Quartz

# Quartz介绍

官网：http://www.quartz-scheduler.org/

## Quzrtz是什么

**Quartz is a richly featured, open source job scheduling library that can be integrated within virtually any Java application - from the smallest stand-alone application to the largest e-commerce system. Quartz can be used to create simple or complex schedules for executing tens, hundreds, or even tens-of-thousands of jobs; jobs whose tasks are defined as standard Java components that may execute virtually anything you may program them to do. The Quartz Scheduler includes many enterprise-class features, such as support for JTA transactions and clustering.**

**Quartz is freely usable, licensed under the Apache 2.0 license.**

**大致翻译**

**Quartz是一个功能非常丰富的开源作业调度框架，我们几乎可以使用它来集成任何java应用，小到单体应用，大到大型电子商务系统；Quartz可以用来执行成千上万的简单或复杂的调度任务；Quartz的任务被定义为标准的java组件，几乎可以执行任何编程任务。Quartz调度器包含了很多企业级的功能，包括JTA的支持和集群支持。**

**Quartz完全免费使用，基于Apache2.0开源。**

## Quartz能做什么

**每天定时触发数据同步任务、定时发送邮件、短信（比如信用卡还款短信）等等；**

## Quartz特点

非常强大的调度功能；

非常灵活的使用方式；（允许咱们灵活配置调度时间等）

Quartz提供了集群的功能；

***了解当当开发的Elastic-job，分布式调度框架***

# Quartz的核心概念

调度器**Scheduler；**

任务Job；

触发器Trigger；

## Quzrtz的核心API

**Job** - 由调度程序执行的组件实现的接口。

**JobDetail** - 用于定义任务的实例。

**JobBuilder** - 用于定义/构建JobDetail实例，用于定义作业的实例。

**JobStore** - 任务的存储

**Scheduler -** 与调度程序交互的主要API，可以定时去执行Jobdeail的信息。

**TriggerBuilder -** 用于定义/构建触发器实例。

**Trigger**（即触发器） - 执行作业计划的组件，可以定义任务何时被调用。**分为SimpleTrigger和CronTrigger；**

# Quartz使用

## 创建第一个Hello World项目

### 引入pom依赖

|  |
| --- |
| **<dependency>**  **<groupId>org.quartz-scheduler</groupId>**  **<artifactId>quartz</artifactId>**  **<version>2.3.0</version>**  **</dependency>** |

### 创建HelloJob

要求创建的任务类要实现Job接口；

### 创建JobDtail

使用构建者模式创建JobDetail

|  |
| --- |
| **// 1.创建JobDetail<-JobBuilder 构建者方式**  **JobDetail jobDetail = JobBuilder//**  **// 使用反射创建HelloJob的实例，每执行一次都会创建一个实例**  **.*newJob*(HelloJob.class)//**  **// .withIdentity("hello", "hello")// 身份信息**  **.build();** |

### 创建Trigger

|  |
| --- |
| **// 2.创建触发器Trigger**  **SimpleTrigger trigger = TriggerBuilder//**  **.*newTrigger*()//**  **.withIdentity("hello", "hello")// 身份信息**  **.startNow()// 立即执行**  **.withSchedule(SimpleScheduleBuilder//**  **.*simpleSchedule*()//**  **.withIntervalInSeconds(3)// 3秒执行一次**  **.repeatForever())//**  **.build();** |

### 创建Scheduler

|  |
| --- |
| **// 3.创建调度器**  **Scheduler scheduler = StdSchedulerFactory.*getDefaultScheduler*();**  **scheduler.scheduleJob(jobDetail, trigger);**  **scheduler.start();** |

## JobDetail给Job任务传参

三种方式:都是用usingJobData方法

1.直接key/value方式.<BR>

2.使用JobDataMap.<BR>

3.使用任务类的成员变量,需要给变量加上getter和setter.<BR>

|  |
| --- |
| **public** **class** **ParamJob** **implements** Job {  // 第三种传参方式,使用任务类的成员变量,需要给变量加上getter和setter  **private** String third;  @Override  **public** **void** execute(JobExecutionContext context)  **throws** JobExecutionException {  SimpleDateFormat sdf = **new** SimpleDateFormat("yyyy-MM-dd HH:mm:ss");  System.***out***.println(sdf.format(**new** Date()) + "--Test Param!");  JobDetail jobDetail = context.getJobDetail();  JobDataMap jobDataMap = jobDetail.getJobDataMap();  System.***out***.println("first-->" + jobDataMap.get("first"));  System.***out***.println("second-->" + jobDataMap.get("second"));  System.***out***.println("third-->" + **this**.third);  }  **public** String getThird() {  **return** third;  }  **public** **void** setThird(String third) {  **this**.third = third;  }  }  ===============================================================================  // 第二种传参：直接使用JobDataMap  **JobDataMap jobDataMap = new JobDataMap();**  **jobDataMap.put("second", "第二种传参方式");**  **JobDetail jobDetail = JobBuilder//**  **.*newJob*(ParamJob.class)//**  **.withIdentity("param", "paramGroup")//**  **.usingJobData("first", "第一种传参方式")// 第一种传参方式**  **.usingJobData(jobDataMap)// 第二种传参方式**  **.usingJobData("third", "第三种方式：使用成员变量传参")// 第三种传参方式**  **.build();** |

## Trigger给Job任务传参

与JonDetail传参方式相同，这里不再赘述，请参考代码；

## 合并JobDetail和Trigger的参数

**注意:如果JobDetail和Trigger同时传入相同名称的参数，则使用Trigger的**

|  |
| --- |
| **public** **class** MergeJob **implements** Job {  @Override  **public** **void** execute(JobExecutionContext context) **throws** JobExecutionException {  SimpleDateFormat sdf = **new** SimpleDateFormat("yyyy-MM-dd HH:mm:ss");  System.***out***.println(sdf.format(**new** Date()) + "Test Merge Param");  // 获取JobDetail和Trigger合并后的数据  JobDataMap mergedMap = context.getMergedJobDataMap();  Set<Entry<String, Object>> entrySet = mergedMap.entrySet();  Iterator<Entry<String, Object>> iterator = entrySet.iterator();  **while** (iterator.hasNext()) {  Entry<String, Object> entry = iterator.next();  System.***out***.println(entry.getKey() + ":" + entry.getValue());  }  }  }  ======================================================================  **public** **class** FourthMain {  **public** **static** **void** main(String[] args) **throws** SchedulerException {  JobDetail jobDetail = JobBuilder//  .*newJob*(MergeJob.**class**)//  .withIdentity("merge")//  .usingJobData("id", 1)//  **.usingJobData("name", "et1812")//**  .build();  SimpleTrigger trigger = TriggerBuilder//  .*newTrigger*()//  .withIdentity("merge")//  .usingJobData("age", "18")//  **.usingJobData("name", "etoak")//**  .usingJobData("address", "趵突泉北路6号")//  .startNow()//  .withSchedule(SimpleScheduleBuilder//  .*simpleSchedule*()//  .withIntervalInSeconds(5)//  .repeatForever()//  ).build();  Scheduler scheduler = StdSchedulerFactory.*getDefaultScheduler*();  scheduler.scheduleJob(jobDetail, trigger);  scheduler.start();  }  } |

## 详细讲解SimpleTrigger

**>在具体的时间点执行一次；**

**>或者在具体的时间点执行，并且以指定的间隔重复执行若干次；**

使用三个例子讲解

1.任务2秒后执行一次，之后每隔3秒执行一次

2.任务2秒之后执行，之后每3秒执行一次，10秒后停止执行

3.任务2秒之后执行，之后每3秒执行一次，执行10次

任务一实现

|  |
| --- |
| **// 获取当前时间**  **Date current = new Date();**  **SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");**  **System.*out*.println("当前时间:" + sdf.format(current));**  **// 1.任务2秒后执行一次，之后每隔3秒执行一次**  **Date twoSecondAfter = new Date(current.getTime() + 2000L);**  **// 创建Trigger**  **SimpleTrigger trigger = TriggerBuilder//**  **.*newTrigger*()//**  **.withIdentity("simple", "simple")//**  **.startAt(twoSecondAfter)**  **.withSchedule(SimpleScheduleBuilder//**  **.*simpleSchedule*()//**  **.withIntervalInSeconds(3)//**  **.repeatForever()//**  **).build();** |

任务二实现

|  |
| --- |
| // 获取当前时间  Date current = **new** Date();  SimpleDateFormat sdf = **new** SimpleDateFormat("yyyy-MM-dd HH:mm:ss");  System.***out***.println("当前时间:" + sdf.format(current));  **// 2.任务2秒之后执行，之后每3秒执行一次，10秒后停止执行**  **Date twoSecondAfter = new Date(current.getTime() + 2000L);**  **Date tenSecondAfter = new Date(current.getTime() + 10000L);**  **SimpleTrigger trigger = TriggerBuilder//**  **.*newTrigger*()//**  **.withIdentity("simple", "simple")//**  **.startAt(twoSecondAfter)// 2秒后执行**  **.endAt(tenSecondAfter)// 10秒后停止**  **.withSchedule(SimpleScheduleBuilder//**  **.*simpleSchedule*()//**  **.withIntervalInSeconds(3)//**  **.repeatForever()//**  **).build();** |

任务三

|  |
| --- |
| // 获取当前时间  Date current = **new** Date();  SimpleDateFormat sdf = **new** SimpleDateFormat("yyyy-MM-dd HH:mm:ss");  System.***out***.println("当前时间:" + sdf.format(current));  // 3.任务2秒之后执行，之后每3秒执行一次，重复执行10次  **Date twoSecondAfter = new Date(current.getTime() + 2000L);**  SimpleTrigger trigger = TriggerBuilder.*newTrigger*()  .withIdentity("simple", "simple")//  .startAt(twoSecondAfter)//  .withSchedule(SimpleScheduleBuilder//  .*simpleSchedule*()//  .withIntervalInSeconds(3)//  .withRepeatCount(10)// 重复10次  ).build(); |

## 详细讲解CronTrigger

Cron表达式详解见Cron表达式.docx文件

|  |
| --- |
| CronTrigger trigger = TriggerBuilder.*newTrigger*()//  .withIdentity("cron", "cron")//  .withSchedule(CronScheduleBuilder//  // 每天15:40-15:41每隔5秒执行一次  // .cronSchedule("0/5 40 15 \* \* ?")//  .*cronSchedule*("\*/5 \* \* \* \* ?")  ).build(); |

# 各种API介绍

## Job接口



只有一个方法：execute方法，用于执行调度任务逻辑；

参数JobExecutionContext：任务执行的上下文

## JobDetail接口

任务细节，Quartz执行Job时，需要新建个Job实例，但是不能直接操作Job类，所以通过JobDetail来设置一些Job的属性，存储Job的状态描述信息等，调度器需要使用JobDetail对象来添加Job实例。

### 创建JobDetail涉及的一些参数

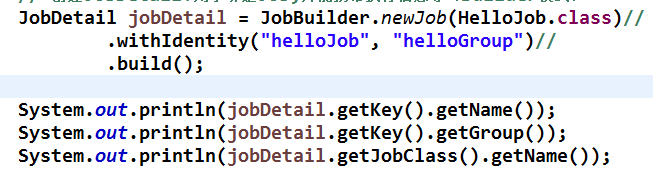
name：任务名称；

group：任务组，有默认值：DEFAULT

jobClass：Job的实现类

jobDataMap：传参作用，由于没办法调用有参构造方法，所以使用这个属性进行传参

使用代码去看看几个参数



## JobExecutionContext接口

当任务调度器Schduler调用一个Job时，就会将JobExecutionContext传递到Job的execute方法中的形参中；

Job任务能够通过JobExecutionContext对象访问Quartz运行时的环境以及Job任务的一些数据。

## JobDataMap类

JobDataMap 可以用来保存数据对象（序列化）。

JobDataMap 是Java Map接口的一个实现，并且添加了一些方便的方法用于存储和获取原始数据类型。

JobMap存储在JobExecutionContext中，在Job中非常容易获取；

### 如何获取JobDataMap

***>直接获取Jobdetail或者Trigger自身的JobDataMap***

***>使用成员变量传入，需要加上getter和setter***

***> 使用mergedDataMap，获取JobDetail和Trigger合并的JobDataMap***

## Trigger接口

用来指定任务何时触发，也就是用来触发要执行的job任务；

Trigger是通过Builder模式创建

## SimpleTrigger接口

SimpleTrigger可以满足的调度需求是：

>在具体的时间点执行一次；

>或者在具体的时间点执行，并且以指定的间隔重复执行若干次；

|  |
| --- |
|  |

## CronTrigger接口

CronTrigger通常比Simple Trigger更有用，适合基于日历的方式进行作业调度。

使用CronTrigger，您可以指定时间表，例如“每周五中午”或“每个工作日上午9:00”，甚至“每周一至周五上午9:00至10点之间每5分钟”。

它和SimpleTrigger一样，有一个startTime，它指定何时生效，以及一个endTime，用于指定何时停止计划。

### Cron表达式

**Cron-Expressions用于配置CronTrigger的实例。Cron Expressions是由七个子表达式组成的字符串，用于描述时间的各个细节。这些子表达式用空格分隔，并表示：**

**[秒] [分] [小时] [日] [月] [周] [年]**

**Seconds Minutes Hours Day-of-Month Month Day-of-Week Year (optional field)**

|  |  |  |  |
| --- | --- | --- | --- |
| **时间字段** | **是否必填** | **允许值** | **特殊字符** |
| **秒** | **是** | **0-59** | **,-\*/** |
| **分** | **是** | **0-59** | **,-\*/** |
| **时** | **是** | **0-23** | **,-\*/** |
| **日** | **是** | **1-31** | **, - \* ? / L W C** |
| **月** | **是** | **1-12或者JAN-DEC** | **,-\*/** |
| **周** | **是** | **1-7或者SUN-SAT** | **,-\* ? / L C #** |
| **年** | **否** | **Empty或者1970-2099-** | **,-\*/** |

**,：表示或的关系**

**-：范围的关系【比如1-21】**

**\*：每秒、每分、每小时等**

**/：每天哪个时间执行**

**表达式举例**

**每天14:30上课时触发：0 30 14 ? \* \***

**每天1点到1点59分，每隔5分执行：0 0/5 1 \* \* ?**

**周一到周五9:00上班的时候执行：0 0 9 ? \* MON-FRI**

? 号只能用在天和周域上，但是不能在这两个域上同时使用。你可以认为 ? 字符是"我并不关心在该域上是什么值。" 这不同于星号，星号是指示着该域上的每一个值。? 是说不为该域指定值。

不能同时这两个域上指定值的理由是难以解释甚至是难以理解的。基本上，假定同时指定值的话，意义就会变得含混不清了：考虑一下，如果一个表达式在日域上有值11，同时在周域上指定了 WED。那么是要 trigger 仅在每个月的11号，且正好又是星期三那天被激发？还是在每个星期三的11号被激发呢？要去除这种不明确性的办法就是不能同时在这两个域上指定值。

***只要记住，假如你为这两域的其中一个指定了值，那就必须在另一个字值上放一个 ?。***

***表达式样例：***

0 10,44 14 ? 3 WEN

意义：在三月中的每个星期三的下午 2:10:00 和下午 2:44:00 被触发。

|  |
| --- |
| 0 0/10 \* \* \* 与 0 \*/10 \* \* \* 的差别在于什么地方。  在说这两者的差别之前，先说下各个字符代表的含义。0代表从0分开始，\*代表任意字符，／代表递增。  也就是说0 0/10 \* \* \*代表从0分钟开始，每10分钟执行任务一次。0 \*/10 \* \* \*代表从任务启动开始每10分钟执行任务一次。有人会问，这不是一样的么？  答案是不一样的。因为起始的时间不一样。例如：从5:07分钟的时候执行该任务第一种写法会在5:10的时候进行执行，写法二会在5:17进行执行。这就是两者的差别。  当然0 0/1 \* \* \* 与0 \*/1 \* \* \*有时会被认为是同一种写法。 |

## Schedler

Scheduler scheduler = StdSchedulerFactory.getDefaultScheduler();

scheduler.start();

scheduler.scheduleJob(jobDetail, trigger);

# 整合Spring

## 写一个任务类

## 配置MethodInvokingJobDetailFactoryBean

## 配置CronTriggerFactoryBean

## 配置SchedulerFactoryBean

|  |
| --- |
| <!-- 定义任务类 -->  <bean id=*"springJob"* class=*"com.etoak.job.FisrtSpringJob"* />  <!-- JobDetail -->  <bean id=*"jobDetail"* class=*"org.springframework.scheduling.quartz.MethodInvokingJobDetailFactoryBean"*>  <property name=*"targetObject"* ref=*"springJob"* />  <property name=*"targetMethod"* value=*"execute"* />  </bean>  <!-- Trigger -->  <bean id=*"springTrigger"*  class=*"org.springframework.scheduling.quartz.CronTriggerFactoryBean"*>  <property name=*"jobDetail"* ref=*"jobDetail"* />  <property name=*"cronExpression"* value=*"0/10 \* \* \* \* ?"* />  </bean>  <!-- SchedulerFactoryBean -->  <bean id=*"scheduler"*  class=*"org.springframework.scheduling.quartz.SchedulerFactoryBean"*>  <property name=*"triggers"*>  <list>  <ref bean=*"springTrigger"* />  </list>  </property>  <property name=*"applicationContextSchedulerContextKey"*  value=*"applicationContext"*/>  </bean> |

# 持久化、集群

## 使用JDBC持久化任务

|  |
| --- |
| **1.引入依赖**  **quartz**  **spring-context-support**  **spring-jdbc**  **spring-tx**  **mysql-connector-java**  **2.找到mysql数据库表文件**  **2.1 创建数据库**  **字符集：(utf8[utf8mb4])**  **排序规则(utf8\_general\_ci[utf8mb4\_general\_ci])**  **2.2 执行sql创建表**  **3.quartz.properties放到src/main/resources文件夹下**  **4.创建任务类**  **要求任务类继承QuartzJobBean**  **public class EtoakJob extends QuartzJobBean{**  **实现方法**  **}**  **5.配置Spring**  **5.1 配置数据源**  **<bean id="dataSource" class="HikariDataSource">**  **<property name="driverClassName" value="com.mysql.jdbc.Driver">**  **<property name="jdbcUrl" value="jdbc:mysql:///quartz" />**  **<property name="username" value="root" />**  **<property name="password" value="etoak" />**  **</bean>**  **5.2 配置事务管理器**  **<bean id="tx" class="DataSourceTransactionManager">**  **<property name="dataSource" ref="dataSource" />**  **</bean>**  **5.3-配置JobDetailFactoryBean**  **<bean id="jobDetail" class="JobDetailFactoryBean">**  **<property name="jobClass" value="com.job.EtoakJob" />**  **<property name="durability" value="true" />**  **<property name="name" value="名称" />**  **<property name="group" value="组名称" />**  **</bean>**    **5.4 配置CronTriggerFactoryBean**  **<bean id="trigger" class="CronTriggerFactoryBean">**  **<property name="jobDetail" ref="jobDetail" />**  **<property name="cronExpression" value="0/5 \* \* \* \* ?" />**  **<property name="name" value="名称" />**  **<property name="group" value="组名称" />**  **</bean>**  **5.5 配置SchedulerFactoryBean**  **<bean id="scheduler" class="SchedulerFactoryBean">**  **<property name="triggers">**  **<list>**  **<ref bean="trigger" />**  **</list>**  **</property>**  **<property name="dataSource" ref="dataSource" />**  **<property name="transactionManager" ref="tx" />**  **<property name="configLocation" value="classpath:quartz.properties"/>**  **<property name="applicationContextSchedulerContextKey" value="applicationContext" />**  **</bean>** |