在集群环境下，大家会碰到一直困扰的问题，即多个 APP 下如何用 quartz 协调处理自动化 JOB 。

**大家想象一下，现在有 A** **， B** **， C3** **台机器同时作为集群服务器对外统一提供 SERVICE** **：**

A ， B ， C 3 台机器上各有一个 QUARTZ ，他们会按照即定的 SCHEDULE 自动执行各自的任务。

我们先不说实现什么功能，就说这样的架构其实有点像多线程。

那多线程里就会存在“资源竞争”的问题，即可能产生脏读，脏写，由于三台 APP SERVER 里都有QUARTZ ，因此会存在重复处理 TASK 的现象。

一般外面的解决方案是只在一台 APP 上装 QUARTZ ，其它两台不装，这样集群就形同虚设了；

另一种解决方案是动代码，这样就要影响到原来已经写好的 QUARTZ JOB 的代码了，这对程序开发人员来说比较痛苦；

本人仔细看了一下 Spring 的结构和 QUARTZ 的文档，结合 Quartz 自身可以实例化进数据的特性找到了相关的解决方案。

本方案优点：

1.       每台作为集群点的 APP SERVER 上都可以布署 QUARTZ ；

2.       QUARTZ 的 TASK （ 12 张表）实例化如数据库，基于数据库引擎及 High-Available 的策略（集群的一种策略）自动协调每个节点的 QUARTZ ，当任一一节点的 QUARTZ 非正常关闭或出错时，另几个节点的 QUARTZ 会自动启动；

3.       无需开发人员更改原已经实现的 QUARTZ ，使用 SPRING+ 类反射的机制对原有程序作切面重构；

本人也事先搜索了一些资料，发觉所有目前在 GOOGLE 上或者在各大论坛里提供的解决方案，要么是只解决了一部分，要么是错误的，要么是版本太老，要么就是完全抄别人的。

尤其是在使用 QUARTZ+SPRING 对数据库对象作实例化时会抛错（源于 SPRING 的一个 BUG ），目前网上的解决方案全部是错的或者干脆没说，本人在此方案中也会提出如何解决。

**解决方案：**

1.       把 QUARTZ 的 TASK 实例化进数据库， QUARTZ 只有实例化进入数据库后才能做集群，外面的解决方案说实例化在内存里全部是错的，把quartz-1.8.4/docs/dbTables/tables\_oracle.sql 在 ORACLE9I2 及以上版本中执行一下会生成 12 张表；

2.       生成 quartz.properties 文件，把它放在工程的 src 目录下，使其能够被编译时纳入 class path 。

一般我们的开发人员都喜欢使用 SPRING+QUARTZ ，因此这个 quartz.properties 都不用怎么去写，但是在集群方案中 quartz.properties 必写，如果不写 quartz 会调用自身 jar 包中的 quartz.properties 作为默认属性文件，同时修改 quartz.xml 文件。

**Quartz.xml** **文件的内容 :**

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

<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN" "<http://www.springframework.org/dtd/spring-beans.dtd> ">

<beans>

                <bean id="mapScheduler" lazy-init="false" autowire="no"

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

**<property name="configLocation" value="classpath:quartz.properties" />**

                                <property name="triggers">

                                                <list>

                                                                <ref bean="cronTrigger" />

                                                </list>

                                </property>

                                <!— 就是下面这句，因为该 bean 只能使用类反射来重构

**<property name="applicationContextSchedulerContextKey" value="applicationContext" />**

</bean>

**quartz.properties** **文件的内容：**

org.quartz.scheduler.instanceName = mapScheduler

org.quartz.scheduler.instanceId = AUTO

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

 org.quartz.jobStore.driverDelegateClass = org.quartz.impl.jdbcjobstore.oracle.weblogic.WebLogicOracleDelegate

 org.quartz.jobStore.dataSource = myXADS

 org.quartz.jobStore.tablePrefix = QRTZ\_

 org.quartz.jobStore.isClustered = true

 org.quartz.dataSource.myXADS.jndiURL=jdbc/TestQuartzDS

 org.quartz.dataSource.myXADS.jndiAlwaysLookup = DB\_JNDI\_ALWAYS\_LOOKUP

 org.quartz.dataSource.myXADS.java.naming.factory.initial = weblogic.jndi.WLInitialContextFactory

 org.quartz.dataSource.myXADS.java.naming.provider.url = t3://localhost:7020

 org.quartz.dataSource.myXADS.java.naming.security.principal = weblogic

 org.quartz.dataSource.myXADS.java.naming.security.credentials = weblogic

3.       重写 quartz 的 QuartzJobBean 类

原因是在使用 quartz+spring 把 quartz 的 task 实例化进入数据库时，会产生： serializable 的错误，原因在于：

<bean id="jobtask" class="org.springframework.scheduling.quartz.**MethodInvokingJobDetailFactoryBean** ">

                                <property name="targetObject">

                                                <ref bean="quartzJob"/>

                                </property>

                                <property name="targetMethod">

                                                <value>execute</value>

                                </property>

</bean>

这个 MethodInvokingJobDetailFactoryBean 类中的 methodInvoking 方法，是不支持序列化的，因此在把QUARTZ 的 TASK 序列化进入数据库时就会抛错。网上有说把 SPRING 源码拿来，修改一下这个方案，然后再打包成 SPRING.jar 发布，这些都是不好的方法，是不安全的。

必须根据 QuartzJobBean 来重写一个自己的类，然后使用 SPRING 把这个重写的类（我们就名命它为：MyDetailQuartzJobBean ）注入 appContext 中后，再使用 AOP 技术反射出原有的 quartzJobx( 就是开发人员原来已经做好的用于执行 QUARTZ 的 JOB 的执行类 ) 。

**下面来看** MyDetailQuartzJobBean **类：**

public class MyDetailQuartzJobBean extends QuartzJobBean {

                protected final Log logger = LogFactory.getLog(getClass());

                private String targetObject;

                private String targetMethod;

                private ApplicationContext ctx;

                protected void executeInternal(JobExecutionContext context)

                                                throws JobExecutionException {

                                try {

                                                logger.info("execute [" + targetObject + "] at once>>>>>>");

                                                Object otargetObject = ctx.getBean(targetObject);

                                                Method m = null;

                                                try {

                                                                m = otargetObject.getClass().getMethod(targetMethod,

                                                                                                new Class[] {});

                                                                m.invoke(otargetObject, new Object[] {});

                                                } catch (SecurityException e) {

                                                                logger.error(e);

                                                } catch (NoSuchMethodException e) {

                                                                logger.error(e);

                                                }

                                } catch (Exception e) {

                                                throw new JobExecutionException(e);

                                }

                }

                public void setApplicationContext(ApplicationContext applicationContext){

                                this.ctx=applicationContext;

                }

                public void setTargetObject(String targetObject) {

                                this.targetObject = targetObject;

                }

                public void setTargetMethod(String targetMethod) {

                                this.targetMethod = targetMethod;

                }

}

**再来看完整的 quartz.xml** **（注意红色加粗部分尤为重要）：**

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

<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN" "<http://www.springframework.org/dtd/spring-beans.dtd> ">

<beans>

                <bean id="mapScheduler" lazy-init="false" autowire="no"

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

**<property name="configLocation" value="classpath:quartz.properties" />**

                                <property name="triggers">

                                                <list>

                                                                <ref bean="cronTrigger" />

                                                </list>

                                </property>

                                <property name=" **applicationContextSchedulerContextKey** " value=" **applicationContext** " />

                </bean>

                <bean id="quartzJob" class="com.testcompany.framework.quartz.QuartzJob">

                </bean>

**<bean id="jobTask" class="org.springframework.scheduling.quartz.JobDetailBean">**

**<property name="jobClass">**

**<value>com.testcompany.framework.quartz. MyDetailQuartzJobBean** **</value>**

**</property>**

**<property name="jobDataAsMap">**

**<map>**

**<entry key="quartzJob" value="quartzJob" />**

**<entry key="targetMethod" value="execute" />**

**</map>**

**</property>**

**</bean>**

                <bean id="cronTrigger" class="org.springframework.scheduling.quartz.CronTriggerBean">

                                <property name="jobDetail">

                                                <ref bean="jobTask" />

                                </property>

                                <property name="cronExpression">

                                                <value>0/5 \* \* \* \* ?</value>

                                </property>

                </bean>

</beans>

4.       下载最新的 quartz1.8 版，把 quartz-all-1.8.4.jar, quartz-oracle-1.8.4.jar,quartz-weblogic-1.8.4.jar 这三个包放到 web-inf/lib 目录下，布署。

**测试：**

几个节点都带有 quartz 任务，此时只有一台 quartz 在运行，另几个节点上的 quartz 没有运行。

此时手动 shutdown 那台运行 QUARTZ （在程序里加 system.out.println(“execute once…”), 运行 quartz的那个节点在后台会打印 execute once ）的节点，过了 7 秒左右，另一个节点的 quartz 自动监测到了集群中运行着的 quartz 的 instance 已经 shutdown ，因此 quartz 集群会自动把任一台可用的 APP 上启动起一个 quartz job 的任务。

自此， QUARTZ 使用 HA 策略的集群大功告成，不用改原有代码，配置一下我们就可作到 QUARTZ 的集群与自动错误冗余。