## 3种作业类型的区别

### Simple类型作业

Simple类型作业意为简单实现，未经任何封装的类型。需要继承AbstractSimpleElasticJob，该类只提供了一个方法用于覆盖，此方法将被定时执行。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。

### ThroughputDataFlow类型作业

ThroughputDataFlow类型作业意为高吞吐的数据流作业。需要继承AbstractThroughputDataFlowElasticJob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类型作业和ThroughputDataFlow作业类型极为相似，所不同的是ThroughputDataFlow作业类型可以将获取到的数据多线程处理，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2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；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2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。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。

分数据处理，需要自己实现，可以根据分片的ID位数，对应数据的位数进行处理。

# 作业分片策略

## 框架提供的分片策略

### AverageAllocationJobShardingStrategy

**全路径：**

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

**全路径：**

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

**全路径：**

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

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。

## 配置分片策略

与配置通常的作业属性相同，在spring命名空间或者JobConfiguration中配置jobShardingStrategyClass属性，属性值是作业分片策略类的全路径。

# 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。

# Elastic-job目录结构说明

## elastic-job-core

elastic-job核心模块，只通过Quartz和Curator就可执行分布式作业。

## elastic-job-api

elastic-job生命周期操作的API，可独立使用。

## elastic-job-spring

elastic-job对spring支持的模块，包括命名空间，依赖注入，占位符等。

## elastic-job-console

elastic-job web控制台，可将编译之后的war放入tomcat等servlet容器中使用。

## elastic-job-example

使用示例。

## elastic-job-doc

使用markdown生成文档的项目，使用方无需关注。

# 作业Job配置

jobName(String)：作业名称

jobType(枚举JobType)：作业类型(SIMPLE, DATA\_FLOW, SCRIPT)，不需要配置作业构造函数中会自动添加

jobClass：分布式作业类型(有Simple类型作业、ThroughputDataFlow类型作业及SequenceDataFlow类型作业）

shardingTotalCount(int)：作业分片总数

cron：cron表达式(如:0/30 \* \* \* \* ?)

shardingItemParameters：设置分片序列号和个性化参数对照表.

\* 分片序列号和参数用等号分隔, 多个键值对用逗号分隔. 类似map.

\* 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

jobParameter：设置作业自定义参数,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

monitorExecution(boolean)：设置监控作业执行时状态，

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. 也无法实现作业失效转移.

每次作业执行时间和间隔时间均较长短的情况, 建议监控作业运行时状态, 可保证数据不会重复选取

maxTimeDiffSeconds(int)：设置最大容忍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表示不检查时间误差

failover(boolean):**设置是否开启失效转移(要开启失效转移，需将monitorExecution设置为true)**

misfire(boolean): 是否开启错过任务重新执行

monitorPort(int)：设置作业辅助监控端口(?)

jobShardingStrategyClass:设置作业分片策略实现类全路径，默认使用{@code com.dangdang.ddframe.job.plugin.sharding.strategy.AverageAllocationJobShardingStrategy}

description:设置作业描述信息

disabled(boolean):设置作业是否禁止启动,可用于部署作业时, 先禁止启动, 部署结束后统一启动

overwrite(boolean)：设置本地配置是否可覆盖注册中心配置

# 基于Zookeeper的注册中心配置

serverLists：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：命名空间(节点名称）

baseSleepTimeMilliseconds：等待重试的间隔时间的初始值(单位毫秒).

maxSleepTimeMilliseconds：等待重试的间隔时间的最大值(单位毫秒)

maxRetries：最大重试次数

sessionTimeoutMilliseconds：会话超时时间(单位毫秒)

connectionTimeoutMilliseconds：连接超时时间(单位毫秒)

digest：连接Zookeeper的权限令牌(缺省为不需要权限验证).

nestedPort：内嵌Zookeeper的端口号，-1表示不开启内嵌Zookeeper.

nestedDataDir：内嵌Zookeepe的数据存储路径,为空表示不开启内嵌Zookeeper.

# Elastic-Job主要功能

## 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。

## 其他功能

* **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。

# 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。