托管Hadoop工作内容交接

## 一、工作交接安排

|  |  |  |
| --- | --- | --- |
| 模块 | 工作内容 | 交接人 |
| 实例管理 | 集群实例的生命周期的管理 | 刘健 |
| 计费相关 | 计费接口以及OpenAPI | 康俊 |
| 监控相关 | 集群状态以及集群节点状态监控 | 李涛 |
| 环境部署相关 | 镜像制作，部署脚本，yum源 | 赵东阳 |

其中，各个模块已有各个交接人负责开发了两个月。大体工作内容也已经分别给每个人讲解完成。

## 二、开发测试发布流程

月初指定本月需求规划 --🡪

jira创建release版本 --🡪

jira创建任务链接release版本 --🡪

开发测试 --🡪

发布到staging环境，更新镜像，测试功能 ---🡪

发版本前2天测试人员测试 ---🡪

发版本前一天产品经理发上线申请 ---🡪

马宗学更新生产环境 ---🡪

测试人员测试通过回复邮件

注意事项：

后端application.yml和前端package.json配置项修改后注意修改部署脚本<http://git.inspur.com/Insight-HD/hdinsight/HDInsight-deploy>

## 三、项目组成

1.后端项目cloudbreak:

Git地址： <http://git.inspur.com/Insight-HD/hdinsight/cloudbreak>

2.功能：

实例的生命周期管理，包括创建、扩容、移除节点、扩容集群和删除集群、绑定解绑EIP

代码模块：

3.Blueprint-manager:

解析处理blueprint json文件，可以动态添加修改blueprint配置项

4.Cloud-api/cloud-template/cloud-openstack:

openstack相关的接口，stack的创建、flavor的查询、vpc的查询

Cloud-reactor/cloud-reactor-api:

反应堆设计模式，比如core模块中调用LaunchStackRequest，会在cloud-reactor的LaunchStackHandler中去处理创建stack的操作。事件的注册是在项目启动的时候CloudPlatformInitializer类中init方法初始化的。

Core/core-api/core-model:

核心代码块，主要是rest API层与业务逻辑处理层。所有的实例的管理操作都是由这初始的。重心是一个实例操作的请求会有状态机的管理

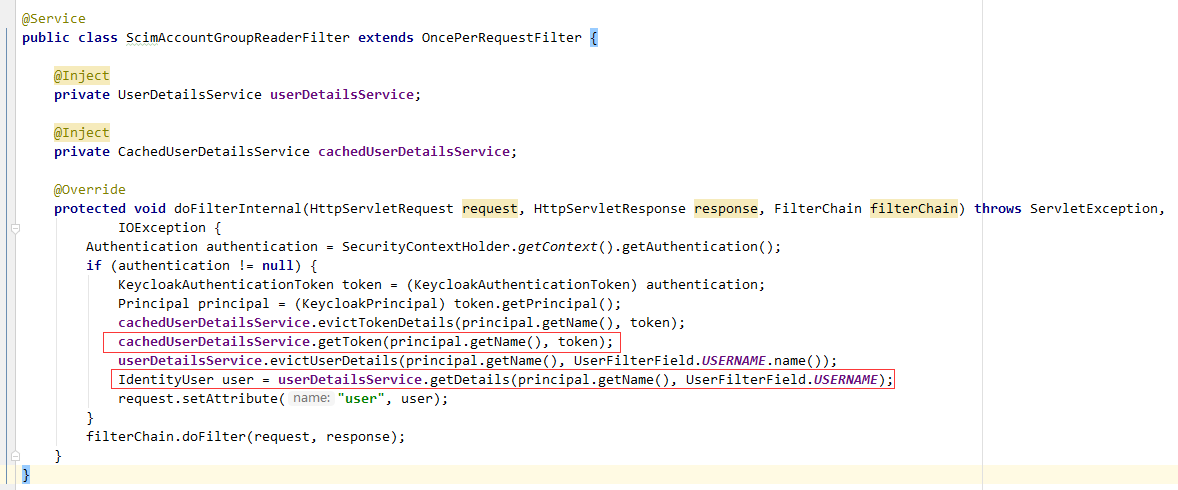
Orchestrator-api/orchestrator-salt:

上层应用组件编排工具saltstack的调用，主要是postgresql、ambari、knox和kerberos以及DNS等的配置

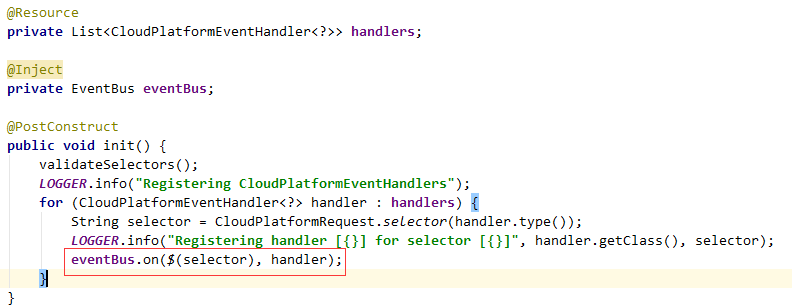
## 四、核心代码讲解

### 项目启动配置

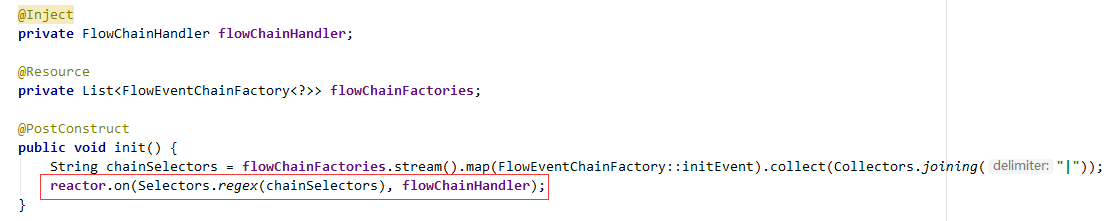
1.SecurityConfig.java 配置keycloak安全以及配置user和token的缓存过滤器ScimAccountGroupReaderFilter



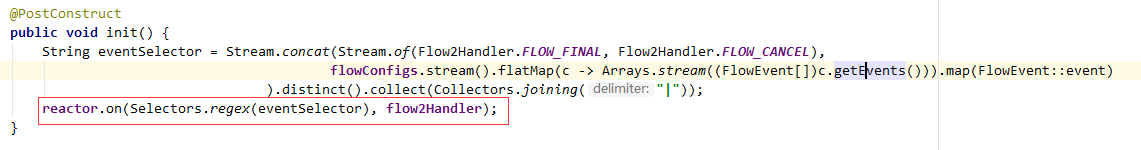
2.CloudPlatformInitializer.java 注册与openstack交互相关的事件和handler到事件总线**eventBus**(事件总线是reactor中的一种异步编程的发布订阅模式，在项目启动时，将相关事件绑定给handler处理器，在代码中调用eventBus.notify(event)发布事件，然后被handler订阅处理)，比如将LaunchStackRequest事件注册给LaunchStackHandler处理器处理，具体逻辑如下：



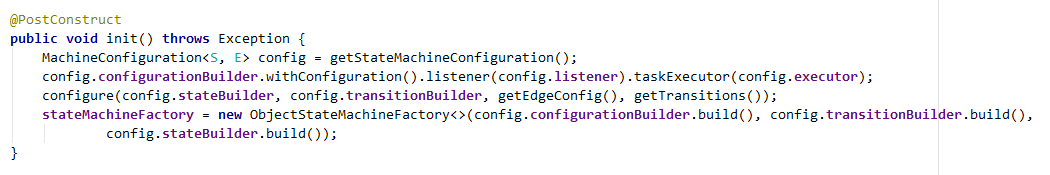
3.FlowChainInitializer.java 注册所有FlowChain的初始化事件到事件总线**eventBus**， 比如集群创建的ProvisionFlowEventChainFactory初始化事件FULL\_PROVISION\_TRIGGER\_EVENT注册给FlowChainHandler，当集群创建时，会发送FULL\_PROVISION\_TRIGGER\_EVENT事件，然后交由FlowChainHandler处理。



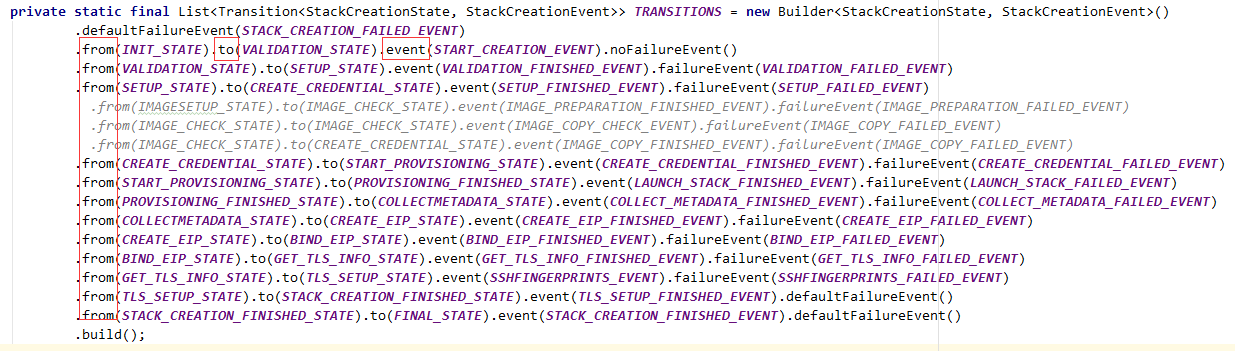
4.Flow2Initializer.java 注册FlowConfiguration的所有FlowEvent事件到事件总线**eventBus**，当发送Flow事件时统一由Flow2Handler处理。



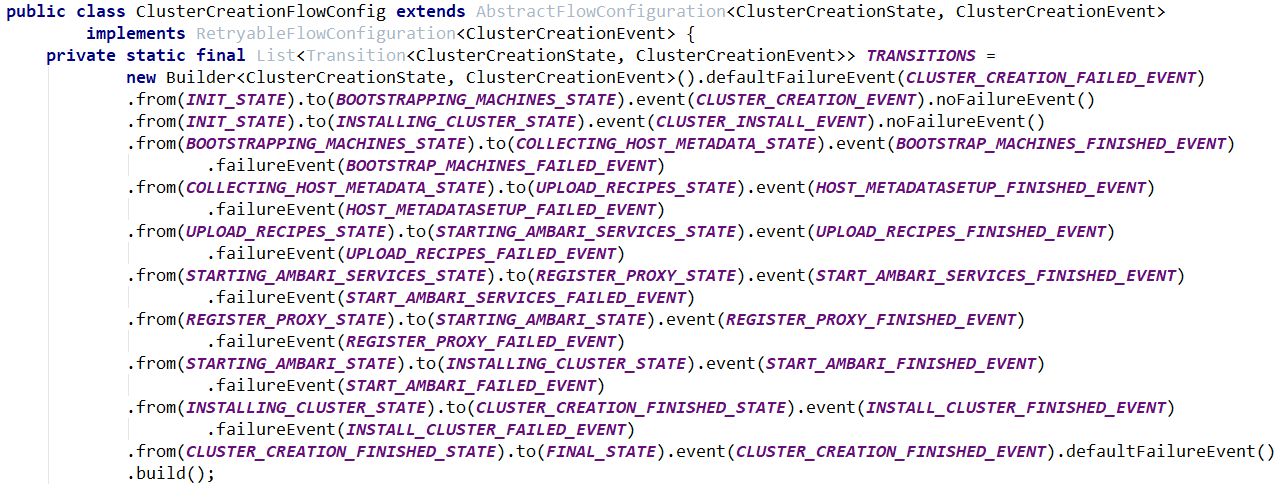
5.AbstractFlowConfiguration.java init方法中配置所有子类的状态机，即



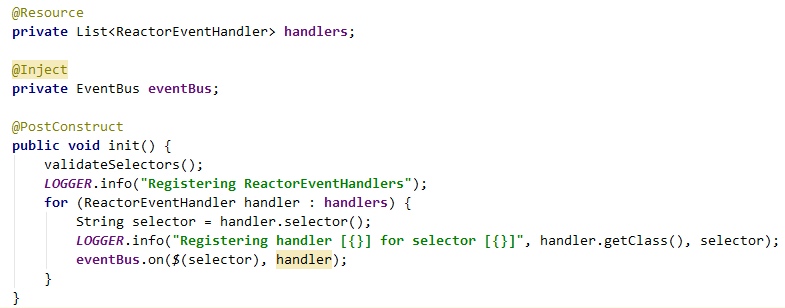
stack创建的Flow配置StackCreationFlowConfig.java



cluster创建的Flow配置ClusterCreationFlowConfig.java



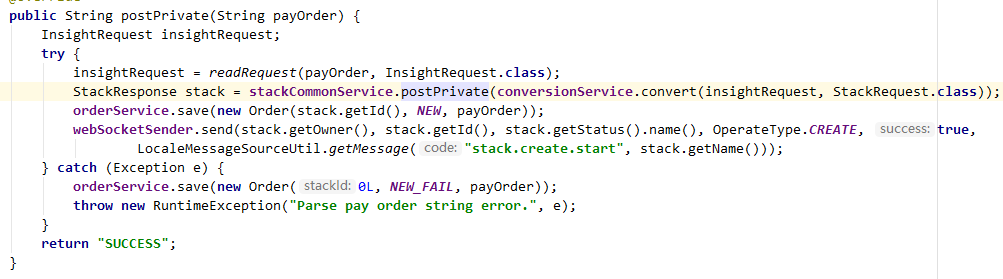
6.ReactorEventHandlerInitializer.java 将与管理节点交互的事件注册到eventBus中



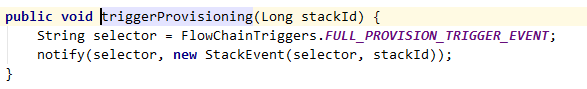
### 五、实例创建过程

1.用户页面选择配置点击确定，跳转到支付页面，支付成功后BSS发送MQ消息

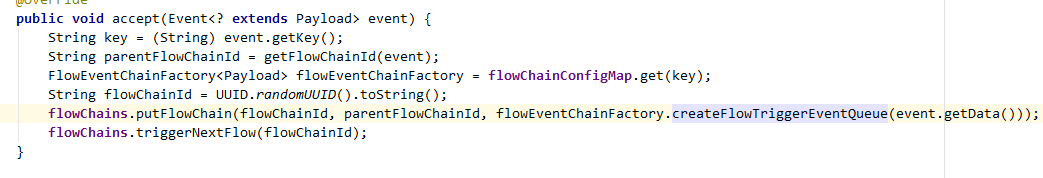
2.Cloudbreak PayOrderListener.java接受mq订单消息，创建数据库元信息（第三步重点讲解），保存订单消息，发送websocket信息



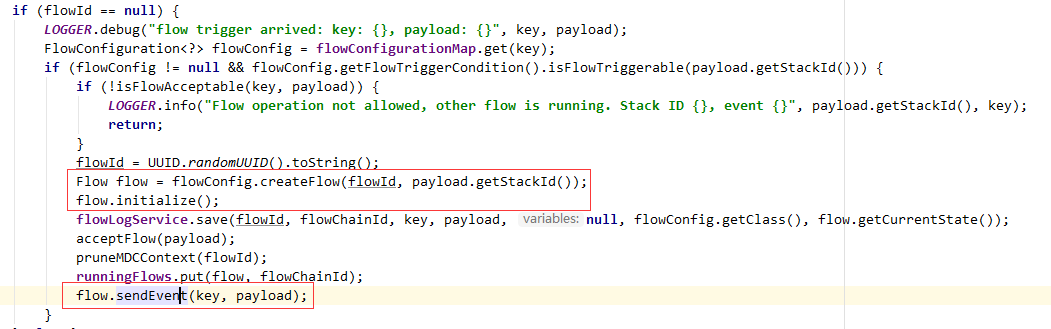
3.关于stack的创建，除了数据库信息的保存，有两个地方需要特别注意。一个是镜像名称和yum源的信息的获取，代码在0.java的187行，逻辑主要是从image-catalog.ftl文件中读取。另一个是Flow机制的运行（flow可以理解为一个工作流，一个工作流对应一个状态机，状态机就是项目启动配置5中预先定义好的状态以及状态之间的转移和动作，一个FlowChain可以对应多个flow，按照先进先出顺序执行），代码调用ReactorFlowManager的triggerProvisioning方法，eventBus发布FULL\_PROVISION\_TRIGGER\_EVENT事件。



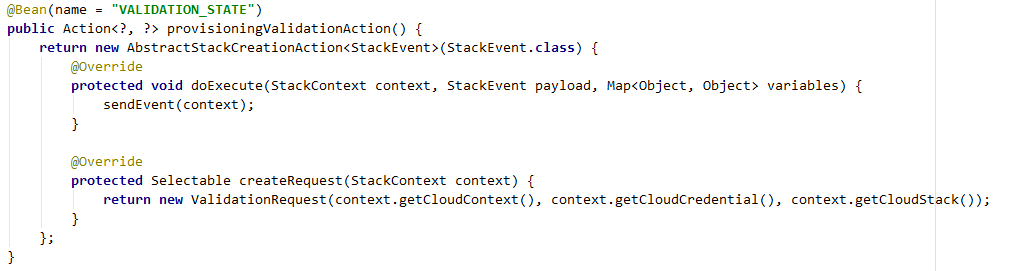
4.根据项目启动配置3中讲解的可知，发送的FULL\_PROVISION\_TRIGGER\_EVENT被FlowChainHandler处理，然后flowChains中放入STACK\_PROVISION\_TRIGGER\_EVENT和CLUSTER\_CREATION\_EVENT两个事件，然后triggerNextFlow中eventBus发送STACK\_PROVISION\_TRIGGER\_EVENT事件。



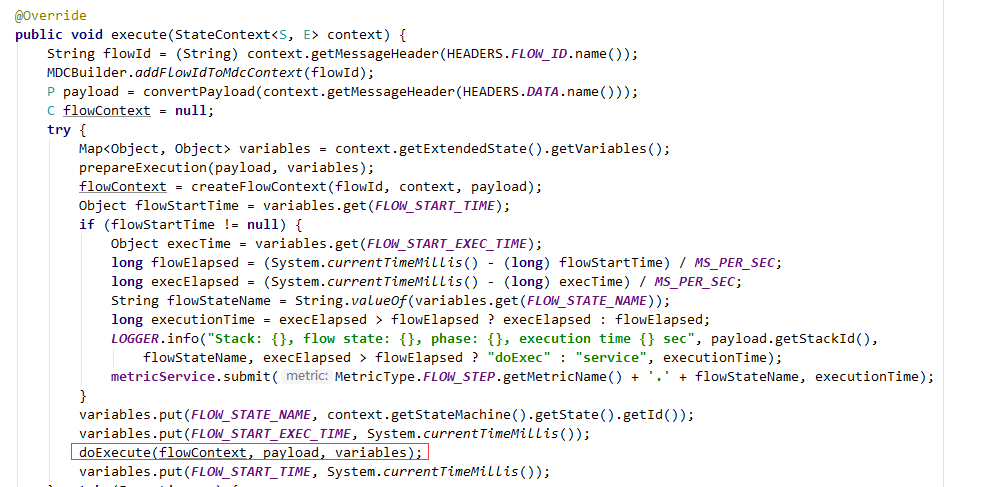
5.根据上文项目启动配置4中讲解的可知，发送的STACK\_PROVISION\_TRIGGER\_EVENT事件被Flow2Handler接收处理。还没有运行的flow的时候会创建一个flow，启动flow的状态机，给状态机flowMachine发送STACK\_PROVISION\_TRIGGER\_EVENT事件（注意此时发送的事件不是eventBus发布，是状态机内部事件机制）



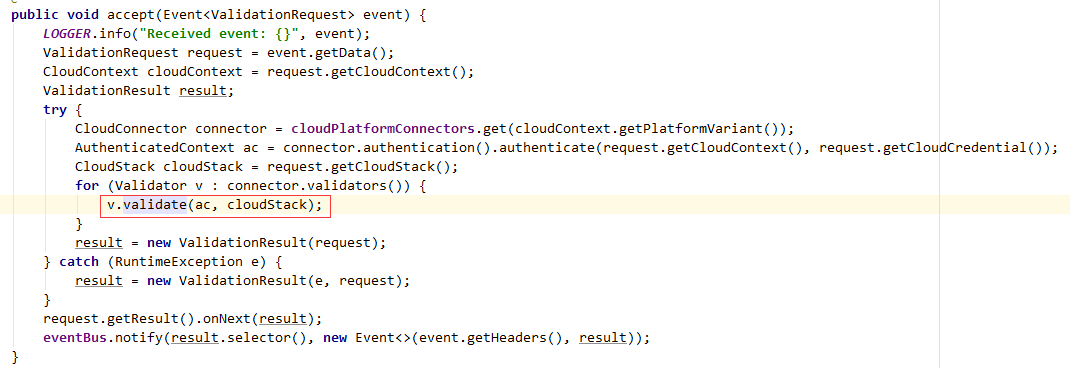
6.根据项目启动配置5可知，STACK\_PROVISION\_TRIGGER\_EVENT事件会触发flowMachine状态机从INIT\_STATE变为VALIDATION\_STATE状态，进入VALIDATION\_STATE状态之前需要执行StackCreationActions.java中定义的VALIDATION\_STATE的action动作。



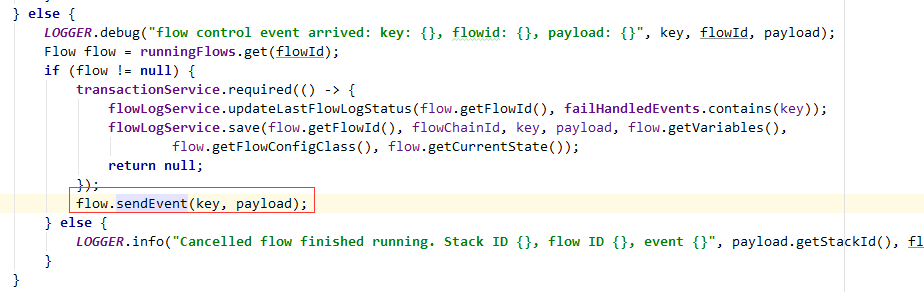
7.执行VALIDATION\_STATE的action的execute方法（AbstractAction.java），该方法主要是调用子类的doExecute方法，执行sendEvent方法，eventBus发送ValidationRequest事件。



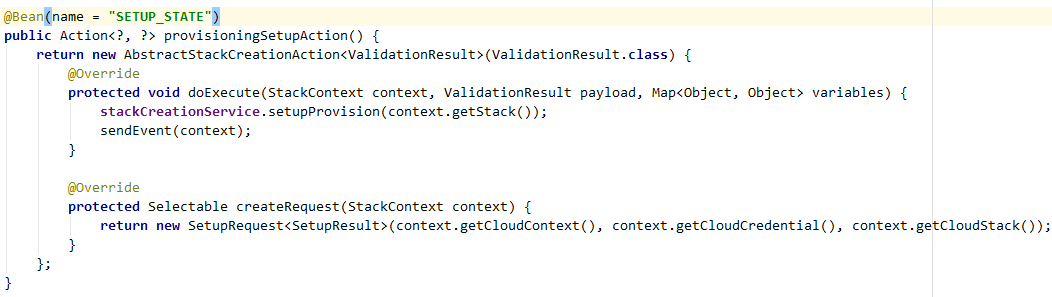
8.根据项目启动配置2可知，ValidationRequest事件被ProvisionValidationHandler处理器处理，主要检查openstack中stack名称存不存在，eventBus发送VALIDATIONRESULT事件。



9.根据项目启动配置 4可知，VALIDATIONRESULT事件被Flow2Handler处理（所有的flow事件都是被Flow2Handler处理），这次Flow2Handler处理，因为已经存在运行中的flow，所以不会生成新的flow，只是给flow的状态机发送VALIDATIONRESULT事件



10.根据StackCreationFlowConfig.java预先配置的状态迁移关系可知，状态机将会变为SETUP\_STATE状态，进入SETUP\_STATE状态之前执行SETUP\_STATE操作的action。

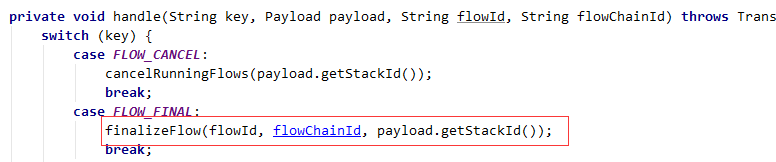


11.到此，整个cloudbreak的操作的内部流转机制就运转起来了。

先是创建一个工作流flow，然后工作流flow中有一个状态机stateMachine，stateMachine在项目启动时配置完成了状态之间的转换关系，以及进入一个状态之前要完成的操作action。

当工作流启动之后，flow2Handler中接着给状态机发送一个FlowEvent，然后状态机发生状态迁移，状态机进入下一个状态之前执行操作action，在执行操作action的时候会发送Request请求被handler处理完后，触发eventBus发步FlowEvent事件，被flow2Handler接收，然后调用状态机发送该FlowEvent事件，开始下一个状态操作。（所以flow2Handler后面的功能就是把eventBus发布的事件发送给stateMachine，触发状态机状态迁移）

最后会迁移到FINAL\_STATE之前，执行FlowFinalizeAction操作时，eventBus发布FLOW\_FINAL事件被Flow2Handler接收，如果flowChain中还有flow，就执行下一个flow，没有就结束。



12.集群创建Flow流程总结：

（1）Stack创建flowchain步骤：

**VALIDATION\_STATE**：

检查stack名称是否重复

**SETUP\_STATE**：

创建hdinsight\_manage\_security\_group安全组

**START\_PROVISIONING\_STATE**：

调用openstack API创建stack

**PROVISIONING\_FINISHED\_STATE**：

更新集群状态

**COLLECTMETADATA\_STATE**：

获取stack虚拟机实例ID

**CREATE\_EIP\_STATE**：

若创建EIP，则调用EIP接口创建

**BIND\_EIP\_STATE**：

绑定EIP接口

**TLS\_SETUP\_STATE**：

连接管理节点nginx，保存nginx服务证书

**STACK\_CREATION\_FINISHED\_STATE**：

stack创建流程完毕

（2）Cluster创建flowchain步骤：

**BOOTSTRAPPING\_MACHINES\_STATE**：

上传saltstack配置文件和密钥，调用配置启动salt-master和salt-minion的API（调用日志查看manage节点的/var/log/saltboot.log）

**COLLECTING\_HOST\_METADATA\_STATE**：

获取虚拟机的FQDN并保存数据库

**UPLOAD\_RECIPES\_STATE**：

如果有自定义脚本，上传脚本（可忽略）

**STARTING\_AMBARI\_SERVICES\_STATE**：

配置saltstack pillar信息主要是unbound、kerberos、yum 源、knox和数据库的配置信息；配置saltstack grains信息，主要是给管理节点添加ambari、postgresql，给主节点添加knox和kerberos（开启的话）角色；执行saltstack命令state.highstate，启动所有相关服务配置，

**REGISTER\_PROXY\_STATE**：

未启用

**STARTING\_AMBARI\_STATE**：

等待ambariserver启动，并创建集群用户

**INSTALLING\_CLUSTER\_STATE**：

按照实际情况跟新blueprint信息，主要是hdfs挂盘、kerberos开启信息；然后去ambari创建blueprint，创建集群。

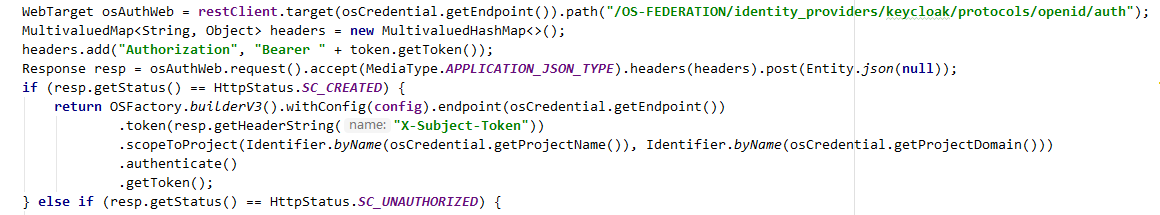
**CLUSTER\_CREATION\_FINISHED\_STATE**：

cluster创建完成

### 六、重点接口调用

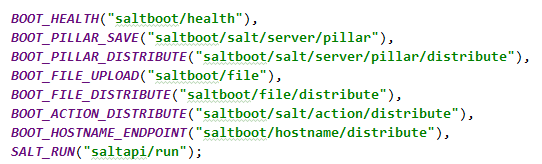
1.Openstack接口调用分析：

主要使用openstack4j进行openstack API的调用，生成客户端OpenStackClient.java，用缓存的用户的登陆token获取keystone的token完成认证



2.管理节点封装的一些用于集群管理的接口：

管理节点配置了nginx的ssl，后端API包含了saltbootstrap和saltapi的一些接口，接口声明如下SaltEndpoint.java



Saltboot/health：

健康检查

Saltboot/salt/server/pillar：

上传pillar配置

Saltboot/salt/server/pillar/distribute：

分发pillar配置到minion节点

Saltboot/file：

上传文件

Saltboot/file/distribute：

分发文件到其他节点

Saltboot/salt/action/distribute：

执行salt-master和salt-minion的启动

Saltboot/hostname/distribute：

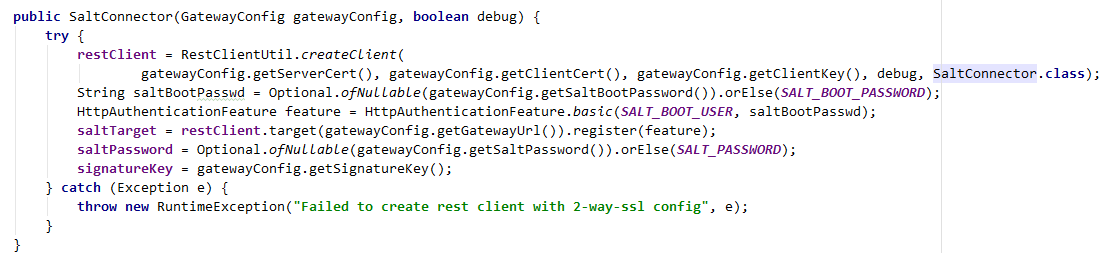
获取host主机信息

Saltapi/run：

执行saltapi操作比如grains.append、jobs.acive、state.highstate。

执行日志保存在节点/var/log/saltboot.log

连接客户端SaltConnector.java，主要涉及https的连接，首先认证模式是basic auth，用户名是cbadmin，密码是数据库securityconfig表中的saltbootpassword，https的clientcert和clientkey也是在数据库表secutiryconfig中。使用Decrypt.java进行解密。



3.AmbariClient进行ambari相关的操作

主要是进行ambari的用户名密码的创建，blueprint的创建和集群的创建等。客户端连接AmbariClientProvider.java，连接方式类似与上，先是basic auth使用集群用户名密码，然后是同样的https配置：

