# 资源申请参考文档

目录

[资源申请参考文档 1](#_Toc480448444)

[一：背景知识： 2](#_Toc480448445)

[1：优云相关知识了解 2](#_Toc480448446)

[2：编排 2](#_Toc480448447)

[3：作业job 3](#_Toc480448448)

[4：chef自动化管理脚本工具 4](#_Toc480448449)

[5：插件 4](#_Toc480448450)

[6：proxy代理 5](#_Toc480448451)

[二：编码流程图 5](#_Toc480448452)

[三：编码细节和实现 6](#_Toc480448453)

[1：根据chef的自动化任务步骤组装chef可以识别的json文件，在本地生成。 7](#_Toc480448454)

[2：组装proxy要去执行的命令(执行主机ip,command, user,pwd) 8](#_Toc480448455)

[3：调用具体的执行器执行命令，返回结果并生成日志 9](#_Toc480448456)

[4：如果需要录入cmdb的就录入cmdb数据 9](#_Toc480448457)

[5：关于多个执行主机执行同一条命令的处理方法： 9](#_Toc480448458)

## 一：背景知识：

我将尽量用简单的语言给开发传递新框架涉及到的所有知识，其实是对顾问陈其炜写的《编排框架设计文档.docx》文档的一些补充。

维石的目的是在于将运维人员日常的运维任务由手动转为自动，所以核心就是调用运维人员日常在系统上操作的所有命令组装成的脚本，自动执行运维人员的配置管理等命令。比如一个mysql创建任务，正常流程是一个运维人员去找到一个主机和ip地址，分配硬盘资源，CPU和内存资源，创建主备库，建立数据库实例，授权，及建立成功后写入cmdb数据库等一系列操作组成，如果维石能够将所有的这些步骤的所有命令全部集合起来，一步一步执行，完成全流程的申请过程，那运维人员的工作就能全部实现了。

其中找主机，ip地址，分配资源的过程就是资源管理和资源分配，当然还有后期的容量管理，库存管理，而这些数据对于维石而言，主要有3种来源，IAMS（架构图系统），CMDB系统，BALANT性能监控平台，对于每个系统，每种资源如何取，存在一个最优化的计算过程，也包含了性能数据的获取-从balant等系统获取，这就是分配规则。对于任何一种资源，分配规则都是不一样的。

每个步骤的输入输出不一样，参数不一样，就需要实现不同的操作功能。当然对于同样的操作，如果只是参数的值不一样，其实就是同一种操作。

### 1：优云相关知识了解

由于新框架非第三方开源的传统开发框架，顾问借鉴了优云的设计思路，没有了解优云背景的可能不太了解这个框架的核心引擎-编排引擎到底是实现了什么功能。

给出网址如下https://automation.uyun.cn/#/?\_k=lhobml，账号gdnpc@163.com和密码220350，有空的抽时间去看一下，对这个框架就能有个大概的印象。

开发团队给出的内部部署网址如下10.202.125.217:8080/vishnu-web/，由于集成了CAS平台，所以用工号即可登录。这个上面部署的比较过时，但对了解编排是有辅助作用的。

### 2：编排

优云的核心功能是编排，编排顾名思义是编辑排序，即将一项任务分成多个步骤来执行，同时还可以排序，即调整顺序，是自由度相对比较高的一种设计。

我们维石的框架核心就是编排，编排即任务的一种定义方式，它有多个步骤，每个步骤都有很多参数，而每个步骤即是一个操作action，背景知识介绍部分讲到了mysql的创建过程，按照编排的做法来做，其实一个mysql申请过程就是一个编排，对应一个编排ID，这个编排分为多步，每一步去调用不同的脚本执行不同的操作。

所以编排是这样设计的，有个arrange表，定义了编排ID，创建人，创建时间，失败策略，定时，执行代理等等信息。而这个编排ID有很多步骤，所以有个arrange\_step表，这个arrange\_step步骤表，应当有自己的输入输出参数，所以有个arrange\_step\_param表，用于放置每个步骤的输入输出参数的具体值。还有个arrange\_param表，用于存放这个编排全局的参数，比如环境参数env，设定值为sit就不能同时设置为prd，即一个任务不可能在测试环境执行一些步骤又跑到生产环境去执行。当然也可以不需要全局参数，将所有参数都丢到步骤参数表去也是可以的。全看你设置的粒度有多小。

如果每个步骤，我们都去一步一步的设置步骤参数，那这个会特别繁琐，于是对于每个arrange\_step,我们可以把它的参数对应放到一个操作里面去，有个action表，与arrange\_step一一对应，那么对应的action\_param表，就与arrange\_step\_param一一对应，为什么这么做，比如mysql主备库，搭建他们时，很多步骤是重复的，如果写步骤，同样的编排步骤参数每次都去配，是没必要的，那么抽象出每个arrange\_step来，设置对应的action，就可以实现功能复用。

讲到这里，那大家应该就知道了，action\_param参数其实只需要定义key就可以了，value在编排表中一定会被修改，即action其实都是一些模板。

接下来说下参数类型，当前编排支持的参数类型有节点类型和文本类型，还有json类型(String或text，json)，节点类型非常重要，节点其实是对执行主机的一种抽象，查看node表就知道一个节点应该有ID，主机的ip地址，uuid，登录它的用户名和密码等等一系列信息。

维石的框架很牛的地方在于，它可以根据节点来执行操作，比如在前台页面，配置了一个mysql将在4台主机（节点参数）上执行，那框架就获取到4个节点类型的ip，分别在一个步骤中，将这个操作分给4个节点分别去执行。当然这个需要在插件代码中去实现的。后面讲具体的代码实现的时候会告诉大家如何按主机进行分组去执行。

这个更深入的信息，可以查看《编排框架设计文档》的第4部分流程图。

很多人理解编排也比较难以理解，但是还有更复杂的设计，就是参数类型了，对于action\_param表，即每个操作的操作，其实是对某一个action操作的输入输出参数的存储，对于一个操作，肯定是需要指明它是输入还是输出参数，所以action\_param表的每个参数，都有一个输入input输出output类型的mode（参数模式）字段标记。

对于action的操作，每个操作的输入和输出都是独立的，是无关的，但是对于具体的编排步骤，上一步和下一步其实是有参数的输入输出依赖关系的，比如mysql第一步申请的ip资源，肯定要放到第二步去使用。所以对于编排的每个步骤，步骤在引用action\_param的arrange\_step\_param表中，每个参数应该知道自己的参数是来自哪的，是否是来自上一步的输出呢，还是不依赖于上一步，或者就是从全局参数那边取来的。于是arrange\_step\_param就有个input\_type字段，分别叫const，表示不依赖于上一步，是常量，有个叫output输出变量，他来自于上x步，还有个arrange参数，表示来自于编排参数，是全局参数。

对于output输出参数，他的value可以定义为x.key，表示前面第x步的key参数。框架会自动取到该参数，但是需要在它的上一步设置该输出参数，不设置输出，就算编排定义了，也是取不到该参数的。需要编码。

**PS：从文档写出来后，有很多人问我为什么表单方式提交资源申请时，各步骤的输入和输出参数在paramlist取不到，其实是因为1是执行主机的IP没有设置成节点参数， 2是因为在action\_param表中，有些输出的参数没有设置为output类型，3是因为编排步骤参数表arrange\_step\_param和action\_param表，和表单下发的参数key，这3者没有同名，任何一个设置出错都会导致数据取不到或者有问题。**

**一定要注意所有的参数的命名的前后一致性。**

### 3：作业job

我们创建编排之后，接下来来肯定要运行它，不然编排就是一个任务的定义，而不是一个实际意义上的任务。创建编排的时候，后台会将所有的数据存储到上面action和arrange等6张表中。

点击编排的执行按钮，框架实际上是先查到上次存储的编排ID对应的6个表的数据，并全部拷贝到jobvo后，再创建这个job，同时也会接受执行时页面上新传下来的的参数。这些新传下来的参数的key一定是和编排步骤参数里面的参数的key同名的，否则是无效参数，不参与到接下来的任务运算中去。

具体可以参考jobController.java中的runArrange方法，(使用ctrl+shift+r快捷键查找这个文件即可搜索到).想要了解细节的，跟一下代码即可，另外可以参考下炜哥的文档。

Job才是真正执行的任务，所以编排就算配好了，你也可以在执行前先编辑，修改了value后再去执行，因此编排，和操作的参数值和job的实际运行值不一样是很有可能的。

实际上编排还有很多特性，细节需要了解，但是由于这个文档定位于资源申请流程，因此不需要知道的就不多写了。

### 4：chef自动化管理脚本工具

在上面我讲到了自动化脚本（自动化脚本其实就是专业组的原子服务这个概念），比如创建db就是一个原子服务，其实对于开发来说，就是调用一个自动化脚本。

但是对于脚本，其实除了少数脚本，大部分其实也是有输入参数的，不然怎么运行，而这些脚本的输入参数，其实就是来自于前端页面的输入参数最后传递给脚本了。

但各种各样的脚本，光定义名字，输入，管理，运行，全丢在一起肯定不好管理，于是chef这种自动化管理脚本工具就出现了。Chef就是管理脚本的工具，它分为chef-server, chef-client, workstation,其实李梦迪写了个ppt讲chef的，我按我的理解先讲一下，chef单词意思是大厨，对于要运行的脚本，它是通过run\_list去管理的，它有个cookbook（菜谱），recipe（菜单），理解一下，以mysql为例，对于cookbook，其实就是一个mysql的文件夹，而recipe就是mysql文件夹下一堆的ruby脚本，如 mysql\_vmount.rb, db\_create.rb，一个run\_list，就是cookbook::repice 这样的一个组合，它会去mysql目录下，找到对应的脚本去。这就是我理解的run\_list。

以一个步骤为例，它的执行命令串是这样的：

chef-client –r run\_list –j

<http://ip:port/dir/taskid-cookbook-recipe-uuidOfExecuteHost-uuidOfRandom.json>

run\_list不用解释了，后面的json文件的ip表示这个json放置在哪个ip主机上（对于当前的维石系统，就是维石部署的生产环境的那个vip地址- 这里做了haproxy），端口port和dir路径（国峰给定的，使用NAS同时挂载到vishnu-web应用主机和chef-server的一个公共路径）是在哪，taskid表示任务id（jobid），cookbook和recipe不用多解释了，就是这个任务要运行哪个ruby脚本（这个开发在配置编排的时候，就应当知道），uuidOfExcecuteHost表示执行主机的uuid，而后面的uuidOfRandom表示随机生成的一个uuid字符串，这些就是为了确保该文件唯一并且可以定位到是哪个任务哪台主机上执行的哪个json文件。

Chef-server，当前维石生产环境平台只用一套(10.202.10.10-sit，10.202.235.8-stg)，chef-client这个agent客户端要在具体的执行主机上去执行，那就必须在这台执行主机上部署chef-client,否则就无法执行chef命令，就好像要执行一个ssh命令一样，前提是这个主机安装了ssh命令，同时，chef-client还必须在chef-server上注册，否则chef-client无法获取到run\_list对应的ruby脚本，自然就无法处理后面的json文件了。

所有的这些ruby脚本，即run\_list,都是workstartion上传给server，这3者也可以部署在同一台主机上。比如235.8上就是3者部署在同一台上。

另外chef命令是通过proxy代理去远程的执行主机上去执行的，它的执行结果要作为日志返回，所以来回的权限都要开通。

这部分的内容在第四章将展开讲述。

### 5：插件

插件是顾问提出来的一种隔绝资源的解决方案。插件的意思是即插即用，当前也的确做到了，他的做法是使用用户自定义类加载器，在需要的时候加载所需要的类，想要了解更多的自行百度“类加载器”，这里不再展开。稍微提一下，使用类加载器，同级之间不可见，而父类和子类之间，父类可见子类，而子类之间不可见，这就能实现资源的隔离，任何一个插件出问题，也不影响别的插件功能，当然如果在编排过程中，调用了无法正常运行的插件，那对应的这个插件的这个步骤就是失败了。

当前我们的插件是按专业组去划分的。比如vishnu-plugin-storage（存储组），这里将放置NAS,SAN等存储插件，比如vishnu-plugin-network(网络组)会放置DNS，防火墙等插件等等，但也不完全是按专业组去做了，有些插件比较庞大的就会单独划分一个插件出来做一个子项目。

类似的设计思路在维石中还有很多，回调接口调用也插件化了。其他可以参考架构设计文档。

### 6：proxy代理

Proxy代理，也是维石中的一个项目工程，它和维石其他项目工程之间不形成相互依赖关系，所有的输入输出都是通过http和https接口传递。

Proxy的设计在我看来，是为了解决当前维石项目多个安全区域之间通信的问题，当前公司有很多网络区域，如BTN, ISN, DCN, DMZ等等，所有的网络区域，如果都能从维石平台直接访问，无疑会加大网络安全的风险。通过proxy去访问，严格设置proxy的访问权限，就能进一步控制网络安全。

所以proxy当初的设计非常简单，就是维石本身通过http和https连接到proxy，当然有用户名和密码和认证方式(支持basic，hmac，还支持cookie，不过这种属于高危漏洞的就不会使用了)。

维石要去执行主机上执行，那按照上面4中的chef命令来说，他需要通过proxy登录到执行主机上去执行命令，所以不但要传递给proxy登录命令所需的用户名和密码，还需要传递命令所需的参数，如命令串和json文件所在的主机等信息。

## 二：编码流程图

有了上述背景知识，再来看一个资源申请需要做哪些事了。



## 三：编码细节和实现

1.以防火墙为例，首先你要创建防火墙的一个编排模板含编排id，并且不能修改和删除（通过设置arrange表和action表的is\_lock字段锁定，如果为0就锁定，为1就不锁定），当页面输入参数并提交资源申请时，要编写表单的申请后台处理接口，即首先要写一个controller，更新对应编排id的编排数据，存储申请清单信息，并调用job创建接口，开始创建编排任务。

@RequestMapping(value = "saveFirewall/{arrangeId}", method = RequestMethod.**POST**)

@Description("保存防火墙申请")

public ResultMessage saveFirewall(@PathVariable Long arrangeId, @RequestBody FirewallVo firewallVo) {

try {

JSONObject **obj** = firewallService.saveFirewall(firewallVo);

ArrangeVo **arrangeVo** = arrangeService.getArragentById(arrangeId);

JobVo **jobVo** = new JobVo(arrangeVo);

if (obj == null) {

obj = new JSONObject();

}

jobVo.setParam(obj.toString(4));

Long **jobId** = JobManager.run(jobVo);

return new ResultMessage(ResultMessage.**Success**, String.valueOf(jobId));

} catch (Exception **e**) {

logger.error(e.getMessage(), e);

return new ResultMessage(ResultMessage.**Fail**, e.getMessage());

}

}

这里解释一下，由于不确定需求方到底要不要读取本地的数据，廖洋的防火墙设置一个表存储防火墙@RequestBody FirewallVo firewallVo，实际上也可以用JSONOBject paramObj去将页面的参数全部取下来，但是需要注意的是，表单申请方式的paramObj,会丢到编排参数类型去，它在job的运行过程中，会替换掉来自jobVo中来自arrange\_step\_param中的参数， 所以，paramObj必须要有编排中定义的key同名的参数，如果不同名，或者漏掉一些参数，那实际运行中就会取不到值。

2：从创建该job任务开始，其实job如何跑，开发不用关心，只需要去实现具体的业务了，对于具体申请资源的job的每个步骤，其实主要是资源分配和插件实现（自动化任务-也叫原子服务实现），资源分配我没写过，后面让秋浩等补充。

对于job本身，它创建后，就会立即返回任务ID，同时创建多个进程去启动各个步骤job去执行任务。而对于这些任务ID，可以重做，具体参考阿炜的文档。

插件实现：（其实应该叫做插件的原子服务类实现-下面仍然以插件代替）

插件上面的背景知识已经介绍过了，对于每个插件，在各自的插件子项目中的plugin对应package下，创建一个继承pluginBase类的具体插件，应用在启动的时候会扫描所有继承PluginBase的类，并加载到系统的plugin表中，（***当前其实已经发现很多插件都是很相似，后面讲考虑抽象出来4个以内的插件，实现所有的功能***）。

getDescription方法返回插件的描述，前端页面加载时可见（有些开发没有写description，在前端页面显示就为空了）。主要要实现的handle方法。

实现了handle方法，那插件就算完成了，插件的主要工作在于设置输入和输出参数，而开发的主要工作在于联调和验证。后续有时间对公共部分再去抽象和完善。

**protected** **void** handle(JobVo jobVo, JobStepVo jobStepVo, JobStepNodeVo jobStepNodeVo, ResultVo resultVo,

List<ParamVo> paramList)

对于插件要实现的handle方法，它的JobVo，包含了整个任务所有的参数信息，jobStepVo，包含了这一步骤的所有信息，jobStepNodeVo,包含了该步骤的运行节点信息，ResultVo只是用来返回数据，输出参数。而List<ParamVo> paramList则包含了所有的数据，也包含了上一步的输入数据。

首先我们说明白了，每个继承pluginBase类的都是一个原子服务的实现类。在chapter二中，我已说明插件中要做的4项工作。

1：根据chef的自动化任务步骤组装json文件，生成在本地

2：组装proxy要去执行的命令(执行主机ip,command) 不需要传递用户名和密码，后面proxy封装了，传了也没事。

3：调用具体的执行器执行命令，返回结果并生成日志

4：如果需要录入cmdb的就录入cmdb数据

接下来分别作说明：

### 1：根据chef的自动化任务步骤组装chef可以识别的json文件，在本地生成。

首先大家要明白，一个资源申请的自动化任务，是分为多个步骤的，每个步骤对应一个原子服务，而这个原子服务是通过chef来调用的，那就首先要去找到chef的这个run\_list对应的json文件是什么结构的，只有找到这个结构，我们在原子服务实现类中实现并传递给chef-client, chef才能用这个json作为输入参数传给run\_list去执行。

因此第一步是找到这个步骤对应的json文件是哪个，根据我们上面讲的这个json文件的命名规则，我们只要知道run\_list对应的文件，就知道对应的json文件结构了。

在10.202.235.38（对应vip是235.8）主机上（用户名root，密码sf123456）下有个目录/app/ngftp/omatp/下，使用命令find ./ -name ‘\*dns\*’ 这个命令，就可以找到dns这个run\_list相关的json文件，其实用run\_list的一半来做搜索是更准确的，提供给开发的所有的需求规格说明书应当说明run\_list是哪个。

如果连run\_list都没提供的，那肯定是原子服务都没写好的，或未提供出来的，当然235.38(对应vip是235.8)上不是所有的run\_list对应的json文件都有，有些比如NAS等新增的原子服务，就没有，必须由专业组对应需求人员提供。

比如文件：

1468983666551-vip-add-4c4c4544-0054-5710-8044-b4c04f463358d44be09f0fef4267a244ffed93bf0495.json

打开里面的内容如下：

{"vip":{"S\_vip\_list":"10.202.44.235"},"task":{"id":"1468983666551"}}

比如文件：

1468998246594-mysql-createslave-4c4c4544-0054-5710-8044-b4c04f46335808f24b903bbd45f8a17250cf14f5f1dc.json

打开里面的内容如下：

{"task":{"id":"1468998246594"},"mysql":{"S\_pri\_vip":"10.202.44.234","S\_my\_serverid":"2","S\_my\_buffer":"4G","S\_my\_version":"5.6.21","S\_my\_sid":"zhouxin","S\_stg\_vip":"10.202.44.235"}}

可以看到，json文件的格式肯定有一个task，里面的id就是taskid(***这里一定要是具体的taskId，不能用系统时间戳做taskid，因为如果这个任务失败了，后面可以根据jobid重跑该任务***)，再然后就是具体的业务字段了（这整个业务字段，我命名为attributes字段），具体的业务字段，根据具体的业务不同，内容和字段也各不相同，因此，我写了一个通用的方法。只要开发人员在页面的每个编排步骤对应的action配置一个类型为json的参数，key为attributes，value为具体的结构，比如上面这个这样写：attributes:

"mysql":{"S\_pri\_vip":"$S\_priv\_vip$","S\_my\_serverid":"$S\_my\_serverid$","S\_my\_buffer":"$ S\_my\_buffer$","S\_my\_version":"$S\_my\_version$","S\_my\_sid":"$S\_my\_sid$","S\_stg\_vip":"S\_stg\_vip"},

其他参数在页面上也配好：如“S\_priv\_vip”:”10.202.44.234”, "S\_my\_serverid":"2”…， 即外面的参数和attributes里面的$key$符号相对应，那么在arrange项目下使用HandleData的replaceParam方法，该方法就会将$key$里面的key参数表示的值全部替换成外面的参数的值。当然前提是用户在前面编排页面配置的参数全是key，value形式，且能对应的上，不然是无法替换的。

举例：

//attrfromweb :　“dns”:{"M\_dnsinfo" : "$surfix\_domain$,$domain$,$ip$"} 这个即页面的配置json结构数据

attrfromweb = HandleData.replaceParam(attrfromweb, paramList);

//attrfromweb替换之后，他的值就全取到了。

接下来要生成文件：上面的属性字段我们取到了，接下来是去组装数据。

JSONObject attrjson= JSONObject.fromJsonObject(attrfromweb);

JSONObject filejson = new JSONObject();

filejson.put(“task”,”{\“id\”,…}”);

filejson.put("attibuttes", attributes);

FileUtil.createChefFile(String taskid, jSONObject json) //这个文件在arrange工程下的util下，去创建文件的时候，需要传给taskid，ip，run\_list和attributes字段。

生成的文件路径，是在vishnu-web的config.properties的chef.file.path定义的目录下，

这样，第一步就算完成了。

### 2：组装proxy要去执行的命令(执行主机ip,command) ps:这里用户名和密码去掉即可

这一步应该调用ProxyCaller.execute方法，

JSONObject **retjson** = new JSONObject();

try {

retjson = ProxyCaller.execute("winrm", json); ….

这其中，user和pwd字段，其实不用在json中给定了，后面的执行器已经封装好了，只给执行机的主机ip和拼接command字符串即可。

这里command字符串就是sudo /usr/bin/chef-client –r run\_list –j

<http://vishnu-web-ip:8080/dir/xx.json>

需要在前面拼接后组装到json中给proxy发送。

***PS:当前也不需要组装这种了，可以直接传run\_list和filename过来就可以了，其他的执行器会做到***

至于调用哪个proxy执行器，其实可以看下proxy执行器下的各个plugin，其中winrm就dns在用，而ssh2，则是当前阶段所有插件类都在用，ssh2会去调用chef到远程主机去执行。

上面的代码，其实定义了retjson，接受proxy执行结果的日志返回情况。根据日志可以追踪到每个jobStep执行后的日志信息。

### 3：调用具体的执行器执行命令，返回结果并生成日志

// 根据执行结果判断添加正确日志和错误日志。

if (retjson.containsKey(ConstVariables.RIGHT\_LOG)) {

resultVo.setResult(retjson.getString(ConstVariables.RIGHT\_LOG));

} else if (retjson.containsKey(ConstVariables.FALSE\_LOG)) {

resultVo.setError(retjson.getString(ConstVariables.FALSE\_LOG));

}

上述这段代码照抄即可。

写完后，需要在写一下这个步骤默认的执行状态。

// DNS正常执行会比较快，默认设置为成功。

resultVo.setStatus(Status.**SUCCEED**.getValue());

不同的任务执行速率不一样，有些会执行的很慢，状态字段分别有：

**PENDING**("pending", "未开始"), **RUNNING**("running", "运行中"), **SUCCEED**("succeed", "已成功"), **FAILED**("failed", "已失败"), **ABORTED**("aborted", "已中止");

如果是会执行很快的任务，可以默认设置执行成功，否则设置运行中即可，等过段时间通过回调函数来查下该步骤是否执行成功。

比如虚拟机，执行要很久，就设置为RUNNING状态即可。

### 4：如果需要录入cmdb的就录入cmdb数据

这个当前匡磊正在编写通用接口，等他提供了再补充。

### 5：关于多个执行主机（节点）并发执行一个操作，分组处理方法：

对于ip地址类型的参数，如果知道是执行主机的，应当在页面配置编排参数的时候，配置ip类型的参数为节点类型，这样框架会识别出来节点类型，并分发到不同节点（执行主机）去执行各个任务。

举例，比如对于一个分配ip资源的流程来说，第一步分配ip资源，实例名，和其他参数，第二步去这些ip上去执行。那么对于这2步对应的插件来说，应当这样写。

第一步的插件这样写

//以主机host做分组

for(ParamVo p: paramList){

if(p.getGroup().equals(jobStepNodeVo.getHost())){

if(p.getKey().equals ("host")){

resultVo.addParam(new JobStepNodeParamVo("host",p.getValue(),p.getValue()));

}

if(p.getKey().equals ("nasip")){

resultVo.addParam(new JobStepNodeParamVo("nasip",p.getValue(),p.getValue()));

}

if(p.getKey().equals ("env")){

resultVo.addParam(new JobStepNodeParamVo("env",p.getValue(),p.getValue()));

}

}

}

resultVo.setStatus(Status.SUCCEED.getValue());

第二步的插件这样写。

//取出所有参数

for(ParamVo p: paramList){

if(p.getGroup().equals( jobStepNodeVo.getHost()){

if(p.getKey().equals("key")){

String host= p.getValue();

}

//获取其他参数

}

}

//组装数据并执行

另外新增了一个HandleData.replace(String input, List<ParamVo> ParamList, String group)方法，处理当参数分组时，替换参数的方法。具体可以参考李明部分的代码，4.20日下午2点刚和他过了一遍他的代码。

## 四：功能测试：

### 物理部署图

要做测试，必须先了解维石的整体物理部署架构，如下所示：



### 如何测试

对于vishnu-web,当前已在10.202.78.4上部署了一套应用，他通过http接口，访问78.51这台sit环境的proxy，78.51是vip，对应实际的proxy的ip地址分别是(50,52)，在这2台主机上部署proxy应用程序，对于不同的业务，执行主机是不同的，比如防火墙的执行主机就只有2个，生成一个，测试一个，对于DNS，由于只能在windows下环境下运行，也只有2个执行主机。

从proxy需要登录到执行主机去，当前，所有的执行主机都应当配置vishnu账号免密登录到执行主机上去。但是也需要注意，所有的执行主机都必须安装chef-client，并且注册到chef-server中。

在执行主机上，如果执行chef命令，那chef-client（执行主机）必须要从chef-server中获取run\_list,同时获取到刚开始vishnu-web生成的json文件。

由于vishnu-web的/app/ngftp/omatp（实际路径）和chef-server的这个路径的NAS是挂载在一起，所以，通过http:chef-server:port/nasurl 这个地址是可以在vishnu-web的生成路径下，取得生成的文件。执行完成整个命令。

其实对于各位开发，完全可以把vishnu-web,proxy部署到10.202.235.8，因为该台主机上chef-server, chef-client是在同一台机器上都部署了的，所以，只要开通执行机的前后访问权限。那应该是可以直接测试的。