**elastic-job入门技术总结**

* **版本历史**

|  |  |  |  |
| --- | --- | --- | --- |
| **版本/状态** | **作者/ 修改者** | **日期** | **说明** |
| 1.0 |  | 2016/07/07 | 建立文档 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

一. 概述 5

1.1 范围 5

1.2 参考资料 5

二. 术语、符号、缩略语 5

2.1 术语定义 5

2.1.1 分布式作业调度 5

2.1.2 ZooKeeper 5

三. Elastic-Job 6

3.1 概述 6

3.2 功能介绍 6

3.2.1 何为分布式作业 6

3.2.2 主要功能 8

3.2.3 其它功能 8

3.2.4 非功能性需求 9

3.3 使用限制 9

四. 快速上手 10

4.1 源码下载 10

4.2 运行环境配置 11

4.2.1 Lombok安装配置 11

4.2.2 Zookeeper安装配置 12

4.3 软件运行 13

4.3.1 启动zookeeper服务器 13

4.3.2 启动elastic-job-example 13

4.3.3 启动管理平台elastic-job-console 13

五. 配置介绍 14

5.1 Spring作业配置 14

5.1.1 job:simple命名空间属性详细说明 16

5.1.2 job:dataflow命名空间属性详细说明 17

5.1.3 job:script命名空间属性详细说明 17

5.1.4 job:listener命名空间属性详细说明 18

5.1.5 Reg注册中心配置 18

5.1.6 Spring 配置信息 19

5.1.7 Spring 配置信息-带占位符配置文件 21

5.2 不使用Spring配置Job 24

5.3 作业分片策略 24

5.3.1 AverageAllocationJobShardingStrategy 24

5.3.2 OdevitySortByNameJobShardingStrategy 25

5.3.3 RotateServerByNameJobShardingStrategy 25

5.3.4 自定义分片策略 26

5.4 作业类型 26

5.4.1 Simple类型作业 26

5.4.2 DataFlow类型作业 26

5.4.3 Script 类型作业 27

5.5 作业批量处理 27

5.6 作业任务监听配置 28

5.6.1 每台作业节点均执行的监听 28

5.6.2 分布式场景中仅单一节点执行的监听 28

六. 实现原理 29

6.1 弹性分布式实现 29

6.2 实现方案及开发理念 29

6.2.1 1 elastic-job的具体模块的底层及如何实现 29

6.3 注册中心数据结构 31

6.3.1 概览 32

6.3.2 config节点 33

6.3.3 servers节点 34

6.3.4 execution节点 34

6.3.5 leader节点 35

6.4 流程图 36

6.4.1 作业启动 36

6.4.2 作业执行 38

七. 拓展 39

7.1 Zookeer介绍 39

# 概述

## 范围

本文档是对elastic-job分布式作业框架的初步研究文档。文档介绍了对elastic-job初步认识和实际如何部署实现。

## 参考资料

# 术语、符号、缩略语

## 术语定义

### 分布式作业调度

任务的分布式执行，需要将一个任务拆分为n个独立的任务项，然后由分布式的服务器分别执行某一个或几个分片项。

例如：有一个遍历数据库某张表的作业，现有2台服务器。为了快速的执行作业，那么每台服务器应执行作业的50%。 为满足此需求，可将作业分成2片，每台服务器执行1片。作业遍历数据的逻辑应为：服务器A遍历ID以奇数结尾的数据；服务器B遍历ID以偶数结尾的数据。 如果分成10片，则作业遍历数据的逻辑应为：每片分到的分片项应为ID%10，而服务器A被分配到分片项0,1,2,3,4；服务器B被分配到分片项5,6,7,8,9，直接的结果就是服务器A遍历ID以0-4结尾的数据；服务器B遍历ID以5-9结尾的数据。

### ZooKeeper

ZooKeeper是一个[分布式](http://baike.baidu.com/view/402382.htm)的，开放源码的[分布式应用程序](http://baike.baidu.com/view/553502.htm)协调服务，是[Google](http://baike.baidu.com/view/105.htm)的Chubby一个[开源](http://baike.baidu.com/view/9664.htm)的实现，是Hadoop和Hbase的重要组件。它是一个为分布式应用提供一致性服务的软件，提供的功能包括：配置维护、域名服务、分布式同步、组服务等。

# Elastic-Job

## 概述

Elastic-Job是ddframe中dd-job的作业模块中分离出来的分布式弹性作业框架。去掉了和dd-job中的监控和ddframe接入规范部分。该项目基于成熟的开源产品Quartz和Zookeeper及其客户端Curator进行二次开发。

## 功能介绍

### 何为分布式作业

#### 分片概念

任务的分布式执行，需要将一个任务拆分为n个独立的任务项，然后由分布式的服务器分别执行某一个或几个分片项。

例如：有一个遍历数据库某张表的作业，现有2台服务器。为了快速的执行作业，那么每台服务器应执行作业的50%。 为满足此需求，可将作业分成2片，每台服务器执行1片。作业遍历数据的逻辑应为：服务器A遍历ID以奇数结尾的数据；服务器B遍历ID以偶数结尾的数据。 如果分成10片，则作业遍历数据的逻辑应为：每片分到的分片项应为ID%10，而服务器A被分配到分片项0,1,2,3,4；服务器B被分配到分片项5,6,7,8,9，直接的结果就是服务器A遍历ID以0-4结尾的数据；服务器B遍历ID以5-9结尾的数据。

#### 分片项与业务处理解耦

Elastic-job并不直接提供数据处理的功能，框架只会将分片项分配至各个运行中的作业服务器，开发者需要自行处理分片项与真实数据的对应关系。

#### 分布式作业的执行

Elastic-job并无作业调度中心节点，而是基于部署作业框架的程序在到达相应时间点时各自触发调度。

注册中心仅用于作业注册和监控信息存储。而主作业节点仅用于处理分片和清理等功能。

#### 个性化参数的适用场景

个性化参数即shardingItemParameters，可以和分片项匹配对应关系，用于将分片项的数字转换为更加可读的业务代码。

例如：按照地区水平拆分数据库，数据库A是北京的数据；数据库B是上海的数据；数据库C是广州的数据。 如果仅按照分片项配置，开发者需要了解0表示北京；1表示上海；2表示广州。 合理使用个性化参数可以让代码更可读，如果配置为0=北京,1=上海,2=广州，那么代码中直接使用北京，上海，广州的枚举值即可完成分片项和业务逻辑的对应关系。

#### 作业高可用

Elastic-job提供最安全的方式执行作业。将分片项设置为1，并使用多于1台的服务器执行作业，作业将会以1主n从的方式执行。

一旦执行作业的服务器崩溃，等待执行的服务器将会在下次作业启动时替补执行。 开启失效转移功能效果更好，可以保证在本次作业执行时崩溃，备机立即启动替补执行。

#### 最大限度利用资源

Elastic-job也提供最灵活的方式，最大限度的提高执行作业的吞吐量。将分片项设置为大于服务器的数量，最好是大于服务器倍数的数量，作业将会合理的利用分布式资源，动态的分配分片项。

例如：3台服务器，分成10片，则分片项分配结果为服务器A=0,1,2;服务器B=3,4,5;服务器C=6,7,8,9。 如果服务器C崩溃，则分片项分配结果为服务器A=0,1,2,3,4;服务器B=5,6,7,8,9。在不丢失分片项的情况下，最大限度的利用现有资源提高吞吐量。

### 主要功能

* **分布式：** 重写Quartz基于数据库的分布式功能，改用Zookeeper实现注册中心。
* **并行调度：** 采用任务分片方式实现。将一个任务拆分为多个独立的任务项，由分布式的服务器并行执行各自分配到的分片项。
* **弹性扩容缩容：** 将任务拆分为多个子任务项后，各个服务器分别执行各自分配到的任务项。一旦有新的服务器加入集群，或现有服务器下线，elastic-job将在保留本次任务执行不变的情况下，下次任务开始前触发任务重分片。
* **集中管理：** 采用基于Zookeeper的注册中心，集中管理和协调分布式作业的状态，分配和监听。外部系统可直接根据Zookeeper的数据管理和监控elastic-job。
* **定制化流程型任务：** 作业可分为简单和数据流处理两种模式，数据流又分为高吞吐处理模式和顺序性处理模式，其中高吞吐处理模式可以开启足够多的线程快速的处理数据，而顺序性处理模式将每个分片项分配到一个独立线程，用于保证同一分片的顺序性，这点类似于Kafka的分区顺序性。

### 其它功能

* **失效转移：** 弹性扩容缩容在下次作业运行前重分片，但本次作业执行的过程中，下线的服务器所分配的作业将不会重新被分配。失效转移功能可以在本次作业运行中用空闲服务器抓取孤儿作业分片执行。同样失效转移功能也会牺牲部分性能。
* **Spring命名空间支持：** elastic-job可以不依赖于Spring直接运行，但是也提供了自定义的命名空间方便与Spring集成。
* **运维平台：** 提供web控制台用于管理作业。

### 非功能性需求

* **稳定性：** 在服务器无波动的情况下，并不会重新分片；即使服务器有波动，下次分片的结果也会根据服务器IP和作业名称哈希值算出稳定的分片顺序，尽量不做大的变动。
* **高性能：** 同一服务器的批量数据处理采用自动切割并多线程并行处理。
* **灵活性：** 所有在功能和性能之间的权衡，都可通过配置开启/关闭。如：elastic-job会将作业运行状态的必要信息更新到注册中心。如果作业执行频度很高，会造成大量Zookeeper写操作，而分布式Zookeeper同步数据可能引起网络风暴。因此为了考虑性能问题，可以牺牲一些功能，而换取性能的提升。
* **一致性：** elastic-job可牺牲部分性能用以保证同一分片项不会同时在两个服务器上运行。
* **容错性：** 作业服务器和Zookeeper断开连接则立即停止作业运行，用于防止分片已经重新分配，而脑裂的服务器仍在继续执行，导致重复执行。

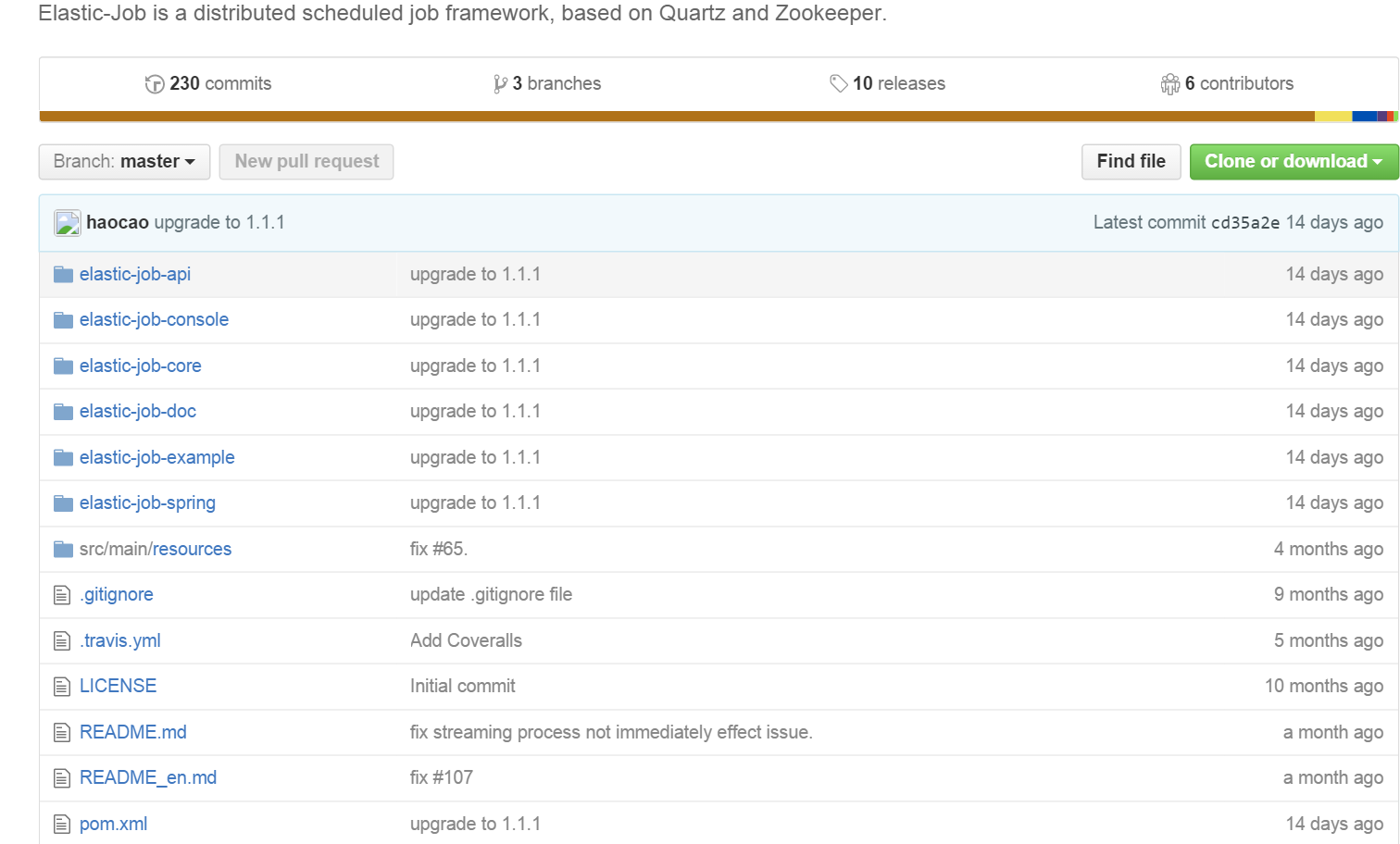
## 使用限制

* **作业一旦启动成功后不能修改作业名称，如果修改名称则视为新的作业**。
* 同一台作业服务器只能运行一个相同的作业实例，因为作业运行时是按照IP注册和管理的。
* 一旦有服务器波动，或者修改分片项，将会触发重新分片；触发重新分片将会导致运行中的流式处理的作业在执行完本次作业后不再继续执行，等待分片结束后再恢复正常。
* 开启monitorExecution才能实现分布式作业幂等性（即不会在多个作业服务器运行同一个分片）的功能，但monitorExecution对短时间内执行的作业（如每5秒一触发）性能影响较大，建议关闭并自行实现幂等性。

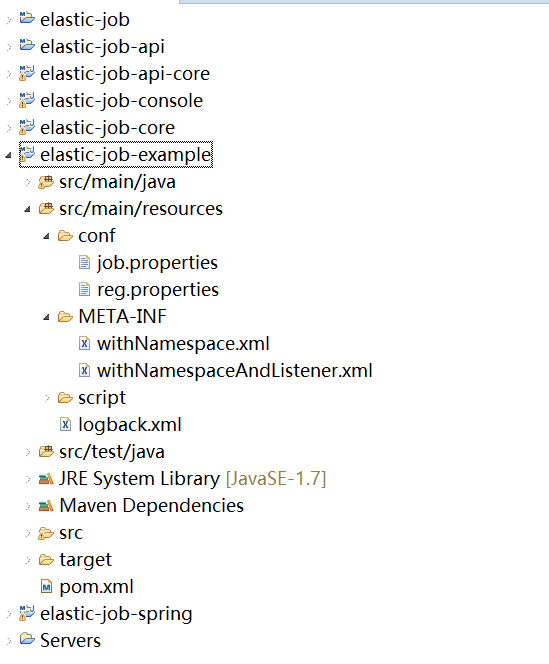
# 快速上手

## 源码下载

从github上（<https://github.com/dangdangdotcom/elastic-job>）下载源码到本地，最新版本为1.1.1，解压到相应文件夹。



Eclipese导入工程后如下图：



## 运行环境配置

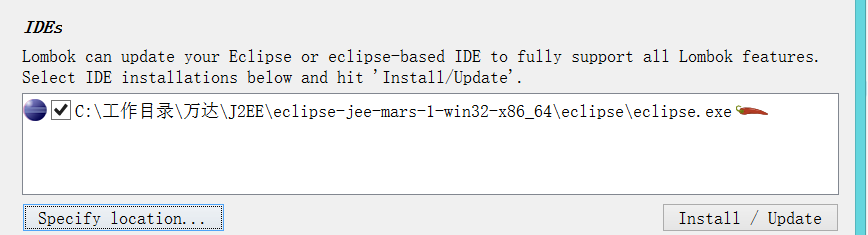
### Lombok安装配置

由于elastic-job源码中省略了get、set、log方法，需要安装lombok才能让源码通过编译。

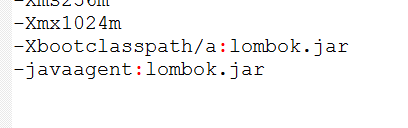
lombok的官方网址：<http://projectlombok.org/>

Eclipse安装步骤：

* 双击下载的lombok.jar压缩包文件。
* 单击Specify location..选择相应的IDE目录



* 点击Install/Update
* 打开Eclipse目录下的eclipse.ini文件



### Zookeeper安装配置

Zookeeper的官方网址：<http://hadoop.apache.org/zookeeper/releases.html>，

本文使用的3.4.8版本。

本地运行Zookeeper可以配置为单机模式和伪集群模式。

#### 单机模式

**配置zoo.cfg**

首次使用 ZooKeeper，需要将ZK\_HOME下的zoo\_sample.cfg文件重命名为zoo.cfg,并进行以下配置:

##Zookeeper最小时间单元，单位毫秒(ms)，默认值为3000

tickTime=2000

##Zookeeper服务器存储快照文件的目录，必须配置

dataDir=D:\\zookeeper\\zookeeper-3.4.8-server1\\data

##Zookeeper服务器存储事务日志的目录，默认为dataDir

dataLogDir= D:\\zookeeper\\zookeeper-3.4.8-server1\\log

##服务器对外服务端口，一般设置为2181

clientPort=2181

##Leader服务器等待Follower启动并完成数据同步的时间，

##默认值10，表示tickTime的10倍

initLimit=5

##Leader服务器和Follower之间进行心跳检测的最大延时时间，

##默认值5，表示tickTime的5倍

syncLimit=2

**启动服务**

使用ZK\_HOME/bin目录下的 zkServer.cmd脚本进行服务的启动。

#### 集群模式

一个 ZooKeeper 集群通常由一组机器组成，一般 3 台以上就可以组成一个可用的 ZooKeeper 集群了。

　　组成 ZooKeeper 集群的每台机器都会在内存中维护当前的服务器状态，并且每台机器之间都会互相保持通信。重要的一点是，只要集群中存在超过一半的机器能够正常工作，那么整个集群就能够正常对外服务。

　　ZooKeeper 的客户端程序会选择和集群中的任意一台服务器创建一个 TCP 连接，而且一旦客户端和服务器断开连接，客户端就会自动连接到集群中的其他服务器

**1.伪集群模式**

　　一种特殊的集群模式，即集群的所有服务器都部署在一台机器上。当你手头上有一台比较好的机器，如果作为单机模式进行部署，就会浪费资源，这种情况下，ZooKeeper允许你在一台机器上通过启动不同的端口来启动多个 ZooKeeper 服务实例，以此来以集群的特性来对外服务。

　　这种模式下，只需要把 zoo.cfg 做如下修改：

tickTime=2000

dataDir=D:\\zookeeper\\zookeeper-3.4.8-server1\\data

dataLogDir= D:\\zookeeper\\zookeeper-3.4.8-server1\\log

clientPort=2181

initLimit=5

syncLimit=2

server.1=IP:2888:3888

server.2=IP:2889:3889

server.3=IP:2890:3890

集群模式 server.id=host:port1:port2 的配置。其中id 被称为 Server ID，用来标识该机器在集群中的机器序号（在每台机器的 dataDir 目录下创建 myid 文件，文件内容即为该机器对应的 Server ID 数字）。host 为机器 IP，port1 用于指定 Follower 服务器与 Leader 服务器进行通信和数据同步的端口，port2 用于进行 Leader 选举过程中的投票通信。

**创建 myid 文件**

　在 dataDir 目录下创建名为 myid 的文件，在文件第一行写上对应的 Server ID。

**按照相同步骤，为其他目录配置 zoo.cfg和myid文件**

**2.集群模式**

**略**

## 软件运行

### 启动zookeeper服务器

* 启动：zkServer.cmd(zkServer.sh)，无报错。
* 验证：JPS，存在QuorumPeerMain为已启动；验证端口netstat -ano|findstr 2181

### 启动elastic-job-example

* 源码conf文件夹下[job.properties](#_job:simple命名空间属性详细说明)为作业相应配置信息值，[reg.properties](#_Reg_注册中心配置_1)文件为注册中心相应配置信息值。
* 源码META-INF文件下withNamespace.xml为作业配置文件，注册了对应的作业和zookeeper配置信息。
* 作业主程序为SpringJobMain.java，位于

com.dangdang.example.elasticjob.spring.main包中，

以Java Application运行。

### 启动管理平台elastic-job-console

* 配置tomact服务器
* 启动tomcat服务
* 在浏览器中打开【http://localhost:8080/elastic-job-console/，用户名密码为root/root】



# 配置介绍

## Spring作业配置

与Spring容器配合使用作业，可以将作业Bean配置为Spring Bean，可在作业中通过依赖注入使用Spring容器管理的数据源等对象。可用placeholder占位符从属性文件中取值。

**Spring命名空间配置**

<?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:reg="http://www.dangdang.com/schema/ddframe/reg"

xmlns:job="http://www.dangdang.com/schema/ddframe/job"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd

http://www.dangdang.com/schema/ddframe/reg

http://www.dangdang.com/schema/ddframe/reg/reg.xsd

http://www.dangdang.com/schema/ddframe/job

http://www.dangdang.com/schema/ddframe/job/job.xsd

">

<!--配置作业注册中心 -->

<reg:zookeeper id="regCenter" server-lists=" yourhost:2181" namespace="dd-job" base-sleep-time-milliseconds="1000" max-sleep-time-milliseconds="3000" max-retries="3" />

<!-- 配置简单作业-->

<job:simple id="simpleElasticJob" class="xxx.MySimpleElasticJob" registry-center-ref="regCenter" cron="0/10 \* \* \* \* ?" sharding-total-count="3" sharding-item-parameters="0=A,1=B,2=C" />

<!-- 配置数据流作业-->

<job:dataflow id="throughputDataFlow" class="xxx.MyThroughputDataFlowElasticJob" registry-center-ref="regCenter" cron="0/10 \* \* \* \* ?" sharding-total-count="3" sharding-item-parameters="0=A,1=B,2=C" process-count-interval-seconds="10" concurrent-data-process-thread-count="10" />

<!-- 配置脚本作业-->

<job:script id="scriptElasticJob" registry-center-ref="regCenter" cron="0/10 \* \* \* \* ?" sharding-total-count="3" sharding-item-parameters="0=A,1=B,2=C" script-command-line="/your/file/path/demo.sh" />

<!-- 配置带监听的简单作业-->

<job:simple id="listenerElasticJob" class="xxx.MySimpleListenerElasticJob" registry-center-ref="regCenter" cron="0/10 \* \* \* \* ?" sharding-total-count="3" sharding-item-parameters="0=A,1=B,2=C">

<job:listener class="xx.MySimpleJobListener"/>

<job:listener class="xx.MyOnceSimpleJobListener" started-timeout-milliseconds="1000" completed-timeout-milliseconds="2000" />

</job:simple>

</beans>

### job:simple命名空间属性详细说明

| **属性名** | **类型** | **必填** | **缺省值** | **描述** |
| --- | --- | --- | --- | --- |
| id | String | 是 |  | 作业名称 |
| class | String | 否 |  | 作业实现类，需实现ElasticJob接口，脚本型作业不需要配置 |
| registry-center-ref | String | 是 |  | 注册中心Bean的引用，需引用reg:zookeeper的声明 |
| cron | String | 是 |  | cron表达式，用于配置作业触发时间 |
| sharding-total-count | int | 是 |  | 作业分片总数 |
| sharding-item-parameters | String | 否 |  | 分片序列号和参数用等号分隔，多个键值对用逗号分隔 分片序列号从0开始，不可大于或等于作业分片总数。如：0=a,1=b,2=c |
| job-parameter | String | 否 |  | 作业自定义参数 可以配置多个相同的作业，但是用不同的参数作为不同的调度实例 |
| monitor-execution | boolean | 否 | true | 监控作业运行时状态 每次作业执行时间和间隔时间均非常短的情况，建议不监控作业运行时状态以提升效率。因为是瞬时状态，所以无必要监控。请用户自行增加数据堆积监控。并且不能保证数据重复选取，应在作业中实现幂等性。 每次作业执行时间和间隔时间均较长的情况，建议监控作业运行时状态，可保证数据不会重复选取。 |
| monitor-port | int | 否 | -1 | 作业监控端口。建议配置作业监控端口, 方便开发者dump作业信息。使用方法: echo “dump” | nc 127.0.0.1 9888 |
| max-time-diff-seconds | int | 否 | -1 | 最大允许的本机与注册中心的时间误差秒数，如果时间误差超过配置秒数则作业启动时将抛异常，配置为-1表示不校验时间误差 |
| failover | boolean | 否 | false | 是否开启失效转移  仅monitorExecution开启，失效转移才有效 |
| misfire | boolean | 否 | true | 是否开启错过任务重新执行 |
| job-sharding-strategy-class | String | 否 | true | 作业分片策略实现类全路径，默认使用平均分配策略 |
| description | String | 否 |  | 作业描述信息 |
| disabled | boolean | 否 | false | 作业是否禁止启动 可用于部署作业时，先禁止启动，部署结束后统一启动 |
| overwrite | boolean | 否 | false | 本地配置是否可覆盖注册中心配置。如果可覆盖，每次启动作业都以本地配置为准 |

### job:dataflow命名空间属性详细说明

job:dataflow命名空间拥有job:simple命名空间的全部属性，以下仅列出特有属性

| **属性名** | **类型** | **是否必填** | **缺省值** | **描述** |
| --- | --- | --- | --- | --- |
| process-count-interval-seconds | int | 否 | 300 | 统计作业处理数据数量的间隔时间 单位：秒 |
| concurrent-data-process-thread-count | int | 否 | CPU核数\*2 | 同时处理数据的并发线程数，不能小于1，仅ThroughputDataFlow作业有效 |
| fetch-data-count | int | 否 | 1 | 每次抓取的数据量 |
| streaming-process | boolean | 否 | false | 是否流式处理数据 如果流式处理数据, 则fetchData不返回空结果将持续执行作业 如果非流式处理数据, 则处理数据完成后作业结束 |

### job:script命名空间属性详细说明

job:script命名空间拥有job:simple命名空间的全部属性，以下仅列出特有属性

| **属性名** | **类型** | **是否必填** | **缺省值** | **描述** |
| --- | --- | --- | --- | --- |
| script-command-line | String | 否 |  | 脚本型作业执行命令行 |

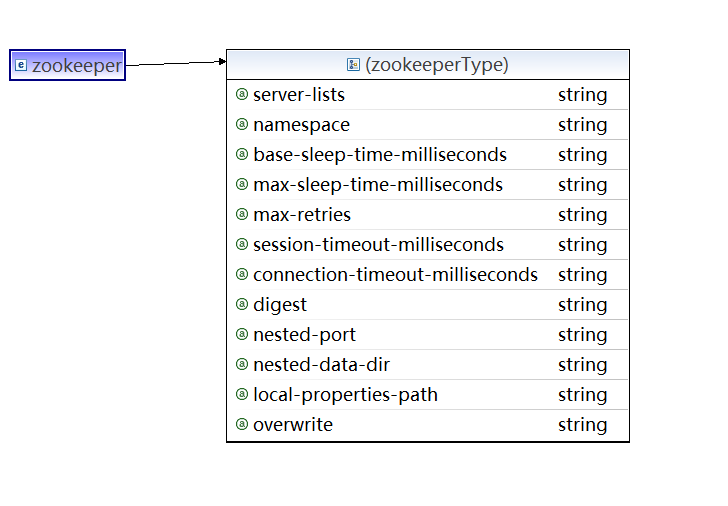
### job:listener命名空间属性详细说明

job:listener必须配置为job:bean的子元素

| **属性名** | **类型** | **必填** | **缺省值** | **描述** |
| --- | --- | --- | --- | --- |
| class | String | 是 |  | 前置后置任务监听实现类，需实现ElasticJobListener接口 |
| started-timeout-milliseconds | long | 否 | Long.MAX\_VALUE | AbstractDistributeOnceElasticJobListener型监听器，最后一个作业执行前的执行方法的超时时间 单位：毫秒 |
| completed-timeout-milliseconds | long | 否 | Long.MAX\_VALUE | AbstractDistributeOnceElasticJobListener型监听器，最后一个作业执行后的执行方法的超时时间 单位：毫秒 |

### Reg注册中心配置

reg.XSD定义了与Zookeeper相关的配置信息。



| **属性名** | **类型** | **必填** | **缺省值** | **描述** |
| --- | --- | --- | --- | --- |
| id | String | 是 |  | 注册中心在Spring容器中的主键 |
| server-lists | String | 是 |  | 连接Zookeeper服务器的列表，包括IP地址和端口号，多个地址用逗号分隔。如: host1:2181,host2:2181 |
| namespace | String | 是 |  | Zookeeper的命名空间 |
| base-sleep-time-milliseconds | int | 否 | 1000 | 等待重试的间隔时间的初始值 单位：毫秒 |
| max-sleep-time-milliseconds | int | 否 | 3000 | 等待重试的间隔时间的最大值 单位：毫秒 |
| max-retries | int | 否 | 3 | 最大重试次数 |
| session-timeout-milliseconds | int | 否 | 60000 | 会话超时时间 单位：毫秒 |
| connection-timeout-milliseconds | int | 否 | 15000 | 连接超时时间 单位：毫秒 |
| digest | String | 否 | 无验证 | 连接Zookeeper的权限令牌 缺省为不需要权限验证 |
| nested-port | int | 否 | -1 | 内嵌Zookeeper的端口号 -1表示不开启内嵌Zookeeper |
| nested-data-dir | String | 否 |  | 内嵌Zookeeper的数据存储路径 为空表示不开启内嵌Zookeeper |

### Spring 配置信息

<?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:reg="http://www.dangdang.com/schema/ddframe/reg"

xmlns:job="http://www.dangdang.com/schema/ddframe/job"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd

http://www.dangdang.com/schema/ddframe/reg

http://www.dangdang.com/schema/ddframe/reg/reg.xsd

http://www.dangdang.com/schema/ddframe/job

http://www.dangdang.com/schema/ddframe/job/job.xsd

">

<!--配置作业注册中心 -->

<reg:zookeeper id="regCenter" server-lists=" yourhost:2181" namespace="dd-job" base-sleep-time-milliseconds="1000" max-sleep-time-milliseconds="3000" max-retries="3" />

<!-- 配置简单作业-->

<job:simple id="simpleElasticJob" class="xxx.MySimpleElasticJob" registry-center-ref="regCenter" cron="0/10 \* \* \* \* ?" sharding-total-count="3" sharding-item-parameters="0=A,1=B,2=C" />

<!-- 配置数据流作业-->

<job:dataflow id="throughputDataFlow" class="xxx.MyThroughputDataFlowElasticJob" registry-center-ref="regCenter" cron="0/10 \* \* \* \* ?" sharding-total-count="3" sharding-item-parameters="0=A,1=B,2=C" process-count-interval-seconds="10" concurrent-data-process-thread-count="10" />

<!-- 配置脚本作业-->

<job:script id="scriptElasticJob" registry-center-ref="regCenter" cron="0/10 \* \* \* \* ?" sharding-total-count="3" sharding-item-parameters="0=A,1=B,2=C" script-command-line="/your/file/path/demo.sh" />

<!-- 配置带监听的简单作业-->

<job:simple id="listenerElasticJob" class="xxx.MySimpleListenerElasticJob" registry-center-ref="regCenter" cron="0/10 \* \* \* \* ?" sharding-total-count="3" sharding-item-parameters="0=A,1=B,2=C">

<job:listener class="xx.MySimpleJobListener"/>

<job:listener class="xx.MyOnceSimpleJobListener" started-timeout-milliseconds="1000" completed-timeout-milliseconds="2000" />

</job:simple>

</beans>

### Spring 配置信息-带占位符配置文件

reg.properties注册中心配置占位符文件

serverLists=10.1.81.45:2181

namespace=elasticjob-example

baseSleepTimeMilliseconds=1000

maxSleepTimeMilliseconds=3000

maxRetries=3

nestedPort=-1

nestedDataDir=target/test\_zk\_data/

**job.properties作业配置占位符文件**

simpleJob.cron=0 0/1 \* \* \* ?

simpleJob.shardingTotalCount=10

simpleJob.shardingItemParameters=0=A,1=B,2=C,3=D,4=E,5=F,6=G,7=H,8=I,9=J

simpleJob.monitorExecution=false

simpleJob.failover=true

simpleJob.description=\u53EA\u8FD0\u884C\u4E00\u6B21\u7684\u4F5C\u4E1A\u793A\u4F8B

simpleJob.disabled=false

simpleJob.overwrite=true

simpleJob.monitorPort=9888

throughputDataFlowJob.cron=0/5 \* \* \* \* ?

throughputDataFlowJob.shardingTotalCount=10

throughputDataFlowJob.shardingItemParameters=0=A,1=B,2=C,3=D,4=E,5=F,6=G,7=H,8=I,9=J

throughputDataFlowJob.monitorExecution=true

throughputDataFlowJob.failover=true

throughputDataFlowJob.processCountIntervalSeconds=10

throughputDataFlowJob.concurrentDataProcessThreadCount=3

throughputDataFlowJob.description=\u4E0D\u505C\u6B62\u8FD0\u884C\u7684\u4F5C\u4E1A\u793A\u4F8B

throughputDataFlowJob.disabled=false

throughputDataFlowJob.overwrite=true

throughputDataFlowJob.streamingProcess=true

sequenceDataFlowJob.cron=0/5 \* \* \* \* ?

sequenceDataFlowJob.shardingTotalCount=10

sequenceDataFlowJob.shardingItemParameters=0=A,1=B,2=C,3=D,4=E,5=F,6=G,7=H,8=I,9=J

sequenceDataFlowJob.maxTimeDiffSeconds=-1

sequenceDataFlowJob.monitorExecution=true

sequenceDataFlowJob.failover=true

sequenceDataFlowJob.processCountIntervalSeconds=10

sequenceDataFlowJob.description=\u6309\u987A\u5E8F\u4E0D\u505C\u6B62\u8FD0\u884C\u7684\u4F5C\u4E1A\u793A\u4F8B

sequenceDataFlowJob.disabled=false

sequenceDataFlowJob.overwrite=true

作业配置信息xml文件，主程序启动，加载配置文件，启动定时作业服务。

<?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"*

xmlns:reg=*"http://www.dangdang.com/schema/ddframe/reg"*

xmlns:job=*"http://www.dangdang.com/schema/ddframe/job"*

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*

*http://www.dangdang.com/schema/ddframe/reg*

*http://www.dangdang.com/schema/ddframe/reg/reg.xsd*

*http://www.dangdang.com/schema/ddframe/job*

*http://www.dangdang.com/schema/ddframe/job/job.xsd*

*"*>

<context:component-scan base-package=*"com.dangdang.example.elasticjob"* />

<context:property-placeholder location=*"classpath:conf/\*.properties"* />

<reg:zookeeper id=*"regCenter"* server-lists=*"${serverLists}"* namespace=*"${namespace}"* base-sleep-time-milliseconds=*"${baseSleepTimeMilliseconds}"* max-sleep-time-milliseconds=*"${maxSleepTimeMilliseconds}"* max-retries=*"${maxRetries}"* nested-port=*"${nestedPort}"* nested-data-dir=*"${nestedDataDir}"* />

<job:simple id=*"simpleElasticJob"* class=*"com.dangdang.example.elasticjob.spring.job.SimpleJobDemo"* registry-center-ref=*"regCenter"* sharding-total-count=*"${simpleJob.shardingTotalCount}"* cron=*"${simpleJob.cron}"* sharding-item-parameters=*"${simpleJob.shardingItemParameters}"* monitor-execution=*"${simpleJob.monitorExecution}"* monitor-port=*"${simpleJob.monitorPort}"* failover=*"${simpleJob.failover}"* description=*"${simpleJob.description}"* disabled=*"${simpleJob.disabled}"* overwrite=*"${simpleJob.overwrite}"* />

<job:dataflow id="throughputDataFlowJob" class="com.dangdang.example.elasticjob.spring.job.ThroughputDataFlowJobDemo" registry-center-ref="regCenter" sharding-total-count="${throughputDataFlowJob.shardingTotalCount}" cron="${throughputDataFlowJob.cron}" sharding-item-parameters="${throughputDataFlowJob.shardingItemParameters}" monitor-execution="${throughputDataFlowJob.monitorExecution}" failover="${throughputDataFlowJob.failover}" process-count-interval-seconds="${throughputDataFlowJob.processCountIntervalSeconds}" concurrent-data-process-thread-count="${throughputDataFlowJob.concurrentDataProcessThreadCount}" description="${throughputDataFlowJob.description}" disabled="${throughputDataFlowJob.disabled}" overwrite="${throughputDataFlowJob.overwrite}" streaming-process="${throughputDataFlowJob.streamingProcess}" />

<job:dataflow id="sequenceDataFlowJob" class="com.dangdang.example.elasticjob.spring.job.SequenceDataFlowJobDemo" registry-center-ref="regCenter" sharding-total-count="${sequenceDataFlowJob.shardingTotalCount}" cron="${sequenceDataFlowJob.cron}" sharding-item-parameters="${sequenceDataFlowJob.shardingItemParameters}" monitor-execution="${sequenceDataFlowJob.monitorExecution}" failover="${sequenceDataFlowJob.failover}" process-count-interval-seconds="${sequenceDataFlowJob.processCountIntervalSeconds}" max-time-diff-seconds="${sequenceDataFlowJob.maxTimeDiffSeconds}" description="${sequenceDataFlowJob.description}" disabled="${sequenceDataFlowJob.disabled}" overwrite="${sequenceDataFlowJob.overwrite}" />

</beans>

## 不使用Spring配置Job

略。详见代码

## 作业分片策略

Elastic-job提供了三种作业分片策略，可以在配置信息文件中设置属性值【job-sharding-strategy-class】来决定采用哪种作业分片策略，默认使用

com.dangdang.ddframe.job.plugin.sharding.strategy.AverageAllocationJobShardingStrategy

### AverageAllocationJobShardingStrategy

**包路径：**

com.dangdang.ddframe.job.plugin.sharding.strategy.AverageAllocationJobShardingStrategy

**策略说明：**

基于平均分配算法的分片策略，也是默认的分片策略。

如果分片不能整除，则不能整除的多余分片将依次追加到序号小的服务器。如：

如果有3台服务器，分成9片，则每台服务器分到的分片是：1=[0,1,2], 2=[3,4,5], 3=[6,7,8]

如果有3台服务器，分成8片，则每台服务器分到的分片是：1=[0,1,6], 2=[2,3,7], 3=[4,5]

如果有3台服务器，分成10片，则每台服务器分到的分片是：1=[0,1,2,9], 2=[3,4,5], 3=[6,7,8]

### OdevitySortByNameJobShardingStrategy

**策略说明：**

根据作业名的哈希值奇偶数决定IP升降序算法的分片策略。

作业名的哈希值为奇数则IP升序。

作业名的哈希值为偶数则IP降序。

用于不同的作业平均分配负载至不同的服务器。

AverageAllocationJobShardingStrategy的缺点是，一旦分片数小于作业服务器数，作业将永远分配至IP地址靠前的服务器，导致IP地址靠后的服务器空闲。而OdevitySortByNameJobShardingStrategy则可以根据作业名称重新分配服务器负载。如：

如果有3台服务器，分成2片，作业名称的哈希值为奇数，则每台服务器分到的分片是：1=[0], 2=[1], 3=[]

如果有3台服务器，分成2片，作业名称的哈希值为偶数，则每台服务器分到的分片是：3=[0], 2=[1], 1=[]

### RotateServerByNameJobShardingStrategy

**策略说明：**

根据作业名的哈希值对服务器列表进行轮转的分片策略。

### 自定义分片策略

Elastic-job允许使用者自动以分片策略，实现JobShardingStrategy接口并实现sharding方法，接口方法参数为作业服务器IP列表和分片策略选项，分片策略选项包括作业名称，分片总数以及分片序列号和个性化参数对照表，可以根据需求定制化自己的分片策略。

## 作业类型

Elastic-job提供了**三种作业类型：Simple、DataFlow和Script。**

### Simple类型作业

Simple类型作业为简单作业模式，定时执行，执行一次，用于执行普通的定时任务，与Quartz原生接口相似，只是增加了弹性扩缩容和分片等功能。

该类型作业实现需要继承AbstractSimpleElasticJob类，在process方法中实现作业。

### DataFlow类型作业

Dataflow 类型作业为**数据流作业模式**，根据同一分片项是否使用多线程处理，可以细分为ThroughputDataFlow类型作业和SequenceDataFlow类型作业。

**由于ThroughputDataFlow作业可以使用多于分片项的任意线程数处理，所以性能调优的可能会优于SequenceDataFlow作业。**

#### ThroughputDataFlow类型作业

ThroughputDataFlow作业类型可以将获取到的数据多线程处理，可以使用多于分片项的线程处理，但不会保证多线程处理数据的顺序。

如：从2个分片共获取到100条数据，第1个分片40条，第2个分片60条，配置为两个**线程**处理，则第1个线程处理前50条数据，第2个线程处理后50条数据，**无视分片项**，也可以采用大于分片项的线程处理，4个线程，每个线程处理25条数据。

该类型作业实现需要继承

AbstractIndividualThroughputDataFlowElasticJob类，并可以指定返回值泛型，该类提供3个方法可覆盖，分别用于抓取数据，处理数据和指定是否流式处理数据。可以获取数据处理成功失败次数等辅助监控信息。如果**流式处理(streamingProcess:true)**数据，**fetchData方法的返回值只有为null或长度为空时，作业才会停止执行，否则作业会一直运行下去**；**非流式处理(streamingProcess:true)**数据则**只会在每次作业执行过程中执行一次fetchData方法和processData方法，即完成本次作业**。流式数据处理参照TbSchedule设计，适用于不间歇的数据处理。

作业执行时会将fetchData的数据传递给processData处理，其中processData得到的数据是通过多线程（线程池大小可配）拆分的。如果采用流式作业处理方式，建议processData处理数据后更新其状态，避免fetchData再次抓取到，从而使得作业永远不会停止。processData的返回值用于表示数据是否处理成功，抛出异常或者返回false将会在统计信息中归入失败次数，返回true则归入成功次数。

#### SequenceDataFlow类型作业

SequenceDataFlow类型作业则根据当前服务器所分配的分片项数量进行多线程处理，每个分片项使用同一线程处理，防止了同一分片的数据被多线程处理，从而导致的顺序问题。

如：从2个分片共获取到100条数据，第1个分片40条，第2个分片60条，则系统自动分配两个线程处理，第1个线程处理第1个分片的40条数据，第2个线程处理第2个分片的60条数据。

该类型作业实现需要继承

AbstractIndividualSequenceDataFlowElasticJob类，实现方法与ThroughputDataFlow作业类型相同。

### Script 类型作业

Script类型作业意为脚本类型作业，支持shell，python，perl等所有类型脚本。只需通过控制台/代码配置scriptCommandLine即可。执行脚本路径可以包含参数，最后一个参数为作业运行时信息.

## 作业批量处理

为了提高数据处理效率，数据流类型作业提供了批量处理数据的功能。

批量处理则使用另外两个接口

AbstractBatchThroughputDataFlowElasticJob和

AbstractBatchSequenceDataFlowElasticJob。不同之处在于processData方法的返回值从boolean类型变为int类型，用于表示一批数据处理的成功数量，第二个入参则转变为List数据集合。

## 作业任务监听配置

Elastic-job可以通过配置多个任务监听器，在任务执行前和执行后执行监听的方法。

监听器分为每台作业节点均执行和分布式场景中仅单一节点执行两种。

### 每台作业节点均执行的监听

每台作业监听器会在任何一台作业服务器任务开始前和任务结束后执行。

监听需要实现ElasticJobListener 接口中的beforeJobExecuted和afterJobExecuted方法

若作业处理作业服务器的文件，处理完成后删除文件，可考虑使用每个节点均执行清理任务。此类型任务实现简单，且无需考虑全局分布式任务是否完成，请尽量使用此类型监听器。

### 分布式场景中仅单一节点执行的监听

分布式场景节点监听器只会在主作业服务器任务开始前和任务结束后执行。

监听需要实现AbstractDistributeOnceElasticJobListener

类中的doBeforeJobExecutedAtLastStarted

和doAfterJobExecutedAtLastCompleted方法

**若作业处理数据库数据，处理完成后只需一个节点完成数据清理任务即可。此类型任务处理复杂，需同步分布式环境下作业的状态同步，提供了超时设置来避免作业不同步导致的死锁，请谨慎使用**。

# 实现原理

## 弹性分布式实现

* **第一台服务器上线触发主服务器选举**。主服务器一旦下线，则重新触发选举，**选举过程中阻塞**，只有主服务器选举完成，才会执行其他任务。
* **某作业服务器上线时会自动将服务器信息注册到注册中心，下线时会自动更新服务器状态。**
* **主节点选举，服务器上下线，分片总数变更均更新重新分片标记**。
* **定时任务触发时，如需重新分片，则通过主服务器分片，分片过程中阻塞，分片结束后才可执行任务。如分片过程中主服务器下线，则先选举主服务器，再分片。**
* 通过上一项说明可知，为了维持作业运行时的稳定性，运行过程中只会标记分片状态，不会重新分片。分片仅可能发生在下次任务触发前。
* **每次分片都会按服务器IP排序，保证分片结果不会产生较大波动**。
* **实现失效转移功能，在某台服务器执行完毕后主动抓取未分配的分片，并且在某台服务器下线后主动寻找可用的服务器执行任务**。

## 实现方案及开发理念

### 1 elastic-job的具体模块的底层及如何实现

elastic-job采用**去中心化**设计，主要分为**注册中心**，**数据分片**，**分布式协调**，定时任务处理和定制化流程型任务等模块。

**a) 去中心化**

去中心化指elastic-job并无调度中心这一概念，每个运行在集群中的作业服务器都是对等的，节点之间通过注册中心进行分布式协调。但elastic-job有主节点的概念，主节点用于处理一些集中式任务，如分片，清理运行时信息等，并无调度功能，定时调度都是由作业服务器自行触发。

下面对比一下各自的优缺点：

|  |  |  |
| --- | --- | --- |
|  | **中心化** | **去中心化** |
| **实现难度** | 高 | 低 |
| **部署难度** | 高 | 低 |
| **触发时间统一控制** | 可以 | 不可以 |
| **触发延迟** | 有 | 无 |
| **异构语言支持** | 容易 | 困难 |

**b) 注册中心**

注册中心模块目前直接使用zookeeper，用于记录作业的配置，服务器信息以及作业运行状态。Zookeeper虽然很成熟，但原理复杂，使用较难，在海量数据支持的情况下也会有性能和网络问题。目前elastic-job已经抽象出注册中心的接口，下一步将会考虑支持多注册中心，如etcd，或由用户自行实现注册中心。无临时节点和监听机制的注册中心需要自行实现定时心跳监测等功能。

**c) 数据分片**

数据分片是elastic-job中实现分布式的重要概念，将真实数据和逻辑分片对应，用于解耦作业框架和数据的关系。作业框架只负责将分片合理的分配给相关的作业服务器，而作业服务器需要根据所分配的分片匹配数据进行处理。服务器分片目前都存储在注册中心中，各个服务器根据自己的IP地址拉取分片。

**d) 分布式协调**

分布式协调模块用于处理作业服务器的动态扩容缩容。一旦集群中有服务器发生变化，分布式协调将自动监测并将变化结果通知仍存活的作业服务器。协调时将会涉及主节点选举，重分片等操作。目前使用的Zookeeper的临时节点和监听器实现主动检查和通知功能。

**e) 定时任务处理**

定时任务处理根据cron表达式定时触发任务，目前有防止任务同时触发，错过任务重出发等功能。主要还是使用Quartz本身的定时调度功能，为了便于控制，每个任务都使用独立的线程池。

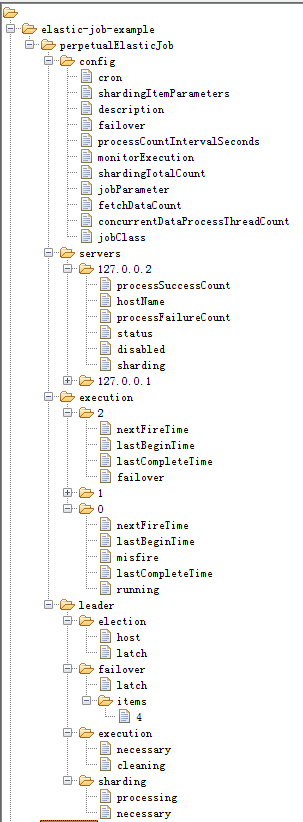
**f) 定制化流程型任务**

定制化流程型任务将定时任务分为多种流程，有不经任何修饰的简单任务；有用于处理数据的fetchData/processData的数据流任务；以后还将增加消息流任务，文件任务，工作流任务等。用户能以插件的形式扩展并贡献代码。

## 注册中心数据结构

**注册中心在定义的命名空间下**，创建作业名称节点，用于区分不同作业，所以**作业一旦创建则不能修改作业名称，如果修改名称将视为新的作业**。**作业名称节点下又包含4个数据子节点**，分别是config, servers, execution和leader。

### 概览



### config节点

作业全局配置信息

| **子节点名** | **临时节点** | **描述** |
| --- | --- | --- |
| jobClass | 否 | 作业实现类名称 |
| shardingTotalCount | 否 | 作业分片总数 |
| cron | 否 | 作业启动时间的cron表达式 |
| shardingItemParameters | 否 | 分片序列号和个性化参数对照表 |
| jobParameter | 否 | 作业自定义参数 |
| monitorExecution | 否 | 监控作业执行时状态 |
| processCountIntervalSeconds | 否 | 统计作业处理数据数量的间隔时间 |
| concurrentDataProcessThreadCount | 否 | 同时处理数据的并发线程数 |
| fetchDataCount | 否 | 每次抓取的数据量 |
| streaming-process | 否 | 是否流式处理数据 如果流式处理数据, 则fetchData不返回空结果将持续执行作业 如果非流式处理数据, 则处理数据完成后作业结束 |
| maxTimeDiffSeconds | 否 | 允许的本机与注册中心的时间误差秒数 |
| failover | 否 | 是否开启失效转移 |
| misfire | 否 | 是否开启错过任务重新执行 |
| description | 否 | 作业描述信息 |
| scriptCommandLine | 否 | 脚本型作业作业执行命令行 |

### servers节点

作业服务器信息，**子节点是作业服务器的IP地址**。**IP地址节点的子节点存储详细信息**。同一台作业服务器只能运行一个相同的作业实例，因为作业运行时是按照IP注册和管理的。

| **子节点名** | **临时节点** | **描述** |
| --- | --- | --- |
| hostName | 否 | 作业服务器名称 |
| status | 是 | 作业服务器状态，分为READY和RUNNING 用于表示服务器在等待执行作业还是正在执行作业 如果status节点不存在则表示作业服务器未上线 |
| disabled | 否 | 作业服务器状态是否禁用 可用于部署作业时，先禁止启动，部署结束后统一启动 |
| sharding | 否 | 该作业服务器分到的作业分片项 多个分片项用逗号分隔 如：0, 1, 2代表该服务器执行第1, 2, 3片分片 |
| processSuccessCount | 否 | 统计一段时间内处理数据成功的数量 统计间隔可通过config\processCountIntervalSeconds配置 |
| processFailureCount | 否 | 统计一段时间内处理数据失败的数量 统计间隔可通过config\processCountIntervalSeconds配置 |
| paused | 否 | 暂停作业的标记，暂停的作业不会终止调度器运行。作业程序再次启动时不会清理此标记 |
| shutdown | 否 | 关闭作业的标记，关闭的作业将停止调度，并可通过控制台删除。只有作业程序再次启动时才会清理此标记 |
| trigger | 否 | 立刻触发作业的标记，作业在不与上次运行中作业冲突的情况下将立刻启动，并在启动后自动清理此标记 |

### execution节点

执行时信息，子节点是分片项序号，从零开始，至分片总数减一。分片项序号的子节点存储详细信息。可通过配置config\monitorExecution为false关闭记录作业执行时信息。

| **子节点名** | **临时节点** | **描述** |
| --- | --- | --- |
| running | 是 | 分片项正在运行的状态 如果没有此节点，并且没有completed节点，表示该分片未运行 |
| completed | 否 | 分片项运行完成的状态 下次作业开始执行时会清理 |
| failover | 是 | 如果该分片项被失效转移分配给其他作业服务器，则此节点值记录执行此分片的作业服务器IP |
| lastBeginTime | 否 | 该分片项最近一次的开始执行时间 |
| nextFireTime | 否 | 该分片项下次作业触发时间 |
| lastCompleteTime | 否 | 该分片项最近一次的结束执行时间 |
| misfire | 否 | 是否开启错过任务重新执行 |

### leader节点

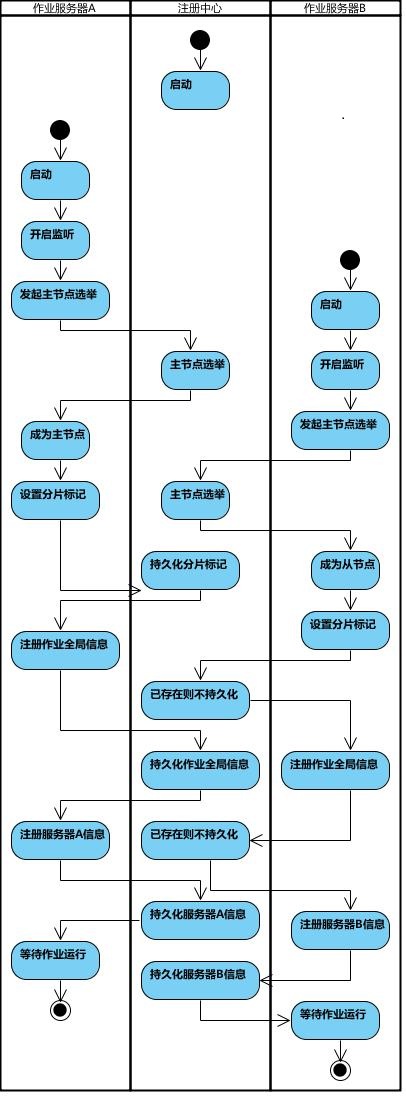
作业服务器主节点信息，分为election，sharding和execution三个子节点。分别用于主节点选举，分片和作业执行时处理。

leader节点是内部使用的节点，如果对作业框架原理不感兴趣，可不关注此节点。

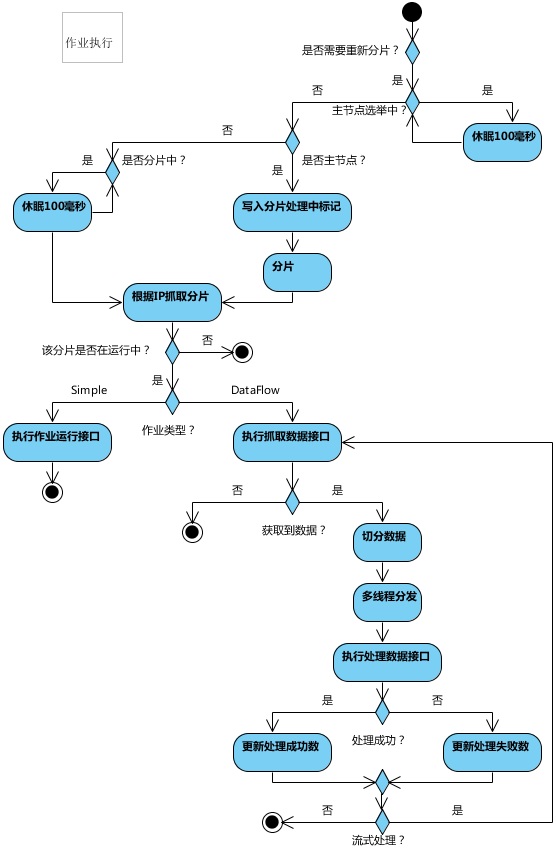
| **子节点名** | **临时节点** | **描述** |
| --- | --- | --- |
| election\host | 是 | 主节点服务器IP地址 一旦该节点被删除将会触发重新选举 重新选举的过程中一切主节点相关的操作都将阻塞 |
| election\latch | 否 | 主节点选举的分布式锁 为curator的分布式锁使用 |
| sharding\necessary | 否 | 是否需要重新分片的标记 如果分片总数变化，或作业服务器节点上下线或启用/禁用，  以及主节点选举，会触发设置重分片标记 作业在下次执行时使用主节点重新分片，且中间不会被打断 作业执行时不会触发分片 |
| sharding\processing | 是 | 主节点在分片时持有的节点 如果有此节点，所有的作业执行都将阻塞，直至分片结束 主节点分片结束或主节点崩溃会删除此临时节点 |
| execution\necessary | 否 | 是否需要修正作业执行时分片项信息的标记 如果分片总数变化，会触发设置修正分片项信息标记 作业在下次执行时会增加或减少分片项数量 |
| execution\cleaning | 是 | 主节点在清理上次作业运行时状态时所持有的节点 每次开始新作业都需要清理上次运行完成的作业信息 如果有此节点，所有的作业执行都将阻塞，直至清理结束 主节点分片结束或主节点崩溃会删除此临时节点 |
| failover\items\分片项 | 否 | 一旦有作业崩溃，则会向此节点记录 当有空闲作业服务器时，会从此节点抓取需失效转移的作业项 |
| failover\items\latch | 否 | 分配失效转移分片项时占用的分布式锁 为curator的分布式锁使用 |

## 流程图

### 作业启动



### 作业执行



# dump作业信息

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| monitor-port | int | 否 | -1 | 作业监控端口。建议配置作业监控端口, 方便开发者dump作业信息。使用方法: echo “dump” | nc 127.0.0.1 9888 |

作业需要配置上monitor-port参数，才能dump执行信息，每个作业应该配置为不一样的端口。

[yangshihong@CDCS-213057126 logs]$ echo "dump" | nc 127.0.0.1 49888

/clearParkingDataSimpleJob/servers |

/clearParkingDataSimpleJob/servers/ip1 |

/clearParkingDataSimpleJob/servers/ip1/status | READY

/clearParkingDataSimpleJob/servers/ip1/sharding | 0,1,2,3,4

/clearParkingDataSimpleJob/servers/ip1/hostName | CDCS-213057126

/clearParkingDataSimpleJob/servers/ip2 |

/clearParkingDataSimpleJob/servers/ip2/status | READY

/clearParkingDataSimpleJob/servers/ip2/sharding | 5,6,7,8,9

/clearParkingDataSimpleJob/servers/ip2/hostName | CDCS-213042186

/clearParkingDataSimpleJob/leader |

/clearParkingDataSimpleJob/leader/sharding |

/clearParkingDataSimpleJob/leader/failover |

/clearParkingDataSimpleJob/leader/failover/latch |

/clearParkingDataSimpleJob/leader/failover/items |

/clearParkingDataSimpleJob/leader/execution |

/clearParkingDataSimpleJob/leader/election |

/clearParkingDataSimpleJob/leader/election/latch |

/clearParkingDataSimpleJob/leader/election/host | ip2

/clearParkingDataSimpleJob/execution |

/clearParkingDataSimpleJob/execution/9 |

/clearParkingDataSimpleJob/execution/9/nextFireTime | 1475058600000

/clearParkingDataSimpleJob/execution/9/misfire |

/clearParkingDataSimpleJob/execution/9/lastCompleteTime | 1475058000353

/clearParkingDataSimpleJob/execution/9/lastBeginTime | 1475058000199

/clearParkingDataSimpleJob/execution/9/completed |

/clearParkingDataSimpleJob/execution/8 |

/clearParkingDataSimpleJob/execution/8/nextFireTime | 1475058600000

/clearParkingDataSimpleJob/execution/8/misfire |

/clearParkingDataSimpleJob/execution/8/lastCompleteTime | 1475058000333

/clearParkingDataSimpleJob/execution/8/lastBeginTime | 1475058000161

/clearParkingDataSimpleJob/execution/8/completed |

/clearParkingDataSimpleJob/execution/7 |

/clearParkingDataSimpleJob/execution/7/nextFireTime | 1475058600000

/clearParkingDataSimpleJob/execution/7/misfire |

/clearParkingDataSimpleJob/execution/7/lastCompleteTime | 1475058000309

/clearParkingDataSimpleJob/execution/7/lastBeginTime | 1475058000141

/clearParkingDataSimpleJob/execution/7/completed |

/clearParkingDataSimpleJob/execution/6 |

/clearParkingDataSimpleJob/execution/6/nextFireTime | 1475058600000

/clearParkingDataSimpleJob/execution/6/misfire |

/clearParkingDataSimpleJob/execution/6/lastCompleteTime | 1475058000287

/clearParkingDataSimpleJob/execution/6/lastBeginTime | 1475058000117

/clearParkingDataSimpleJob/execution/6/completed |

/clearParkingDataSimpleJob/execution/5 |

/clearParkingDataSimpleJob/execution/5/nextFireTime | 1475058600000

/clearParkingDataSimpleJob/execution/5/misfire |

/clearParkingDataSimpleJob/execution/5/lastCompleteTime | 1475058000257

/clearParkingDataSimpleJob/execution/5/lastBeginTime | 1475058000101

/clearParkingDataSimpleJob/execution/5/completed |

/clearParkingDataSimpleJob/execution/4 |

/clearParkingDataSimpleJob/execution/4/nextFireTime | 1475058600000

/clearParkingDataSimpleJob/execution/4/misfire |

/clearParkingDataSimpleJob/execution/4/lastCompleteTime | 1475058000870

/clearParkingDataSimpleJob/execution/4/lastBeginTime | 1475058000310

/clearParkingDataSimpleJob/execution/4/completed |

/clearParkingDataSimpleJob/execution/3 |

/clearParkingDataSimpleJob/execution/3/nextFireTime | 1475058600000

/clearParkingDataSimpleJob/execution/3/misfire |

/clearParkingDataSimpleJob/execution/3/lastCompleteTime | 1475058000848

/clearParkingDataSimpleJob/execution/3/lastBeginTime | 1475058000258

/clearParkingDataSimpleJob/execution/3/completed |

/clearParkingDataSimpleJob/execution/2 |

/clearParkingDataSimpleJob/execution/2/nextFireTime | 1475058600000

/clearParkingDataSimpleJob/execution/2/misfire |

/clearParkingDataSimpleJob/execution/2/lastCompleteTime | 1475058000824

/clearParkingDataSimpleJob/execution/2/lastBeginTime | 1475058000226

/clearParkingDataSimpleJob/execution/2/completed |

/clearParkingDataSimpleJob/execution/1 |

/clearParkingDataSimpleJob/execution/1/nextFireTime | 1475058600000

/clearParkingDataSimpleJob/execution/1/misfire |

/clearParkingDataSimpleJob/execution/1/lastCompleteTime | 1475058000802

/clearParkingDataSimpleJob/execution/1/lastBeginTime | 1475058000193

/clearParkingDataSimpleJob/execution/1/completed |

/clearParkingDataSimpleJob/execution/0 |

/clearParkingDataSimpleJob/execution/0/nextFireTime | 1475058600000

/clearParkingDataSimpleJob/execution/0/misfire |

/clearParkingDataSimpleJob/execution/0/lastCompleteTime | 1475058000782

/clearParkingDataSimpleJob/execution/0/lastBeginTime | 1475058000136

/clearParkingDataSimpleJob/execution/0/completed |

/clearParkingDataSimpleJob/config |

/clearParkingDataSimpleJob/config/shardingTotalCount | 10

/clearParkingDataSimpleJob/config/shardingItemParameters | 0=A,1=B,2=C,3=D,4=E,5=F,6=G,7=H,8=I,9=J

/clearParkingDataSimpleJob/config/monitorPort | 49888

/clearParkingDataSimpleJob/config/monitorExecution | true

/clearParkingDataSimpleJob/config/misfire | true

/clearParkingDataSimpleJob/config/maxTimeDiffSeconds | -1

/clearParkingDataSimpleJob/config/jobType | SIMPLE

/clearParkingDataSimpleJob/config/jobShardingStrategyClass | com.dangdang.ddframe.job.plugin.sharding.strategy.OdevitySortByNameJobShardingStrategy

/clearParkingDataSimpleJob/config/jobParameter |

/clearParkingDataSimpleJob/config/jobClass | com.wanda.cloudparking.service.job.ClearParkingDataSimpleJob

/clearParkingDataSimpleJob/config/failover | true

/clearParkingDataSimpleJob/config/description | 实时停车数据清理

/clearParkingDataSimpleJob/config/cron | 0 0/10 \* \* \* ?

# 项目集成

* **直接引入maven坐标**

<!-- 引入elastic-job核心模块 -->

<dependency>

<groupId>com.dangdang</groupId>

<artifactId>elastic-job-core</artifactId>

<version>${lasted.release.version}</version>

</dependency>

<!-- 使用springframework自定义命名空间时引入 -->

<dependency>

<groupId>com.dangdang</groupId>

<artifactId>elastic-job-spring</artifactId>

<version>${lasted.release.version}</version>

</dependency>

* **作业开发**

public class MyElasticJob extends AbstractSimpleElasticJob {

@Override

public void process(JobExecutionMultipleShardingContext context) {

// do something by sharding items

}

}

* **作业配置**

<?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:reg="http://www.dangdang.com/schema/ddframe/reg"

xmlns:job="http://www.dangdang.com/schema/ddframe/job"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd

http://www.dangdang.com/schema/ddframe/reg

http://www.dangdang.com/schema/ddframe/reg/reg.xsd

http://www.dangdang.com/schema/ddframe/job

http://www.dangdang.com/schema/ddframe/job/job.xsd

">

<!--配置作业注册中心 -->

<reg:zookeeper id="regCenter" server-lists="yourhost:2181" namespace="dd-job" base-sleep-time-milliseconds="1000" max-sleep-time-milliseconds="3000" max-retries="3" />

<!-- 配置作业-->

<job:simple id="myElasticJob" class="xxx.MyElasticJob" registry-center-ref="regCenter" cron="0/10 \* \* \* \* ?" sharding-total-count="3" sharding-item-parameters="0=A,1=B,2=C" />

</beans>

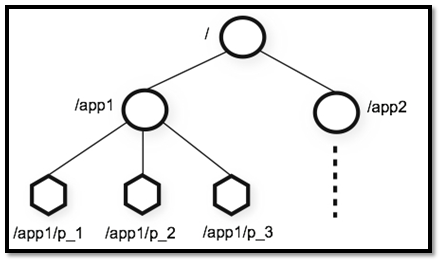
# 拓展

## Zookeer介绍

Zookeeper的视图结构类似标准的Unix文件系统，但是没有引入文件系统相关概念：目录和文件，而是使用了自己特有的节点(node)概念，称为znode。Znode是ZooKeeper中数据的最小单元，每个znode上都可以保存数据，同时还可以挂载子节点，也构成了一个层次化的命名空间，我们称之为树。

**树**

首先我们来看下面这张图，对ZooKeeper上的数据节点有一个大概的认识。这个图来自zookeeper官方文档中，相信很多开发者在网上也都看到过很多次了。在ZK中，每一个数据节点称为一个znode，所有znode按层次化进行组织，形成一棵树。znode是一个跟Unix文件系统路径相似的节点，由一系列由斜杠（/ )进行分割的路径表示。可以向这个节点中写入数据，也可以在节点下面创建子节点。

[](http://img1.51cto.com/attachment/201211/085951545.png)

**节点类型**

每个节点是有生命周期的，这取决于节点的类型。在ZooKeeper中，节点类型可以分为持久节点（PERSISTENT ）、临时节点（EPHEMERAL），以及时序节点（SEQUENTIAL ），具体在节点创建过程中，一般是组合使用，可以生成以下4种节点类型：

持久节点（PERSISTENT）

所谓持久节点，是指在节点创建后，就一直存在，直到有删除操作来主动清除这个节点——不会因为创建该节点的客户端会话失效而消失。

持久顺序节点（PERSISTENT\_SEQUENTIAL ）

这类节点的基本特性和上面的节点类型是一致的。额外的特性是，在ZK中，每个父节点会为他的第一级子节点维护一份时序，会记录每个子节点创建的先后顺序。基于这个特性，在创建子节点的时候，可以设置这个属性，那么在创建节点过程中，ZK会自动为给定节点名加上一个数字后缀，作为新的节点名。这个数字后缀的上限是整型的最大值。

临时节点（EPHEMERAL ）

和持久节点不同的是，临时节点的生命周期和客户端会话绑定。也就是说，如果客户端会话失效，那么这个节点就会自动被清除掉。注意，这里提到的是会话失效，而非连接断开。另外，在临时节点下面不能创建子节点。