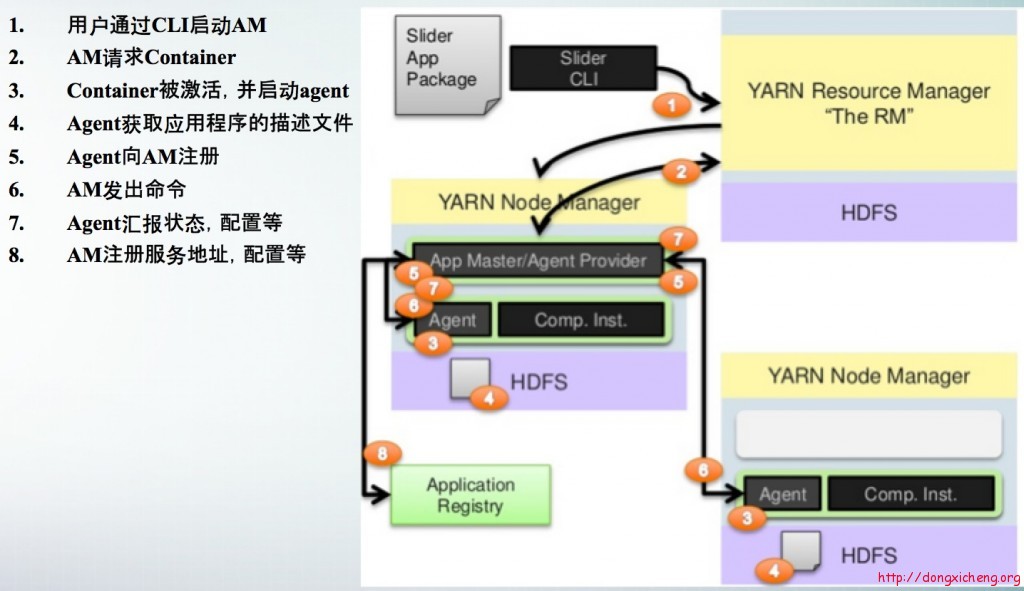
Slider源码解析

# 1、Apache Slider组件

Slider的基本组成模块及关系如下图所示：

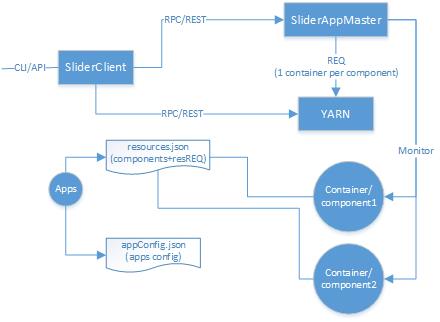


如上图，Slider的基本模块为App Master、Agent Provider、Slider Agent、CLI、Application Registry和App Package，下面进行详细介绍：

Slider AppMaster: 根据App配置向Yarn申请Container，每个Component对应一个Container，并监控Container状态

Slider Client:与SlientAppMaster和Yarn通过REST/RPC通信，对外提供CLI和底层API

App Package: App应用程序安装包，需要提供resource.json（各个components以及mem/cpu资源需求），appConfig（各个components的配置信息）及.zip（app应用的压缩文件）。这三个组件的应用关系，如下图所示：



## 1.1 Slider AppMaster

Slider AppMaster有以下几个模块组成，AM Engine、Agent Provider、Apps State及App Registry。其类图如下所示：



### 1.1.1 Agent Provider

用于用户App的部署和启动，Provider会在Container部署Slider Agent，然后该agent调用Provider的REST API来执行返回的命令。具体包括，从HDFS上下载并解压app安装文档，运行Python脚本来启动服务。Provider分为两个部分，AbstractClientProvider及ProviderService。

1）AbstractClientProvider，其实现类为AgentClientProvider及SliderAMClientProvider，ClientProvider用于管理App的配置文件。

类的结构图如下所示：



AbstractClientProvider用于从配置文件中获取配置参数（instanceConf），并校验其是否可用，其中参数包括appconf.json、resources.json、internal.json及metainfo.xml及相应App的配置目录信息。

SliderAMClientProvider，用于处理Client端获取的Slider AM启动时相关的参数，包括role,cluster及cluster描述等参数。其ProviderRole为APPMASTER，特定Role。

2）ProviderService，SliderAppMaster中启动Component的核心类

 根据resource.json的定义，启动Componet，在对应的Container部署Slider Agent。Agent会将心跳汇报给AgentProviderService，根据心跳信息进行Component的操作。

SliderAMProviderService，将AppMaster中的一些功能放到该类中进行处理，包括配置文件的处理，创建ContainerLauncContext，启动AppMaster。

其中AgentProviderSerivce的handleHeartBeat是核心方法，实现功能依次：

1. 根据roleName从MetaInfo.xml中获取其执行脚本路径
2. 构建componet的状态机
3. 处理端口
4. 添加InstallCommond
5. 添加startCommand，会根据component的状态进行判断
6. 返回心跳性信息

### 1.1.2 Provider AppState

AppState用于记录应用实例的信息，包括resource需求说明，当前各role运行哪些实例Instance的对应关系信息，Role History用以记录哪些Role部署在哪些nodes上的信息，该信息持久存储到磁盘中，以便后续快速启动服务。类图如下所示：



在{app-def}的history的目录中，存储了rolehistory。

以flex为例，描述AppState的前后变化，执行流程如下所示：

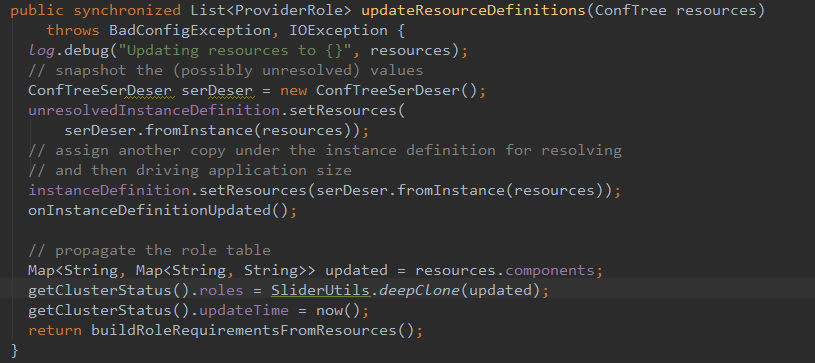
SliderClient.actionFlex —> SliderClient.flex:clusterName,Map<String,Integer>

—> SliderClusterOperations#flex()

—> SliderAppMaster#flexCluster:ConfTree, updatedResources

—>appState.updateResourceDefinition(resources)，appState.resetFailureCounts

updateResourceDefinitions的源码如下所示：



注意：

flex操作中，SliderAppMaster会调用reviewRequestAndReleaseNodes，来根据Resources中定义的roles数目来扩展或缩小Cluster的规模：

SliderAppMaster#reviewRequestAndReleaseNodes

—>RMOperationHandler.addContainerRequest

—>AMRMClientImpl#addContainerRequest

### 1.1.3 Slider Registry

Slider Registry用于App的注册过程，与AM通信，类图如下所示：



Slider Registry封装了Hadoop中的RegistryOperations，为App提供了Component,Service的注册操作。

1）Service的注册流程

Slider.main —> ServiceLauncher#serviceMain —> ServiceLauncher#launchService

—> SlideAppMaster#runSerivce —> #createAndRunCluster

—> #queue(new ActionRegisterServiceInstance)

—> SliderAppMaster#registerSerivceInstance

—> YarnRegistryViewForProviders#register

在启动Service之前，首先将Service注册到Zookeeper中

2）Component的注册流程

SliderAppMaster#onContainerStarted

—>SliderAppMaster#registerComponent

—>YarnRegistryViewForProviders#putComponent

### 1.1.4 LaunchService

LaunchService用于启动Container，类图如下所示：



调用流程为：

SliderAppMaster#onContainerAllocated

—>RoleLaunchService#launchRole

—>RoleLauncher#run

—> actionQueue.scheduler(new ActionStartContainer)

—>SliderAppMaster#startContainer

—>NMClientAsync#startContainerAsync

## 1.2 Slider Agent

Slider Agent是Python实现的线程，其运行在Container中，根据APP的定义文件及描述进行App的安装及维护。

AM启动后，会向RM请求Container，激活Container后会启动agent。Agent启动后获取应用程序的描述及定义文件，并向AM注册。注册后AM发出命令，这一步是通过Agent心跳信息进行通信。执行命令后，Agent向AM汇报状态，配置等。

1）AgentClientProvider将slider-agent.tar.gz上传到对应APP启动的HDFS的路径下，例如

{app-xxx}/tmp/application\_1431331396278\_0008/agent/slider-agent.tar.gz

调用栈如下所示:SliderClient.launchApplication —> AgentClientProvider.prepareAMAnd

ConfigForLaunch —>ProviderUtils.addAgentTar

2）在SliderAppMaster启动Container之前，将Agent启动脚本，添加到ContainerLauncher中：

* 1. 首先获取main.py的路径{app-home}/slider-agent/agent/main.py
  2. 构建Agent启动命令，CommandLineBuilder.add(.....)，大概的命令格式如下所示：

${python-home}/python main.py --label xx --zk-quorum xx --zk-reg-path xx ...

3）SliderAppMaster，从RM获取Container后，启动Agent，调用栈如下：

SliderAppMaster#onContainerAllocated —> RoleLaunchService.launchRole —>

RoleLauncher#run —> AgentProviderService#buildContainerLauncContext —>

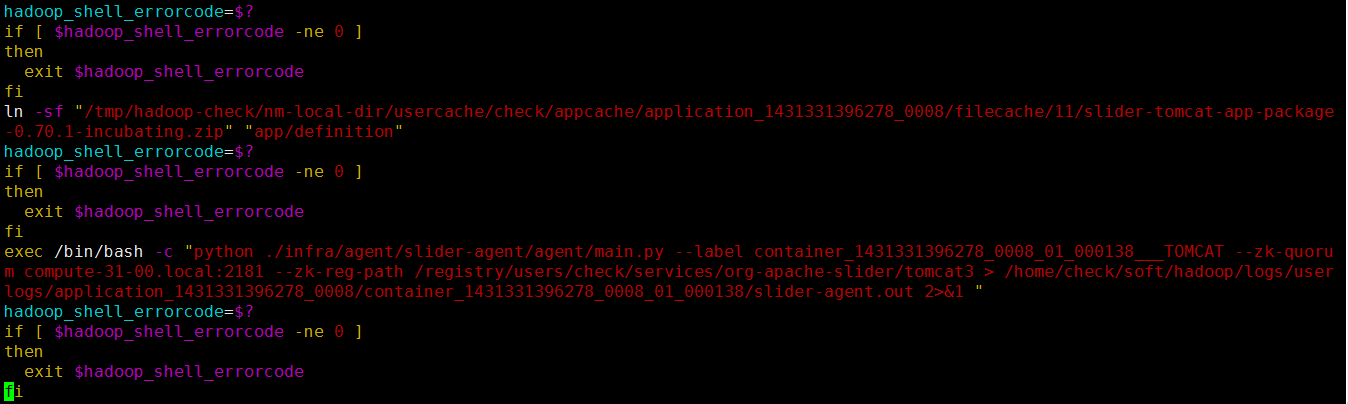
QueueService#schedule —> actionQueue.put —> SliderAppMaster#startQueueProcessing

—> QueueExecutory#run —> ActionStartContainer#execute —>

SliderAppMaster#startContainer —> NMClientAsync#startContainerAsync

4）NodeMananger收到container start命令后，生成launch\_container.sh脚本，然后

ContainerLaunch会调用launch\_container启动一个container。脚本如下所示：



可以看到exec /bin/bash -c "python .../agent/main.py"启动一个agent。

5）Slider agent基本功能介绍

类图如下所示：

## 1.3 Slider Client

用户通过CLI创建AM，例如通过命令slider create及start等。在Slider中客户端是slider.py脚本，在脚本中根据配置JAVA\_HOME/HADOOP\_CONF等目录，创建客户端JVM，

JVM类为org.apache.slider.Slider。

调用流程：slider.py -> Slider -> ServiceLauncher -> SliderClient -> SliderClient.exec

根据传递的参数选择执行的命令。

# 2、Slider执行流程

## 2.1 SliderAppMaster启动流程

以App的create为例，分析其执行流程，如下图所示：

 SliderClient，通过SliderYarnClientImpl（继承YarnClientImpl）向RM提交App，RM根据配置分配Container给SlientAppMaster，并启动AppMaster。

SliderClient create集群时，通过SliderYarnClientImpl向ResourceManager提交应用请求。RM根据集群情况，分配Container。SliderAppMaster启动后，根据resources.json的定义，创建并启动container。

## 2.2 Slider Port的分配

由于Slider可以创建多个相同的APP，因此port的管理很重要，要为每个应用分配一个唯一的端口，其实现类为PortScanner。

SliderAppMaster#createAndRunCluster —>

SliderAppMaster#startSliderRPCServer : instanceDefinition —>

SliderAppMaster#getPortRequest ，PortScanner#getAvailablePort

new WorkFlowRpcService —> deployChildService(rpcService )

## 2.3 App Config使用流程

### 2.3.1 config文件

以HBase为例，介绍应用的配置文件，hbase的create命令为：

$slider create hbase --template appConfig.json --resources resources.json

命令中，有两个配置文件appConfig.json和resouces.json，由用户配置，appConfig.json配置基本参数，resources.json配置集群中启动的服务组件数量及资源占用量。

创建后，在应用创建目录hdfs:///user/check/.slider/cluster/hbase/中会生成三个配置文件，分别是app\_config.json、internal.json和resouces.json，文件中是处理后的参数，下次启动时使用这个配置。（在安装包中，有软件的配置文件，这部分当前不做处理，如果想对配置文件里面的参数进行修改，需要在app\_config.json中添加配置）。

metainfo.xml文件是app运行的核心文件，其中定义了app的启动脚本和命令执行的command order。

### 2.3.2配置持久化

创建App后，需要将配置文件持久化到HDFS中，服务停止后，通过这些配置文件进行重启。配置文件的处理服务为SliderClientProvider及SliderAMClientProvider，调用流程如下所示：

SliderClient#actionCreate:args

—> actionBuild:createArgs

—> buildInstanceDefinition:buildInfo

—>builder: new InstanceBuilder

—>SliderAMClientPorvider#prepareInstanceConfiguration

—> SliderAMClientProvider#mergeTemplate:internal.json,appConf.json,resources.json

—> instanceDescription :new AggregateConf，

resources.json(resource:ConfTreeOperations),

appConfig.json(appConf:ConfTreeOperations)

—> persistInstanceDefinition

—>builder#persist

—>persister：ConfPersister #save(AggregateConf conf)

—>confTreeSerDeser#save：fileSystem,internal,resources,app\_conf

—>confTreeSerDeser#writeJsonAsBytes

通过上面的步骤，在程序中初始化了instanceDefinition:AggregateConf及将instanceDe

finition持久化到HDFS中。

参数在cluster的启动过程中使用，前提是参数已经持久化到了HDFS中(actionCreate或者重新启动cluster，过程都是相同的，都是调用startCluster)，流程如下所示：

SliderClient# startCluster

—> loadInstanceDefinitionUnresolved(clusterName,ClusterDirectory)

—> InstanceIO#loadInstanceDefinitionUnresolved(fileSystem,ClusterDirectory)

—> ConfPersister#load(instanceDefiniton)，confTreeSerDeser.load:internal,

resources,app\_conf

—> launchApplication(clusterName,ClusterDirectory,instanceDefinition,..)

—> new AppMasterLauncher

—> SliderAMClientProvider#prepareAMAndConfigForLaunch，preflightValidateClusterCon

figuration（fileSystem,instanceDefinition,clusterName）

—> SliderClientProvider#prepareAMAndConfigForLaunch, preflightValidateClusterCon

figuration（fileSystem,instanceDefinition,clusterName）

—> AppMasterLauncher #submitApplication

—> SliderYarnClientImpl#submitApplication(ApplicationSubmissionContext)

其中提交应用请求的commandLine中的很多参数都是从instanceDefinition中获取

### 2.3.4参数的赋值

在配置文件中，参数很多是${TOMCAT.ALLOCATED\_PORT}这种临时形式，在实际运行中进行赋值。SliderClient提交submitApplication，RM分配Container后，在Container中执行SliderAppMaster.main，以Tomcat为例，介绍其赋值的过程：

SliderAppMaster#main —> runService —> createAndRunCluster

—> InstanceIO.loadInstanceDefinitionUnresolved，:instanceDefiniton

—> SliderProviderFactory#createSliderProviderFactory

—> AgentProviderService，SliderAMProviderServerice #initAndAddService

—> PortScannder，buildPortScanner

—> startSliderRPCServer: <rpchostname，rpcPort>，initial TrackingURL

—> RegistryOperations# init

—> SliderAppMaster # launchProviderService

—> SliderAppMaster # onContainersAllocated

—> RoleLaunchService # launchRole //SliderAppMaster

—> AppState#prepareAllocationList(allocatedContainers) //确定启动哪个Role

—> ordered:List<Container>: RoleHistory.prepareAllocationList<allocatedContainers>

—> role:RoleStatus: lookupRoleStatus()

—> RoleLaunchService.launcheRole：assignment, role,instanceDefintion

—> RoleLauncher.run

—> AgentProviderServer#buildContainerLaunchContext //初始化python执行环境，包括确定agent的执行脚本 main.py

—> RoleInstance#initial

—> SliderAppMaster#startContainer

—> NMClientAsync#startContainerAsync //启动SliderAgent，仅执行main.py，里面没有运行命令，仅向AgentProviderServer发送心跳信息

Slider Agent启动后向AgentProviderServer发送心跳信息

AgentProviderService#handleHeartBeat，

//进行的操作，有获取心跳中的hostname,roleName及containeID

—> AgentProviderService#getScriptPathFromMetainfo(roleName) //根据roleName获取执行脚本，例如Tomcat中，Tomcat Master的执行脚本为Tomcat.py，HBase中HBase Master的执行脚本为hbase\_master.py。

—> Agent Heartbeat中发送信息包括分配的port，report.agetAllocatedPort

—> AgentProviderService # processAllocatedPorts

—> RoleInstance < AppState#getOwnedContainer

—> RoleInstance#registerPortEndPoint -> endpoints.add(endpoint) //这里仅注册了TCP port

# 3. Agent与SliderAppMaster通信

SliderAppMaster通过AgentProviderService与Agent通信，并进行控制。Agent与SliderAppMaster的心跳交互，如下图所示：

 AgentWebApp为提供Rest服务的Web应用，Agent发送心跳的REST请求，最终调用AgentProviderServerice#handleHeartbeat，其中Agent的注册过程相同。

Agent端与Rest相关的类图如下：



Agent在启动后，执行main.py，调用Controller.py中的方法，发送Heartbeat及register Rest API调用，返回心跳信息，信息中包括要执行的操作。

# 4. Command的执行

在第3部分介绍了Agent与SliderAppMaster的通信过程，Agent通过REST API发送心跳信息到SliderAppMaster。SliderAppMaster会返回JSON形式的应答信息，信息中包含了要执行的命令。执行流程如下所示：

Controller.py #heartbeatWithServer

—> AgentResource#hearbeat

—> AgentProviderService#handlerHeartbeat //返回Json,response

—> Controller.py，addToQueue(response['executionCommands'])

—> ActionQueue #processCommand

—> CustomServiceOrchestrator #runCommand

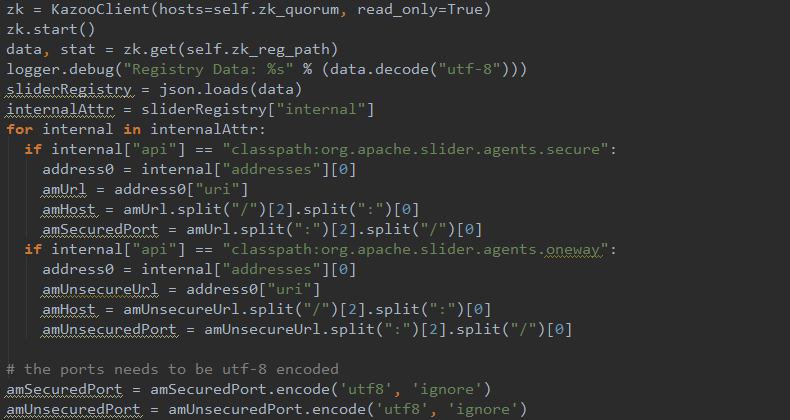
命令以python文件的形式，定义在App-package中。

# 5. Slider Registry信息

## 5.1 Agent Registry及Heartbeat EndPoint

在Agent的发送心跳信息之前会获取Slider的注册信息，获取方法为Registry#readAM

HostPort，源码如下：



获取的amHost及port，是Agent与AM的REST的端口，其中出现的EndPoint，描述例子如下所示：

"internal": [

{

"api": "classpath:org.apache.slider.agents.secure",

"addressType": "uri",

"protocolType": "REST",

"addresses": [

{

"uri": "https://compute-31-00.local:55410/ws/v1/slider/agents"

}

]

},

{

"api": "classpath:org.apache.slider.agents.oneway",

"addressType": "uri",

"protocolType": "REST",

"addresses": [

{

"uri": "https://compute-31-00.local:38154/ws/v1/slider/agents"

}

]

}

]

这两个EndPoint用于Agent与AM之间发送心跳信息：

1. "api": classpath:org.apache.slider.agents.secure

REST服务的加密URI

1. "api":classpath:org.apache.slider.agents.onway

REST服务的非加密URI，

从源码中可以看到，heartbeat及register使用加密端口，从URI中可以解析出AMhost及端口号，而且从信息中可以看出，这两个EndPoint为Internal，用于内部通信机制。

## 5.2 AM IPC endpoint

SliderAppMaster会启动RPC Server用于接收客户端请求，IPC endpoint用于暴露Server的主机名和端口，示例如下所示：

{

"api": "classpath:org.apache.slider.appmaster.ipc",

"addressType": "host/port",

"protocolType": "hadoop/IPC",

"addresses": [

{

"port": "1024",

"host": "compute-31-00.local"

}

]

}

SliderAppMaster的RPC类图如下所示：



如类图所示，可以通过SliderClient对Cluster进行操作。其中port和hostname是提供给客户端，通过SliderYarnClientImpl可以根据clusterName获取port和hostname。但是自己定制客户端，可以通过endpoint来进行操作。

## 5.3 AM REST endpoint

AM提供给外部REST管理及操作接口，类图如下所示：



通过Rest接口可以完成registry、publisher、management及应用相关等操作。

### 5.3.1 AM webUI endpoint

AM提供给外部REST接口，供用户使用，endpoint如下图所示：

{

"api": "http://",

"addressType": "uri",

"protocolType": "webui",

"addresses": [

{

"uri": "http://compute-31-00.local:1025"

}

]

},

{

"api": "classpath:org.apache.slider.client.rest",

"addressType": "uri",

"protocolType": "webui",

"addresses": [

{

"uri": "http://compute-31-00.local:1025"

}

]

}

在SliderAppMaster#createAndRunCluster中，启动WebApp，随机获取port，并调用deployWebApplication启动一个WebAppApiImpl。上面的两个REST endpoint是提供给外部的接口。

### 5.3.2 Management REST endpoint

Slider Cluster的管理功能接口，endpoint示例如下：

{

"api": "classpath:org.apache.slider.management",

"addressType": "uri",

"protocolType": "REST",

"addresses": [

{

"uri": "http://compute-31-00.local:1025/ws/v1/slider/mgmt"

}

]

}

其对应ApplicationResources中定义的操作

### 5.3.3 publisher REST endpoint

该Rest接口用于返回，配置信息，示例如下：

{

"api": "classpath:org.apache.slider.publisher",

"addressType": "uri",

"protocolType": "REST",

"addresses": [

{

"uri": "http://compute-31-00.local:1025/ws/v1/slider/publisher"

}

]

}

其对应两个具体的分类， slider配置信息，REST接口如下所示：

{

"api": "classpath:org.apache.slider.publisher.configurations",

"addressType": "uri",

"protocolType": "REST",

"addresses": [

{

"uri": "http://compute-31-00.local:1025/ws/v1/slider/publisher/slider"

}

]

}

exports endpoint接口，示例如下：

{

"api": "classpath:org.apache.slider.publisher.exports",

"addressType": "uri",

"protocolType": "REST",

"addresses": [

{

"uri": "http://compute-31-00.local:1025/ws/v1/slider/publisher/exports"

}

]

}

]

### 5.3.4 registry REST endpoint

该Rest接口用于查询zookeeper中的path对应的数据，示例如下：

{

"api": "classpath:org.apache.slider.registry",

"addressType": "uri",

"protocolType": "REST",

"addresses": [

{

"uri": "http://compute-31-00.local:1025/ws/v1/slider/registry"

}

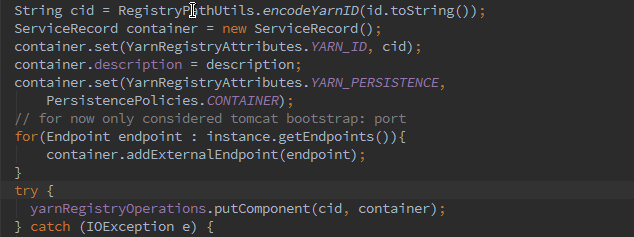
]

}

## 5.4 Agent endpoint

在yarn中启动集群服务后，服务可能有多种对外Endpoint，例如REST,IPC及TCP等，当前版本中，没有在zookeeper中进行注册，但是可以修改源码，进行注册，当前仅支持TCP的注册(RoleInstance# registerPortEndPoint)。

以Tomcat为例，在SliderAppMaster#registerComponent中注册TCP端口，源码如下：



在registry中获取endpoint信息，如下：

{

"type": "JSONServiceRecord",

"description": "TOMCAT",

"external": [

{

"api": "server-xml.http.port",

"addressType": "host/port",

"protocolType": "tcp",

"addresses": [

{

"port": "42537",

"host": "compute-31-00.local"

}

]

}

}

可以访问http://compute-31-00.local:42537，Tomcat的端口

# 6、Slider Rest API

上面是当前Slider中具有的REST等接口，下面是REST的使用（目前Slider支持的大部分REST都是GET应用），在命令行使用curl来执行。

## 6.1 GET

### 6.1.1 Registry

以registry为例，介绍其使用

访问/ws/v1/slider/{path\*\*}，对应RegistryResources，在分析RegistryResource中可以看到registry可以在slider中进行配置，默认/registry

命令如下：

curl -i -X GET "http://compute-32-12.local:1025/ws/v1/slider/registry"

返回结果：

{"nodes":["users"]}

可以通过zkCli.sh，查看结果是相同

获取service信息的命令如下：

curl -i -X GET "http://compute-32-12.local:1025/ws/v1/slider/registry/users/check/services/

org-apache-slider/tomcate4"

返回内容如下：

{

"nodes": [

"components"

],

"service": {

"type": "JSONServiceRecord",

"description": "Slider Application Master",

"external": [

{

"api": "classpath:org.apache.slider.appmaster.ipc",

"addressType": "host/port",

........

}

"yarn:persistence": "application",

"yarn:id": "application\_1432086695267\_0004"

}

}

分析:

service中的内容为该ZKnode对应的数据，nodes下的components是子节点，可以进一步查看。

查看component的container信息，命令如下：

curl -i -X GET "http://compute-32-12.local:1025/ws/v1/slider/registry/users/check/services/org-apache-slider/tomcate4/components/container-1432086695267-0004-01-000002"

返回信息：

{

"nodes": [],

"service": {

"type": "JSONServiceRecord",

"description": "TOMCAT",

"external": [

{

"api": "server-xml.http.port",

"addressType": "host/port",

"protocolType": "tcp",

"addresses": [

{

"port": "42537",

"host": "compute-31-00.local"

}

]

}

],

"internal": [],

"yarn:persistence": "container",

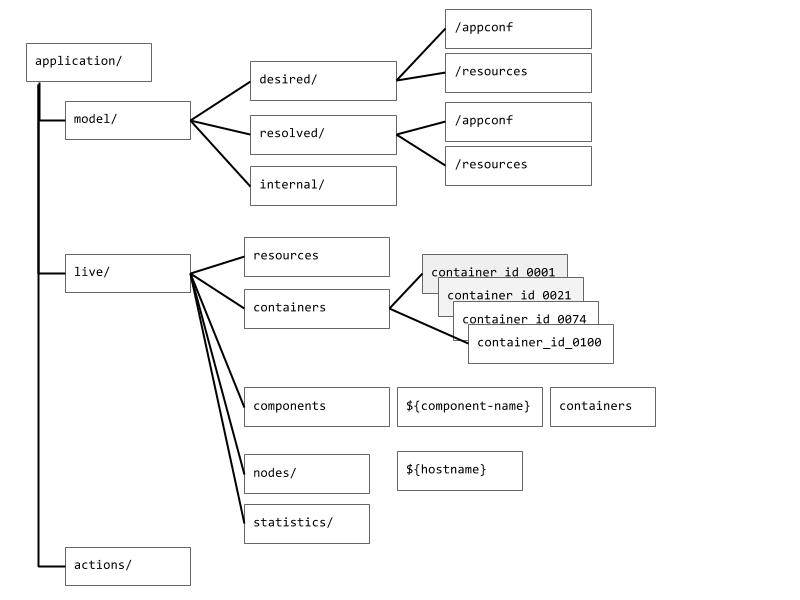
"yarn:id": "container-1432086695267-0004-01-000002"

}

}

### 6.1.2 Application

上面获取的是Registry信息，ApplicationResource对应的结构如下图所示：



通过下面的命令，可以看到第一层结构：

curl -i -X GET "http://compute-32-12.local:1025/ws/v1/slider/application"

返回的结果如下：

["model","live","actions"]

## 6.2 PUT

下面的命令，查看desired信息

curl -i -X GET "http://compute-32-12.local:1025/ws/v1/slider/application/model/desired"

service的配置信息中，包括internal,appconf,resources。

可以通过修改resources/数据，触发flex operation，但是写configuration在服务重启或者upgrade的时候才会生效，使用Put命令如下：

curl -i -X PUT -d "xxx" "http://compute-32-12.local:1025/ws/v1/slider/application/model/desired/resources"

/application/model/internal的配置只读。

## 6.3 POST

在ApplicationResource中，支持的POST有actionUpgrade、actionPing\*,actionStop。

action/stop： 停止某个应用

action/upgrade: 升级某个应用

action/ping，简单的Ping操作，用于测试用

## 6.4 DELETE

在ApplicationResource中，支持的Delete操作仅有，actionPingDelete

# Reference