QuartZ

介绍

Quartz是OpenSymphony开源组织在Job scheduling领域又一个开源项目，是完全由java开发的一个开源的任务日程管理系统，“任务进度管理器”就是一个在预先确定（被纳入日程）的时间到达时，负责执行（或者通知）其他软件组件的系统。 Quartz用一个小Java库发布文件（.jar文件），这个库文件包含了所有Quartz核心功能。这些功能的主要接口(API)是Scheduler接口。它提供了简单的操作，例如：将任务纳入日程或者从日程中取消，开始/停止/暂停日程进度。

定时器种类

Quartz 中五种类型的 Trigger：SimpleTrigger，CronTirgger，DateIntervalTrigger，NthIncludedDayTrigger和 Calendar 类（ org.quartz.Calendar）。 最常用的： SimpleTrigger：用来触发只需执行一次或者在给定时间触发并且重复N次且每次执行延迟一定时间的任务。 CronTrigger：按照日历触发，例如“每个周五”，每个月10日中午或者10：15分。

存储方式

RAMJobStore和JDBCJobStore 对比：

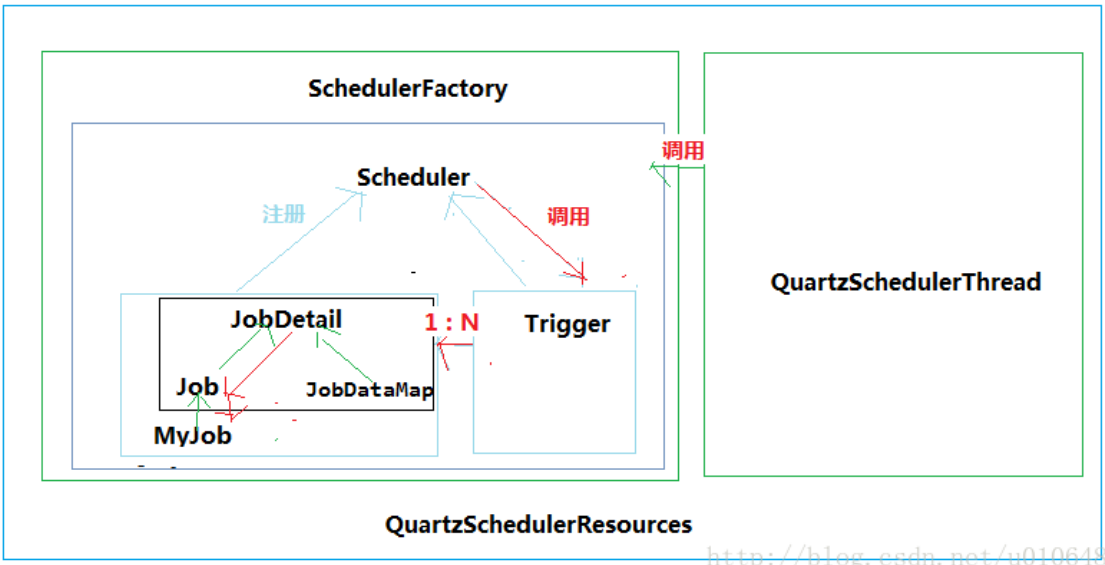
类型	优点	缺点
RAMJobStore	不要外部数据库，配置容易，运行速度快	因为调度程序信息是存储在被分配给JVM的内存里面，所以，当应用程序停止运行时，所有调度信息将被丢失。另外因为存储到JVM内存里面，所以可以存储多少个Job和Trigger将会受到限制
JDBCJobStore	支持集群，因为所有的任务信息都会保存到数据库中，可以控制事物，还有就是如果应用服务器关闭或者重启，任务信息都不会丢失，并且可以恢复因服务器关闭或者重启而导致执行失败的任务	运行速度的快慢取决与连

表关系和解释

表名称	说明
qrtz_blob_triggers	Trigger作为Blob类型存储(用于Quartz用户用JDBC创建他们自己定制的Trigger类型，JobStore 并不知道如何存储实例的时候)
qrtz_calendars	以Blob类型存储Quartz的Calendar日历信息， quartz可配置一个日历来指定一个时间范围
qrtz_cron_triggers	存储Cron Trigger，包括Cron表达式和时区信息。
qrtz_fired_triggers	存储与已触发的Trigger相关的状态信息，以及相联Job的执行信息
qrtz_job_details	存储每一个已配置的Job的详细信息
qrtz_locks	存储程序的非观锁的信息(假如使用了悲观锁)
qrtz_paused_trigger_graps	存储已暂停的Trigger组的信息
qrtz_scheduler_state	存储少量的有关 Scheduler的状态信息，和别的 Scheduler 实例(假如是用于一个集群中)
qrtz_simple_triggers	存储简单的 Trigger，包括重复次数，间隔，以及已触的次数
qrtz_triggers	存储已配置的 Trigger的信息
qrzt_simprop_triggers	

核心类和关系

1. 核心类 （1）核心类 **QuartzSchedulerThread** ： 负责执行向QuartzScheduler注册的触发Trigger的工作的线程。 **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供运行效率。 **QuartzSchedulerResources**：包含创建QuartzScheduler实例所需的所有资源（JobStore，ThreadPool等）。 **SchedulerFactory** ： 提供用于获取调度程序实例的客户端可用句柄的机制。 **JobStore**： 通过类实现的接口，这些类要为org.quartz.core.QuartzScheduler的使用提供一个org.quartz.Job和org.quartz.Trigger存储机制。作业和触发器的存储应该以其名称和组的组合为唯一性。 **QuartzScheduler** ： 这是Quartz的核心，它是org.quartz.Scheduler接口的间接实现，包含调度org.quartz.Jobs，注册org.quartz.JobListener实例等的方法。 **Scheduler** ： 这是Quartz Scheduler的主要接口，代表一个独立运行容器。调度程序维护JobDetails和触发器的注册表。 一旦注册，调度程序负责执行作业，当他们的相关联的触发器触发（当他们的预定时间到达时）。 **Trigger** ： 具有所有触发器通用属性的基本接口，描述了job执行的时间出发规则。 - 使用TriggerBuilder实例化实际触发器。 **JobDetail** ： 传递给定作业实例的详细信息属性。 JobDetails将使用JobBuilder创建/定义。 **Job**：要由表示要执行的“作业”的类实现的接口。只有一个方法 void execute(jobExecutionContext context) (jobExecutionContext 提供调度上下文各种信息，运行时数据保存在jobDataMap中) Job有个子接口StatefulJob ,代表有状态任务。 有状态任务不可并发，前次任务没有执行完，后面任务处于阻塞等到。
2. 关系-自己理解



注：一个job可以被多个Trigger 绑定，但是一个Trigger只能绑定一个job

配置文件

```
//调度标识名 集群中每一个实例都必须使用相同的名称 （区分特定的调度器实例）
org.quartz.scheduler.instanceName: DefaultQuartzScheduler
//ID设置为自动获取 每一个必须不同 （所有调度器实例中是唯一的）
org.quartz.scheduler.instanceId : AUTO
//数据保存方式为持久化
org.quartz.jobStore.class : org.quartz.impl.jdbcjobstore.JobStoreTX
//表的前缀
org.quartz.jobStore.tablePrefix : QRTZ_
//设置为TRUE不会出现序列化非字符串类到 BLOB 时产生的类版本问题
//org.quartz.jobStore.useProperties : true
//加入集群 true 为集群 false不是集群
org.quartz.jobStore.isClustered : false
//调度实例失效的检查时间间隔
org.quartz.jobStore.clusterCheckinInterval: 20000
//容许的最大作业延长时间
org.quartz.jobStore.misfireThreshold : 60000
//ThreadPool 实现的类名
org.quartz.threadPool.class: org.quartz.simpl.SimpleThreadPool
//线程数量
org.quartz.threadPool.threadCount : 10
//线程优先级
org.quartz.threadPool.threadPriority : 5 (threadPriority 属性的最大值是常量
java.lang.Thread.MAX_PRIORITY, 等于10。最小值为常量 java.lang.Thread.MIN_PRIORITY, 为1)
//自创建父线程
//org.quartz.threadPool.threadsInheritContextClassLoaderOfInitializingThread: true
//数据库别名
org.quartz.jobStore.dataSource : qzDS
//设置数据源
```

```
org.quartz.dataSource.qzDS.driver:com.mysql.jdbc.Driver
org.quartz.dataSource.qzDS.URL:jdbc:mysql://localhost:3306/quartz
org.quartz.dataSource.qzDS.user:root
org.quartz.dataSource.qzDS.password:123456
org.quartz.dataSource.qzDS.maxConnection:10
```

JDBC插入表顺序

主要的JDBC操作类，执行sql顺序。

Simple_trigger ： 插入顺序 qrtz_job_details —> qrtz_triggers —> qrtz_simple_triggers —> qrtz_fired_triggers
Cron_Trigger: 插入顺序 qrtz_job_details —> qrtz_triggers —> qrtz_cron_triggers —> qrtz_fired_triggers

Spring整合QuartZ

Spring是一个很优秀的框架，它无缝的集成了Quartz，简单方便的让企业级应用更好的使用Quartz进行任务的调度。在企业级开发过程中，正常情况下不会使用RAM方式进行任务的存储，都是使用JDBC方式。方便学习，这两种方式都进行介绍 。

RAM方式

一. Jar包依赖 使用Maven进行Jar包的管理，使用的jar包如下：

```
<!-- spring相关依赖 -->

<!-- quartz -->
<dependency>
    <groupId>org.quartz-scheduler</groupId>
    <artifactId>quartz</artifactId>
    <version>2.3.0</version>
</dependency>
<dependency>
    <groupId>org.quartz-scheduler</groupId>
    <artifactId>quartz-jobs</artifactId>
    <version>2.3.0</version>
</dependency>
```

二.任务类Job

```
/**
 * QuartZ任务调度
 *
 * 1.实现Job接口
 * 2.重写execute方法
 *
 * @version 2018年8月2日下午2:28:07
```

```
 * @author zhuwenbin
 */
public class MyJob implements Job {

    private Logger log = LoggerFactory.getLogger(this.getClass());

    @Override
    public void execute(JobExecutionContext context) throws JobExecutionException {
        log.info(">>>>>正在使用QuartZ任务调度框架");
    }

}
```

三.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"
xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context.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="myJob"/>
        <!-- 指定job的分组 -->
        <property name="group" value="myJob"/>
        <!-- 指定具体的job类 -->
        <property name="jobClass" value="xxx.xxx.xxx.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>

    <!-- 2.1: 定义触发器的bean，定义一个Simple的Trigger，一个触发器只能和一个任务进行绑定 -->
    <!-- <bean name="simpleTrigger"
class="org.springframework.scheduling.quartz.SimpleTriggerFactoryBean">
        指定Trigger的名称
        <property name="name" value="hw_trigger"/>
        指定Trigger的名称
```

```
        <property name="group" value="hw_trigger_group"/>
        指定Tirgger绑定的Job
        <property name="jobDetail" ref="hwJob"/>
        指定Trigger的延迟时间 1s后运行
        <property name="startDelay" value="1000"/>
        指定Trigger的重复间隔  5s
        <property name="repeatInterval" value="5000"/>
        指定Trigger的重复次数
        <property name="repeatCount" value="5"/>
    </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="myJob_trigger"/>
        <!-- 指定Trigger的名称 -->
        <property name="group" value="myJob_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="simpleTrigger"/> -->
                <ref bean="cronTrigger"/>
            </list>
        </property>
    </bean>

</beans>
```

JDBC方式

一、执行sql

```
#
# In your Quartz properties file, you'll need to set
# org.quartz.jobStore.driverDelegateClass = org.quartz.impl.jdbcjobstore.StdJDBCDelegate
#
#
# By: Ron Cordell - roncordell
#  I didn't see this anywhere, so I thought I'd post it here. This is the script from Quartz to
create the tables in a MySQL database, modified to use INNODB instead of MYISAM.

DROP TABLE IF EXISTS QRTZ_FIRED_TRIGGERS;
DROP TABLE IF EXISTS QRTZ_PAUSED_TRIGGER_GRPS;
DROP TABLE IF EXISTS QRTZ_SCHEDULER_STATE;
DROP TABLE IF EXISTS QRTZ_LOCKS;
```

```
DROP TABLE IF EXISTS QRTZ_SIMPLE_TRIGGERS;
DROP TABLE IF EXISTS QRTZ_SIMPROP_TRIGGERS;
DROP TABLE IF EXISTS QRTZ_CRON_TRIGGERS;
DROP TABLE IF EXISTS QRTZ_BLOB_TRIGGERS;
DROP TABLE IF EXISTS QRTZ_TRIGGERS;
DROP TABLE IF EXISTS QRTZ_JOB_DETAILS;
DROP TABLE IF EXISTS QRTZ_CALENDARS;

CREATE TABLE QRTZ_JOB_DETAILS(
SCHED_NAME VARCHAR(120) NOT NULL,
JOB_NAME VARCHAR(200) NOT NULL,
JOB_GROUP VARCHAR(200) NOT NULL,
DESCRIPTION VARCHAR(250) NULL,
JOB_CLASS_NAME VARCHAR(250) NOT NULL,
IS_DURABLE VARCHAR(1) NOT NULL,
IS_NONCONCURRENT VARCHAR(1) NOT NULL,
IS_UPDATE_DATA VARCHAR(1) NOT NULL,
REQUESTS_RECOVERY VARCHAR(1) NOT NULL,
JOB_DATA BLOB NULL,
PRIMARY KEY (SCHED_NAME,JOB_NAME,JOB_GROUP))
ENGINE=InnoDB;

CREATE TABLE QRTZ_TRIGGERS (
SCHED_NAME VARCHAR(120) NOT NULL,
TRIGGER_NAME VARCHAR(200) NOT NULL,
TRIGGER_GROUP VARCHAR(200) NOT NULL,
JOB_NAME VARCHAR(200) NOT NULL,
JOB_GROUP VARCHAR(200) NOT NULL,
DESCRIPTION VARCHAR(250) NULL,
NEXT_FIRE_TIME BIGINT(13) NULL,
PREV_FIRE_TIME BIGINT(13) NULL,
PRIORITY INTEGER NULL,
TRIGGER_STATE VARCHAR(16) NOT NULL,
TRIGGER_TYPE VARCHAR(8) NOT NULL,
START_TIME BIGINT(13) NOT NULL,
END_TIME BIGINT(13) NULL,
CALENDAR_NAME VARCHAR(200) NULL,
MISFIRE_INSTR SMALLINT(2) NULL,
JOB_DATA BLOB NULL,
PRIMARY KEY (SCHED_NAME,TRIGGER_NAME,TRIGGER_GROUP),
FOREIGN KEY (SCHED_NAME,JOB_NAME,JOB_GROUP)
REFERENCES QRTZ_JOB_DETAILS(SCHED_NAME,JOB_NAME,JOB_GROUP))
ENGINE=InnoDB;

CREATE TABLE QRTZ_SIMPLE_TRIGGERS (
SCHED_NAME VARCHAR(120) NOT NULL,
TRIGGER_NAME VARCHAR(200) NOT NULL,
TRIGGER_GROUP VARCHAR(200) NOT NULL,
REPEAT_COUNT BIGINT(7) NOT NULL,
REPEAT_INTERVAL BIGINT(12) NOT NULL,
TIMES_TRIGGERED BIGINT(10) NOT NULL,
PRIMARY KEY (SCHED_NAME,TRIGGER_NAME,TRIGGER_GROUP),
FOREIGN KEY (SCHED_NAME,TRIGGER_NAME,TRIGGER_GROUP)
```

```
REFERENCES QRTZ_TRIGGERS(SCHED_NAME,TRIGGER_NAME,TRIGGER_GROUP))
ENGINE=InnoDB;

CREATE TABLE QRTZ_CRON_TRIGGERS (
SCHED_NAME VARCHAR(120) NOT NULL,
TRIGGER_NAME VARCHAR(200) NOT NULL,
TRIGGER_GROUP VARCHAR(200) NOT NULL,
CRON_EXPRESSION VARCHAR(120) NOT NULL,
TIME_ZONE_ID VARCHAR(80),
PRIMARY KEY (SCHED_NAME,TRIGGER_NAME,TRIGGER_GROUP),
FOREIGN KEY (SCHED_NAME,TRIGGER_NAME,TRIGGER_GROUP)
REFERENCES QRTZ_TRIGGERS(SCHED_NAME,TRIGGER_NAME,TRIGGER_GROUP))
ENGINE=InnoDB;

CREATE TABLE QRTZ_SIMPROP_TRIGGERS
  (
    SCHED_NAME VARCHAR(120) NOT NULL,
    TRIGGER_NAME VARCHAR(200) NOT NULL,
    TRIGGER_GROUP VARCHAR(200) NOT NULL,
    STR_PROP_1 VARCHAR(512) NULL,
    STR_PROP_2 VARCHAR(512) NULL,
    STR_PROP_3 VARCHAR(512) NULL,
    INT_PROP_1 INT NULL,
    INT_PROP_2 INT NULL,
    LONG_PROP_1 BIGINT NULL,
    LONG_PROP_2 BIGINT NULL,
    DEC_PROP_1 NUMERIC(13,4) NULL,
    DEC_PROP_2 NUMERIC(13,4) NULL,
    BOOL_PROP_1 VARCHAR(1) NULL,
    BOOL_PROP_2 VARCHAR(1) NULL,
    PRIMARY KEY (SCHED_NAME,TRIGGER_NAME,TRIGGER_GROUP),
    FOREIGN KEY (SCHED_NAME,TRIGGER_NAME,TRIGGER_GROUP)
    REFERENCES QRTZ_TRIGGERS(SCHED_NAME,TRIGGER_NAME,TRIGGER_GROUP))
ENGINE=InnoDB;

CREATE TABLE QRTZ_BLOB_TRIGGERS (
SCHED_NAME VARCHAR(120) NOT NULL,
TRIGGER_NAME VARCHAR(200) NOT NULL,
TRIGGER_GROUP VARCHAR(200) NOT NULL,
BLOB_DATA BLOB NULL,
PRIMARY KEY (SCHED_NAME,TRIGGER_NAME,TRIGGER_GROUP),
INDEX (SCHED_NAME,TRIGGER_NAME, TRIGGER_GROUP),
FOREIGN KEY (SCHED_NAME,TRIGGER_NAME,TRIGGER_GROUP)
REFERENCES QRTZ_TRIGGERS(SCHED_NAME,TRIGGER_NAME,TRIGGER_GROUP))
ENGINE=InnoDB;

CREATE TABLE QRTZ_CALENDARS (
SCHED_NAME VARCHAR(120) NOT NULL,
CALENDAR_NAME VARCHAR(200) NOT NULL,
CALENDAR BLOB NOT NULL,
PRIMARY KEY (SCHED_NAME,CALENDAR_NAME))
ENGINE=InnoDB;
```

```
CREATE TABLE QRTZ_PAUSED_TRIGGER_GRPS (
SCHED_NAME VARCHAR(120) NOT NULL,
TRIGGER_GROUP VARCHAR(200) NOT NULL,
PRIMARY KEY (SCHED_NAME,TRIGGER_GROUP))
ENGINE=InnoDB;

CREATE TABLE QRTZ_FIRED_TRIGGERS (
SCHED_NAME VARCHAR(120) NOT NULL,
ENTRY_ID VARCHAR(95) NOT NULL,
TRIGGER_NAME VARCHAR(200) NOT NULL,
TRIGGER_GROUP VARCHAR(200) NOT NULL,
INSTANCE_NAME VARCHAR(200) NOT NULL,
FIRED_TIME BIGINT(13) NOT NULL,
SCHED_TIME BIGINT(13) NOT NULL,
PRIORITY INTEGER NOT NULL,
STATE VARCHAR(16) NOT NULL,
JOB_NAME VARCHAR(200) NULL,
JOB_GROUP VARCHAR(200) NULL,
IS_NONCONCURRENT VARCHAR(1) NULL,
REQUESTS_RECOVERY VARCHAR(1) NULL,
PRIMARY KEY (SCHED_NAME,ENTRY_ID))
ENGINE=InnoDB;

CREATE TABLE QRTZ_SCHEDULER_STATE (
SCHED_NAME VARCHAR(120) NOT NULL,
INSTANCE_NAME VARCHAR(200) NOT NULL,
LAST_CHECKIN_TIME BIGINT(13) NOT NULL,
CHECKIN_INTERVAL BIGINT(13) NOT NULL,
PRIMARY KEY (SCHED_NAME,INSTANCE_NAME))
ENGINE=InnoDB;

CREATE TABLE QRTZ_LOCKS (
SCHED_NAME VARCHAR(120) NOT NULL,
LOCK_NAME VARCHAR(40) NOT NULL,
PRIMARY KEY (SCHED_NAME,LOCK_NAME))
ENGINE=InnoDB;

CREATE INDEX IDX_QRTZ_J_REQ_RECOVERY ON QRTZ_JOB_DETAILS(SCHED_NAME,REQUESTS_RECOVERY);
CREATE INDEX IDX_QRTZ_J_GRP ON QRTZ_JOB_DETAILS(SCHED_NAME,JOB_GROUP);

CREATE INDEX IDX_QRTZ_T_J ON QRTZ_TRIGGERS(SCHED_NAME,JOB_NAME,JOB_GROUP);
CREATE INDEX IDX_QRTZ_T_JG ON QRTZ_TRIGGERS(SCHED_NAME,JOB_GROUP);
CREATE INDEX IDX_QRTZ_T_C ON QRTZ_TRIGGERS(SCHED_NAME,CALENDAR_NAME);
CREATE INDEX IDX_QRTZ_T_G ON QRTZ_TRIGGERS(SCHED_NAME,TRIGGER_GROUP);
CREATE INDEX IDX_QRTZ_T_STATE ON QRTZ_TRIGGERS(SCHED_NAME,TRIGGER_STATE);
CREATE INDEX IDX_QRTZ_T_N_STATE ON
QRTZ_TRIGGERS(SCHED_NAME,TRIGGER_NAME,TRIGGER_GROUP,TRIGGER_STATE);
CREATE INDEX IDX_QRTZ_T_N_G_STATE ON QRTZ_TRIGGERS(SCHED_NAME,TRIGGER_GROUP,TRIGGER_STATE);
CREATE INDEX IDX_QRTZ_T_NEXT_FIRE_TIME ON QRTZ_TRIGGERS(SCHED_NAME,NEXT_FIRE_TIME);
CREATE INDEX IDX_QRTZ_T_NFT_ST ON QRTZ_TRIGGERS(SCHED_NAME,TRIGGER_STATE,NEXT_FIRE_TIME);
CREATE INDEX IDX_QRTZ_T_NFT_MISFIRE ON QRTZ_TRIGGERS(SCHED_NAME,MISFIRE_INSTR,NEXT_FIRE_TIME);
CREATE INDEX IDX_QRTZ_T_NFT_ST_MISFIRE ON
QRTZ_TRIGGERS(SCHED_NAME,MISFIRE_INSTR,NEXT_FIRE_TIME,TRIGGER_STATE);
```

```
CREATE INDEX IDX_QRTZ_T_NFT_ST_MISFIRE_GRP ON
QRTZ_TRIGGERS(SCHED_NAME,MISFIRE_INSTR,NEXT_FIRE_TIME,TRIGGER_GROUP,TRIGGER_STATE);

CREATE INDEX IDX_QRTZ_FT_TRIG_INST_NAME ON QRTZ_FIRED_TRIGGERS(SCHED_NAME,INSTANCE_NAME);
CREATE INDEX IDX_QRTZ_FT_INST_JOB_REQ_RCVRY ON
QRTZ_FIRED_TRIGGERS(SCHED_NAME,INSTANCE_NAME,REQUESTS_RECOVERY);
CREATE INDEX IDX_QRTZ_FT_J_G ON QRTZ_FIRED_TRIGGERS(SCHED_NAME,JOB_NAME,JOB_GROUP);
CREATE INDEX IDX_QRTZ_FT_JG ON QRTZ_FIRED_TRIGGERS(SCHED_NAME,JOB_GROUP);
CREATE INDEX IDX_QRTZ_FT_T_G ON QRTZ_FIRED_TRIGGERS(SCHED_NAME,TRIGGER_NAME,TRIGGER_GROUP);
CREATE INDEX IDX_QRTZ_FT_TG ON QRTZ_FIRED_TRIGGERS(SCHED_NAME,TRIGGER_GROUP);

commit;
```

二、quartz.properties

```
#============================================================================
# Configure Main Scheduler Properties
#============================================================================
org.quartz.scheduler.instanceName: dufy_test
org.quartz.scheduler.instanceId = AUTO
org.quartz.scheduler.rmi.export: false
org.quartz.scheduler.rmi.proxy: false
org.quartz.scheduler.wrapJobExecutionInUserTransaction: false
#============================================================================
# Configure ThreadPool
#============================================================================
org.quartz.threadPool.class: org.quartz.simpl.SimpleThreadPool
org.quartz.threadPool.threadCount: 2
org.quartz.threadPool.threadPriority: 5
org.quartz.threadPool.threadsInheritContextClassLoaderOfInitializingThread: true

org.quartz.jobStore.misfireThreshold: 60000
#============================================================================
# Configure JobStore
#============================================================================
#default config
#org.quartz.jobStore.class: org.quartz.simpl.RAMJobStore
#持久化配置
org.quartz.jobStore.class:org.quartz.impl.jdbcjobstore.JobStoreTX
org.quartz.jobStore.driverDelegateClass:org.quartz.impl.jdbcjobstore.StdJDBCDelegate
org.quartz.jobStore.useProperties:true

#============================================================================
#havent cluster spring
#============================================================================
org.quartz.jobStore.isClustered = false
#数据库表前缀
org.quartz.jobStore.tablePrefix:qrtz_
org.quartz.jobStore.dataSource:qzDS

#============================================================================
# Configure Datasources
```

```
#============================================================================
#JDBC驱动
org.quartz.dataSource.qzDS.driver:com.mysql.jdbc.Driver
org.quartz.dataSource.qzDS.URL:jdbc:mysql://localhost:3306/quartz_test
org.quartz.dataSource.qzDS.user:root
org.quartz.dataSource.qzDS.password:root
org.quartz.dataSource.qzDS.maxConnection:10
```

二.任务类Job

```
/**
 * QuartZ任务调度
 *
 * 1.实现Job接口
 * 2.重写execute方法
 *
 * @version 2018年8月2日下午2:28:07
 * @author zhuwenbin
 */
public class MyJob implements Job {

    private Logger log = LoggerFactory.getLogger(this.getClass());

    @Override
    public void execute(JobExecutionContext context) throws JobExecutionException {
        log.info(">>>>>正在使用QuartZ任务调度框架");
    }

}
```

三.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"
xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context.xsd">

    <!-- 1.配置jdbc文件 -->
    <bean id="propertyConfigurer"
class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
            <property name="locations" value="classpath:jdbc.properties"/>
    </bean>

    <!-- 2.配置数据源 , 使用的alibba的数据库-->
     <bean id="dataSource" class="com.alibaba.druid.pool.DruidDataSource" init-method="init"
destroy-method="close">
```

```
        <!-- 基本属性 url、user、password -->
        <property name="driverClassName" value="${jdbc_driverClassName}"/>
        <property name="url" value="${jdbc_url}"/>
        <property name="username" value="${jdbc_username}"/>
        <property name="password" value="${jdbc_password}"/>

        <!-- 配置初始化大小、最小、最大 -->
        <property name="initialSize" value="10"/>
        <property name="minIdle" value="10"/>
        <property name="maxActive" value="50"/>

        <!-- 配置获取连接等待超时的时间 -->
        <property name="maxWait" value="60000"/>
        <!-- 配置间隔多久才进行一次检测，检测需要关闭的空闲连接，单位是毫秒 -->
        <property name="timeBetweenEvictionRunsMillis" value="60000" />

        <!-- 配置一个连接在池中最小生存的时间，单位是毫秒 -->
        <property name="minEvictableIdleTimeMillis" value="300000" />

        <property name="validationQuery" value="SELECT 'x'" />
        <property name="testWhileIdle" value="true" />
        <property name="testOnBorrow" value="false" />
        <property name="testOnReturn" value="false" />

        <!-- 打开PSCache，并且指定每个连接上PSCache的大小  如果用Oracle，则把poolPreparedStatements
配置为true，mysql可以配置为false。-->
        <property name="poolPreparedStatements" value="false" />
        <property name="maxPoolPreparedStatementPerConnectionSize" value="20" />

        <!-- 配置监控统计拦截的filters -->
        <property name="filters" value="wall,stat" />
    </bean>

    <!-- quartz调度器 -->
    <bean name="quartzScheduler"
class="org.springframework.scheduling.quartz.SchedulerFactoryBean" >
        <property name="dataSource" ref ="dataSource" />
        <property name="applicationContextSchedulerContextKey" value="applicationContextKey"/>
        <property name="configLocation" value="classpath:quartz.properties"/>
    </bean>

</beans>
```

四.任务运行类

```
@Service
public interface QuartzServiceImpl implements QuartzService {

    @Autowired
    private Scheduler quartzScheduler;

    /**
```

```
     * addJob(方法描述：添加一个定时任务) <br />
     * (方法适用条件描述： – 可选)
     *
     * @param jobName
     *            作业名称
     * @param jobGroupName
     *            作业组名称
     * @param triggerName
     *            触发器名称
     * @param triggerGroupName
     *            触发器组名称
     * @param cls
     *            定时任务的class
     * @param cron
     *            时间表达式 void
     * @exception
     * @since 1.0.0
     */
    public void addJob(String jobName, String jobGroupName,String triggerName, String
triggerGroupName, Class cls, String cron) {
        try {
            // 获取调度器
            Scheduler sched = quartzScheduler;
            // 创建一项作业
            JobDetail job = JobBuilder.newJob(cls)
                    .withIdentity(jobName, jobGroupName).build();
            // 创建一个触发器
            CronTrigger trigger = TriggerBuilder.newTrigger()
                    .withIdentity(triggerName, triggerGroupName)
                    .withSchedule(CronScheduleBuilder.cronSchedule(cron))
                    .build();
            // 告诉调度器使用该触发器来安排作业
            sched.scheduleJob(job, trigger);
            // 启动
            if (!sched.isShutdown()) {
                sched.start();
            }
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
    }

    /**
     *
     * @param oldjobName 原job name
     * @param oldjobGroup 原job group
     * @param oldtriggerName 原 trigger name
     * @param oldtriggerGroup 原 trigger group
     * @param jobName
     * @param jobGroup
     * @param triggerName
     * @param triggerGroup
     * @param cron
```

```
     */
    public boolean modifyJobTime(String oldjobName,String oldjobGroup, String oldtriggerName,
String oldtriggerGroup, String jobName, String jobGroup,String triggerName, String triggerGroup,
String cron) {
        try {
            Scheduler sched = quartzScheduler;
            CronTrigger trigger = (CronTrigger) sched.getTrigger(TriggerKey
                    .triggerKey(oldtriggerName, oldtriggerGroup));
            if (trigger == null) {
                return false;
            }

            JobKey jobKey = JobKey.jobKey(oldjobName, oldjobGroup);
            TriggerKey triggerKey = TriggerKey.triggerKey(oldtriggerName,
                    oldtriggerGroup);

            JobDetail job = sched.getJobDetail(jobKey);
            Class jobClass = job.getJobClass();
            // 停止触发器
            sched.pauseTrigger(triggerKey);
            // 移除触发器
            sched.unscheduleJob(triggerKey);
            // 删除任务
            sched.deleteJob(jobKey);

            addJob(jobName, jobGroup, triggerName, triggerGroup, jobClass,
                    cron);

            return true;
        } catch (Exception e) {
            throw new RuntimeException(e);
        }

    }

    /**
     * 修改触发器调度时间
     * @param triggerName  触发器名称
     * @param triggerGroupName  触发器组名称
     * @param cron cron表达式
     */
    public void modifyJobTime(String triggerName,
            String triggerGroupName, String cron) {
        try {
            Scheduler sched = quartzScheduler;
            CronTrigger trigger = (CronTrigger) sched.getTrigger(TriggerKey
                    .triggerKey(triggerName, triggerGroupName));
            if (trigger == null) {
                return;
            }
            String oldTime = trigger.getCronExpression();
            if (!oldTime.equalsIgnoreCase(time)) {
                CronTrigger ct = (CronTrigger) trigger;
```

```
                // 修改时间
                ct.getTriggerBuilder()
                        .withSchedule(CronScheduleBuilder.cronSchedule(time))
                        .build();
                // 重启触发器
                sched.resumeTrigger(TriggerKey.triggerKey(triggerName,
                        triggerGroupName));
            }
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
    }

    /**
     * 暂停指定的任务
     * @param jobName 任务名称
     * @param jobGroupName 任务组名称
     * @return
     */
    public void pauseJob(String jobName,String jobGroupName) {
        try {
            quartzScheduler.pauseJob( JobKey.jobKey(jobName, jobGroupName));
        } catch (SchedulerException e) {
            e.printStackTrace();
        }

    }

    /**
     * 恢复指定的任务
     * @param jobName 任务名称
     * @param jobGroupName 任务组名称
     * @return
     */
    public void resumeJob(String jobName,String jobGroupName) {
        try {
            quartzScheduler.resumeJob(JobKey.jobKey(jobName, jobGroupName));
        } catch (SchedulerException e) {
            e.printStackTrace();
        }
    }

    /**
     * 删除指定组任务
     * @param jobName 作业名称
     * @param jobGroupName 作业组名称
     * @param triggerName 触发器名称
     * @param triggerGroupName 触发器组名称
     */
    public void removeJob(String jobName, String jobGroupName,
            String triggerName, String triggerGroupName) {
        try {
            Scheduler sched = quartzScheduler;
```

```
            // 停止触发器
            sched.pauseTrigger(TriggerKey.triggerKey(triggerName,
                    triggerGroupName));
            // 移除触发器
            sched.unscheduleJob(TriggerKey.triggerKey(triggerName,
                    triggerGroupName));
            // 删除任务
            sched.deleteJob(JobKey.jobKey(jobName, jobGroupName));
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
    }


    /**
     * 开始所有定时任务。启动调度器
     */
    public void startSchedule() {
        try {
            Scheduler sched = quartzScheduler;
            sched.start();
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
    }

    /**
     * 关闭调度器
     */
    public void shutdownSchedule() {
        try {
            Scheduler sched = quartzScheduler;
            if (!sched.isShutdown()) {
                sched.shutdown();
            }
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
    }
}
```

cron表达式详解

cron表达式用于配置cronTrigger的实例。cron表达式实际上是由七个子表达式组成。这些表达式之间用空格分隔。

```
1.Seconds （秒）

2.Minutes（分）
```

```
3.Hours（小时）

4.Day-of-Month  （天）

5.Month（月）

6.Day-of-Week  （周）

7.Year（年）
```

例："0 0 12 ? * WED” 意思是：每个星期三的中午12点执行。

个别子表达式可以包含范围或者列表。例如：上面例子中的WED可以换成"MON-FRI"，"MON,WED,FRI"，甚至"MON-WED,SAT"。

子表达式范围：

```
1.Seconds (0~59)

2.Minutes (0~59)

3.Hours (0~23)

4.Day-of-Month (1~31,但是要注意有些月份没有31天)

5.Month (0~11，或者"JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV,DEC")

6.Day-of-Week (1~7,1=SUN 或者"SUN, MON, TUE, WED, THU, FRI, SAT”)

7.Year (1970~2099)
```

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 #
年(可选字段)	empty	1970-2099 , - * /

字符含义：

* ：代表所有可能的值。因此，“*”在Month中表示每个月，在Day-of-Month中表示每天，在Hours表示每小时

- ：表示指定范围。

, ：表示列出枚举值。例如：在Minutes子表达式中，“5,20”表示在5分钟和20分钟触发。

/ ：被用于指定增量。例如：在Minutes子表达式中，“0/15”表示从0分钟开始，每15分钟执行一次。"3/20"表示从第三分钟开始，每20分钟执行一次。和"3,23,43"（表示第3，23，43分钟触发）的含义一样。

? ：用在Day-of-Month和Day-of-Week中，指“没有具体的值”。当两个子表达式其中一个被指定了值以后，为了避免冲突，需要将另外一个的值设为“?”。例如：想在每月20日触发调度，不管20号是星期几，只能用如下写法：0 0 0 20 * ?，其中最后以为只能用“?”，而不能用“*”。

L ：用在day-of-month和day-of-week字串中。它是单词“last”的缩写。它在两个子表达式中的含义是不同的。

在day-of-month中，“L”表示一个月的最后一天，一月31号，3月30号。

在day-of-week中，“L”表示一个星期的最后一天，也就是“7”或者“SAT”

如果“L”前有具体内容，它就有其他的含义了。例如：“6L”表示这个月的倒数第六天。“FRIL”表示这个月的最后一个星期五。

注意：在使用“L”参数时，不要指定列表或者范围，这样会出现问题。

W ：“Weekday”的缩写。只能用在day-of-month字段。用来描叙最接近指定天的工作日（周一到周五）。例如：在day-of-month字段用“15W”指“最接近这个月第15天的工作日”，即如果这个月第15天是周六，那么触发器将会在这个月第14天即周五触发；如果这个月第15天是周日，那么触发器将会在这个月第 16天即周一触发；如果这个月第15天是周二，那么就在触发器这天触发。注意一点：这个用法只会在当前月计算值，不会越过当前月。“W”字符仅能在 day-of-month指明一天，不能是一个范围或列表。也可以用“LW”来指定这个月的最后一个工作日，即最后一个星期五。

：只能用在day-of-week字段。用来指定这个月的第几个周几。例：在day-of-week字段用"6#3" or "FRI#3"指这个月第3个周五（6指周五，3指第3个）。如果指定的日期不存在，触发器就不会触发。

表达式例子：

0 * * * * ? 每1分钟触发一次

0 0 * * * ? 每天每1小时触发一次

0 0 10 * * ? 每天10点触发一次

0 * 14 * * ? 在每天下午2点到下午2:59期间的每1分钟触发

0 30 9 1 * ? 每月1号上午9点半

```
0 15 10 15 * ? 每月15日上午10:15触发

*/5 * * * * ? 每隔5秒执行一次

0 */1 * * * ? 每隔1分钟执行一次

0 0 5-15 * * ? 每天5-15点整点触发

0 0/3 * * * ? 每三分钟触发一次

0 0-5 14 * * ? 在每天下午2点到下午2:05期间的每1分钟触发

0 0/5 14 * * ? 在每天下午2点到下午2:55期间的每5分钟触发

0 0/5 14,18 * * ? 在每天下午2点到2:55期间和下午6点到6:55期间的每5分钟触发

0 0/30 9-17 * * ? 朝九晚五工作时间内每半小时

0 0 10,14,16 * * ? 每天上午10点，下午2点，4点

0 0 12 ? * WED 表示每个星期三中午12点

0 0 17 ? * TUES,THUR,SAT 每周二、四、六下午五点

0 10,44 14 ? 3 WED 每年三月的星期三的下午2:10和2:44触发

0 15 10 ? * MON-FRI 周一至周五的上午10:15触发

0 0 23 L * ? 每月最后一天23点执行一次

0 15 10 L * ? 每月最后一日的上午10:15触发

0 15 10 ? * 6L 每月的最后一个星期五上午10:15触发

0 15 10 * * ? 2005 2005年的每天上午10:15触发

0 15 10 ? * 6L 2002-2005 2002年至2005年的每月的最后一个星期五上午10:15触发

0 15 10 ? * 6#3 每月的第三个星期五上午10:15触发
```

参考

官方文档