编排框架设计文档

|  |  |  |  |
| --- | --- | --- | --- |
| 版本 | 变更概要 | 作者 | 修订日期 |
| V1.0 | 创建文档 | 陈其炜 | 2017-03-28 |
|  |  |  |  |

[1 概述 4](#_Toc481082462)

[2 技术框架 4](#_Toc481082463)

[3 名词解释 4](#_Toc481082464)

[4 编排流程 4](#_Toc481082465)

[5 核心线程序列图 5](#_Toc481082466)

[6 代码规范 6](#_Toc481082467)

[6.1 代码结构 6](#_Toc481082468)

[6.2 模块规范 7](#_Toc481082469)

[7 管理编排 9](#_Toc481082470)

[7.1 失败策略 9](#_Toc481082471)

[7.2 编排参数 9](#_Toc481082472)

[7.3 编排步骤参数 9](#_Toc481082473)

[8 调用编排 9](#_Toc481082474)

[8.1 发起单个作业 10](#_Toc481082475)

[8.1.1 参数格式 10](#_Toc481082476)

[8.2 发起多个作业 11](#_Toc481082477)

[8.3 重新执行已有作业 11](#_Toc481082478)

[8.4 继续待确认作业 12](#_Toc481082479)

[9 编排插件 12](#_Toc481082480)

[9.1 接口定义 12](#_Toc481082481)

[9.1.1 handle方法 12](#_Toc481082482)

[9.1.2 callback方法 12](#_Toc481082483)

[9.1.3 getType方法 13](#_Toc481082484)

[9.1.4 getDescription方法 13](#_Toc481082485)

[9.2 编写范例 13](#_Toc481082486)

[9.3 获取参数 13](#_Toc481082487)

[9.4 返回参数 14](#_Toc481082488)

[9.5 执行代理自动分配 15](#_Toc481082489)

[9.6 获取当前节点 15](#_Toc481082490)

[9.7 调用执行代理 15](#_Toc481082491)

[9.8 返回状态 16](#_Toc481082492)

[9.9 返回日志和错误 16](#_Toc481082493)

[9.10 执行锁策略 17](#_Toc481082494)

[10 定时调度插件 18](#_Toc481082495)

[10.1 接口定义 18](#_Toc481082496)

[10.1.1 executeInternal方法 18](#_Toc481082497)

[10.1.2 getJobClassId方法 18](#_Toc481082498)

[10.1.3 getName方法 18](#_Toc481082499)

[10.2 编写范例 18](#_Toc481082500)

[11 restful接口插件 19](#_Toc481082501)

[11.1 调用方式： 19](#_Toc481082502)

[11.2 接口定义 19](#_Toc481082503)

[11.2.1 myDoService()方法 19](#_Toc481082504)

[11.2.2 getToken()方法 20](#_Toc481082505)

[11.2.3 getName()方法 20](#_Toc481082506)

[11.3 编写范例 20](#_Toc481082507)

[12 restful接口 21](#_Toc481082508)

[12.1 回调接口：jobcallback 21](#_Toc481082509)

[12.2 取锁接口：getlock 22](#_Toc481082510)

[12.3 释放锁接口：releaselock 22](#_Toc481082511)

[13 数据库设计 22](#_Toc481082512)

[13.1 数据字典 22](#_Toc481082513)

# 概述

编排框架分成调度和执行两大模块，主要处理编排作业从发起到结束的整个业务流程，包括传递参数，更新状态，写日志等操作。

整个项目使用了web fragment实现各模块之间解耦部署。

# 技术框架

开发语言：java，后台框架：Spring4.x，数据持久层框架：Mybatis，数据库：MySql

# 名词解释

模块（module）：一个web-fragment工程，命名方式：vishnu-plugin-xxx，在eclipse里是一个单独的工程，允许有自己的页面。模块的划分原则可以根据专业组，例如：中间件、数据库、网络等进行，也可以根据比较大的技术条线，例如MySql、Oracle、Jetty等进行。模块和模块之间不建议互相依赖。

编排插件（plugin）：一个java类，需要继承PluginBase抽象接，用于提供原子操作服务，一个模块可以拥有多个插件。

原子操作：一个不可分割的操作，通过插件提供，通过动作暴露出来给用户配置。一个插件可以实现多个原子操作，也可以只实现一个原子操作，建议一个插件只实现一个原子操作，让不同的操作可以相互解耦。所有原子操作都必须遵循幂等性原则。

动作（action）：一个具体的动作，和插件是N:1关系，需要根据关联插件的特性配置入参和出参，不同的动作可以设置不同的参数。

编排（arrange）：用于组织编排步骤，接受输入参数并传递到编排步骤。

编排步骤（arrange step）：属于某个编排的步骤，和action是1:1关系，可以配置入参的输入方式。

作业（job）：一个编排实例，每次调用编排都会创建一个作业。作业会复制编排的所有信息，作业一旦创建完毕，无论编排如何修改配置，作业的配置都不会受影响。

作业步骤（job step）：作业步骤，作业创建时会根据编排步骤创建对应的作业步骤，作业步骤会复制编排步骤的所有信息，作业步骤一旦创建完毕，无论编排步骤如何修改配置，作业步骤的配置都不会受影响。

节点（Node）：编排步骤的远程执行目标，可以是一台主机或任何能远程控制的设备。每个编排步骤可能对应0到N个节点。节点数为0代表无需进行远程操作，例如分配资源等动作。

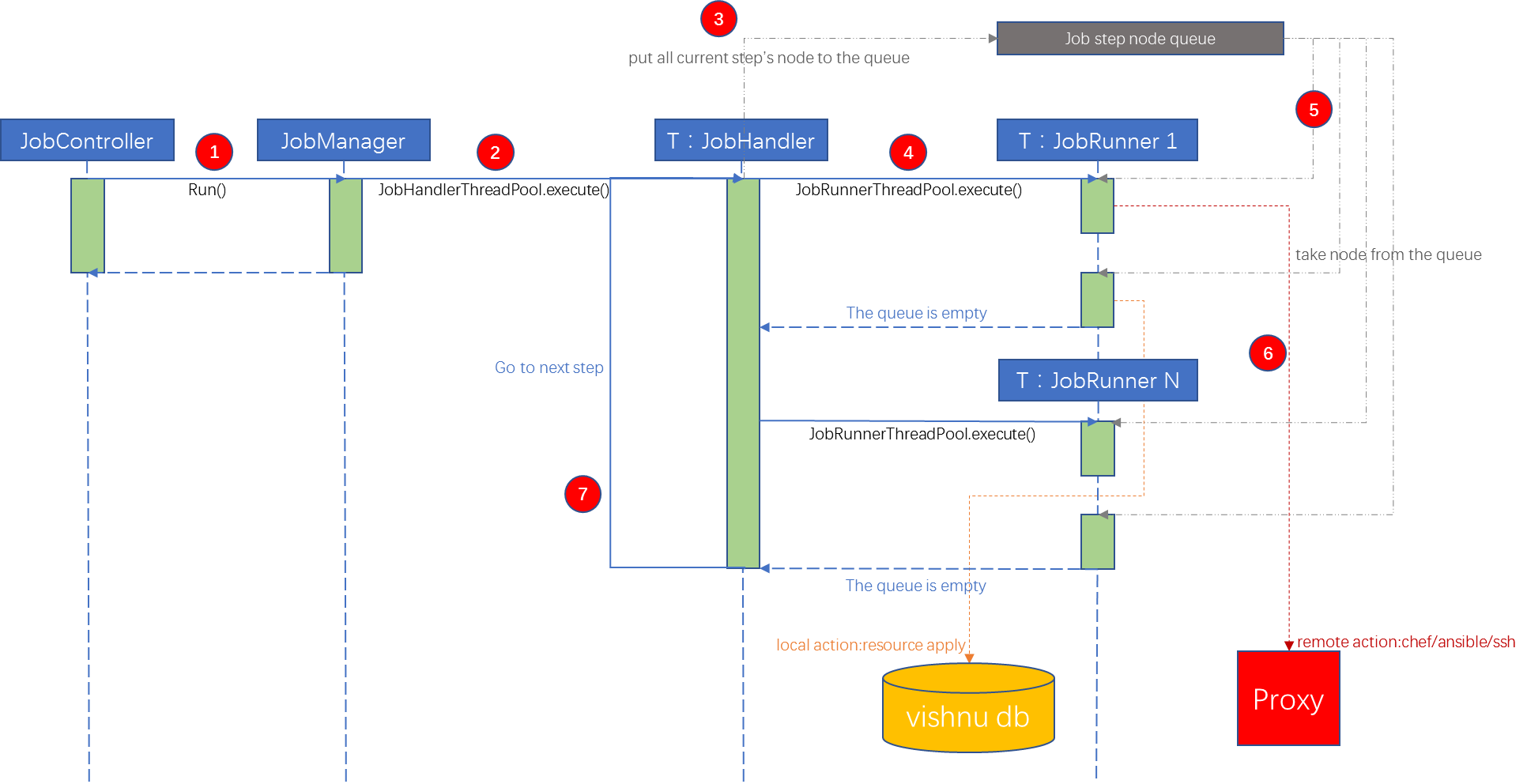
作业任务（jobtask）：一个作业任务是多个作业的组合，作业任务中的多个作业可以以串行或并行的方式进行。作业任务不需要事先声明，支持动态创建。作业任务应用在需要多个编排协作的场景。

定时调度插件：一个java类，需要集成JobBase。插件编写完发布上去，启动时系统会自动加载调度插件，每个插件可以配置多个定时作业，每个作业可以设置不同的时间计划和参数。

# 编排流程



# 核心线程序列图



1. 外部调用通过JobManager.run()方法启动一个job，系统会根据arrange信息和参数信息创建一个job并保存到数据库。
2. JobManager会创建一个线程：JobHandler并使用JobHandlerThreadPool进行管理，JobHandlerThreadPool是一个有限线程池，默认设置job.handler.minsize=3，job.handler.maxsize=10，队列长度100，如需修改请到config.properties更新以上参数，JobManager完成以上逻辑后马上返回，job交由JobHandler进行管理。
3. JobHanderl先获取一个空队列：job step node queue。再根据job step sort找到第一个未处理step，从参数从获取当前step的node列表，如果没有找到node，就创建一个nodeId=0的node对象，然后把这些node对象全部放到job step node queue中。
4. 根据step的threadcount参数和job step node queue当前容量决定创建多少个JobRunner线程来执行当前step。JobRunner由JobRunnerThreadPool进行管理，JobRunnerThreadPool是无限线程池，会优先复用空闲线程，如果没有空闲线程则创建新的线程，空闲线程在一段时间后会被销毁。
5. 每个JobRunner都会从job step node queue获取node执行当前步骤，如果job step node queue为空，线程返回。
6. JobRunner获取node后，根据当前step配置调动plugin进行实际操作，如果需要远程执行，则通过proxy进行调用。
7. 等当批次的JobRunner全部返回后，JobHandler更新当前step状态，检查是否还有未完成步骤，如果有，则回到第3步，如果没有，则销毁job step node queue，更新job的状态并返回。

# 代码规范

## 代码结构

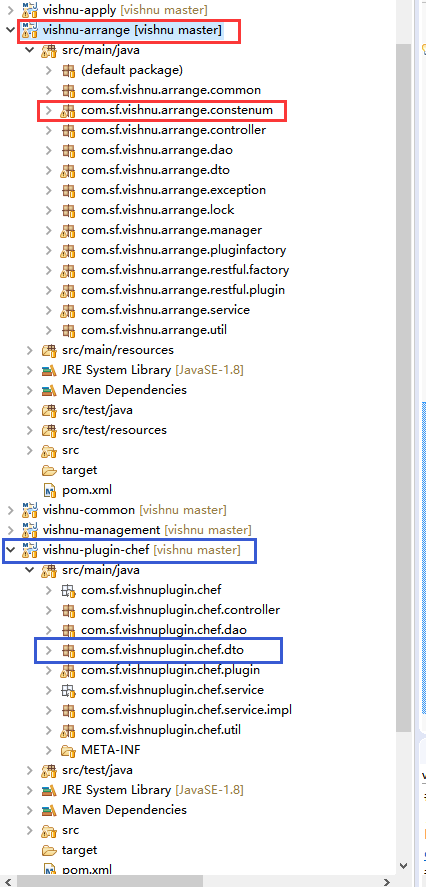
代码分核心模块和插件模块两大部分，核心模块主要实现编排框架逻辑等公共功能，插件模块用于维护各专业组自身需要的个性化功能。

核心模块的package命名需要以：com.sf.vishnu.开头。

插件模块的package命名需要以：com.sf.vishnuplugin.xxx开头。

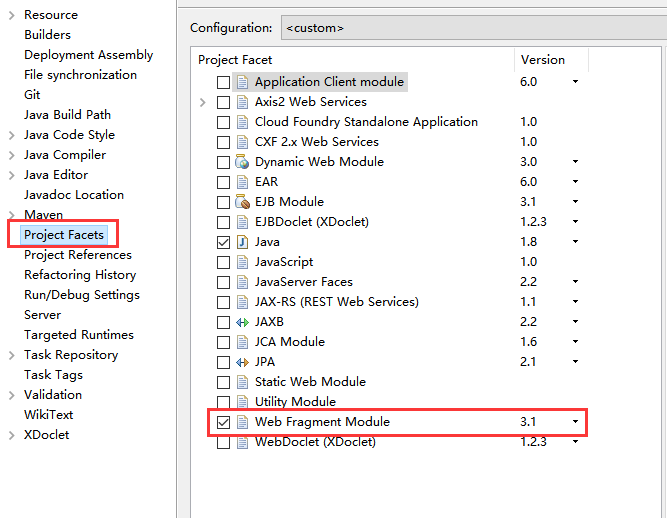
插件模块的工程命名需要以：vishnu-plugin-xxx方式命名。

注意：插件模块之间应该做到相互隔离，原则上不能互相依赖，部署的时候需要做到插件模块可以随意加载或卸载，而不会影响其他插件模块。如果存在需要被其他插件模块依赖的类或方法，应该放到核心模块代码里。

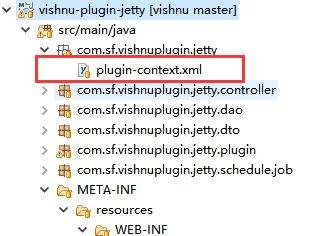


## 模块规范

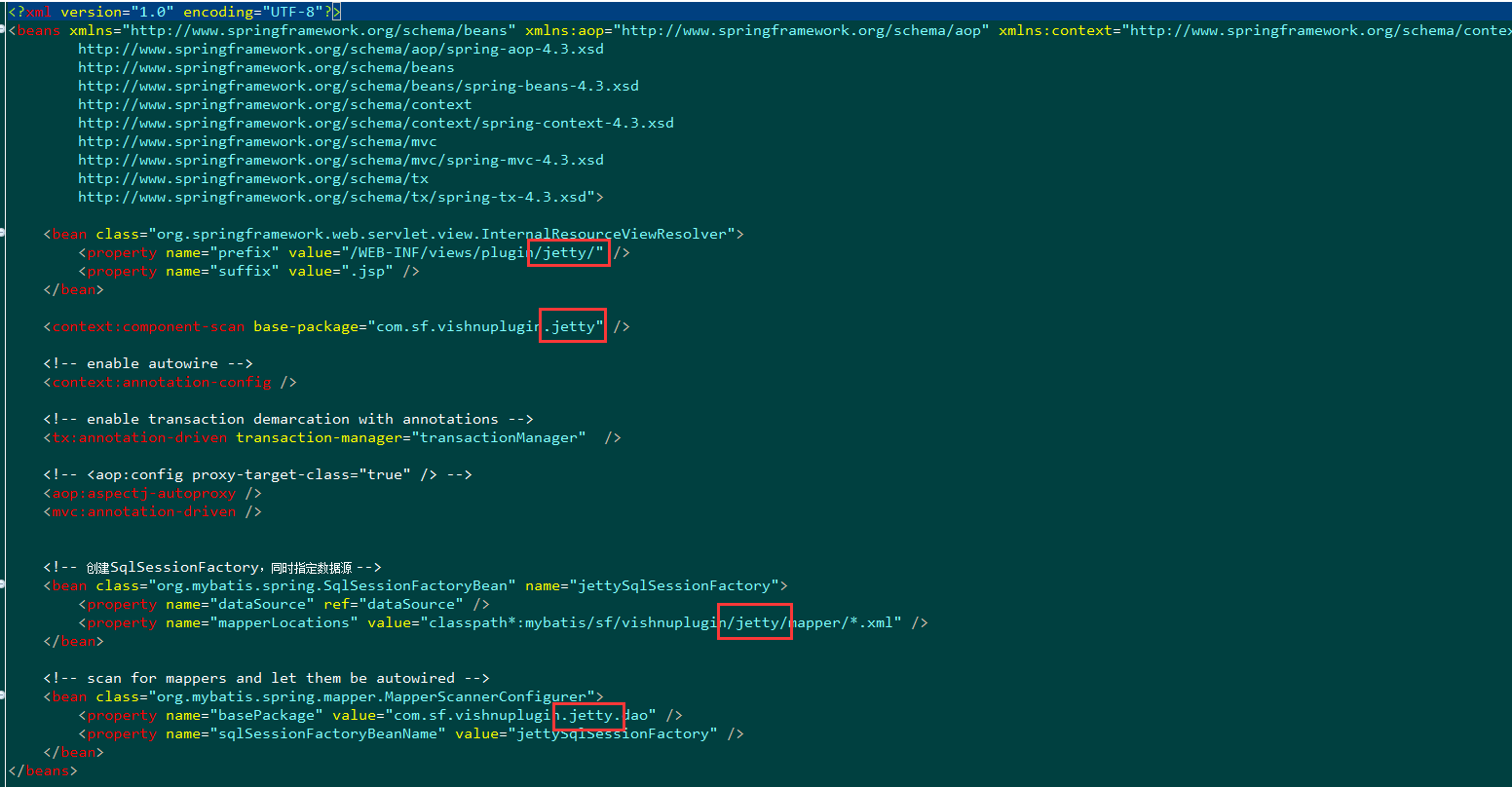
模块如果有页面，需要设置为web fragment工程。



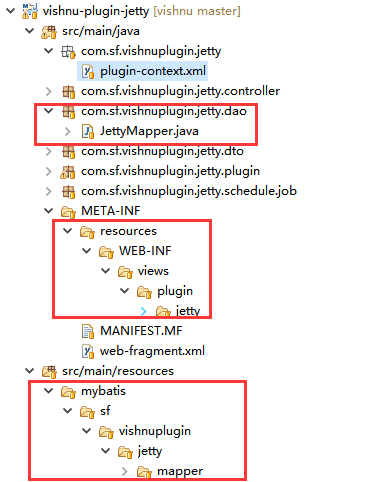
模块专用的spring配置文件路径如下图，名字必须设为：plugin-context.xml



一般情况下，plugin-context.xml配置文件只需要修改以下几个地方。这几个地方都和具体的文件路径有关，因此相关文件也要放到相应的文件夹中。



对应的页面文件路径：



提示：使用了web-fragment发布的模块会把所有资源打包到jar包中，即使修改jsp文件，也需要重启才会生效，如果想调试方便，可以安装eclipse的filesync插件，把静态文件自动拷到发布目录里，这样就不需要每次重启了。

# 管理编排

## 失败策略

编排执行过程中，各个地方都有可能出现异常，譬如作业，作业步骤，作业步骤节点。作业和作业步骤出现的异常一般都是状态异常，一旦遇到编排即会中止执行。作业步骤节点的异常大部分属于插件内部异常，针对这种异常系统有两种失败策略应对。

策略break：一旦遇到失败的作业步骤节点，整个编排将会中止执行。

策略continue：假如在某一步，步骤节点A发生异常，那么节点A将不会执行余下的步骤，但其他步骤不受影响。

注意：无论失败策略怎么选择，如果某一步没有一个成功的节点，那么余下的步骤都不会执行。

## 编排参数

编排参数用于封装当前编排所有步骤执行的所需参数，用户调用编排时，只需要关注编排参数，而无需关注编排步骤的参数。编排步骤执行过程中需要的参数，可以来自编排，也可以来自前置步骤的返回值，具体的策略由专业组管理员进行配置，普通用户无需关注。

编排参数来自两个地方：

1. 用户输入。
2. 通过${变量}表达式从作业任务公共参数列表中获取，如果获取不到则为空。

作业任务公共参数来自作业步骤节点的返回值，如果作业属于某个作业任务管理，那么在作业执行过程中，所有返回值都会存放到jobtask\_job\_param表中。系统在组装编排参数时遇到${变量}表达式，会从jobtask\_job\_param中提取参数值，如果找不到且此参数属于必填项，则返回报错。

## 编排步骤参数

编排步骤参数用于提供当前插件执行所需要的参数列表，编排步骤参数不是由用户直接输入，而是通过编排参数传递。

编排步骤参数来自三个地方：

1. 常量：const，例如username=admin，此处admin即为常量
2. 编排参数：arrange（来自用户调用编排时的输入）,例如username=admin，此处admin代表编排参数的名称，如果编排没有admin参数，那么username则为空。
3. 其他步骤的返回值：output，例如：username=1.admin，1代报步骤1，admin代表步骤1的返回参数名（需要在action\_param中配置），如果步骤1对应的action没有配置返回参数，那么即使插件返回了key为admin的参数，系统只会保存到job\_step\_node\_param表，但不会传递到下一步。

# 调用编排

## 发起单个作业



参数paramObj通过payload的方式传递，根据编排id获取编排的所有信息（包括编排步骤），实例化一个新的作业对象，然后把参数设进作业对象。

调用入口：JobManager.run(jobVo)，返回值是新建的作业id。

### 参数格式

发起单个作业的参数形式有两种模式。

简单模式：

{“key1”:”value1”,”key2”:”value2”,”key3”:[“value3”,”value4”]}

复杂模式：

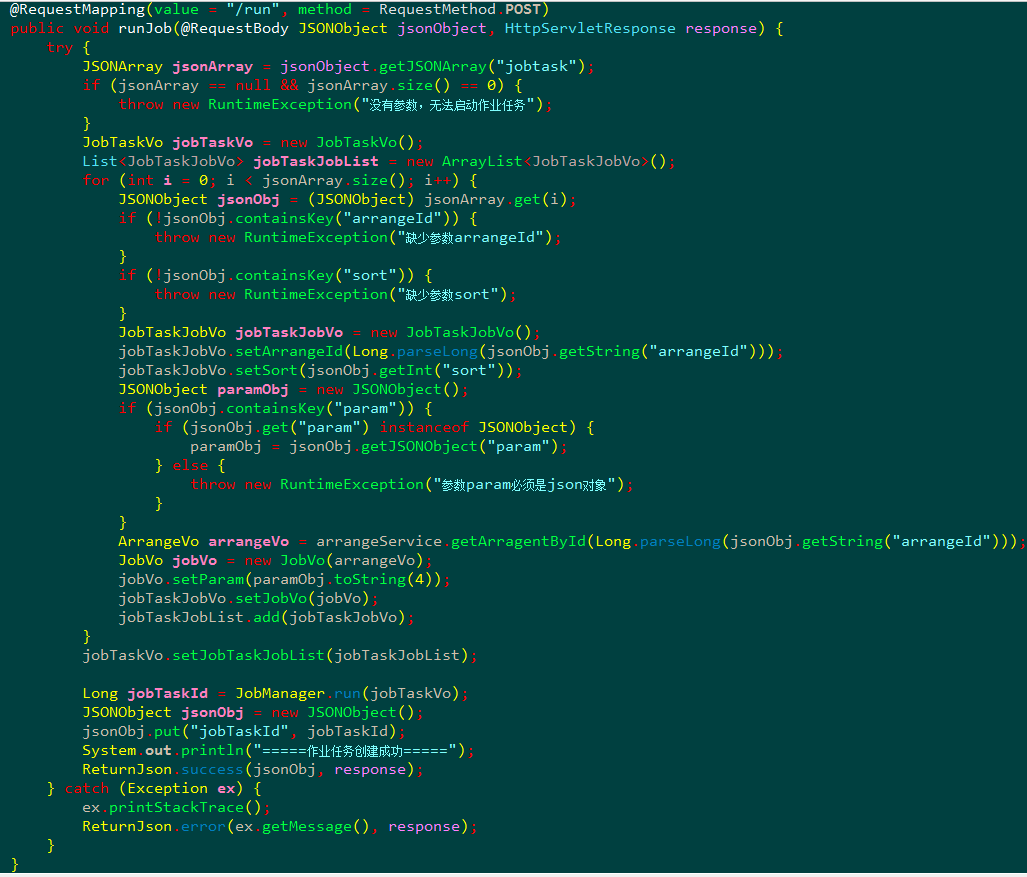
[{“key”:”key1”,”value”:”value1”,”host”:”host1”,”group1”:”group1”},

{“key”:”key1”,”value”:”value2”},

{“key”:”key2”:”value”:”value3”}]

注意：在复杂模式下，value值不能有数组结构，host和group属性根据需要设置

## 发起多个作业



系统支持动态发起多个作业，多个作业之间可以以串行或并行的方式进行，每次调用系统会自动创建一个作业任务管理这些作业。发起作业任务的参数范例：

{"jobtask":[

{"arrangeId":1,"sort":1,"param":{"env":"stg","ip\_count":1}},

{"arrangeId":1,"sort":2,"param":{"env":"stg","ip\_count":1}},

{"arrangeId":2,"sort":2,"param":{"env":"stg","ip\_count":1}}]

}

参数需要以payload的方式传递。

arrangeId表示编排id，同一个作业任务中的编排id可以重复。

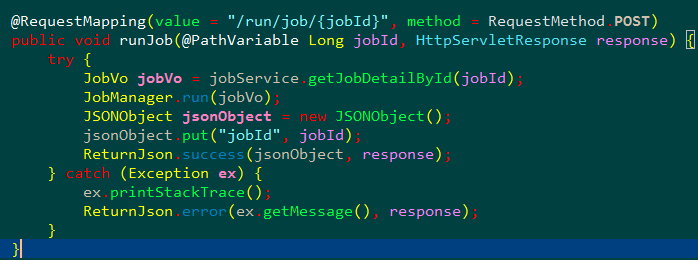
sort表示执行次序，系统会以从小到大的顺序调度作业，如果sort值一样，代表并行执行。

param表示当前编排需要的参数，必须是个json对象。

调用入口：JobManager.run(jobTaskVo)，返回值是新建的作业任务id。

## 重新执行已有作业

当作业执行失败时，可能需要重复执行已经创建过的作业，如果作业状态是已成功，则不能再次执行。执行过程中，只有状态不是已成功的步骤节点才会再次执行。



执行已有作业不能重新输入参数，只能从数据库中获取。

调用入口：JobManager.run(jobVo)，返回值是当前作业id。

系统判断当前调用是新作业还是老作业的唯一标识是，jobVo的id属性是否为null，为null代表新作业，不为null代表已有作业。

如果当前作业属于作业任务中的其中一个作业，那么作业执行完毕后，系统会自行判断是否执行其余作业。

## 继续待确认作业

编排步骤如果打开等待确认开关后，作业步骤在完成后不会自动流转到下一步，需要人工确认后才会继续流转。流转方法：找到待确认的作业步骤，将is\_confirm字段改成1，然后仿照8.3获取作业详细信息，调用JobManager.run(jobVo)，作业即会继续流转。

# 编排插件

## 接口定义

抽象类：com.sf.vishnu.arrange.pluginfactory.PluginBase

### handle方法

编排调度的主体方法，当编排执行到当前步骤时，会调用对应插件的handle方法。所有插件都必须实现handle方法

入参：

JobVo：当前作业信息

JobStepVo：当前步骤信息

JobStepNodeVo：当前节点信息，如果无需节点，则nodeId=0

ResultVo：结果对象

ParamList：参数列表，ParamVo集合的封装类，提供一些常用的方法以便获取参数

### callback方法

当前步骤如果需要回调时，系统会调用当前步骤对应插件的callback方法。此方法不是必须实现，只有需要callback的插件，并且在callback时需要执行一些自定义操作时，才需要实现callback方法。即使不实现callback方法，框架也会处理返回状态和输出参数，并根据情况流转到下一步。

入参：

JobVo：当前作业信息

JobStepVo：当前步骤信息

JobStepNodeVo：当前节点信息，如果无需节点，则nodeId=0

ResultVo：结果对象

ParamList：参数列表，ParamVo集合的封装类，提供一些常用的方法以便获取参数

JSONObject：callback时，发起方提供的原始json参数。如果发起方提供的参数比系统要求的规范更复杂，可以在此处进行二次处理。

### getType方法

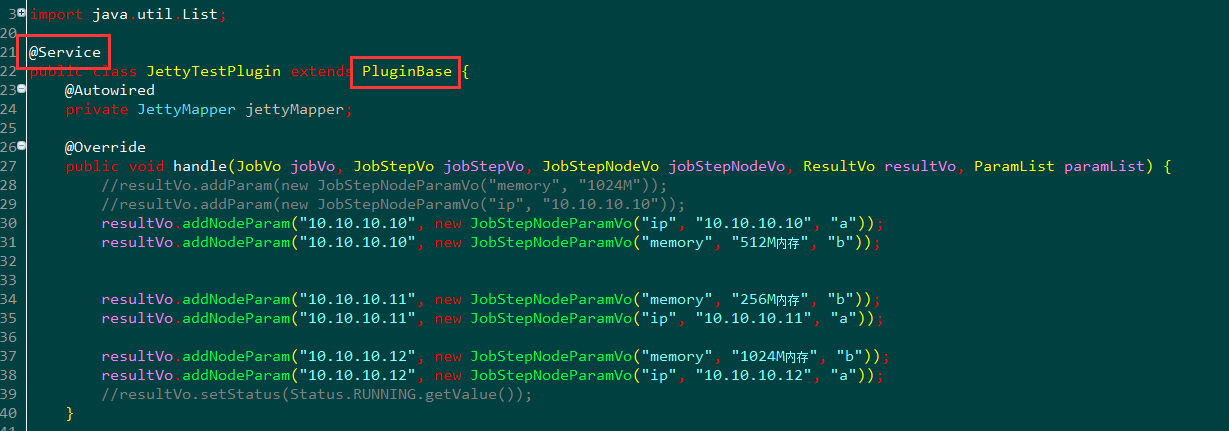
getType方法用于返回插件的唯一标记，可以直接使用当前插件的类全名作为唯一标识。

### getDescription方法

getDescription方法用于返回插件的中文描述，在配置动作（action）的时候，用户看到的是getDescription的返回值。

## 编写范例

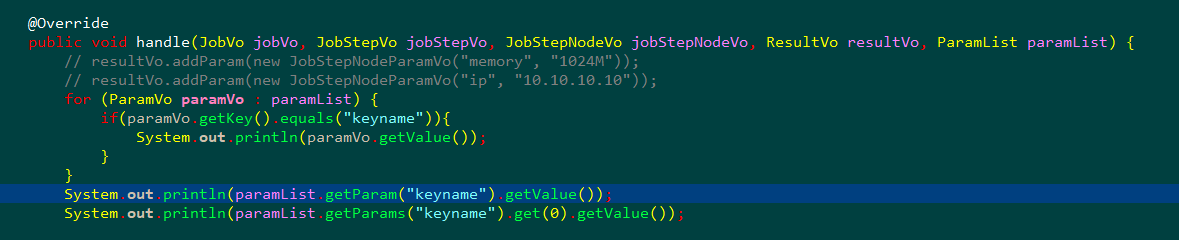
插件是一个java类，并且是一个Spring bean，需要继承PluginBase抽象类，并且需要添加Annotation @Component或@Service以便在启动过程中能被Spring自动加载。



编写插件和写普通的Spring bean类似，可以通过@Autowired注入其他对象，因此可以随时调用MyBatis进行数据库操作。

## 获取参数

参数获取需要自行迭代paramList获取，或者使用ParamList的api进行获取。框架会在进入插件之前组织好所有所需参数，包括常量，来自编排的参数和来自其他步骤的返回值（数据库中的arrange step需要配置相应的入参，框架不会传递没有配置的参数进入插件）。

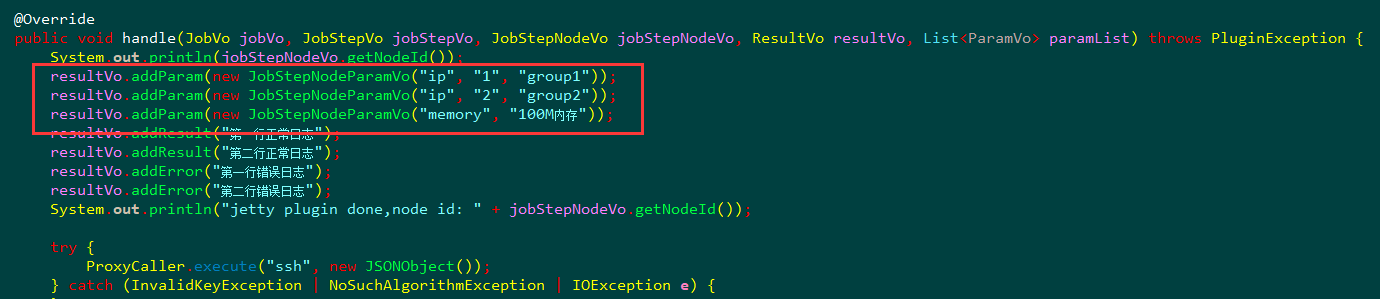


## 返回参数

返回的参数通过入参ResultVo进行传递。返回时只需要定义Key和Value，而无需关注参数的数据类型，框架会通过key在数据库中查找数据类型并补充进去。注意：如果是Node类型的参数，请返回数据库node表中的id值或host值。框架把当前步骤的返回参数传递到下一个步骤时会检查node是否存在，如果查不到Node信息，将会丢弃。所有通过ResultVo返回的参数无论是否合法，是否有在arrange step\_param中配置，都会统一写入job\_step\_node\_param表中以便查询。

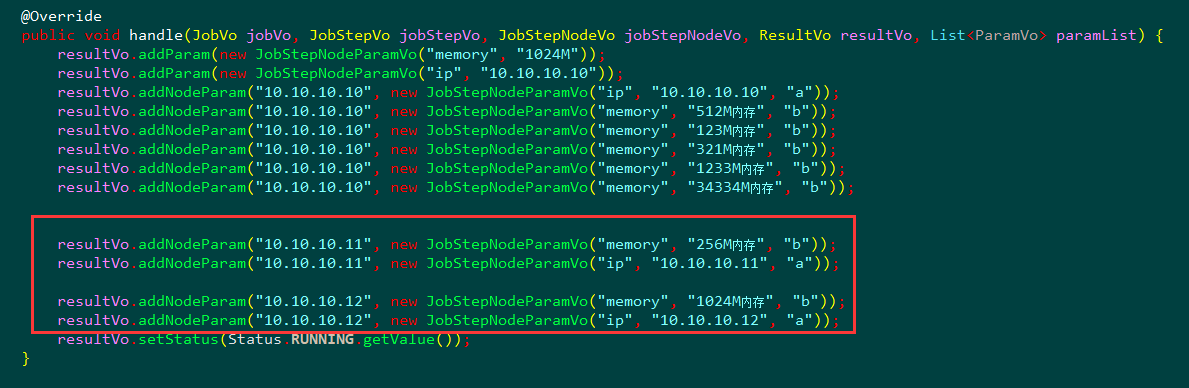
如果当前节点步骤已经执行过，在再次执行时，上次输出的参数会重新灌入ResultVo中，插件可以通过resultVo.getParamList()重新获取上次的输出参数。插件要根据幂等性原则自行判断是否需要重新产生的新的返回值。

注意：如果返回需要返回相同key和相同value的参数，但又不想被系统自动去重，可以通过group来给参数分组。



如果某些参数需要指定给某些后续步骤的节点使用，可以使用另外一个方法resultVo.addNodeParam()。

使用范例如下：



addNodeParam第一个参数是节点的host，需要和node表的host字段对应，第二个参数就是JobStepNodeParamVo。后续步骤节点开始执行前，框架会自动抽取对应的节点参数放到paramList里，这样每个节点都可以获取到不同的参数。

注意：通过addNodeParam返回的参数，只有对应的host和nodeId等于0的虚拟节点才能接收，相应的host只能接受到自己的参数，而虚拟节点能获取所有参数。

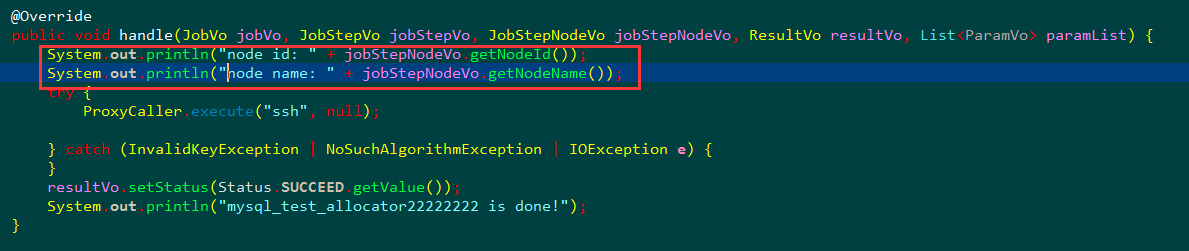
## 执行代理自动分配

如果当前步骤的执行代理为空，且当前节点的节点ID大于0，框架会根据节点的host字段和proxy\_network的network\_ip和mask字段来计算需要调用哪个执行代理来执行。如果同时匹配多个网段，框架会选择找到的第一个代理。

如果当前步骤的执行代理不为空，则框架不再为当前步骤节点分配执行代理。

## 获取当前节点

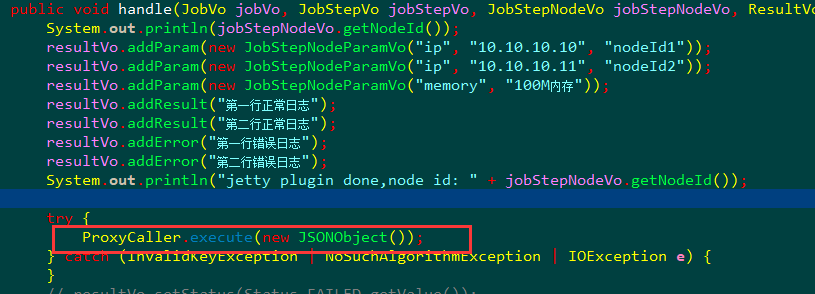
节点代表编排步骤需要执行的目标，它可以是一台主机或任何可以远程操作的设备。编排步骤的节点信息是通过参数的方式进行配置，参数类型是Node，输入方式可以是常量，来自编排或来自前置步骤的返回值。如果编排步骤没有指定Node参数，框架会构造一个虚拟节点，特征是NodeId等于0。NodeId大于0都代表一个真实的节点，但必须要在Node表中有记录，如果返回了不存在nodeId，并且后面的步骤需要引用，那么系统将会抛出NodeNotExistsException，编排会中止执行。



## 调用执行代理

如果当前步骤需要进行远程操作，绝大多数需要通过执行代理进行。执行代理也是个tomcat程序，它通过restful方式提供服务。访问执行代理的restful接口是需要用户验证的，验证方式配在vishnu-proxy的config.properties文件里，编排引擎则通过proxy表对的执行代理进行管理，调用前要确保编排引擎和执行代理的验证配置是一致的。

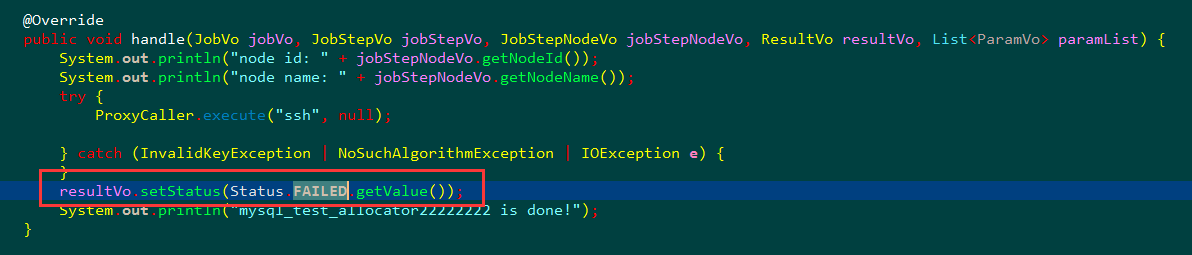
编排引擎在调度过程中会把当前步骤需要用到的执行代理信息保存的Threadlocal里，因此调用时只需要如下方式：



调用执行代理是通过json的方式传参，json的格式要求有执行代理对应的执行插件约定。

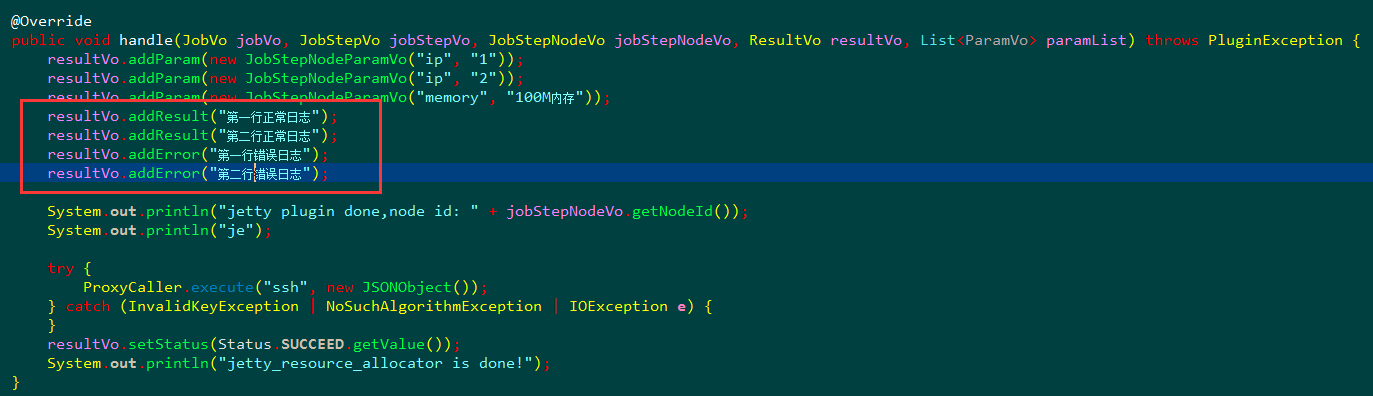
## 返回状态

通过ResultVo返回当前步骤的执行状态，可以返回SUCCEED、FAILED、RUNNING和IGNORED四种状态，如果不显式返回，则默认是SUCCEED。如果当前步骤执行条件不满足，但又希望可以继续执行后续步骤，可以返回IGNORED状态，代表当前步骤没有执行。注意：如果当前步骤需要通过异步方式进行，发起异步调用后请返回RUNNING状态，框架会停止调度编排，直到Callback返回SUCCEED或FAILED，才会继续调度。



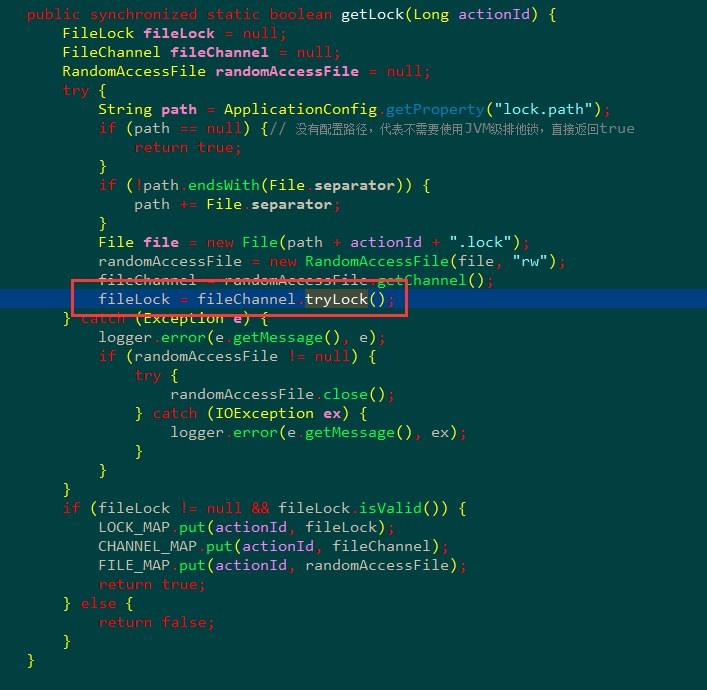
## 返回日志和错误

通过ResultVo返回日志或错误，addResult用于返回正常日志，addError用于返回错误日志。每调用一次addRestult或addError代表返回一行记录，最终输出日志时框架会根据这个来换行。也可以直接通过setResultList或setErrorList返回整个日志列表，每个成员代表一行。日志会通过文件的方式保存，数据库只会保存文件路径。注意：文件路径通过Config.properties中的jobstepnode.result.path参数进行配置。



## 执行锁策略

配置action时，如果不启用并发执行，那么在当前action执行前需要先获取执行锁，才能调用插件。执行锁在框架中实现，开发插件时无需关注。执行锁有两层，第一层是线程级，用于在同一JVM中控制并发，这一层锁可以设置获取超时时间，如果指定时间内获取不到锁，会抛异常返回，超时时间通过action表的timeout字段提供。第二层锁是共享文件锁，属于进程级，用于多个JVM之间实现并发控制，如果在config.properties里配置了lock.path参数，代表激活了第二层锁。,lock.path的路径应该是多个jvm进程都能访问的共享文件目录路径，不然失去并发控制的意义。第二层锁考虑安全的因素，获取锁没有采用阻塞方式，因此如果获取不到，将会马上抛出异常返回，不会等待。如需要改成阻塞方式，可以修改ActionLockGenerator.java，将tryLock改成lock即可。



# 定时调度插件

定时调度需要依赖quartz框架，定时调度插件需要集成JobBase类，实现executeInternal，getJobClassId和getName三个方法。如果是模块需要开发定时调度插件，maven的pom.xml需要配置对vishnu-schedule项目的依赖。

## 接口定义

接口：com.sf.vishnu.schedule.job.JobBase

### executeInternal方法

executeInternal(JobExecutionContext context, ScheduleJobVo jobVo)：调度的主体函数，所有需要定时执行的逻辑需要写到此方法内。JobExecutionContext是quartz执行的上下文，具体成员请查询quartz文档。ScheduleJobVo是当前作业的实体类，通过jobVo.getPropList()可以获取当前作业的自定义属性，自定义属性放在schedule\_job\_prop表里。

### getJobClassId方法

getJobClassId：返回一个全局唯一的id，如果id不唯一，会导致作业冲突。

### getName方法

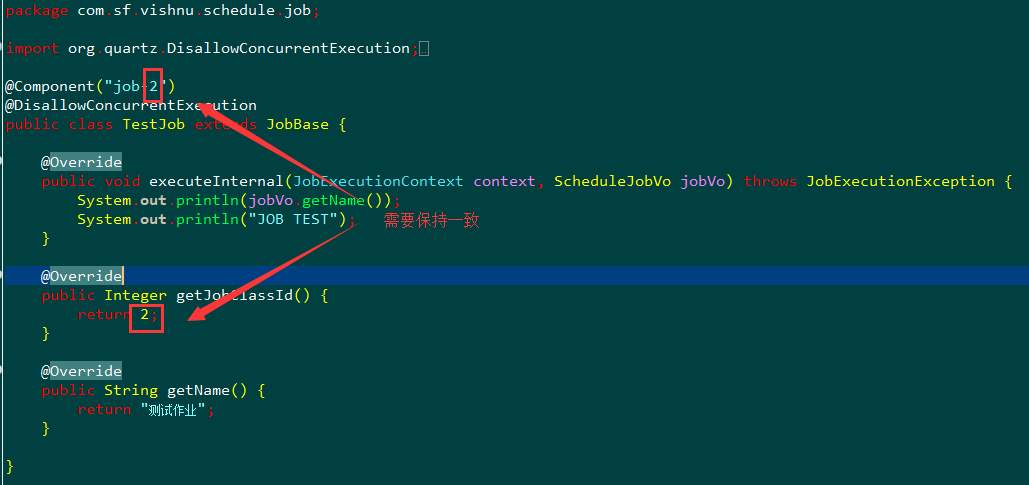
getName：返回定时调度插件中文名称。

## 编写范例

插件是一个java类，并且是一个Spring bean，需要继承JobBase抽象类，并且需要添加Annotation @Component或@Service以便在启动过程中能被Spring自动加载。

注意：@DisallowConcurrentExecution注解如果不加，quartz会无视插件的执行时间，严格按照调度计划来启动作业，如果作业执行很慢，会造成大量线程拥塞，强烈建议加上。

注意：每个定时调度插件需要定义bean name，具体方法参照下面截图，bean name的命名规则必须遵循“job-”+jobclassid的格式。这个设计是为了让quartz调度类能使用spring ioc的解决方案，请务必遵守。



# restful接口插件

restful插件用于对外提供restful服务，框架已经实现了用户认证，时效管理，参数转换，返回值转换等功能，插件只需要实现具体的逻辑。restful插件需要实现RestfulPluginBase接口。每个restful插件可以配置多个接口实例，每个实例可以分别设置不一样的认证方式和超时时间。

## 调用方式：

访问地址：<http://xxx/vishnu-web/restservices/token>，必须使用POST方式调用，参数要以json格式通过payload的方式输入。

需要查看接口帮助，需要在后面增加help，例如<http://xxx/vishnu-web/restservices/token/help>，查看帮助可以使用GET方法

## 接口定义

接口：com.sf.vishnu.arrange.restful.factory.RestfulPluginBase

### myDoService()方法

入参：

RestfulInterfaceVo：当前接口实例的实体类。

JSONObject：调用方传进来的参数，必须符合json格式

出参：

Object：框架会自动转成json格式，格式如下：{

"Return": 插件返回的Object值（可以是字符串或对象）,

"Status": "OK"

}

### getToken()方法

返回当前接口实例的唯一名称，正常情况下token是系统自动生成的md5散列值，开发人员也可以给出有意义的名字作为token，但无论哪种轻狂，所有的接口实例token都不能重复。

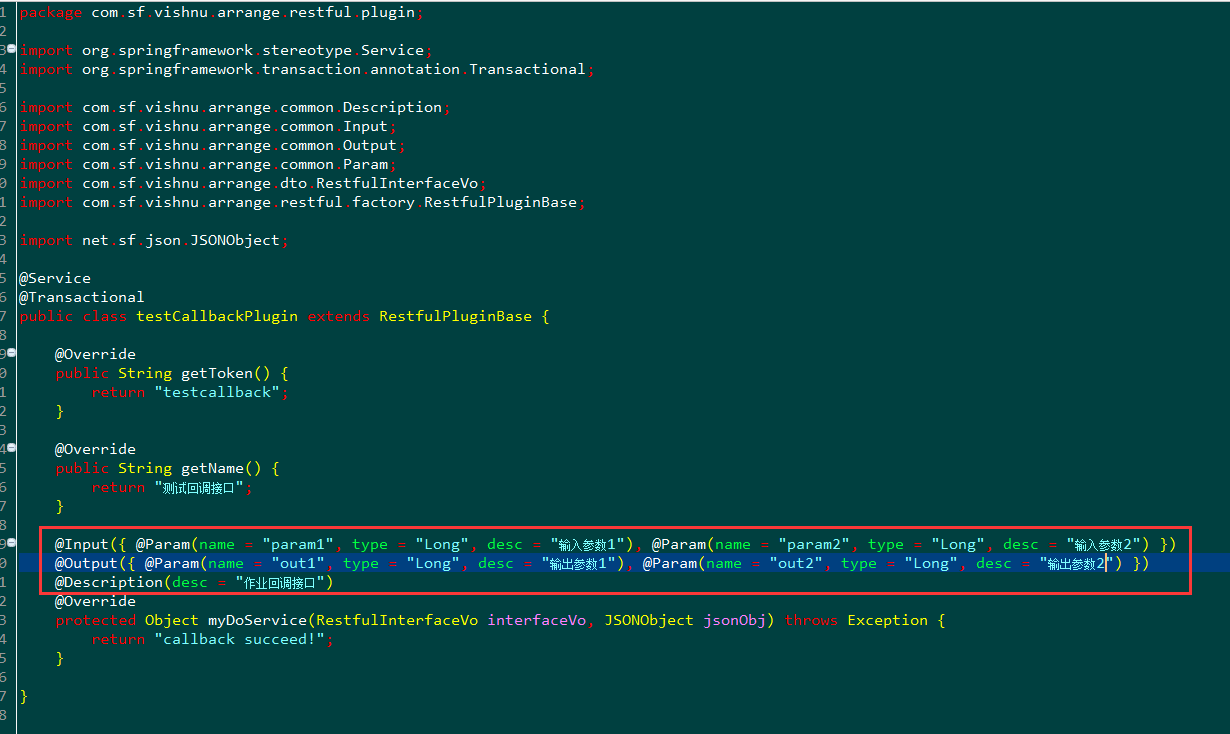
注意：如果返回null或空字符串，系统启动时只会加载当前插件，如果返回一个字符串，系统启动时除了加载当前插件，还会自动配置一个token等于此返回值的接口实例。

### getName()方法

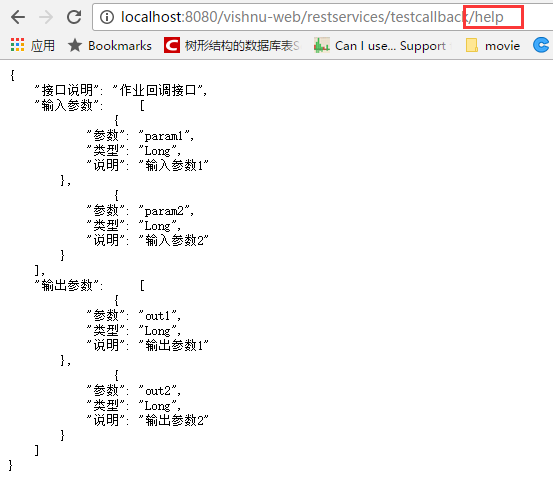
返回restful插件的中文说明

## 编写范例

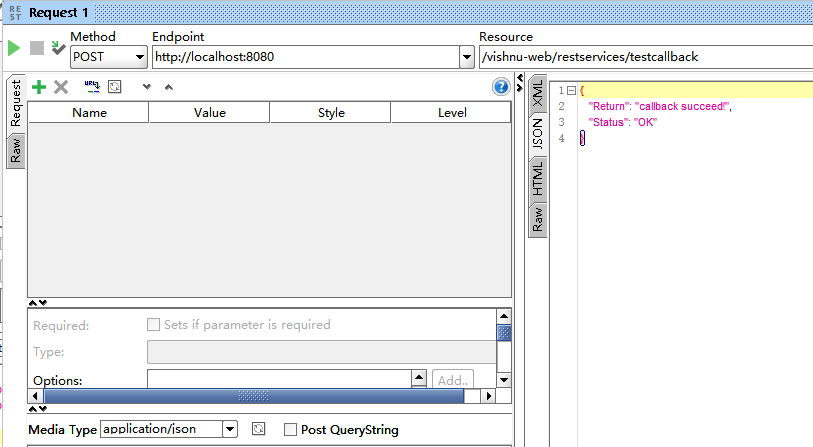
插件是一个java类，并且是一个Spring bean，需要继承RestfulPluginBase抽象类，并且需要添加Annotation @Component或@Service以便在启动过程中能被Spring自动加载。



红框部分的注解会自动生成帮助文档，如下图所示：



调用后的返回值如下图所示：



# restful接口

## 回调接口：jobcallback

如果某些原子操作如果耗时太长，可以通过异步的方式发起调用，执行完毕后需要回调编排引擎让编排可以可以继续进行。

访问地址：http://xxx/vishnu/restservices/jobcallback

帮助地址：http://xxx/vishnu/restservices/jobcallback/help

参数范例：{"jobStepNodeId":1,"status":"succeed","stdout":"正常日志内容","stderr":"错误日志内容","return":[{"key":"ip","value":1,”group”:”group1”,”host”:”10.10.10.10”},{"key":"ip","value":2,”group”:”group2”,”host”:”11.11.11.11”}]}

## 取锁接口：getlock

某些原子操作如果需要竞争独占资源，开始实际操作前可以先通过此接口获取锁。只要给出任意json格式的参数，系统会根据参数返回一个uuid。如果相同的参数已经生成过锁并且没有释放，则会收到锁已存在的报错。

访问地址：http://xxx/vishnu/restservices/getlock

帮助地址：http://xxx/vishnu/restservices/getlock/help

参数范例：json格式的任意参数

## 释放锁接口：releaselock

提供getlock接口返回的uuid，系统会释放对应的锁。

访问地址：http://xxx/vishnu/restservices/releaselock

帮助地址：http://xxx/vishnu/restservices/releaselock/help

参数范例：{“uuid”:”xxx-xxx-xxx-xxx-xxx”}

# 数据库设计

## 数据字典

load\_module：模块表，此表在系统启动时会自动更新

|  |  |  |
| --- | --- | --- |
| 列名 | 数据类型 | 说明 |
| id | bigint(22) | id |
| name | varchar(100) | 名称 |
| description | varchar(255) | 描述 |
| status | tinyint(1) | 0:未加载，1:加载成功，-1:加载失败 |
| error | text | 异常信息 |
| is\_active | int(1) | 0禁用；1激活 |
| init\_time | timestamp | 加载时间 |
| startup | int(2) | 启动顺序 |

plugin：插件表，此表在系统启动时会自动更新

|  |  |  |
| --- | --- | --- |
| 列名 | 数据类型 | 说明 |
| name | varchar(100) | 名字 |
| description | varchar(200) | 描述 |
| status | tinyint(1) | 0:未加载，1:已加载 |
| is\_active | tinyint(1) | 是否激活 |
| init\_time | timestamp | 加载时间 |
| error | text |  |

action：操作表

|  |  |  |
| --- | --- | --- |
| 列名 | 数据类型 | 说明 |
| id | bigint(18) | 主键 |
| name | varchar(50) | 操作名称 |
| description | varchar(200) | 操作描述 |
| tags | varchar(200) | 操作标签 |
| exec\_plugin | enum('chef','ssh','ansible','local') | 对应proxy的插件名 |
| plugin\_class | varchar(255) | 插件类 |
| category\_code | varchar(32) | 操作分类编码，通用操作/云环境/容器等等 |
| is\_parallel | tinyint(1) | 是否可并行 |
| timeout | bigint(20) | 超时时间（单位：分钟） |
| created\_by | varchar(20) | 新增人 |
| created\_date | timestamp | 新增时间 |
| updated\_by | varchar(20) | 修改人 |
| updated\_date | timestamp | 修改时间 |

ation\_param：操作参数表

|  |  |  |
| --- | --- | --- |
| 列名 | 数据类型 | 说明 |
| id | bigint(18) | 主键 |
| action\_id | bigint(18) | 操作id |
| key | varchar(2000) | 参数名称 |
| description | varchar(200) | 参数描述 |
| type | enum('text','json','number','node','proxy') | 参数类型 |
| value | varchar(1024) | 参数默认值 |
| mode | enum('input','output') | 参数模式 |
| is\_required | tinyint(1) | 是否必填项 |
| sort\_num | smallint(6) | 顺序 |
| created\_by | varchar(50) | 新增人 |
| created\_date | timestamp | 新增时间 |
| updated\_by | varchar(50) | 修改人 |
| updated\_date | timestamp | 修改时间 |

arrange：编排表

|  |  |  |
| --- | --- | --- |
| 列名 | 数据类型 | 说明 |
| id | bigint(18) | 主键 |
| name | varchar(50) | 编排名称 |
| fail\_policy | enum('break','continue') | 失败策略 |
| exec\_type | enum('at','mt') | 执行方式，自动或手动 |
| exec\_rule | varchar(50) | 定时策略（cron表达式） |
| description | varchar(200) | 编排描述 |
| created\_by | varchar(50) | 新增人 |
| created\_date | timestamp | 新增时间 |
| updated\_by | varchar(50) | 修改人 |
| updated\_date | timestamp | 修改时间 |

arrange\_step：编排步骤表

|  |  |  |
| --- | --- | --- |
| 列名 | 数据类型 | 说明 |
| id | bigint(18) | 主键 |
| arrange\_id | bigint(18) | 编排ID |
| name | varchar(50) | 名称 |
| action\_id | bigint(18) | 操作ID |
| proxy\_id | bigint(18) | 执行代理ID |
| sort\_num | smallint(6) | 执行顺序 |
| need\_confirm | tinyint(4) | 是否完成后确认 |
| thread\_count | int(11) | 线程数 |
| created\_by | varchar(50) | 新增人 |
| created\_date | timestamp | 新增时间 |
| updated\_by | varchar(50) | 修改人 |
| updated\_date | timestamp | 修改时间 |

arrange\_param：编排参数表

|  |  |  |
| --- | --- | --- |
| 列名 | 数据类型 | 说明 |
| id | bigint(18) | 主键 |
| arrange\_id | bigint(18) | 编排ID |
| key | varchar(50) | 参数名称 |
| description | varchar(200) | 参数描述 |
| type | enum('text','number','node','proxy') | 参数类型 |
| value | varchar(200) | 参数默认值 |
| is\_required | tinyint(1) | 是否必填项 |
| sort\_num | smallint(6) | 顺序 |
| created\_by | varchar(50) | 新增人 |
| created\_date | timestamp | 新增时间 |
| updated\_by | varchar(50) | 修改人 |
| updated\_date | timestamp | 修改时间 |

arrange\_step\_param：编排步骤参数表

|  |  |  |
| --- | --- | --- |
| 列名 | 数据类型 | 说明 |
| id | bigint(18) | 主键 |
| action\_param\_id | bigint(18) | action\_param ID |
| step\_id | bigint(18) | 步骤主键ID |
| input\_type | enum('const','output','arrange','') | 输入方式 |
| value | varchar(200) | 参数默认值 |
| created\_by | varchar(50) | 新增人 |
| created\_date | timestamp | 新增时间 |
| updated\_by | varchar(50) | 修改人 |
| updated\_date | timestamp | 修改时间 |

jobtask：作业任务表

|  |  |  |
| --- | --- | --- |
| 列名 | 数据类型 | 说明 |
| id | bigint(20) | 主键 |
| start\_time | timestamp | 开始时间 |
| end\_time | timestamp | 结束时间 |

jobtask\_job：作业任务和作业的关系表

|  |  |  |
| --- | --- | --- |
| 列名 | 数据类型 | 说明 |
| jobtask\_id | bigint(20) |  |
| job\_id | bigint(20) |  |
| sort\_num | int(4) |  |

jobtask\_job\_param：作业任务参数表（此表存放作业任务下所有作业的所有步骤节点的返回值）

|  |  |  |
| --- | --- | --- |
| 列名 | 数据类型 | 说明 |
| id | bigint(18) | 主键 |
| jobtask\_id | bigint(18) | 作业任务id |
| job\_id | bigint(18) | 作业id |
| job\_step\_node\_id | bigint(18) | 作业步骤节点id |
| job\_step\_id | bigint(18) | 作业步骤ID |
| node\_id | bigint(18) | 节点ID |
| key | varchar(50) | 参数名称 |
| value | varchar(200) | 参数值 |

job：作业表

|  |  |  |
| --- | --- | --- |
| 列名 | 数据类型 | 说明 |
| id | bigint(18) | 主键 |
| status | enum('pending','running','succeed','failed','aborted') | 作业状态：2-执行成功 3-等待确认 |
| name | varchar(50) | 作业名称 |
| fail\_policy | enum('continue','break') | 失败策略 |
| start\_time | timestamp | 开始时间 |
| end\_time | timestamp | 结束时间 |
| arrange\_id | bigint(18) | 编排ID |
| user\_id | varchar(20) | 执行用户 |
| created\_by | varchar(20) | 新增人 |
| created\_date | timestamp | 新增时间 |
| updated\_by | varchar(20) | 修改人 |
| updated\_date | timestamp | 修改时间 |
| param | text | 一段json，原始输入参数 |
| error | text | 错误信息 |
| is\_running | tinyint(1) | 是否执行中 |

job\_step：作业步骤表

|  |  |  |
| --- | --- | --- |
| 列名 | 数据类型 | 说明 |
| id | bigint(18) | 主键 |
| arrange\_step\_id | bigint(18) | 步骤主键 |
| name | varchar(50) | 任务名称 |
| job\_id | bigint(18) | 作业ID |
| proxy\_id | bigint(18) |  |
| status | enum('pending','running','succeed','failed','aborted') | 任务状态 |
| start\_time | timestamp | 任务开始时间 |
| end\_time | timestamp | 任务结束时间 |
| sort\_num | smallint(6) | 任务执行顺序 |
| need\_confirm | tinyint(1) | 是否需要完成后确认 |
| is\_confirm | tinyint(1) | 是否已确认 |
| action\_id | bigint(18) | 操作ID |
| confirm\_time | timestamp | 确认时间 |
| thread\_count | int(11) | 线程数 |
| plugin\_class | varchar(255) | 插件类 |

job\_step\_param：作业步骤参数表

|  |  |  |
| --- | --- | --- |
| 列名 | 数据类型 | 说明 |
| id | bigint(18) | 主键 |
| job\_step\_id | bigint(18) | 作业参数ID |
| arrange\_step\_param\_id | bigint(18) | 编排参数ID |
| input\_type | enum('const','output','arrange','') | 输入方式 |
| mode | enum('input','output') | 入参或出参 |
| key | varchar(50) | 参数名称 |
| description | varchar(200) | 参数描述 |
| value | varchar(200) | 参数值 |
| type | enum('text','number','node','json','proxy') | 参数类型 |
| sort\_num | smallint(6) | 顺序 |
| is\_required | tinyint(1) | 是否必填 |

job\_step\_node：作业步骤节点表

|  |  |  |
| --- | --- | --- |
| 列名 | 数据类型 | 说明 |
| id | bigint(18) | 主键 |
| job\_step\_id | bigint(18) | 任务ID |
| status | enum('pending','running','succeed','failed','aborted') | 执行状态 |
| node\_id | bigint(18) | 执行主机ID,为0代表没有主机 |
| start\_time | timestamp | 任务开始时间 |
| end\_time | timestamp | 任务结束时间 |
| std\_msg\_path | varchar(255) | 标准日志信息 |
| err\_msg\_path | varchar(255) | 错误日志信息 |

job\_step\_node\_param：作业步骤节点参数表（此表只存放作业步骤节点的返回参数）

|  |  |  |
| --- | --- | --- |
| 列名 | 数据类型 | 说明 |
| id | bigint(18) | 主键 |
| job\_step\_node\_id | bigint(18) | 作业步骤节点ID |
| job\_step\_id | bigint(18) | 作业步骤ID |
| node\_id | bigint(18) | 节点ID |
| key | varchar(50) | 参数名称 |
| value | varchar(200) | 参数值 |

node：节点表

|  |  |  |
| --- | --- | --- |
| 列名 | 数据类型 | 说明 |
| id | bigint(18) | 主键 |
| host | varchar(255) | 主机IP |
| type | varchar(50) | 资源类型（从基础信息表） |
| name | varchar(200) | 主机名称 |
| tags | varchar(200) | 标签 |
| res\_info | text | 资源信息 |
| created\_by | varchar(50) | 新增人 |
| created\_date | timestamp | 新增时间 |
| updated\_by | varchar(50) | 修改人 |
| updated\_date | timestamp | 修改时间 |

node\_credential：节点用户密码表（密码字段不能存放密码明文，需要用AES加密）

|  |  |  |
| --- | --- | --- |
| 列名 | 数据类型 | 说明 |
| node\_id | bigint(20) | 节点id |
| username | varchar(255) | 用户名 |
| password | varchar(500) | 密码 |

restful\_plugin：restful接口插件表

|  |  |  |
| --- | --- | --- |
| 列名 | 数据类型 | 说明 |
| id | varchar(255) |  |
| name | varchar(50) | 名称 |
| config | varchar(4000) | 配置 |

restful\_interface：restful接口实例表

|  |  |  |
| --- | --- | --- |
| 列名 | 数据类型 | 说明 |
| id | int(11) |  |
| name | varchar(50) | 名称 |
| plugin\_id | varchar(255) | restful接口插件id |
| username | varchar(50) | 用户名 |
| password | varchar(50) | 密码 |
| config | varchar(4000) | json格式 |
| is\_active | tinyint(4) | 是否激活 |
| token | varchar(50) | 接口实例名称 |
| expire | timestamp | 使用期限 |
| description | varchar(4000) | 说明 |
| authtype | enum('basic','hmac-sha1','cookie','') | 认证方式 |
| timeout | int(4) | 访问重放时间，单位秒 |

schedule\_jobclass：定时调度插件表

|  |  |  |
| --- | --- | --- |
| 列名 | 数据类型 | 说明 |
| id | int(4) | id |
| name | varchar(100) | 名称 |
| classpath | varchar(255) | 类名 |

schedule\_job\_prop：定时调度插件属性表：

|  |  |  |
| --- | --- | --- |
| 列名 | 数据类型 | 说明 |
| job\_id | bigint(20) | 引用schedule\_job的id |
| prop\_name | varchar(100) | 属性名称 |
| prop\_value | varchar(200) | 属性值 |
| id | bigint(20) | id |

schedule\_job：定时调度作业表

|  |  |  |
| --- | --- | --- |
| 列名 | 数据类型 | 说明 |
| id | bigint(20) | id |
| name | varchar(100) | 名称 |
| class\_id | int(4) | 引用schedule\_jobclass的id |
| start\_time | timestamp | 开始时间 |
| end\_time | timestamp | 结束时间 |
| repeat | int(11) | 重复次数 |
| cron | varchar(100) | corn表达式 |
| interval | int(11) | 间隔时间，单位是秒 |
| status | int(1) | 1:运行中;0:暂停;-1:停止;-2未加载 |
| nextfire\_time | timestamp | 下一次被唤醒时间 |
| pause\_time | timestamp | 暂停时间 |
| lastfire\_time | timestamp | 最后一次被唤醒时间 |
| lastfinish\_time | timestamp | 最后一次完成时间 |
| exec\_count | int(11) | 执行次数 |
| is\_active | int(1) | 0:禁用，1：激活 |
| server\_id | int(1) | server\_id，一台主机一个serverid |