# Quartz改造适应集群环境

## 1、配置quartz.properties文件，如下

#==========================================================================

# Configure Main Scheduler Properties

#=========================================================================

org.quartz.scheduler.instanceName: ClusterScheduler

org.quartz.scheduler.instanceId: AUTO

org.quartz.scheduler.skipUpdateCheck: true

#==========================================================================

# Configure ThreadPool

#==========================================================================

org.quartz.threadPool.class: org.quartz.simpl.SimpleThreadPool

org.quartz.threadPool.threadCount: 20

org.quartz.threadPool.threadPriority: 5

#==========================================================================

# Configure JobStore

#==========================================================================

org.quartz.jobStore.misfireThreshold: 60000

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

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

org.quartz.jobStore.useProperties=true

org.quartz.jobStore.dataSource=myDS

org.quartz.jobStore.tablePrefix=QRTZ\_

org.quartz.jobStore.isClustered=true

#=========================================================================

# Configure Datasources

#==========================================================================

org.quartz.dataSource.myDS.driver: net.sourceforge.jtds.jdbc.Driver

org.quartz.dataSource.myDS.URL: amRiYzpqdGRzOnNxbHNlcnZlcjovL2xvY2FsaG9zdDoxNDMzO0RhdGFiYXNlTmFtZT1jaWNz

org.quartz.dataSource.myDS.user: Y2ljcw==

org.quartz.dataSource.myDS.password: Y2ljcw==

org.quartz.dataSource.myDS.maxConnections: 25

org.quartz.dataSource.myDS.validationQuery: select 0

#集群下添加该设置有助于阻止某些数据库在高负载和长时间事物时锁的超时

org.quartz.jobStore.txIsolationLevelSerializable=true

**说明：数据库配置myDS为加密配置重写jar包中的org.quartz.impl. StdSchedulerFactory类**

## 2、创建调度器，并启动

import java.sql.Connection;

import java.sql.Statement;

import java.text.ParseException;

import java.util.List;

import javax.servlet.ServletContext;

import org.apache.log4j.Logger;

import org.quartz.Job;

import org.quartz.JobKey;

import org.quartz.Scheduler;

import org.quartz.SchedulerException;

import org.quartz.SchedulerFactory;

import org.quartz.TriggerKey;

import org.quartz.impl.JobDetailImpl;

import org.quartz.impl.StdSchedulerFactory;

import org.quartz.impl.triggers.CronTriggerImpl;

import org.springframework.orm.hibernate3.SessionFactoryUtils;

import resource.bean.report.ReportJobConfig;

import com.huateng.ebank.business.common.ROOTDAO;

import com.huateng.ebank.business.common.ROOTDAOUtils;

import com.huateng.ebank.business.innerinterface.ITimedScheduler;

import com.huateng.ebank.framework.exceptions.CommonException;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* 定时器类

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

public class CICSTimedScheduler implements ITimedScheduler {

private static Logger logger = Logger.getLogger(CICSTimedScheduler.class);

private static String JOB\_GROUP\_NAME = "CICS\_PICS\_JOB";

private static String TRIGGER\_GROUP\_NAME = "CICS\_PICS\_JOB\_TRIGGER";

private static ServletContext ctx = null;

private static SchedulerFactory sf = new StdSchedulerFactory();

public void runNow(String jobid) {

try {

Scheduler sched = sf.getScheduler();

JobKey JobKey = new JobKey(jobid, JOB\_GROUP\_NAME);

sched.triggerJob(JobKey);

} catch (SchedulerException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

// 初始化加载

public void run(ServletContext context) throws CommonException {

ctx = context;

Connection conn = null;

Statement stmt = null;

try {

ROOTDAO rootdao = ROOTDAOUtils.getROOTDAO();

conn = SessionFactoryUtils.getDataSource(rootdao.getSessionFactory()).getConnection();

stmt = conn.createStatement();

stmt.addBatch("delete from QRTZ\_CALENDARS");

stmt.addBatch("delete from QRTZ\_FIRED\_TRIGGERS");

stmt.addBatch("delete from QRTZ\_PAUSED\_TRIGGER\_GRPS");

stmt.addBatch("delete from QRTZ\_SCHEDULER\_STATE");

stmt.addBatch("delete from QRTZ\_LOCKS");

stmt.addBatch("delete from QRTZ\_SIMPLE\_TRIGGERS");

stmt.addBatch("delete from QRTZ\_CRON\_TRIGGERS");

stmt.addBatch("delete from QRTZ\_BLOB\_TRIGGERS");

stmt.addBatch("delete from QRTZ\_TRIGGERS");

stmt.addBatch("delete from QRTZ\_JOB\_DETAILS");

stmt.addBatch("delete from QRTZ\_SIMPROP\_TRIGGERS");

stmt.executeBatch(); //执行批处理

String hql = "from ReportJobConfig model where model.jobStauts='" + ITimedScheduler.TIMED\_STATUS\_1 + "'";

List<ReportJobConfig> list = rootdao.queryByQL2List(hql);

if (list.size() > 0) {

for (ReportJobConfig jobConfig : list) {

CICSTimedScheduler.addJob(jobConfig.getId(),

(Job) Class.forName(jobConfig.getJobClassName()).newInstance(), jobConfig.getJobTime());

}

}

} catch (Exception e) {

e.printStackTrace();

}finally{

if (stmt != null) {

try {

stmt.close();

} catch (Exception e) {

e.printStackTrace();

}

}

if (conn != null) {

try {

conn.close();

} catch (Exception e) {

e.printStackTrace();

}

}

try {

Scheduler sched = sf.getScheduler();

sched.start();

} catch (SchedulerException e) {

e.printStackTrace();

}

}

}

/\*\*

\* 添加一个定时任务，使用默认的任务组名，触发器名，触发器组名

\*

\* @param jobName

\* 任务名

\* @param job

\* 任务

\* @param time

\* 时间设置，参考quartz说明文档

\* @throws SchedulerException

\* @throws ParseException

\*/

public static void addJob(String jobId, Job job, String time) throws SchedulerException, ParseException {

Scheduler sched = sf.getScheduler();

sched.getContext().put("serContext", ctx);

JobDetailImpl jobDetail = new JobDetailImpl(jobId, JOB\_GROUP\_NAME, job.getClass());// 任务名，任务组，任务执行类

jobDetail.getJobDataMap().put("jobId", jobId);

// 触发器

CronTriggerImpl trigger = new CronTriggerImpl(jobId, TRIGGER\_GROUP\_NAME);// 触发器名,触发器组

trigger.setCronExpression(time);// 触发器时间设定

sched.scheduleJob(jobDetail, trigger);

logger.info("添加任务： " + jobId);

}

/\*\*

\* 移除一个任务

\*

\* @param jobName

\* @param jobGroupName

\* @param triggerName

\* @param triggerGroupName

\* @throws SchedulerException

\*/

public static void removeJob(String jobId) throws SchedulerException {

Scheduler sched = sf.getScheduler();

TriggerKey TriggerKey = new TriggerKey(jobId, TRIGGER\_GROUP\_NAME);

JobKey JobKey = new JobKey(jobId, JOB\_GROUP\_NAME);

sched.pauseTrigger(TriggerKey);// 停止触发器

sched.unscheduleJob(TriggerKey);// 移除触发器

sched.deleteJob(JobKey);// 删除任务

logger.info("移除任务：" + jobId);

}

}

## 3、创建quartz持久化的表（SqlServer版）

IF EXISTS (SELECT \* FROM sysobjects WHERE id = OBJECT\_ID(N'[FK\_QRTZ\_TRIGGERS\_QRTZ\_JOB\_DETAILS]') AND OBJECTPROPERTY(id, N'ISFOREIGNKEY') = 1)

ALTER TABLE [QRTZ\_TRIGGERS] DROP CONSTRAINT FK\_QRTZ\_TRIGGERS\_QRTZ\_JOB\_DETAILS

GO

IF EXISTS (SELECT \* FROM sysobjects WHERE id = OBJECT\_ID(N'[FK\_QRTZ\_CRON\_TRIGGERS\_QRTZ\_TRIGGERS]') AND OBJECTPROPERTY(id, N'ISFOREIGNKEY') = 1)

ALTER TABLE [QRTZ\_CRON\_TRIGGERS] DROP CONSTRAINT FK\_QRTZ\_CRON\_TRIGGERS\_QRTZ\_TRIGGERS

GO

IF EXISTS (SELECT \* FROM sysobjects WHERE id = OBJECT\_ID(N'[FK\_QRTZ\_SIMPLE\_TRIGGERS\_QRTZ\_TRIGGERS]') AND OBJECTPROPERTY(id, N'ISFOREIGNKEY') = 1)

ALTER TABLE [QRTZ\_SIMPLE\_TRIGGERS] DROP CONSTRAINT FK\_QRTZ\_SIMPLE\_TRIGGERS\_QRTZ\_TRIGGERS

GO

IF EXISTS (SELECT \* FROM sysobjects WHERE id = OBJECT\_ID(N'[FK\_QRTZ\_SIMPROP\_TRIGGERS\_QRTZ\_TRIGGERS]') AND OBJECTPROPERTY(id, N'ISFOREIGNKEY') = 1)

ALTER TABLE [QRTZ\_SIMPROP\_TRIGGERS] DROP CONSTRAINT FK\_QRTZ\_SIMPROP\_TRIGGERS\_QRTZ\_TRIGGERS

GO

IF EXISTS (SELECT \* FROM sysobjects WHERE id = OBJECT\_ID(N'[QRTZ\_CALENDARS]') AND OBJECTPROPERTY(id, N'ISUSERTABLE') = 1)

DROP TABLE [QRTZ\_CALENDARS]

GO

IF EXISTS (SELECT \* FROM sysobjects WHERE id = OBJECT\_ID(N'[QRTZ\_CRON\_TRIGGERS]') AND OBJECTPROPERTY(id, N'ISUSERTABLE') = 1)

DROP TABLE [QRTZ\_CRON\_TRIGGERS]

GO

IF EXISTS (SELECT \* FROM sysobjects WHERE id = OBJECT\_ID(N'[QRTZ\_BLOB\_TRIGGERS]') AND OBJECTPROPERTY(id, N'ISUSERTABLE') = 1)

DROP TABLE [QRTZ\_BLOB\_TRIGGERS]

GO

IF EXISTS (SELECT \* FROM sysobjects WHERE id = OBJECT\_ID(N'[QRTZ\_FIRED\_TRIGGERS]') AND OBJECTPROPERTY(id, N'ISUSERTABLE') = 1)

DROP TABLE [QRTZ\_FIRED\_TRIGGERS]

GO

IF EXISTS (SELECT \* FROM sysobjects WHERE id = OBJECT\_ID(N'[QRTZ\_PAUSED\_TRIGGER\_GRPS]') AND OBJECTPROPERTY(id, N'ISUSERTABLE') = 1)

DROP TABLE [QRTZ\_PAUSED\_TRIGGER\_GRPS]

GO

IF EXISTS (SELECT \* FROM sysobjects WHERE id = OBJECT\_ID(N'[QRTZ\_SCHEDULER\_STATE]') AND OBJECTPROPERTY(id, N'ISUSERTABLE') = 1)

DROP TABLE [QRTZ\_SCHEDULER\_STATE]

GO

IF EXISTS (SELECT \* FROM sysobjects WHERE id = OBJECT\_ID(N'[QRTZ\_LOCKS]') AND OBJECTPROPERTY(id, N'ISUSERTABLE') = 1)

DROP TABLE [QRTZ\_LOCKS]

GO

IF EXISTS (SELECT \* FROM sysobjects WHERE id = OBJECT\_ID(N'[QRTZ\_JOB\_DETAILS]') AND OBJECTPROPERTY(id, N'ISUSERTABLE') = 1)

DROP TABLE [QRTZ\_JOB\_DETAILS]

GO

IF EXISTS (SELECT \* FROM sysobjects WHERE id = OBJECT\_ID(N'[QRTZ\_SIMPLE\_TRIGGERS]') AND OBJECTPROPERTY(id, N'ISUSERTABLE') = 1)

DROP TABLE [QRTZ\_SIMPLE\_TRIGGERS]

GO

IF EXISTS (SELECT \* FROM sysobjects WHERE id = OBJECT\_ID(N'[QRTZ\_SIMPROP\_TRIGGERS]') AND OBJECTPROPERTY(id, N'ISUSERTABLE') = 1)

DROP TABLE [QRTZ\_SIMPROP\_TRIGGERS]

GO

IF EXISTS (SELECT \* FROM sysobjects WHERE id = OBJECT\_ID(N'[QRTZ\_TRIGGERS]') AND OBJECTPROPERTY(id, N'ISUSERTABLE') = 1)

DROP TABLE [QRTZ\_TRIGGERS]

GO

CREATE TABLE [QRTZ\_CALENDARS] (

[SCHED\_NAME] [VARCHAR] (120) NOT NULL ,

[CALENDAR\_NAME] [VARCHAR] (200) NOT NULL ,

[CALENDAR] [IMAGE] NOT NULL

) ON [PRIMARY]

GO

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)

) ON [PRIMARY]

GO

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] NOT NULL ,

[SCHED\_TIME] [BIGINT] ,

[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

) ON [PRIMARY]

GO

CREATE TABLE [QRTZ\_PAUSED\_TRIGGER\_GRPS] (

[SCHED\_NAME] [VARCHAR] (120) NOT NULL ,

[TRIGGER\_GROUP] [VARCHAR] (200) NOT NULL

) ON [PRIMARY]

GO

CREATE TABLE [QRTZ\_SCHEDULER\_STATE] (

[SCHED\_NAME] [VARCHAR] (120) NOT NULL ,

[INSTANCE\_NAME] [VARCHAR] (200) NOT NULL ,

[LAST\_CHECKIN\_TIME] [BIGINT] NOT NULL ,

[CHECKIN\_INTERVAL] [BIGINT] NOT NULL

) ON [PRIMARY]

GO

CREATE TABLE [QRTZ\_LOCKS] (

[SCHED\_NAME] [VARCHAR] (120) NOT NULL ,

[LOCK\_NAME] [VARCHAR] (40) NOT NULL

) ON [PRIMARY]

GO

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] [IMAGE] NULL

) ON [PRIMARY]

GO

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] NOT NULL ,

[REPEAT\_INTERVAL] [BIGINT] NOT NULL ,

[TIMES\_TRIGGERED] [BIGINT] NOT NULL

) ON [PRIMARY]

GO

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,

) ON [PRIMARY]

GO

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] [IMAGE] NULL

) ON [PRIMARY]

GO

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] NULL ,

[PREV\_FIRE\_TIME] [BIGINT] NULL ,

[PRIORITY] [INTEGER] NULL ,

[TRIGGER\_STATE] [VARCHAR] (16) NOT NULL ,

[TRIGGER\_TYPE] [VARCHAR] (8) NOT NULL ,

[START\_TIME] [BIGINT] NOT NULL ,

[END\_TIME] [BIGINT] NULL ,

[CALENDAR\_NAME] [VARCHAR] (200) NULL ,

[MISFIRE\_INSTR] [SMALLINT] NULL ,

[JOB\_DATA] [IMAGE] NULL

) ON [PRIMARY]

GO

ALTER TABLE [QRTZ\_CALENDARS] WITH NOCHECK ADD

CONSTRAINT [PK\_QRTZ\_CALENDARS] PRIMARY KEY CLUSTERED

(

[SCHED\_NAME],

[CALENDAR\_NAME]

) ON [PRIMARY]

GO

ALTER TABLE [QRTZ\_CRON\_TRIGGERS] WITH NOCHECK ADD

CONSTRAINT [PK\_QRTZ\_CRON\_TRIGGERS] PRIMARY KEY CLUSTERED

(

[SCHED\_NAME],

[TRIGGER\_NAME],

[TRIGGER\_GROUP]

) ON [PRIMARY]

GO

ALTER TABLE [QRTZ\_FIRED\_TRIGGERS] WITH NOCHECK ADD

CONSTRAINT [PK\_QRTZ\_FIRED\_TRIGGERS] PRIMARY KEY CLUSTERED

(

[SCHED\_NAME],

[ENTRY\_ID]

) ON [PRIMARY]

GO

ALTER TABLE [QRTZ\_PAUSED\_TRIGGER\_GRPS] WITH NOCHECK ADD

CONSTRAINT [PK\_QRTZ\_PAUSED\_TRIGGER\_GRPS] PRIMARY KEY CLUSTERED

(

[SCHED\_NAME],

[TRIGGER\_GROUP]

) ON [PRIMARY]

GO

ALTER TABLE [QRTZ\_SCHEDULER\_STATE] WITH NOCHECK ADD

CONSTRAINT [PK\_QRTZ\_SCHEDULER\_STATE] PRIMARY KEY CLUSTERED

(

[SCHED\_NAME],

[INSTANCE\_NAME]

) ON [PRIMARY]

GO

ALTER TABLE [QRTZ\_LOCKS] WITH NOCHECK ADD

CONSTRAINT [PK\_QRTZ\_LOCKS] PRIMARY KEY CLUSTERED

(

[SCHED\_NAME],

[LOCK\_NAME]

) ON [PRIMARY]

GO

ALTER TABLE [QRTZ\_JOB\_DETAILS] WITH NOCHECK ADD

CONSTRAINT [PK\_QRTZ\_JOB\_DETAILS] PRIMARY KEY CLUSTERED

(

[SCHED\_NAME],

[JOB\_NAME],

[JOB\_GROUP]

) ON [PRIMARY]

GO

ALTER TABLE [QRTZ\_SIMPLE\_TRIGGERS] WITH NOCHECK ADD

CONSTRAINT [PK\_QRTZ\_SIMPLE\_TRIGGERS] PRIMARY KEY CLUSTERED

(

[SCHED\_NAME],

[TRIGGER\_NAME],

[TRIGGER\_GROUP]

) ON [PRIMARY]

GO

ALTER TABLE [QRTZ\_SIMPROP\_TRIGGERS] WITH NOCHECK ADD

CONSTRAINT [PK\_QRTZ\_SIMPROP\_TRIGGERS] PRIMARY KEY CLUSTERED

(

[SCHED\_NAME],

[TRIGGER\_NAME],

[TRIGGER\_GROUP]

) ON [PRIMARY]

GO

ALTER TABLE [QRTZ\_TRIGGERS] WITH NOCHECK ADD

CONSTRAINT [PK\_QRTZ\_TRIGGERS] PRIMARY KEY CLUSTERED

(

[SCHED\_NAME],

[TRIGGER\_NAME],

[TRIGGER\_GROUP]

) ON [PRIMARY]

GO

ALTER TABLE [QRTZ\_CRON\_TRIGGERS] ADD

CONSTRAINT [FK\_QRTZ\_CRON\_TRIGGERS\_QRTZ\_TRIGGERS] FOREIGN KEY

(

[SCHED\_NAME],

[TRIGGER\_NAME],

[TRIGGER\_GROUP]

) REFERENCES [QRTZ\_TRIGGERS] (

[SCHED\_NAME],

[TRIGGER\_NAME],

[TRIGGER\_GROUP]

) ON DELETE CASCADE

GO

ALTER TABLE [QRTZ\_SIMPLE\_TRIGGERS] ADD

CONSTRAINT [FK\_QRTZ\_SIMPLE\_TRIGGERS\_QRTZ\_TRIGGERS] FOREIGN KEY

(

[SCHED\_NAME],

[TRIGGER\_NAME],

[TRIGGER\_GROUP]

) REFERENCES [QRTZ\_TRIGGERS] (

[SCHED\_NAME],

[TRIGGER\_NAME],

[TRIGGER\_GROUP]

) ON DELETE CASCADE

GO

ALTER TABLE [QRTZ\_SIMPROP\_TRIGGERS] ADD

CONSTRAINT [FK\_QRTZ\_SIMPROP\_TRIGGERS\_QRTZ\_TRIGGERS] FOREIGN KEY

(

[SCHED\_NAME],

[TRIGGER\_NAME],

[TRIGGER\_GROUP]

) REFERENCES [QRTZ\_TRIGGERS] (

[SCHED\_NAME],

[TRIGGER\_NAME],

[TRIGGER\_GROUP]

) ON DELETE CASCADE

GO

ALTER TABLE [QRTZ\_TRIGGERS] ADD

CONSTRAINT [FK\_QRTZ\_TRIGGERS\_QRTZ\_JOB\_DETAILS] FOREIGN KEY

(

[SCHED\_NAME],

[JOB\_NAME],

[JOB\_GROUP]

) REFERENCES [QRTZ\_JOB\_DETAILS] (

[SCHED\_NAME],

[JOB\_NAME],

[JOB\_GROUP]

)

GO

表与字段说明：

quartz框架中T\_TASK\_TRIGGERS表 TRIGGER\_STATE 字段显示任务的属性大概状态有这几种：

WAITING:等待

PAUSED:暂停

ACQUIRED:正常执行

BLOCKED：阻塞

ERROR：错误

主要使用以上这几种状态控制调度各节点定时任务

QRTZ\_CALENDARS 以 Blob 类型存储 Quartz 的 Calendar 信息

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\_PAUSED\_TRIGGER\_GRPS 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\_LOCKS 存储程序的悲观锁的信息(假如使用了悲观锁)

QRTZ\_JOB\_DETAILS 存储每一个已配置的 Job 的详细信息

QRTZ\_JOB\_LISTENERS 存储有关已配置的 JobListener的信息

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\_BLOG\_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\_TRIGGER\_LISTENERS 存储已配置的 TriggerListener的信息

QRTZ\_TRIGGERS 存储已配置的 Trigger 的信息

表qrtz\_job\_details:保存job详细信息,该表需要用户根据实际情况初始化

job\_name:集群中job的名字,该名字用户自己可以随意定制,无强行要求

job\_group:集群中job的所属组的名字,该名字用户自己随意定制,无强行要求

job\_class\_name:集群中个notejob实现类的完全包名,quartz就是根据这个路径到classpath找到该job类

is\_durable:是否持久化,把该属性设置为1，quartz会把job持久化到数据库中

job\_data:一个blob字段，存放持久化job对象

表qrtz\_triggers: 保存trigger信息

trigger\_name:trigger的名字,该名字用户自己可以随意定制,无强行要求

trigger\_group:trigger所属组的名字,该名字用户自己随意定制,无强行要求

job\_name:qrtz\_job\_details表job\_name的外键

job\_group:qrtz\_job\_details表job\_group的外键

trigger\_state:当前trigger状态，设置为ACQUIRED,如果设置为WAITING,则job不会触发

trigger\_cron:触发器类型,使用cron表达式

表qrtz\_cron\_triggers:存储cron表达式表

trigger\_name:qrtz\_triggers表trigger\_name的外键

trigger\_group:qrtz\_triggers表trigger\_group的外键

cron\_expression:cron表达式

表qrtz\_scheduler\_state:存储集群中note实例信息，quartz会定时读取该表的信息判断集群中每个实例的当前状态

instance\_name:之前配置文件中org.quartz.scheduler.instanceId配置的名字，就会写入该字段，如果设置为AUTO,quartz会根据物理机名和当前时间产生一个名字

last\_checkin\_time:上次检查时间

checkin\_interval:检查间隔时间

Job 实例

你可以创建一个 Job 类，然后通过创建多个 JobDetail 实例与 Job 关联，并保存到调度器中（每个任务都有自己的属性和 JobDataMap），这个 JobDetail 称为 Job 实例。

例如，你可以创建一个实现了 Job 接口的类，命名为“SalesReportJob”。这个类可以接收一个参数（通过 JobDataMap）用于定义销售报表基于哪个销售人员。它们可以创建多个 Job 实例（使用 JobDetail），例如 “SalesReportForJoe” 和 “SalesReportForMike”，这里使用了由 JobDataMap 传入 “joe” 和 “mike” 作为参数。

当 Trigger 被触发，关联的 JobDetail 将会被加载，并且 Job 类会通过 JobFactory 配置到 Scheduler。默认的 JobFactory 将会简单地调用 Job Class 的 newInstance() 方法，并尝试调用 set 方法将 JobDataMap 中同名的属性设置到 Job 中。

Job 状态和并发

有一组可添加到 Job 的 Annotation，可以影响 Quartz 的行为。

@DisallowConcurrentExecution 添加到 Job 类后，Quartz 将不会同时执行多个 Job 实例（什么是 Job 实例可参看上一节）。

注意措辞。我们用上一节的例子来讲解，如果 “SalesReportJob” 上添加了这个 Annotation，那么同时只能执行一个“SalesReportForJoe”，但是却可以同时执行“SalesReportForMike”。因此，可以说这个约束是基于 JobDetail 的而不是基于 Job 的。

@PersistJobDataAfterExecution 添加到 Job 类后，表示 Quartz 将会在成功执行 execute() 方法后（没有抛出异常）更新 JobDetail 的 JobDataMap，下一次执行相同的任务（JobDetail）将会得到更新后的值，而不是原始的值。就像 @DisallowConcurrentExecution 一样，这个注释基于 JobDetail 而不是 Job 类的实例。

如果你使用了 @PersistJobDataAfterExecution 注释，那么强烈建议你使用 @DisallowConcurrentExecution 注释，这是为了避免出现并发问题，当多个 Job 实例同时执行的时候，到底使用了哪个数据将变得很混乱。