# 前言

## 版本与环境说明

本笔记使用的Quartz版本：2.2.1，需要在JDK1.6及其以上版本运行

<!-- <dependency>

<groupId>org.quartz-scheduler</groupId>

<artifactId>quartz</artifactId>

<version>2.2.1</version>

</dependency>

Spring 版本：4.3.8.RELEASE

# Quartz

## Quartz的三个核心概念

调度器：负责调度作业和触发器

触发器：设置作业执行的时间、参数、条件等(简单触发器和Cron触发器)

作业：定时任务内容，被执行的程序

## Quartz的核心接口和类

Job接口：具体任务需要实现的接口Job还有一类为有状态的StatefulJob接口，如果我们需要在上一个作业执行完后，根据其执行结果再进行下次作业的执行，则需要实现此接口。

Trigger抽象类：调度类(Scheduler)在时间到时调用此类，再由trigger类调用指定的定时程序。

Quertz中提供了两类触发器为：SimpleTrigger,CronTrigger。前者用于实现比较简单的定时功能，例如几点开始，几点结束，隔多长时间执行，共执行多少次等，后者提供了使用表达式来描述定时功能，因此适用于比较复杂的定时描述，例如每个月的最后一个周五，每周的周四等。

JobDetail类：具体某个定时程序的详细描述，包括Name,Group,JobDataMap等。

JobExecutionContext类：定时程序执行的run-time的上下文环境，用于得到当前执行的Job的名字，配置的参数等。

JobDataMap类：用于描述一个作业的参数，参数可以为任何基本类型例如String,float等，也可为某个对象的引用.

JobListener,TriggerListener接口：用于监听触发器状态和作业扫行状态，在特写状态执行相应操作。

JobStore类：在哪里执行定进程序，可选的有在内存中，在数据库中。

## 简单示例:2.2.1版本

**package** com.wangdh.quartz;

**import** org.quartz.Job;

**import** org.quartz.JobExecutionContext;

**import** org.quartz.JobExecutionException;

/\*\*

\* 简单的调度作业

\*

\* **@author** wdhcxx

\*/

**public** **class** SimpleQuartzJob **implements** Job {

@Override

**public** **void** **execute**(JobExecutionContext context) **throws** JobExecutionException {

System.***out***.println("简单的调度作业:"+System.*currentTimeMillis*());

}

}

**package** com.wangdh.quartz;

**import** java.util.Date;

**import** org.quartz.JobBuilder;

**import** org.quartz.JobDetail;

**import** org.quartz.Scheduler;

**import** org.quartz.SchedulerException;

**import** org.quartz.SchedulerFactory;

**import** org.quartz.SimpleScheduleBuilder;

**import** org.quartz.Trigger;

**import** org.quartz.TriggerBuilder;

**import** org.quartz.impl.StdSchedulerFactory;

**public** **class** SimpleApp {

**public** **static** **void** **main**(String[] args) **throws** SchedulerException {

**new** SimpleApp().task();

}

/\*\*

\* 触发器和调度器

\*

\* **@throws** SchedulerException

\*/

**public** **void** **task**() **throws** SchedulerException {

// 构造一个标准的调度器工厂，并生成调度器

SchedulerFactory **schedulerFactory** = **new** StdSchedulerFactory();

Scheduler **scheduler** = schedulerFactory.getScheduler();

// 引进作业程序

JobDetail **jobDetail** = JobBuilder.*newJob*(SimpleQuartzJob.**class**)

.withIdentity("simpleJob", "simpleJob")

.build();

// 构建一个触发器并设置启动时间

Trigger **simpleTrigger** = TriggerBuilder.*newTrigger*()

.forJob(jobDetail)

.startAt(**new** Date())

.withSchedule(

SimpleScheduleBuilder.*simpleSchedule*()

.withIntervalInSeconds(2)

.withRepeatCount(5))

.build();

// 设置作业和触发器到调度器中并于1秒后触发

scheduler.scheduleJob(jobDetail, simpleTrigger);

scheduler.startDelayed(1);

}

}

## JobDataMap

在Job类中定义数据属性是没有意义的，因为这些属性值并不会在执行期间保存。如果需要传递数据该怎么做？就是使用JobDataMap，它是JobDetail的一个属性。JobDataMap是Map接口的一个实现，并且它有一些便利的方法来储存和检索基本类型数据。

## 注解

@DisallowConcurrentExecution

使用该注解，那么同一时间将只有一个Job实例被执行

@PersistJobDataAfterExecution

使用该注解，在Job被执行结束后，将会更新JobDataMap

## Trigger

Job与JobDetail是Quartz用来定义具体任务的，而Trigger则是用来定义任务如何执行的。

Quartz提供了Trigger接口来定义公共属性，使用TriggerBuilder可以创建具体类型的Trigger。

最常见的两种Trigger分别是SimpleTrigger、CronTrigger。

### Trigger的公共属性

key，该属性是为了标识Trigger的。

startTime，Trigger第一次被Scheduler触发的时间；该属性的值是指定某个时间点的java.util.Date对象。

endTime，Trigger不再被执行的时间。

priority，优先级；通过设置优先级属性可以控制Trigger被执行的顺序，该属性默认值是5，可以为正整数也可以为负整数。需要注意的是，只有在触发时间相同时，优先级属性才会有效；比如10:59执行的任务总是会在11:00执行的任务之前执行；另外，如果Trigger是可恢复的，那么恢复后，优先级是不会改变的。

misfire，如果因为某些原因，错过触发时间，就需要使用该属性来调整。不同类型的Trigger拥有不同的misfire，但是默认的是smart policy，这种情况下会根据Trigger的类型与配置来动态的调整行为。

Calendars，该属性并不是java.util.Calendar类型，它的作用是排除某些时间，比如在周末不执行任务。

### CronTrigger

CronTrigger使用cron表达式来设置触发时间。CronTrigger创建方式：

// 1秒执行一次CronTrigger

Trigger **cronTrigger** = TriggerBuilder.*newTrigger*()

.forJob(jobDetail)

.startAt(**new** Date())

.withSchedule(CronScheduleBuilder.*cronSchedule*("0/1 \* \* ? \* \*")).build();

### cron表达式

cron表达式的格式为：秒 分 时 日 周 年；其中年是可选的，其它为必填。

每个属性允许的值：

秒， 0-59；

分，0-59；

时，0-23；

日，1-31；

周 ，1-7；

年，可选，1970-2099

cron表达式使用的一些符号：

“/”：指定增加值；比如上面的”0/5”，意思就是从第0秒开始，然后每隔5秒执行一次。

“\*”：表示所有值；比如”5 \* ？ \* \*”，意思就是每一分钟的第5秒执行一次。

“?”：没有指定具体值，表示某月的某一天或者每周的某一天；只能在”日”和”周”上使用”?”，而且必须使用”?”；比如”5 \* \* ？ \* “或”5 \* \* \* ?”都是正确的且效果一样，而”5 \* \* \* \* \*”则是错误的。

“,”：表示多选；比如”1,2,3,5 \* \* ? \* \*”，意思就是每分钟的第1、2、3、5秒执行一次任务。

“-“：表示范围；比如”1-5 \* \* ? \* \*”，意思就是每分钟的第1到5秒执行一次任务。

“L”：只允许在”日”和”周”上使用，分别有不同的作用；如果在”日”上使用，则表示该月的最后一天，如：1月31日；如果在”周”上使用，则意味着7或者SAT；另外，如果数字和L在”周”上组合使用则有其它含义，如：”6L”，则代表当前月的最后一个星期五。L用在”日”上还可以指定偏移，如”L-3”，表示该月的第3天到最后一天。

“W”：表示给定日期最近的工作日，只能用在”日”上面；例如：”15W”，表示每月15号最近的一个工作日，如果15号是周六，则在14号周五执行；如果15号是周日，则在16号周一执行。需要注意的是结合W使用时，不能使用范围，只能使用单个值。另外W也能和L联合使用，表示当月的最后一个工作日。

“#”：表示当月第几个周X，只能在”周”上使用；如：”1#3”，表示当月的第3个星期一。

## 属性文件：quartz.properties

扔到maven的resources目录下

org.quartz.scheduler.instanceName= MainScheduler //实例名称

org.quartz.scheduler.instanceId= instance\_one //实例Id,集群时请设置不同的Id

org.quartz.threadPool.class = org.quartz.simpl.SimpleThreadPool //使用quartz线程池

org.quartz.threadPool.threadCount = 10

org.quartz.threadPool.threadPriority = 5

org.quartz.threadPool.threadsInheritContextClassLoaderOfInitializingThread = true

org.quartz.jobStore.class = org.quartz.impl.jdbcjobstore.JobStoreTX //使用数据库持久化

org.quartz.jobStore.tablePrefix = QRTZ\_ //表前缀,所需要的表在quartz下载的包中可以找到

org.quartz.jobStore.driverDelegateClass = org.quartz.impl.jdbcjobstore.StdJDBCDelegate

org.quartz.jobStore.dataSource = qzDS //数据源

org.quartz.jobStore.useProperties=true //参数类型只能为String

org.quartz.jobStore.isClustered=false //是否集群

org.quartz.dataSource.qzDS.driver = com.mysql.jdbc.Driver

org.quartz.dataSource.qzDS.URL = jdbc:mysql://localhost:3306/quartz?useUnicode=true&characterEncoding=utf-8

org.quartz.dataSource.qzDS.user = root

org.quartz.dataSource.qzDS.password = 123456

org.quartz.dataSource.qzDS.maxConnections = 30

## 定时任务持久化到数据库：Sqlserver为例

### 添加数据库驱动

<!-- SqlServer 驱动 -->

<dependency>

<groupId>com.hynnet</groupId>

<artifactId>sqljdbc4-chs</artifactId>

<version> 4.0.2206.100 </version>

</dependency>

### 创建数据库表

所需要的表在quartz下载的包中可以找到，根据数据库找到需要的建表脚本，此处选择：tables\_sqlServer.sql

### 配置quartz.properties

# 实例名称

org.quartz.scheduler.instanceName= MainScheduler

# 实例Id,集群时请设置不同的Id

org.quartz.scheduler.instanceId= instance\_one

# 使用quartz线程池

org.quartz.threadPool.class = org.quartz.simpl.SimpleThreadPool

org.quartz.threadPool.threadCount = 10

org.quartz.threadPool.threadPriority = 5

org.quartz.threadPool.threadsInheritContextClassLoaderOfInitializingThread = true

# 使用内存持久化

# org.quartz.jobStore.class = org.quartz.simpl.RAMJobStore

# 使用数据库持久化

org.quartz.jobStore.class = org.quartz.impl.jdbcjobstore.JobStoreTX

# 表前缀,所需要的表在quartz下载的包中可以找到

org.quartz.jobStore.tablePrefix = QRTZ\_

org.quartz.jobStore.driverDelegateClass = org.quartz.impl.jdbcjobstore.StdJDBCDelegate

# 数据源

org.quartz.jobStore.dataSource = qzDS

# 参数类型只能为String

org.quartz.jobStore.useProperties=true

# 是否集群

org.quartz.jobStore.isClustered=false

org.quartz.dataSource.qzDS.driver = com.microsoft.sqlserver.jdbc.SQLServerDriver

org.quartz.dataSource.qzDS.URL = jdbc:sqlserver://127.0.0.17:1433;databasename=QuartzDB;

org.quartz.dataSource.qzDS.user = sa

org.quartz.dataSource.qzDS.password = 1q2w#E$R

org.quartz.dataSource.qzDS.maxConnections = 30

### OK

上面步骤做完就OK了,运行程序，在数据库QRTZ\_FIRED\_TRIGGERS中会有相应记录

## 简单集群配置

在许多情况，希望定时任务是可靠的，不会因系统故障、机器宕机而导致某一定时任务不能按时运行。这种情况下，就需要为Quartz做个集群。

最简单的情况，有两台机器或两个应用，同时维护一批定时任务，假如其中一个机器或应用出现问题，还有另外一个应用保底使用。

### 集群配置

# 集群配置

org.quartz.jobStore.isClustered=true

org.quartz.scheduler.instanceId= AUTO

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

### 启动两个应用

启动两个应用，其中一个应用需要注册定时任务；另外一个应用因集群关系则无需注册定时任务。

**try** {

// 获取Scheduler实例

Scheduler **scheduler** = StdSchedulerFactory.*getDefaultScheduler*();

scheduler.start();

} **catch** (SchedulerException **se**) {

se.printStackTrace();

}

上面代码是无需注册定时任务的启动代码。

先后启动应用A和应用B，运行并观察一段时间后，关闭应用A，继续观察。通过日志观察可见，首先定时任务有应用A执行，在应用A被关闭后，定时任务由应用B继续触发执行。

这里观察到，似乎不是负载均衡，因为在应用A关闭后应用B才有机会运行。而理想中，应该是应用A与应用B互相补充、交替运行。这里主要因为只用了一个任务（只有一个触发器）做测试，如果运用多个触发器设置多个任务做测试，应用A与应用B互相补充、交替运行。

# Spring定时任务

## 定时任务实现概述

### 实现的技术分类

Java自带的java.util.Timer类，这个类允许你调度一个java.util.TimerTask任务。使用这种方式可以让你的程序按照某一个频度执行，但不能在指定时间运行。一般用的较少。

Quartz，这是一个功能比较强大的的调度器，可以让你的程序在指定时间执行，也可以按照某一个频度执行，配置起来稍显复杂。

Spring3.0以后自带的task，可以将它看成一个轻量级的Quartz，而且使用起来比Quartz简单许多。

### 作业类的继承方式

作业类需要继承自特定的作业类基类，如Quartz中需要继承自org.springframework.scheduling.quartz.QuartzJobBean；java.util.Timer中需要继承自java.util.TimerTask。

作业类即普通的java类，不需要继承自任何基类。

注:个人推荐使用第二种方式，因为这样所以的类都是普通类，不需要事先区别对待。

### 任务调度的触发时机

每隔指定时间则触发一次，在Quartz中对应的触发器为：

org.springframework.scheduling.quartz.SimpleTriggerBean

每到指定时间则触发一次，在Quartz中对应的调度器为：

org.springframework.scheduling.quartz.CronTriggerBean

注：并非每种任务都可以使用这两种触发器，如java.util.TimerTask任务就只能使用第一种。Quartz和spring task都可以支持这两种触发条件。

## 基于Spring注解的定时任务

需要启用定时任务注解支持

### xml配置

<?xml version="1.0" encoding="UTF-8" ?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:tx="http://www.springframework.org/schema/tx"

xmlns:context="http://www.springframework.org/schema/context"

xmlns:mvc="http://www.springframework.org/schema/mvc"

xmlns:p="http://www.springframework.org/schema/p"

xmlns:task="http://www.springframework.org/schema/task"

xmlns:aop="http://www.springframework.org/schema/aop"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd

http://www.springframework.org/schema/tx

http://www.springframework.org/schema/tx/spring-tx.xsd

http://www.springframework.org/schema/task

http://www.springframework.org/schema/task/spring-task.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context.xsd

http://www.springframework.org/schema/mvc

http://www.springframework.org/schema/mvc/spring-mvc.xsd

http://www.springframework.org/schema/aop

http://www.springframework.org/schema/aop/spring-aop.xsd">

<!-- 配置定时器 -->

<task:annotation-driven scheduler="qbScheduler"

mode="proxy" />

<task:scheduler id="qbScheduler" pool-size="10" />

<!-- <task:scheduled-tasks>

<task:scheduled ref="taskJob" method="job1" cron="0 \* \* \* \* ?" />

</task:scheduled-tasks> -->

<!-- 使用Annotation自动注册Bean -->

<context:component-scan base-package="com.wangdh.spring.quartz" />

</beans>

### 任务类

**package** com.wangdh.spring.quartz;

**import** org.springframework.scheduling.annotation.Scheduled;

**import** org.springframework.stereotype.Service;

@Service

**public** **class** SpringTaskService {

/\*\*

\* 可配置多个属性：

\* cron:每到什么时候执行一次

\* fixedDelay:延时多少毫秒，多少毫秒执行一次

\* fixedRate:为每隔一段时间执行一次

\*/

@Scheduled(fixedDelay = 1000 \* 3)

**public** **void** **task**(){

System.***out***.println("延迟3秒，每隔3秒执行一次:" + System.*currentTimeMillis*());

}

@Scheduled(fixedRate = 1000 \* 5)

**public** **void** **task1**(){

System.***out***.println("每隔5秒执行一次:" + System.*currentTimeMillis*());

}

@Scheduled(cron = "0/2 \* \* \* \* \*")

**public** **void** **task2**(){

System.***out***.println("cron每隔2秒执行一次:" + System.*currentTimeMillis*());

}

}

### 启动类

**package** com.wangdh.spring.quartz;

**import** org.springframework.context.ApplicationContext;

**import** org.springframework.context.support.ClassPathXmlApplicationContext;

/\*\*

\* 基于Spring注解的定时任务测试

\* **@author** wdhcxx

\*

\*/

**public** **class** SpringTaskApp {

@SuppressWarnings("unused")

**public** **static** **void** **main**(String[] args) {

@SuppressWarnings("resource")

ApplicationContext **applicationContext** = **new** ClassPathXmlApplicationContext("spring-task.xml");

}

}

## Spring集成Quartz

### maven引用

<properties>

<spring.version>4.3.8.RELEASE</spring.version>

<quartz.version>2.2.1</quartz.version>

</properties>

i<!-- Spring依赖 -->ng --

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>${spring.version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context-support</artifactId>

<version>${spring.version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-tx</artifactId>

<version>${spring.version}</version>

</dependency>

<!-- 任务调度 -->

<dependency>

<groupId>org.quartz-scheduler</groupId>

<artifactId>quartz</artifactId>

<version>${quartz.version}</version>

</dependency>

<dependency>

<groupId>org.quartz-scheduler</groupId>

<artifactId>quartz-jobs</artifactId>

<version>${quartz.version}</version>

</dependency>

### 编写任务类

任务类的编写可以有两种方式：普通Java类或继承

org.springframework.scheduling.quartz.QuartzJobBean

\* 继承org.springframework.scheduling.quartz.QuartzJobBean

**package** com.wangdh.spring.quartz;

**import** org.quartz.JobExecutionContext;

**import** org.quartz.JobExecutionException;

**import** org.springframework.scheduling.quartz.QuartzJobBean;

/\*\*

\* spring 定時任务

\*

\* **@author** wangdh 2017年7月7日上午10:48:14

\*/

**public** **class** SpringQtz **extends** QuartzJobBean {

**private** **static** **int** *counter* = 0;

@Override

**protected** **void** **executeInternal**(JobExecutionContext context) **throws** JobExecutionException {

System.***out***.println("开始执行任务...");

**long** **ms** = System.*currentTimeMillis*();

System.***out***.println(ms);

System.***out***.println("(" + *counter*++ + ")");

String **s** = (String) context.getMergedJobDataMap().get("service");

System.***out***.println(s);

}

}

\* 普通Java类方式

**package** com.wangdh.spring.quartz;

**public** **class** SimpleJob {

**public** **void** **doJob**() {

System.***out***.println(System.*currentTimeMillis*());

System.***out***.println("不继承QuartzJobBean方式-调度进行中：" + System.*currentTimeMillis*());

}

}

### 配置作业类

\* 有继承任务类的配置方式

<!-- 配置作业类 -->

<bean id="myjob"

class="org.springframework.scheduling.quartz.JobDetailFactoryBean">

<property name="name" value="exampleJob"></property>

<property name="jobClass" value="com.wangdh.spring.quartz.SpringQtz"></property>

<property name="jobDataMap">

<map>

<entry key="service">

<value>simple is the beat</value>

</entry>

</map>

</property>

</bean>

\* 普通任务类的配置方式

<bean id="simpleJob"

class="org.springframework.scheduling.quartz.MethodInvokingJobDetailFactoryBean">

<!-- 具体作业类 -->

<property name="targetObject">

<bean class="com.wangdh.spring.quartz.SimpleJob" />

</property>

<!-- 作业类要执行的方法 -->

<property name="targetMethod" value="doJob" />

<!-- 作业不并发调度 -->

<property name="concurrent" value="false" />

</bean>

### 配置作业触发器

\* 普通触发器：SimpleTrigger

<!-- 配置作业调度的触发方式（触发器） -->

<bean id="simpleTrigger"

class="org.springframework.scheduling.quartz.SimpleTriggerFactoryBean">

<!-- 要执行的作业 -->

<property name="jobDetail" ref="myjob" />

<!-- 调度工厂实例化后，经过0秒开始执行调度 -->

<property name="startDelay" value="0" />

<!-- 每2秒调度一次 -->

<property name="repeatInterval" value="2000" />

</bean>

\* 定时触发器：CronTrigger

- 配置作业调度的触发方式（触发器） -->

<bean id="cronTrigger"

class="org.springframework.scheduling.quartz.CronTriggerFactoryBean">

<!-- 要执行的作业 -->

<property name="jobDetail" ref="simpleJob" />

<!-- 每5秒运行一次 -->

<property name="cronExpression" value="0/5 \* \* \* \* ?" />

</bean>

### 配置调度工厂

<!-- 配置调度工厂 -->

<bean class="org.springframework.scheduling.quartz.SchedulerFactoryBean">

<property name="triggers">

<list>

<ref bean="cronTrigger" />

<ref bean="simpleTrigger" />

</list>

</property>

</bean>

### 启动应用

大功告成，启动应用，执行任务…