

可直接修改为:

**Xml代码  [IMG_264](http://haiziwoainixx.iteye.com/blog/javascript:void())**

1. **<property** name="configLocation" value="/WEB-INF/quartz.properties" **/>**

5.在spring配置文件中

**Xml代码  [IMG_265](http://haiziwoainixx.iteye.com/blog/javascript:void())**

1. **<bean** id="quartzServiceImpl"
2. class="com.sursen.test.service.impl.QuartzServiceImpl"**>**
3. **<property** name="scheduler"**>**
4. **<ref** bean="scheduler" **/>**
5. **</property>**
6. **</bean>**

 以这种方式将scheduler即StdScheduler  scheduler注入到相关service中报找不到该property的错误

而service中是肯定存在的 : private StdScheduler  scheduler;

改为spring的注解形式注入

**Java代码  [IMG_266](http://haiziwoainixx.iteye.com/blog/javascript:void())**

1. @Autowired
2. **private** StdScheduler  scheduler;

也不行,再改为

**Java代码  [IMG_267](http://haiziwoainixx.iteye.com/blog/javascript:void())**

1. @Resource
2. **private** StdScheduler  scheduler;

 或者

**Java代码  [IMG_268](http://haiziwoainixx.iteye.com/blog/javascript:void())**

1. @Qualifier("scheduler")
2. **private** StdScheduler  scheduler;

则没有问题,具体原因还在查找