**基于akka技术栈的任务调度引擎功能与实现技术介绍**

目 录

[1 简介 4](#_Toc48293077)

[2 类似产品特性对比 4](#_Toc48293078)

[3 架构介绍 5](#_Toc48293079)

[3.1 架构图 5](#_Toc48293080)

[3.2 Rest API 5](#_Toc48293081)

[3.3 Java API 6](#_Toc48293082)

[3.4 Workflow Builder（工作流构建器） 6](#_Toc48293083)

[3.5 Workflow Runner（工作流执行器） 6](#_Toc48293084)

[3.5.1 工作流运行 6](#_Toc48293085)

[3.5.2 工作流的重跑 7](#_Toc48293086)

[3.6 Task Client（任务客户端） 8](#_Toc48293087)

[3.7 Task Executor（任务执行端） 8](#_Toc48293088)

[3.7.1 任务执行 8](#_Toc48293089)

[3.7.2 优先级管理： 9](#_Toc48293090)

[3.8 Pub/Sub System（发布订阅系统） 9](#_Toc48293091)

[3.9 Metrics Listener（集群指标监听器） 9](#_Toc48293092)

[3.10 Fallback Listener（故障恢复监听器） 9](#_Toc48293093)

[3.11 二次开发模组 10](#_Toc48293094)

[3.11.1 Trigger Module（触发器模组） 10](#_Toc48293095)

[3.11.2 Repository Module（持久化仓库模组） 11](#_Toc48293096)

[3.11.3 FileSystem Module（文件系统模组） 11](#_Toc48293097)

[3.11.4 StdHandler Module（任务控制台输出管理模组） 11](#_Toc48293098)

[3.11.5 Expression Evaluator Module（表达式模组） 11](#_Toc48293099)

[3.11.6 Pluggable Executor Module（自定义任务模组） 11](#_Toc48293100)

[4 附录 12](#_Toc48293101)

[4.1 表达式语法 12](#_Toc48293102)

[4.1.1 duration（持续时间） 12](#_Toc48293103)

[4.1.1.1 函数 12](#_Toc48293104)

[4.1.1.2 运算符 12](#_Toc48293105)

[4.1.2 date（日期） 13](#_Toc48293106)

[4.1.2.1 函数 13](#_Toc48293107)

[4.1.2.2 运算符 13](#_Toc48293108)

[4.1.3 fs（本地文件） 14](#_Toc48293109)

[4.1.4 hdfs 14](#_Toc48293110)

[4.1.5 node（节点运行环境） 15](#_Toc48293111)

[4.1.6 graph（工作流） 15](#_Toc48293112)

[4.1.7 task（任务） 16](#_Toc48293113)

# 简介

基于akka的任务调度引擎是一个响应式，完全去中心化，可动态缩扩容，故障恢复，支持跨集群任务调度，指定节点执行任务等功能的调度引擎，目前正在开发中。已可以实现工作流的正常执行、停止、重试；工作流定时调度，暂停，继续，停止；状态查看等功能。

# 类似产品特性对比

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Oozie** | **Dolphin** | **此调度引擎** |
| 多租户 | 不支持 | 支持 | 支持 |
| 算子支持 | 不支持python，flink；对shell，java支持度不高 | 好 | 好 |
| 表达式支持 | 支持el表达式，难以二次开发 | 不支持 | 支持开源组件Aviator，可二次开发。 |
| 文件存储方式 | hdfs | hdfs，s3 | hdfs，s3  提供二次开发接口 |
| 定时任务策略 | cron表达式  hdfs事件监听 | cron表达式 | cron表达式  hdfs事件监听  提供二次开发接口 |
| 工作流失败策略 | 可自定义 | 不支持，若任务失败则工作流直接停止。 | 可自定义 |
| 工作流重跑策略 | 可仅重新运行失败节点，也可跳过指定的节点。并且重跑时可修改运行参数。 | 不支持，直接重新运行所有节点，并且没有运行参数的设置。 | 提供不同的重跑策略，并且重跑时可修改运行参数 |
| 高可用性 | 无 | 调度服务保证高可用，日志服务、告警服务不保证。 | 基于Akka Cluster完全去中心化，保证高可用。 |
| 插件开发 | 可自定义算子 | 可自定义算子 | 可自定义多个模块  详见3.10节 |

# 架构介绍

## 架构图

调度引擎的架构如图所示：



下面依次分析图中的各个模块：

## Rest API

调度引擎提供整一套Rest API来管理工作流和调度任务的执行，以及集群，工作流或调度任务的状态查看。类似Elasticsearch，在集群的所有节点都可以使用Rest API进行操作。

## Java API

Java API基于Rest API，用于java应用程序的开发。

## Workflow Builder（工作流构建器）

该模块在收到Rest Service提交的工作流DAG图后，会通过Akka Stream Graph构建一个可执行的工作流，工作流由算子组成，算子种类有：

|  |  |  |  |
| --- | --- | --- | --- |
| **算子名称** | **输入端口数** | **输出端口数** | **功能说明** |
| start | 0 | 1 | 启动一个动作流 |
| end | 1 | 0 | 结束一个工作流 |
| task | 1 | 1 | 执行一个任务，比如java，shell等 |
| fork | 1 | n | 并行的通知下游算子执行 |
| decision | 1 | n | 根据算子的条件表达式选择一个下游算子执行 |
| join | n | 1 | 收集上游的算子执行情况，当所有上游算子都执行完后，通知下游算子执行。用在fork算子的下游 |
| converge | n | 1 | 当上游的其中一个算子执行完后，就会通知下游的算子执行。  用在decision算子的下游。 |

## Workflow Runner（工作流执行器）

Workflow Builder中构建好的工作流会进入到该模块，负责工作流的运行。

### 工作流运行

工作流的运行包括两种失败策略：

* CONTINUE：当一个任务算子失败时继续执行下游算子，直到工作流结束；
* SHUTDOWN：当一个任务算子失败时立即停止。

运行流程如图所示：



当工作流开始执行时，只需要在工作流的输入端提交一个GraphMeta对象，GraphMeta对象内包含了工作流的运行参数和运行时的状态信息。当Task算子获取GraphMeta对象后会生成一个任务，提交给下方模块Task Client来执行，并监听任务的执行状态。当任务完成后会返回一个TaskMeta对象。GraphMeta和TaskMeta结合生成一个新的GraphMeta并通知到下游，然后下游的Task算子会继续这个过程。当整个工作流的流程结束后，GraphMeta对象中会存有所有运行的算子的状态信息。通过这种方式，可以实现工作流的运行。

### 工作流的重跑

工作流的重跑包括三种重试策略：

* RUN\_ALL：运行所有算子；
* RUN\_ONLY\_FAILED：只运行失败的算子；
* RUN\_ALL\_AFTER\_FAILED：运行失败算子和所有在其下游的算子。

第1种重跑策略与直接运行一次工作流类似，第2、3种则需要根据工作流前一次运行的状态来确定运行策略，其流程如图所示：



由于工作流第一次运行时输出的Graph Meta已经有了所有的Task Meta信息，所以只需要将这个Graph Meta添加上重跑策略重新输入到工作流即可，task算子会根据重跑策略和Graph Meta中上一次运行的结果判断是否要再运行一遍，由此来实现不同策略的工作流重试。

## Task Client（任务客户端）

Workflow Runner的task算子生成的任务会提交到该模块，该模块基于Akka Cluster Client，负责将任务提交到目标节点进行任务的执行或关闭，并实时监控节点的运行状态。

任务的详细执行流程参考2.1.6节。

## Task Executor（任务执行端）

### 任务执行

流程如下图所示：



任务客户端首先会询问执行端是否可用。执行端回复可用的同时会创建一个ExecutionInstance。任务客户端收到确认后会创建一个ClientInstance。之后遍由两个instance来进行该任务的交互。其中：

* ClientInstance负责监听状态信息和提交kill命令；
* ExecutionInstance负责向ClientInstance推送任务的状态信息并向**RunnerManager(执行管理器)**提交任务，RunnerManager会返回一个hook用来kill掉任务。
* RunnerManager(执行管理器)实际上是一个拥有固定线程的线程池，线程会根据任务配置将其交给一个TaskRunner(执行器)对象来执行任务，TaskRunner每次执行一个任务都会通过java的Process类创建一个子进程。这样能够保证资源隔离，并且可以使用不同的系统用户或kerberos租户来执行任务，java任务也能使用jvm参数。

### 优先级管理：

每个任务都带有一个优先级值，当任务提交进RunnerManager时首先会进入一个优先级队列会根据优先级排序，线程会循环从队列中拉取任务并执行，从而实现按优先级运行。

## Pub/Sub System（发布订阅系统）

任务调度引擎支持Akka的发布订阅系统，每一个节点都会订阅一个或多个topic。当TaskCient请求任务时，请求会根据任务配置的topic来随机找到一个拥有相同topic的节点。用户可以根据需求来配置需要的topic，比如：

* 按业务划分节点的topic；
* 对于一个跨集群的调度引擎，各个集群的节点都订阅一个各自的topic，这样就可以实现跨集群的任务调度；
* 每个节点都订阅一个独立的topic，这样就可以已节点为粒度来执行任务，不过在这种情况下若某节点故障可能会导致工作流执行失败。

## Metrics Listener（集群指标监听器）

用于监控集群节点的运行指标，比如堆内存，cpu占用等。

## Fallback Listener（故障恢复监听器）

当集群的某个节点故障或失效时该模块负责将失效节点管理的定时调度任务转移至其他节点运行。

调度引擎的定时调度任务在集群中的执行方式如图所示：



图中的Coordinator就是定时调度任务，由Trigger来进行管理，每个集群的节点都会管理一部分的Coordinator，且每个coordinator都有一个版本号，初始为0。Trigger根据coordinator定时触发工作流运行，工作流的任务算子会由Task Client分发给对应节点的Task Executor来执行。

当集群的其中一个节点失效时，Fallback Listener会将该节点管理的Coordinator重新提交到其他节点运行，并将版本号加1，失效节点如果还在运行，则会发现自己管理的Coordinator版本号要低于当前提交的版本号，于是停止调度。如图所示：



## 二次开发模组

### Trigger Module（触发器模组）

工作流需要周期性的执行，周期性执行的策略可能是每隔一段时间执行一次，也可能是监听某个文件目录出现变化就执行一次等等。

调度引擎提供com.oceanum.trigger.Trigger接口来供开发者实现不同的触发策略，接口包含启动、停止、暂停、继续四个方法。

### Repository Module（持久化仓库模组）

负责持久化和查询工作流的dag配置、工作流运行状态、任务运行状态等等。

调度引擎提供com.oceanum.persistence.RepositoryFactory接口来供开发者实现不同数据库下的持久化方案，推荐使用json格式存储的数据库。

### FileSystem Module（文件系统模组）

任务算子的执行文件需要存放在一个方便管理、上传、下载的地方，由该模组管理。

调度引擎提供com.oceanum.file.FileSystem接口来供开发者实现不同的文件系统。

### StdHandler Module（任务控制台输出管理模组）

Runner Manager中执行的任务的控制台输出会由该模组来管理，也可以通过该模组查看输出内容。

调度引擎提供com.oceanum.exec.StdHandlerFactory接口来供开发者实现不同的任务控制台输出管理方案，接口方法包括对stdout和stderr输出的存储和查看。

### Expression Evaluator Module（表达式模组）

调度引擎支持的表达式求值组件基于开源组件Aviator开发，可以动态生成任务的配置。目前已支持多种函数及表达式，以下为几个例子：

* 输出一小时后的时间的格式化字符串：

|  |
| --- |
| date.format('yyyy-MM-dd HH:mm:ss', date.now() + duration.hour(1)) |

* 判断文件是否存在并且文件大小大于1KB：

|  |
| --- |
| fs.exists('/tmp/test/file') && (fs.size('/tmp/test/file') > 1024) |

详细内容可以参考附录4.1

### Pluggable Executor Module（自定义任务模组）

用于自定义任务的开发。

# 附录

## 表达式语法

常规语法参考Aviator文档，下面为系统自定义的函数和操作符。

### duration（持续时间）

#### 函数

|  |  |  |  |
| --- | --- | --- | --- |
| 语法 | 例子 | 输出类型 | 注释 |
| duration.day([int]) | duration.day(1) | duration | 天数 |
| duration.hour([int]) | duration.hour(1) | duration | 小时数 |
| duration.minute([int]) | duration.minute(1) | duration | 分钟数 |
| duration.second([int]) | duration.second(1) | duration | 秒数 |
| duration.mill([int]) | duration.mill(1) | duration | 毫秒数 |
| duration([string]) | duration('2 days')  duration('1h')  duration('3s') | duration | 解析字符串 |

#### 运算符

|  |  |  |  |
| --- | --- | --- | --- |
| 符号 | 语法 | 例子 | 输出类型 |
| + | [duration] + [duration] | duration('1h') + duration('1d') | duration |
| - | [duration] - [duration] | duration('1h') - duration('1d') | duration |
| - | -[duration] | -duration('1h') | duration |
| / | [duration] / [float] | duration('1h') / 3.0 | duration |
| / | [duration] / [duration] | duration('1d') / duration('1h') | float |
| \* | [duration] \* [float] | duration('1h') \* 3.0 | duration |
| > | [duration] > [duration] | duration('1h') > duration('1d') | boolean |
| >= | [duration] >= [duration] | duration('1h') >= duration('1d') | boolean |
| < | [duration] < [duration] | duration('1h') < duration('1d') | boolean |
| <= | [duration] <= [duration] | duration('1h') <= duration('1d') | boolean |
| == | [duration] == [duration] | duration('1h') == duration('1d') | boolean |
| != | [duration] != [duration] | duration('1h') != duration('1d') | boolean |

### date（日期）

#### 函数

|  |  |  |  |
| --- | --- | --- | --- |
| 语法 | 例子 | 输出类型 | 注释 |
| date.now() |  | date | 当前时间 |
| date([long]) |  | date | 时间戳转日期 |
| date.parse([string], [string]) | date.parse('yyyy-MM-dd HH:mm:ss', '2020-08-12 14:10:00') | date | 解析时间字符串 |
| date.format([string], [date]) | date.format('yyyy-MM-dd HH:mm:ss',date.now()) | string | 生成格式化时间字符串 |

#### 运算符

|  |  |  |  |
| --- | --- | --- | --- |
| 符号 | 语法 | 例子 | 输出类型 |
| + | [date] + [duration] | date.now() + duration('1d') | date |
| - | [date] - [duration] | date.now() - duration('1d') | date |
| > | [date] > [date] |  | boolean |
| >= | [date] >= [date] |  | boolean |
| < | [date] < [date] |  | boolean |
| <= | [date] <= [date] |  | boolean |
| == | [date] == [date] |  | boolean |
| != | [date] != [date] |  | boolean |

### fs（本地文件）

|  |  |  |  |
| --- | --- | --- | --- |
| 语法 | 例子 | 输出类型 | 注释 |
| fs.exits([string]) | fs.exists('/tmp/file.txt') | boolean | 文件或目录是否存在 |
| fs.isDir([string]) | fs.isDir('/tmp/file.txt') | boolean | 是否为目录 |
| fs.isFile([string]) | fs.isFile('/tmp/file.txt') | boolean | 是否为文件 |
| fs.canExecute([string]) | fs.canExecute('/tmp/file.txt') | boolean | 是否可执行 |
| fs.canRead([string]) | fs.canRead('/tmp/file.txt') | boolean | 是否可读 |
| fs.canWrite([string]) | fs.canWrite('/tmp/file.txt') | boolean | 是否可写 |
| fs.name([string]) | fs.name('/tmp/file.txt') | string | 文件或目录名称 |
| fs.parent([string]) | fs.parent('/tmp/file.txt') | string | 父目录路径 |
| fs.size([string]) | fs.size('/tmp/file.txt') | long | 文件字节数 |
| fs.modifiedTime([string]) | fs.modifiedTime('/tmp/file.txt') | long | 最近修改时间 |

### hdfs

|  |  |  |  |
| --- | --- | --- | --- |
| 语法 | 例子 | 输出类型 | 注释 |
| hdfs.exits([string]) |  | boolean | 文件或目录是否存在 |
| hdfs.isDir([string]) |  | boolean | 是否为目录 |
| hdfs.isFile([string]) |  | boolean | 是否为文件 |
| hdfs.size([string]) |  | long | 文件字节数 |
| hdfs.blockSize([string]) |  | long | 文件块字节数 |
| hdfs.accessTime([string]) |  | long | 最近访问时间 |
| hdfs.modifiedTime([string]) |  | long | 最近修改时间 |
| hdfs.owner([string]) |  | string | 文件所属用户 |

### node（节点运行环境）

|  |  |  |  |
| --- | --- | --- | --- |
| 语法 | 例子 | 输出类型 | 注释 |
| node.host() |  | string | 节点地址 |
| node.port() |  | boolean | 节点端口 |
| node.baseDir() |  | string | 节点程序所在目录 |
| node.workDir() |  | string | 节点任务运行根目录 |
| node.logDir() |  | string | 节点日志所在目录 |
| node.getEnv([string]) |  | string | 获取节点环境参数 |

### graph（工作流）

|  |  |  |  |
| --- | --- | --- | --- |
| 语法 | 例子 | 输出类型 | 注释 |
| graph.id() |  | int | 工作流运行id， |
| graph.rerunId() |  | int | 工作流重跑id |
| graph.name() |  | string | 工作流名称 |
| graph.createTime() |  | date | 工作流创建时间 |
| graph.scheduleTime() |  | date | 工作流的调度时间 |
| graph.startTime() |  | date | 工作流实际开始时间 |
| graph.task([string]) |  | task | 根据任务名称找到工作流中的任务 |
| graph.latestTask() |  | task | 获取最近运行的任务 |

### task（任务）

|  |  |  |  |
| --- | --- | --- | --- |
| 语法 | 例子 | 输出类型 | 注释 |
| task.id() |  | int | 当前任务运行id， |
| task.id([string]) |  | int | 根据任务名称找到任务运行id， |
| task.rerunId() |  | int | 当前任务重跑id |
| task.rerunId([string]) |  | int | 根据任务名称找到任务重跑id |
| task.name() |  | string | 当前任务名称 |
| task.name([string]) |  | string | 根据任务名称找到任务名称 |
| task.createTime() |  | date | 当前任务创建时间 |
| task.createTime([string]) |  | date | 根据任务名称找到任务创建时间 |
| task.startTime() |  | date | 当前任务的开始时间 |
| task.startTime([string]) |  | date | 根据任务名称找到任务的开始时间 |
| task.endTime() |  | date | 当前任务结束时间 |
| task.user() |  | date | 任务的执行用户 |
| task.user([string]) |  | date | 根据任务名称找到任务的执行用户 |
| task.type() |  | date | 当前任务类型 |
| task.type([string]) |  | date | 根据任务名称找到任务类型 |
| task.execDir() |  | date | 当前任务的执行目录 |
| task.execDir([string]) |  | date | 根据任务名称找到任务的执行目录 |