## Elastic-job 开发及自测使用说明简述

## 一，概要

Elastic-Job是一个分布式调度解决方案，由两个相互独立的子项目Elastic-Job-Lite和Elastic-Job-Cloud组成。目前项目中只使用到Elastic-Job-Lite ，所以在本文档中只涉及到Elastic-Job-Lite 的内容。

Elastic-Job-Lite定位为轻量级无中心化解决方案，使用jar包的形式提供分布式任务的协调服务。按我自己的理解，就是定时任务管理中心。它提供了两种作业模式，SimpleJob和DataflowJob。在当前使用的2.1.5版本中两种工作模式均可实现分片作业。分片作业需要多台服务器同时工作才能体现出能力。

## 二，任务执作说明

*注：本文档旨在让开发人员快速进入开发，如需详细了解，可自行学习。*

### 任务的启动

任务的启动过程，就是任务实例和zookeeper进行交互的过程。每个实例在启动过程中，会把自身的信息注册到zookeeper中去。并完成选举和分片策略的设置，也就是完成上文一些zookeeper节点的创建和持久化。整个启动的过程都在JobScheduler.init()方法中完成。其中最重要的方法registerStartUpInfo完成了监听，选举持久化数据，以及设置分片标志位（为了任务执行是主节点进行分片算法）等工作。init()方法中完成了配置和注册之后，相关的参数被传递给了JobScheduleController类，这个类就是quartz的封装。之前配置的任务执行类和cron表达式被转换成JobDetail和Trigger这两个quartz的类，然后通过quartz的scheduler.start触发任务。等待任务的执行。

### 任务的执行

任务的执行依赖于quartz job的触发。elastic-job的LiteJob类继承自quartz的Job类，在任务触发的时候，添加额外的逻辑处理。LiteJob的执行器AbstractElasticJobExecutor有两个具体的实现，SimpleJobExecutor和DataflowJobExecutor，各自执行SimpleJob和DataflowJob两种Job类型。Job触发的时候，SimpleJobExecutor或者DataflowJobExecutor会被new一次，重新从缓存中加载Job配置并执行。

LiteJob把任务的执行分为执行前，执行（业务代码的执行），和执行后三个阶段。在执行前阶段中主要实现分片策略的执行（shardingIfNecessary方法），记录事件，执行监听事件等等。而执行后阶段主要处理错过执行的相关任务以及执行监听事件。

## 三，代码编写及配置

#### 代码编写

1. job分为2中类型，分别实现com.dangdang.ddframe.job.api.simple.SimpleJob和com.dangdang.ddframe.job.api.dataflow.DataflowJob。

SimpleJob和DataFlowJob的区别：

*SimpleJob重写接口的execute接口，业务逻辑在execute方法中实现，目前当使用DataFlowJob时默认为流数据处理模式。DataFlowJob 重写fetchData 和 processData方法，processData的第二个参数data是fetchData的返回结果，DataFlowJob默认流式处理，只有当fetchData的返回结果为null或者空集合时，processData停止执行。*

2. SimpleJob的实现放置于crm-job/src/main/java/com/aii/crm/job/instance/simple/路径下

例如：com\aii\crm\job\instance\simple\ResPhoneReleaseJob.java

注：该job中暂不能使用注解方式调用获取待数据的方法

3. DataflowJob的实现放置/crm-job/src/main/java/com/aii/crm/job/instance/dataflow路径下

例如：com\aii\crm\job\instance\dataflow\ExpirationHandling4NotifyJob.java（千言万语不及一个栗子）

注：该job中暂不能使用注解方式获取待数据的方法

4. Pom.xml文件中加入dubbo服务的api

5. aplication-job-consume.xml中配置注入的dubbo的服务接口

6. Base数据库job\_config表中配置对应的数据

|  |  |
| --- | --- |
| JOB\_ID | 必填，表主键 |
| JOB\_NAME | 必填，任务的名称，唯一值 |
| JOB\_PATH | 任务所在代码中的路径 |
| JOB\_CRON | Cron表达式，使用quartz能解析的表达式（定时器如果不懂，可自行查阅学习） |
| JOB\_SHARDING\_COUNT | 任务的分片总数，job不支持同一台服务器多实例。当多个服务器部署job时配置该字段 |
| JOB\_SHARDING\_PARAMETER | 分片参数（与分片总数相对应，格式：0=a,1=b,… … ‘=’前用数字按顺序从0开始，值不能大于等于分片总数，’=’后面的随意定） |
| JOB\_TYPE | 必填，0 是 SimpleJob， 1是DataFlowJob |
| JOB\_PARAMETER | 配置job内的参数（*建议：配置在该表，方便管理及修改*） |
| STREAMING\_PROCESS | 目前当使用DataFlowJob时默认为流数据处理模式，后面可能会放开该功能 |
| STATE | 必填，0 是不可用，1 是可用 |
| JOB\_DESCRIPTION | 任务描述 |

#### 配置数据

INSERT INTO `base`.`job\_config` (`JOB\_ID`, `JOB\_NAME`, `JOB\_PATH`, `JOB\_CRON`, `JOB\_SHARDING\_COUNT`, `JOB\_SHARDING\_PARAMETER`, `JOB\_TYPE`, `JOB\_PARAMETER`, `JOB\_DESCRIPTION`, `FAILOVER`, `MISFIRE`, `STREAMING\_PROCESS`, `STATE`, `TRACE`, `RECOVER`) VALUES ('10023', 'ExpirationHandling4NotifyJob', 'com.aii.crm.job.instance.dataflow.ExpirationHandling4NotifyJob', '0/30 \* \* \* \* ?', '2', '0=A,1=B', '1', 'day=3,row=100', NULL, '1', '1', '0', '1', '0', '0');

## 四，本地自测说明

下载开发环境（rd） job 域的配置文件及与测试job相关域的配置文件

下载地址：<http://10.1.239.47:8081/login.html> admin/admin

配置文件放到对应域的resource目录下面，然后修改配置文件从本地启动 zookeeper，

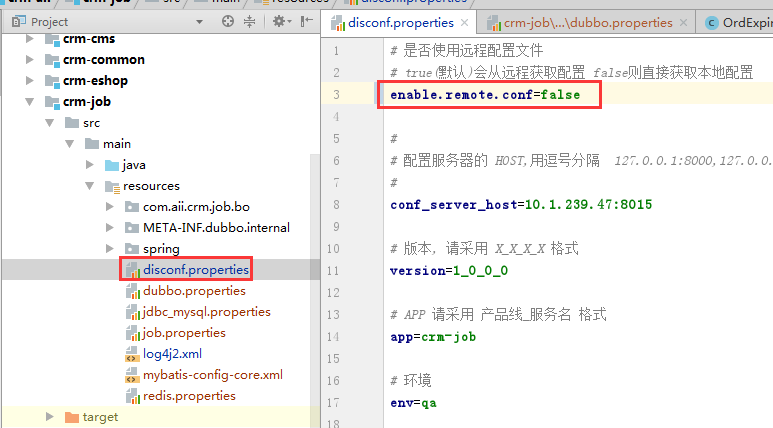
配置文件修改后，先启动相关域的dubbo，最后再启动job域的dubbo。启动job域的dubbo 后job会自动被zookeeper分配执行。

*注：JobScheduler.init()中可检测job配置及启动，job代码中可看到执行情况*

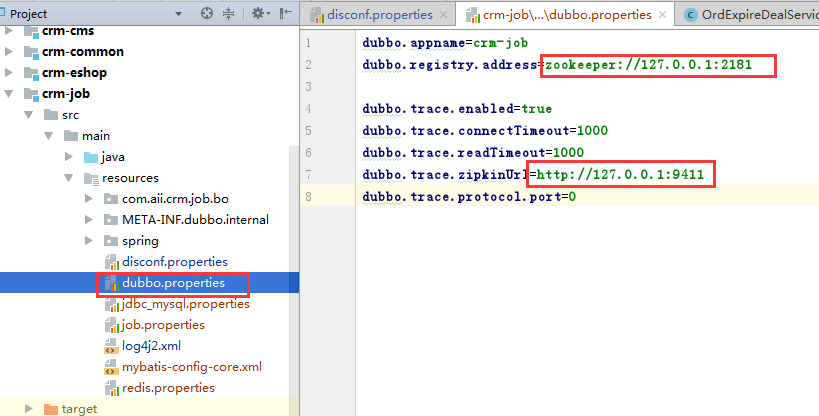
*建议：当测试一个job时，可以先把本地的其它job类删除，等测试完成后再全量更新，可以提高测试效率。*

#### 配置文件修改

1，修改enable.remote.conf = false



2，修改zookeeper 为本机启动的，根据自己的zookeeper进行配置



3，修改job的注册地址为本机的zookeeper

