# 介绍

**Quartz体系结构：**

明白Quartz怎么用，首先要了解Scheduler(调度器)、Job(任务)和Trigger(触发器)这3个核心的概念。

**1. Job：**是一个接口，只定义一个方法execute(JobExecutionContext context)，在实现接口的execute方法中编写所需要定时执行的Job(任务)， JobExecutionContext类提供了调度应用的一些信息。Job运行时的信息保存在JobDataMap实例中；

**2. JobDetail：**Quartz每次调度Job时， 都重新创建一个Job实例， 所以它不直接接受一个Job的实例，相反它接收一个Job实现类(JobDetail:描述Job的实现类及其它相关的静态信息，如Job名字、描述、关联监听器等信息)，以便运行时通过newInstance()的反射机制实例化Job。

**3. Trigger：**是一个类，描述触发Job执行的时间触发规则。主要有SimpleTrigger和CronTrigger这两个子类。当且仅当需调度一次或者以固定时间间隔周期执行调度，SimpleTrigger是最适合的选择；而CronTrigger则可以通过Cron表达式定义出各种复杂时间规则的调度方案：如工作日周一到周五的15：00~16：00执行调度等；

  Cron表达式的格式：秒 分 时 日 月 周 年(可选)。  
               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 #      SUN, MON, TUE, WED, THU, FRI and SAT  
               年 (可选字段)     empty, 1970-2099            , - \* /  
  
               “?”字符：表示不确定的值  
               “,”字符：指定数个值  
               “-”字符：指定一个值的范围  
               “/”字符：指定一个值的增加幅度。n/m表示从n开始，每次增加m  
               “L”字符：用在日表示一个月中的最后一天，用在周表示该月最后一个星期X  
               “W”字符：指定离给定日期最近的工作日(周一到周五)  
               “#”字符：表示该月第几个周X。6#3表示该月第3个周五

         Cron表达式范例：  
                 每隔5秒执行一次：\*/5 \* \* \* \* ?  
                 每隔1分钟执行一次：0 \*/1 \* \* \* ?  
                 每天23点执行一次：0 0 23 \* \* ?  
                 每天凌晨1点执行一次：0 0 1 \* \* ?  
                 每月1号凌晨1点执行一次：0 0 1 1 \* ?  
                 每月最后一天23点执行一次：0 0 23 L \* ?  
                 每周星期六凌晨1点实行一次：0 0 1 ? \* L  
                 在26分、29分、33分执行一次：0 26,29,33 \* \* \* ?  
                 每天的0点、13点、18点、21点都执行一次：0 0 0,13,18,21 \* \* ?

**4. Calendar：**org.quartz.Calendar和java.util.Calendar不同， 它是一些日历特定时间点的集合（可以简单地将org.quartz.Calendar看作java.util.Calendar的集合——java.util.Calendar代表一个日历时间点，无特殊说明后面的Calendar即指org.quartz.Calendar）。 一个Trigger可以和多个Calendar关联， 以便排除或包含某些时间点。

假设，我们安排每周星期一早上10:00执行任务，但是如果碰到法定的节日，任务则不执行，这时就需要在Trigger触发机制的基础上使用Calendar进行定点排除。针对不同时间段类型，Quartz在org.quartz.impl.calendar包下提供了若干个Calendar的实现类，如AnnualCalendar、MonthlyCalendar、WeeklyCalendar分别针对每年、每月和每周进行定义；

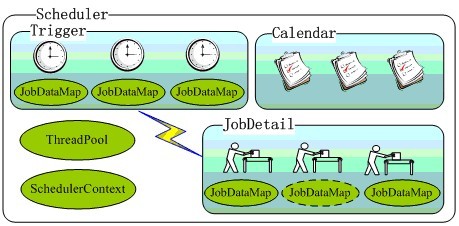
**5. Scheduler：**代表一个Quartz的独立运行容器， Trigger和JobDetail可以注册到Scheduler中， 两者在Scheduler中拥有各自的组及名称， 组及名称是Scheduler查找定位容器中某一对象的依据， Trigger的组及名称必须唯一， JobDetail的组和名称也必须唯一（但可以和Trigger的组和名称相同，因为它们是不同类型的）。Scheduler定义了多个接口方法， 允许外部通过组及名称访问和控制容器中Trigger和JobDetail。

Scheduler可以将Trigger绑定到某一JobDetail中， 这样当Trigger触发时， 对应的Job就被执行。一个Job可以对应多个Trigger， 但一个Trigger只能对应一个Job。可以通过SchedulerFactory创建一个Scheduler实例。Scheduler拥有一个SchedulerContext，它类似于ServletContext，保存着Scheduler上下文信息，Job和Trigger都可以访问SchedulerContext内的信息。SchedulerContext内部通过一个Map，以键值对的方式维护这些上下文数据，SchedulerContext为保存和获取数据提供了多个put()和getXxx()的方法。可以通过Scheduler# getContext()获取对应的SchedulerContext实例；

**6. ThreadPool：**Scheduler使用一个线程池作为任务运行的基础设施，任务通过共享线程池中的线程提高运行效率。  
Job有一个StatefulJob子接口，代表有状态的任务，该接口是一个没有方法的标签接口，其目的是让Quartz知道任务的类型，以便采用不同的执行方案。无状态任务在执行时拥有自己的JobDataMap拷贝，对JobDataMap的更改不会影响下次的执行。而有状态任务共享共享同一个JobDataMap实例，每次任务执行对JobDataMap所做的更改会保存下来，后面的执行可以看到这个更改，也即每次执行任务后都会对后面的执行发生影响。  
正因为这个原因，无状态的Job可以并发执行，而有状态的StatefulJob不能并发执行，这意味着如果前次的StatefulJob还没有执行完毕，下一次的任务将阻塞等待，直到前次任务执行完毕。有状态任务比无状态任务需要考虑更多的因素，程序往往拥有更高的复杂度，因此除非必要，应该尽量使用无状态的Job。  
如果Quartz使用了数据库持久化任务调度信息，无状态的JobDataMap仅会在Scheduler注册任务时保持一次，而有状态任务对应的JobDataMap在每次执行任务后都会进行保存。  
Trigger自身也可以拥有一个JobDataMap，其关联的Job可以通过JobExecutionContext#getTrigger().getJobDataMap()获取Trigger中的JobDataMap。不管是有状态还是无状态的任务，在任务执行期间对Trigger的JobDataMap所做的更改都不会进行持久，也即不会对下次的执行产生影响。

Quartz拥有完善的事件和监听体系，大部分组件都拥有事件，如任务执行前事件、任务执行后事件、触发器触发前事件、触发后事件、调度器开始事件、关闭事件等等，可以注册相应的监听器处理感兴趣的事件。

下图描述了Scheduler的内部组件结构，SchedulerContext提供Scheduler全局可见的上下文信息，每一个任务都对应一个JobDataMap，虚线表达的JobDataMap表示对应有状态的任务：



# 简单的定时任务

使用Quartz需要导入jar包：

<dependency>  
 <groupId>org.quartz-scheduler</groupId>  
 <artifactId>quartz</artifactId>  
 <version>2.2.3</version>  
</dependency>

## 创建工作类

创建一个工作类，需要实现Job接口

**package simple**;  
  
**import org.quartz.**Job;  
**import org.quartz.**JobDetail;  
**import org.quartz.**JobExecutionContext;  
**import org.quartz.JobExecutionException**;  
  
**public class HelloQuartz implements** Job **{  
 //执行  
 public void** execute(JobExecutionContext **context**) **throws JobExecutionException {  
 //创建工作详情** JobDetail **detail=context**.**getJobDetail**();  
 **//获取工作的名称  
 String name=detail**.**getJobDataMap**().getString(**"name"**);  
 **String job=detail**.**getKey**().getGroup();  
 **System**.***out***.println(**"任务调度：组："+job+",工作名："+name+"---->今日整点抢购，不容错过！"**);  
 **}  
}**

## 测试任务调度

**package simple**;  
  
**import org.quartz.\***;  
**import org.quartz.impl.StdSchedulerFactory**;  
  
**public class QuartzTest {  
 public static void** main(**String**[] **args**) **{  
 try{  
 //创建scheduler，执行计划** Scheduler **scheduler = StdSchedulerFactory**.getDefaultScheduler();  
 **//定义一个Trigger,触发条件类** Trigger **trigger = TriggerBuilder**.newTrigger().  
 withIdentity(**"trigger1"**, **"group1"**) **//定义name/group** .startNow()**//一旦加入scheduler，立即生效** .withSchedule(**SimpleScheduleBuilder**.simpleSchedule() **//使用SimpleTrigger** .withIntervalInSeconds(**1**) **//每隔一秒执行一次** .repeatForever()) **//一直执行，奔腾到老不停歇** .build();  
 **//定义一个JobDetail** JobDetail **job = JobBuilder**.newJob(**HelloQuartz**.**class**) **//定义Job类为HelloQuartz类，这是真正的执行逻辑所在** .withIdentity(**"job1"**, **"group1"**) **//定义name/group** .usingJobData(**"name"**, **"quartz"**) **//定义属性** .build();  
 **//加入这个调度  
 scheduler**.**scheduleJob**(**job**, **trigger**);  
 **//启动任务调度  
 scheduler**.**start**();  
 **}catch** (**Exception ex**)**{  
 ex**.printStackTrace();  
 **}  
 }  
}**

# 动态操作任务

## 新建任务类

**package spring**;  
  
**import org.quartz.**Job;  
**import org.quartz.**JobExecutionContext;  
**import org.quartz.JobExecutionException**;  
  
**public class MyJob implements** Job **{  
 private static int** counter **= 1**;  
  
 **public void** execute(JobExecutionContext **jobExecutionContext**) **throws JobExecutionException {  
 System**.***out***.println(**"(第 " +** counter **+ " 次,预告通知)"**);  
 counter**++**;  
 **}  
}**

## 编写任务工具类

**package spring**;  
  
**import org.quartz.\***;  
**import org.quartz.impl.StdSchedulerFactory**;  
  
**public class QuartzManager {  
  
 private static** SchedulerFactory schedulerFactory **= new** StdSchedulerFactory();  
  
 **/\*\*  
 \* @Description: 添加一个定时任务  
 \*  
 \* @param jobName 任务名  
 \* @param jobGroupName 任务组名  
 \* @param triggerName 触发器名  
 \* @param triggerGroupName 触发器组名  
 \* @param jobClass 任务  
 \* @param cron 时间设置，参考quartz说明文档  
 \*/  
 @SuppressWarnings**(**{ "unchecked"**, **"rawtypes" }**)  
 **public static void** addJob(**String jobName**, **String jobGroupName**,  
 **String triggerName**, **String triggerGroupName**, **Class jobClass**, **String cron**) **{  
 try {** Scheduler **sched =** schedulerFactory.**getScheduler**();  
 **// 任务名，任务组，任务执行类** JobDetail **jobDetail= JobBuilder**.newJob(**jobClass**).withIdentity(**jobName**, **jobGroupName**).build();  
 **// 触发器  
 TriggerBuilder<**Trigger**> triggerBuilder = TriggerBuilder**.newTrigger();  
 **// 触发器名,触发器组  
 triggerBuilder**.withIdentity(**triggerName**, **triggerGroupName**);  
 **triggerBuilder**.startNow();  
 **// 触发器时间设定  
 triggerBuilder**.withSchedule(**CronScheduleBuilder**.cronSchedule(**cron**));  
 **// 创建Trigger对象** CronTrigger **trigger =** (CronTrigger) **triggerBuilder**.build();  
 **// 调度容器设置JobDetail和Trigger  
 sched**.**scheduleJob**(**jobDetail**, **trigger**);  
 **// 启动  
 if** (**!sched**.**isShutdown**()) **{  
 sched**.**start**();  
 **}  
 } catch** (**Exception e**) **{  
 throw new** RuntimeException(**e**);  
 **}  
   
 }  
  
 /\*\*  
 \* @Description: 修改一个任务的触发时间  
 \*  
 \* @param jobName  
 \* @param jobGroupName  
 \* @param triggerName 触发器名  
 \* @param triggerGroupName 触发器组名  
 \* @param cron 时间设置，参考quartz说明文档  
 \*/  
 public static void** modifyJobTime(**String jobName**,  
 **String jobGroupName**, **String triggerName**, **String triggerGroupName**, **String cron**) **{  
 try {** Scheduler **sched =** schedulerFactory.**getScheduler**();  
 **TriggerKey triggerKey = TriggerKey**.triggerKey(**triggerName**, **triggerGroupName**);  
 CronTrigger **trigger =** (CronTrigger) **sched**.**getTrigger**(**triggerKey**);  
 **if** (**trigger == null**) **{  
 return**;  
 **}  
  
 String oldTime = trigger**.**getCronExpression**();  
 **if** (**!oldTime**.equalsIgnoreCase(**cron**)) **{  
 /\*\* 方式一 ：调用 rescheduleJob 开始 \*/  
 // 触发器  
 TriggerBuilder<**Trigger**> triggerBuilder = TriggerBuilder**.newTrigger();  
 **// 触发器名,触发器组  
 triggerBuilder**.withIdentity(**triggerName**, **triggerGroupName**);  
 **triggerBuilder**.startNow();  
 **// 触发器时间设定  
 triggerBuilder**.withSchedule(**CronScheduleBuilder**.cronSchedule(**cron**));  
 **// 创建Trigger对象  
 trigger =** (CronTrigger) **triggerBuilder**.build();  
 **// 方式一 ：修改一个任务的触发时间  
 sched**.**rescheduleJob**(**triggerKey**, **trigger**);  
 **/\*\* 方式一 ：调用 rescheduleJob 结束 \*/  
 /\*\* 方式二：先删除，然后在创建一个新的Job \*/  
 //JobDetail jobDetail = sched.getJobDetail(JobKey.jobKey(jobName, jobGroupName));  
 //Class<? extends Job> jobClass = jobDetail.getJobClass();  
 //removeJob(jobName, jobGroupName, triggerName, triggerGroupName);  
 //addJob(jobName, jobGroupName, triggerName, triggerGroupName, jobClass, cron);  
 }  
 } catch** (**Exception e**) **{  
 throw new** RuntimeException(**e**);  
 **}  
 }  
  
 /\*\*  
 \* @Description: 移除一个任务  
 \*  
 \* @param jobName  
 \* @param jobGroupName  
 \* @param triggerName  
 \* @param triggerGroupName  
 \*/  
 public static void** removeJob(**String jobName**, **String jobGroupName**,  
 **String triggerName**, **String triggerGroupName**) **{  
 try {** Scheduler **sched =** schedulerFactory.**getScheduler**();  
 **TriggerKey triggerKey = TriggerKey**.triggerKey(**triggerName**, **triggerGroupName**);  
 **sched**.**pauseTrigger**(**triggerKey**);**// 停止触发器  
 sched**.**unscheduleJob**(**triggerKey**);**// 移除触发器  
 sched**.**deleteJob**(**JobKey**.jobKey(**jobName**, **jobGroupName**));**// 删除任务  
 } catch** (**Exception e**) **{  
 throw new** RuntimeException(**e**);  
 **}  
 }  
  
 /\*\*  
 \* @Description:启动所有定时任务  
 \*/  
 public static void** startJobs() **{  
 try {** Scheduler **sched =** schedulerFactory.**getScheduler**();  
 **sched**.**start**();  
 **} catch** (**Exception e**) **{  
 throw new** RuntimeException(**e**);  
 **}  
 }  
  
 /\*\*  
 \* @Description:关闭所有定时任务  
 \*/  
 public static void** shutdownJobs() **{  
 try {** Scheduler **sched =** schedulerFactory.**getScheduler**();  
 **if** (**!sched**.**isShutdown**()) **{  
 sched**.**shutdown**();  
 **}  
 } catch** (**Exception e**) **{  
 throw new** RuntimeException(**e**);  
 **}  
 }  
}**

## 测试类

**package spring**;  
  
**public class SimpleTest {  
 public static String** JOB\_NAME **= "动态任务调度"**;  
 **public static String** TRIGGER\_NAME **= "动态任务触发器"**;  
 **public static String** JOB\_GROUP\_NAME **= "XLXXCC\_JOB\_GROUP"**;  
 **public static String** TRIGGER\_GROUP\_NAME **= "XLXXCC\_JOB\_GROUP"**;  
  
 **public static void** main(**String**[] **args**) **{  
 try {  
 System**.***out***.println(**"【系统启动】开始(每1秒输出一次)..."**);  
 **QuartzManager**.addJob(JOB\_NAME, JOB\_GROUP\_NAME, TRIGGER\_NAME, TRIGGER\_GROUP\_NAME, **MyJob**.**class**, **"0/1 \* \* \* \* ?"**);  
  
 **Thread**.sleep(**5000**);  
 **System**.***out***.println(**"【修改时间】开始(每5秒输出一次)..."**);  
 **QuartzManager**.modifyJobTime(JOB\_NAME, JOB\_GROUP\_NAME, TRIGGER\_NAME, TRIGGER\_GROUP\_NAME, **"0/5 \* \* \* \* ?"**);  
  
 **Thread**.sleep(**6000**);  
 **System**.***out***.println(**"【移除定时】开始..."**);  
 **QuartzManager**.removeJob(JOB\_NAME, JOB\_GROUP\_NAME, TRIGGER\_NAME, TRIGGER\_GROUP\_NAME);  
 **System**.***out***.println(**"【移除定时】成功"**);  
 **} catch** (**Exception e**) **{  
 e**.printStackTrace();  
 **}  
 }  
}**

# Spring整合Quartz

导入Spring核心包：

<dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-context</artifactId>  
 <version>4.3.11.RELEASE</version>  
</dependency>  
<dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-context-support</artifactId>  
 <version>4.3.11.RELEASE</version>  
</dependency>  
<dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-tx</artifactId>  
 <version>4.3.11.RELEASE</version>  
</dependency>

## 创建工作类

**package spring**;  
  
**import org.quartz.**Job;  
**import org.quartz.**JobExecutionContext;  
**import org.quartz.JobExecutionException**;  
  
**public class MyJob implements** Job **{  
 private static int** counter **= 1**;  
  
 **public void** execute(JobExecutionContext **jobExecutionContext**) **throws JobExecutionException {  
 System**.***out***.println(**"(第 " +** counter **+ " 次,预告通知)"**);  
 counter**++**;  
 **}  
}**

## Spring配置文件

*<?*xml version="1.0" encoding="UTF-8"*?>*<beans xmlns="http://www.springframework.org/schema/beans"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd">  
 <!--  
 Spring整合Quartz进行配置遵循下面的步骤：  
 1：定义工作任务的Job  
 2：定义触发器Trigger，并将触发器与工作任务绑定  
 3：定义调度器，并将Trigger注册到Scheduler  
 -->  
  
 <!-- 1：定义任务的bean ，这里使用JobDetailFactoryBean,也可以使用MethodInvokingJobDetailFactoryBean ，配置类似-->  
 <bean name="myJob" class="org.springframework.scheduling.quartz.JobDetailFactoryBean">  
 <!-- 指定job的名称 -->  
 <property name="name" value="job1"/>  
 <!-- 指定job的分组 -->  
 <property name="group" value="group1"/>  
 <!-- 指定具体的job类 -->  
 <property name="jobClass" value="spring.MyJob"/>  
 <!-- 必须设置为true，如果为false，当没有活动的触发器与之关联时会在调度器中会删除该任务 -->  
 <property name="durability" value="true"/>  
 <!-- 指定spring容器的key，如果不设定在job中的jobmap中是获取不到spring容器的 -->  
 <property name="applicationContextJobDataKey" value="applicationContext"/>  
 </bean>  
  
 <!-- 2.2：定义触发器的bean，定义一个Cron的Trigger，一个触发器只能和一个任务进行绑定 -->  
 <bean id="cronTrigger" class="org.springframework.scheduling.quartz.CronTriggerFactoryBean">  
 <!-- 指定Trigger的名称 -->  
 <property name="name" value="my\_trigger"/>  
 <!-- 指定Trigger的名称 -->  
 <property name="group" value="my\_trigger\_group"/>  
 <!-- 指定Tirgger绑定的Job -->  
 <property name="jobDetail" ref="myJob"/>  
 <!-- 指定Cron 的表达式 ，当前是每隔5s运行一次 -->  
 <property name="cronExpression" value="0/5 \* \* \* \* ?" />  
 </bean>  
  
 <!-- 3.定义调度器，并将Trigger注册到调度器中 -->  
 <bean id="scheduler" class="org.springframework.scheduling.quartz.SchedulerFactoryBean">  
 <property name="triggers">  
 <list>  
 <ref bean="cronTrigger"/>  
 </list>  
 </property>  
 <property name="autoStartup" value="false" />  
 </bean>  
</beans>

## 测试代码

**package spring**;  
  
**import org.quartz.impl.StdScheduler**;  
**import org.springframework.context.**ApplicationContext;  
**import org.springframework.context.support.ClassPathXmlApplicationContext**;  
  
**public class SpringTest {  
 public static void** main(**String**[] **args**) **throws Exception {** ApplicationContext **applicationContext =  
 new** ClassPathXmlApplicationContext(**"spring-quartz.xml"**);  
 **StdScheduler scheduler =** (**StdScheduler**) **applicationContext**.**getBean**(**"scheduler"**);  
 **scheduler**.start();  
 **System**.***in***.**read**();  
 **}  
}**