QUARTZ

Author：Shine

Version：9.0.2

一、引言

1.1 简介

> Quartz： http://www.quartz-scheduler.org/
>
> 是一个 **定时任务调度框架** 。比如你遇到这样的问题：
>
> - 想在30分钟后，查看订单是否支付，未支付则取消订单
> - 想在每月29号，信用卡自动还款
> - ...
> - 想定时在某个时间，去做某件事(任务)。

> Quartz是要做定时任务的调度，设置好触发时间规则，以及相应的任务(Job)即可。

二、Quartz使用

2.1 导入依赖

```
<dependencies>
    <!--Quartz任务调度-->
    <!-- https://mvnrepository.com/artifact/org.quartz-scheduler/quartz -->
    <dependency>
        <groupId>org.quartz-scheduler</groupId>
        <artifactId>quartz</artifactId>
        <version>2.2.3</version>
    </dependency>
</dependencies>
```

2.2 定义Job

```
/**
 * 工作类的具体实现，即需要定时执行的“某件事”
```

```
 * */
public class HelloQuartz implements Job {
    //执行
    public void execute(JobExecutionContext context) throws JobExecutionException {
         //创建工作详情
        JobDetail jobDetail=context.getJobDetail();
        //获取工作的名称
        String jobName = jobDetail.getKey().getName();//任务名
        String jobGroup = jobDetail.getKey().getGroup();//任务group
        System.out.println("job执行, job: "+jobName+" group:"+jobGroup);
        System.out.println(new Date());
    }
}
```

2.3 API测试

```
public static void main(String[] args) {
        try{
            //创建scheduler, 调度器
            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, 即开始计时
                  .withSchedule(SimpleScheduleBuilder.simpleSchedule()
                                .withIntervalInSeconds(1) //每隔一秒执行一次
                                .repeatForever()) //一直执行, 直到结束时间
                  .endAt(new GregorianCalendar(2019,7,15,16,7,0).getTime());//设置结束时间
            //定义一个JobDetail
            //定义Job类为HelloQuartz类, 这是真正的执行逻辑所在
            JobDetail job = JobBuilder.newJob(HelloQuartz.class)
                    .withIdentity("job1","group1") //定义name/group
                    .build();
            //调度器 中加入 任务和触发器
            scheduler.scheduleJob(job, trigger);
            //启动任务调度
            scheduler.start();
        }catch (Exception ex){
            ex.printStackTrace();
        }
}
```

2.4 配置

```
# 名为: quartz.properties, 放置在classpath下, 如果没有此配置则按默认配置启动
# 指定调度器名称, 非实现类
org.quartz.scheduler.instanceName = DefaultQuartzScheduler
# 指定线程池实现类
org.quartz.threadPool.class = org.quartz.simpl.SimpleThreadPool
# 线程池线程数量
org.quartz.threadPool.threadCount = 10
# 优先级, 默认5
org.quartz.threadPool.threadPriority = 5
# 非持久化job
org.quartz.jobStore.class = org.quartz.simpl.RAMJobStore
```

2.5 核心类说明

> Scheduler: 调度器。所有的调度都是由它控制, 是Quartz的大脑, 所有任务都是由它来管理
>
> Job: 任务, 想定时执行的事情(定义业务逻辑)
>
> JobDetail: 基于Job, 进一步包装。其中关联一个Job, 并为Job指定更详细的属性, 比如标识等
>
> Trigger: 触发器。可以指定给某个任务, 指定任务的触发机制。

三、Trigger

3.1 SimpleTrigger

> 以一定的时间间隔（单位是毫秒）执行的任务。
>
> - 指定起始和截止时间(时间段)
> - 指定时间间隔、执行次数

示例:

```
SimpleTrigger trigger = TriggerBuilder.newTrigger()
                .withIdentity("trigger1", "group1")
                .startNow()
               .withSchedule(SimpleScheduleBuilder.simpleSchedule()
                           .withIntervalInSeconds(1) //每秒执行一次
                           .repeatForever())// 不限执行次数
                .endAt(new GregorianCalendar(2020, 4, 7, 2, 24, 0).getTime())
                .build();
```

```
SimpleTrigger trigger = TriggerBuilder.newTrigger()
                .withIdentity("trigger1", "group1")
                .startNow()
               .withSchedule(SimpleScheduleBuilder.simpleSchedule()
                           .withIntervalInMinutes(3) // 每3分钟执行一次
                           .withRepeatCount(3)) // 执行次数不超过3次
                .endAt(new GregorianCalendar(2020, 4, 7, 2, 24, 0).getTime())
                .build();
```

3.2 CronTrigger 【 重点 】

> 适合于更复杂的任务，它支持类型于Linux Cron的语法（并且更强大）。
>
> - 指定Cron表达式即可

示例：

```
// 每天10:00-12:00，每隔2分钟执行一次
CronTrigger trigger = TriggerBuilder.newTrigger()
                .withIdentity("t1", "g1")
                .withSchedule(CronScheduleBuilder.cronSchedule("0 0/2 10-12 * * ?")).build();
```

3.2.1 Cron表达式组成

> 表达式组成："秒 分 时 日 月 星期几 [年]"，其中"年" 是可选的，一般不指定。
>
> - 如："10 20 18 3 5 ?"代表"5月3日18点20分10秒，星期几不确定 "

位置	时间域	允许值	特殊值
1	秒	0-59	, - * /
2	分钟	0-59	, - * /
3	小时	0-23	, - * /
4	日期	1-31	, - * ? / L W
5	月份	1-12	, - * /
6	星期	1-7	, - * ? / L #
7	年份（可选）		, - * /

3.2.2 Cron表达式符号

> 表达式中可使用的特殊符号的含义如下

符号	语义
星号(*)	可用在所有字段中，表示对应时间域的每一个时刻，例如， 在分钟字段时，表示“每分钟”
问号(?)	该字符只在日期和星期字段中使用，它通常指定为“不确定值”
减号(-)	表达一个范围，如在小时字段中使用“10-12”，则表示从10到12点，即10,11,12
逗号(,)	表达一个列表值，如在星期字段中使用“MON,WED,FRI”，则表示星期一，星期三和星期五
斜杠(/)	x/y表达一个等步长序列，x为起始值，y为增量步长值。如在分钟字段中使用0/15，则表示为0,15,30和45秒，而5/15在分钟字段中表示5,20,35,50
井号(#)	该字符只用在星期字段中，"4#2"代表第二个星期3，“5#4”代表第4个星期四
L	该字符只在日期和星期字段中使用，代表“Last”的意思，但它在两个字段中意思不同。
	如果L用在星期字段里，则表示星期六，等同于7
	L出现在星期字段里，而且在前面有一个数值x，则表示“这个月的最后一个周x”，例如，6L表示该月的最后星期五
	L在日期字段中，表示这个月份的最后一天，如一月的31号，非闰年二月的28号
W	该字符只能出现在日期字段里，是对前导日期的修饰，表示离该日期最近的工作日
	例如15W表示离该月15号最近的工作日，如果该月15号是星期六，则匹配14号星期五；如果15日是星期日，则匹配16号星期一；如果15号 是星期二，那结果就是15号星期二；但必须注意关联的匹配日期不能够跨月
LW组合	在日期字段可以组合使用LW，它的意思是当月的最后一个工作日

3.2.3 Cron表达式示例

> 演示实例

表示式	说明
0 0 12 * * ?	每天12点运行
0 15 10 * * ?	每天10:15运行
0 15 10 * * ? 2008	在2008年的每天10：15运行
0 * 14 * * ?	每天14点到15点之间每分钟运行一次，开始于14:00，结束于14:59。
0 0/5 14 * * ?	每天14点到15点每5分钟运行一次，开始于14:00，结束于14:55。
0 0/5 14,18 * * ?	每天14点到15点每5分钟运行一次，此外每天18点到19点每5钟也运行一次。
0 0-5 14 * * ?	每天14:00点到14:05，每分钟运行一次。
0 0-5/2 14 * * ?	每天14:00点到14:05，每2分钟运行一次。
0 10,44 14 ? 3 4	3月每周三的14:10分和14:44，每分钟运行一次。
0 15 10 ? * 2-6	每周一，二，三，四，五的10:15分运行。
0 15 10 15 * ?	每月15日10:15分运行。
0 15 10 L * ?	每月最后一天10:15分运行。
0 15 10 ? * 6L	每月最后一个星期五10:15分运行。【此时天必须是"?"】
0 15 10 ? * 6L 2007-2009	在2007,2008,2009年每个月的最后一个星期五的10:15分运行。

四、Spring整合Quartz 【 重点 】

4.1 依赖

```
<properties>
  <springframework.version>5.1.6.RELEASE</springframework.version>
  <quartz.version>2.2.3</quartz.version>
</properties>
```

```
<dependencies>

  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context-support</artifactId>
    <version>${springframework.version}</version>
  </dependency>

   <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-tx</artifactId>
    <version>${springframework.version}</version>
  </dependency>

  <dependency>
    <groupId>org.quartz-scheduler</groupId>
    <artifactId>quartz</artifactId>
    <version>${quartz.version}</version>
  </dependency>

  </dependencies>
</project>
```

4.2 定义Job

> 定义一个Job类

```
public class MyJob implements Job {
    public void execute(JobExecutionContext jobExecutionContext) throws JobExecutionException {
        System.err.println("job 执行"+new Date());
    }
}
```

4.3 配置

> 调度器 SchedulerFactoryBean
>
> 触发器 CronTriggerFactoryBean
>
> JobDetail JobDetailFactoryBean

```
<?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="lxJob" 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="job_group1"/>
        <!-- 指定具体的job类 -->
        <property name="jobClass" value="com.qf.quartz.MyJob"/>
    </bean>

    <!-- 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="trigger1"/>
        <!-- 指定Trigger的名称 -->
        <property name="group" value="trigger_group1"/>
        <!-- 指定Tirgger绑定的JobDetail -->
        <property name="jobDetail" ref="lxJob"/>
        <!-- 指定Cron 的表达式 , 当前是每隔5s运行一次 -->
        <property name="cronExpression" value="*/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>
        <!-- 添加 quartz 配置, 如下两种方式均可 -->
        <!--<property name="configLocation" value="classpath:quartz.properties"></property>-->
        <property name="quartzProperties">
```

```
            <value>
                # 指定调度器名称，实际类型为：QuartzScheduler
                org.quartz.scheduler.instanceName = MyScheduler
                # 指定连接池
                org.quartz.threadPool.class = org.quartz.simpl.SimpleThreadPool
                # 连接池线程数量
                org.quartz.threadPool.threadCount = 11
                # 优先级
                org.quartz.threadPool.threadPriority = 5
                # 不持久化job
                org.quartz.jobStore.class = org.quartz.simpl.RAMJobStore
            </value>
        </property>
    </bean>
</beans>
```

4.4 操作

4.4.1 启动任务

> 工厂启动，调度器启动，任务调度开始

```
public static void main(String[] args) throws InterruptedException, SchedulerException {
    // 工厂启动，任务启动，工厂关闭，任务停止
  ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");
}
```

4.4.2 任务操作

4.4.2.1 删除任务

```
public static void main(String[] args) throws InterruptedException, SchedulerException {
    ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");
    System.out.println("==============");
    StdScheduler scheduler = (StdScheduler) context.getBean("scheduler");
    System.out.println(scheduler.getClass());
    Thread.sleep(3000);
    // 删除Job
    scheduler.deleteJob(JobKey.jobKey("job1","job_group1"));
}
```

4.4.2.2 暂停、恢复

```
public static void main(String[] args) throws InterruptedException, SchedulerException {
    ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");
    System.out.println("==============");
    StdScheduler scheduler = (StdScheduler) context.getBean("scheduler");
    System.out.println(scheduler.getClass());
    Thread.sleep(3000);
    // 暂停，恢复工作
    scheduler.pauseJob(JobKey.jobKey("job1","job_group1"));// 暂停工作
    Thread.sleep(3000);
    scheduler.resumeJob(JobKey.jobKey("job1","job_group1"));// 恢复工作
}
```

4.4.2.3 批量操作

```
public static void main(String[] args) throws InterruptedException, SchedulerException {
    ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");
    System.out.println("==============");
    StdScheduler scheduler = (StdScheduler) context.getBean("scheduler");
    System.out.println(scheduler.getClass());
    Thread.sleep(3000);
    GroupMatcher<JobKey> group1 = GroupMatcher.groupEquals("group1");
    scheduler.pauseJobs(group1); // 暂停组中所有工作
    Thread.sleep(2000);
    scheduler.resumeJobs(group1); // 恢复组中所有工作
}
```