Dshell实现详情

# 源码分析：

源码类：ApplicationMaster Client DSConstants Log4jPropertyHelper

**其中**： Client为提交任务时在客户端执行的主方法类，主要功能包括接收参数，向hdfs传文件，给appmaster传相关的环境变量，启动appmaster，和appmaster通信。

Appmstaer接收Client传来的环境变量，和ResourceManager通信申请Container并且启动执行，编写相关回调函数，实现其中的各种监控管理方法。

DSConstants 为Dshell中用到的字符常量，用来做判断使用。

Log4jPropertyHelper为日志配置，当用户没有log4j.property时，默认使用该类中的配置加载输出。

**原生Client的入口main函数如下：**

public static void main(String[] args) {

…

Client client = new Client();

boolean doRun = client.init(args);

if (!doRun) {

System.exit(0);

}

**result = client.run();**

…

}

Client中最重要的是函数为run()，该函数实现过程如下：

1.  YarnClient.getNewApplication

2. 创建ApplicationSubmissionContext,ContainerLaunchContext（启动AM的Container）​

3. YarnClient.submitApplication​

4. 每隔一段时间调用YarnClient.getApplicationReport获得Application Status

// 创建AM的上下文信息

ContainerLaunchContext amContainer = Records.newRecord(ContainerLaunchContext.class);

// 设置本地资源，AppMaster.jar包，log4j.properties

amContainer.setLocalResources(localResources);

// 环境变量,shell脚本在hdfs的地址, CLASSPATH

amContainer.setEnvironment(env);

// 设置启动AM的命令和参数

Vector<CharSequence> vargs = new Vector<CharSequence>(30);

vargs.add("${JAVA\_HOME}" + "/bin/java");

vargs.add("-Xmx" + amMemory + "m");

// AM主类

vargs.add("org.apache.hadoop.yarn.applications.distributedshell.ApplicationMaster?");

vargs.add("--container\_memory " + String.valueOf(containerMemory));

vargs.add("--num\_containers " + String.valueOf(numContainers));

vargs.add("--priority " + String.valueOf(shellCmdPriority));

if (!shellCommand.isEmpty()) {

vargs.add("--shell\_command " + shellCommand + "");

}

if (!shellArgs.isEmpty()) {

vargs.add("--shell\_args " + shellArgs + "");

}

for (Map.Entry<String, String> entry : shellEnv.entrySet()) {

vargs.add("--shell\_env " + entry.getKey() + "=" + entry.getValue());

}

vargs.add("1>" + ApplicationConstants.LOG\_DIR\_EXPANSION\_VAR + "/AppMaster.stdout");

vargs.add("2>" + ApplicationConstants.LOG\_DIR\_EXPANSION\_VAR + "/AppMaster.stderr");

amContainer.setCommands(commands);

// 设置Resource需求，目前只设置memory

capability.setMemory(amMemory);

amContainer.setResource(capability);

appContext.setAMContainerSpec(amContainer);

// 提交application到RM

super.submitApplication(appContext);

ApplicationMaster类的main方法如下：

public static void main(String[] args) {

boolean result = false;

ApplicationMaster appMaster = new ApplicationMaster();

boolean doRun = appMaster.init(args);

if (!doRun) {

System.exit(0);

}

**appMaster.run();**

result = appMaster.finish();

}

所有的功能操作同样是在run方法中：

1.  AMRMClient.registerApplicationMaster​​

2. 通过amRMClient.addContainerRequest​提交ContainerRequest

3. container放入新建的LaunchContainerRunnable线程内执行

4. 创建ContainerLaunchContext​，设置localResource，shellcommand, shellArgs等​​container启动信息

5.  nmClientAsync.startContainerAsync(container, ctx);

6. 调用相关回调函数查询container状态：

applicationMaster.nmClientAsync.getContainerStatusAsync(containerId, container.getNodeId());

7.  销毁appMaster：amRMClient.unregisterApplicationMaster(appStatus, appMessage, null);

# 实现目标：

1. 在每次提交单机算子时只创建一个container，在这个container中执行一个普通的java程序。
2. 能够通过dshell的参数把执行java程序时所需的文件传输至container，同时支持目录。

# 开发与实现：

1. 每次启动只创建一个container：

numContainers = Integer.parseInt(cliParser.getOptionValue("num\_containers", "1"));

//container默认值赋为1，并且注掉命令帮助中的container个数选项

1. 添加container\_files选项和container\_archives选项

以container\_files为例解释程序：

在client中：

if (cliParser.hasOption("container\_files")) {

containerJarPaths = cliParser.getOptionValue("container\_files").split(",");

}//解析命令行中的container\_files参数，解析赋给局部变量

在run方法中把相关file上传到hdfs并且添加至appMaster的环境变量中：

if (containerJarPaths.length != 0) {

for (int i = 0; i < containerJarPaths.length; i++) {

String hdfsJarLocation = "";

String[] jarNameSplit = containerJarPaths[i].split("/");

String jarName = jarNameSplit[jarNameSplit.length - 1];

long hdfsJarLen = 0;

long hdfsJarTimestamp = 0;

if (!containerJarPaths[i].isEmpty()) {

Path jarSrc = new Path(containerJarPaths[i]);

String jarPathSuffix = appName + "/" + appId.toString() + "/" + jarName;

Path jarDst = new Path(fs.getHomeDirectory(), jarPathSuffix);

**fs.copyFromLocalFile(false, true, jarSrc, jarDst)**;

hdfsJarLocation = jarDst.toUri().toString();

FileStatus jarFileStatus = fs.getFileStatus(jarDst);

hdfsJarLen = jarFileStatus.getLen();

hdfsJarTimestamp = jarFileStatus.getModificationTime();

**env.put(DshellDSConstants.DISTRIBUTEDJARLOCATION + i,**

**hdfsJarLocation);**

**env.put(DshellDSConstants.DISTRIBUTEDJARTIMESTAMP + i,**

**Long.toString(hdfsJarTimestamp));**

**env.put(DshellDSConstants.DISTRIBUTEDJARLEN + i,**

**Long.toString(hdfsJarLen));**

}

}

}

在appMaster中：

解析环境变量，得到file相关信息(包括hdfs地址，时间戳，大小)

// 遍历envs，把所有的jars的HDFS路径，时间戳，LEN全部保存到jarPaths对象数组中。

for (String key : envs.keySet()) {

if (key.contains(DshellDSConstants.DISTRIBUTEDJARLOCATION)) {

DshellFile scistorJar = new DshellFile();

scistorJar.setJarPath(envs.get(key));

String num = key

.split(DshellDSConstants.DISTRIBUTEDJARLOCATION)[1];

scistorJar.setTimestamp(Long.parseLong(envs

.get(DshellDSConstants.DISTRIBUTEDJARTIMESTAMP + num)));

scistorJar.setSize(Long.parseLong(envs

.get(DshellDSConstants.DISTRIBUTEDJARLEN + num)));

scistorJars.add(scistorJar);

}

}

在appMaster的内部类LaunchContainerRunnable中的run方法中把相关文件加载到container的localresource中：

for (DshellFile perJar : scistorJars) {

LocalResource jarRsrc = Records.newRecord(LocalResource.class);

jarRsrc.setType(LocalResourceType.FILE);

jarRsrc.setVisibility(LocalResourceVisibility.APPLICATION);

try {

jarRsrc.setResource(ConverterUtils

.getYarnUrlFromURI(new URI(perJar.getJarPath()

.toString())));

} catch (URISyntaxException e1) {

LOG.error("Error when trying to use JAR path specified"

+ " in env, path=" + perJar.getJarPath(), e1);

numCompletedContainers.incrementAndGet();

numFailedContainers.incrementAndGet();

return;

}

jarRsrc.setTimestamp(perJar.getTimestamp());

jarRsrc.setSize(perJar.getSize());

String[] tmp = perJar.getJarPath().split("/");

**localResources.put(tmp[tmp.length - 1], jarRsrc);**

//第一个参数为下载后的文件名，第二个参数为hdfs路径

}

对于archive进行和file类似的操作则可以实现上传目录到container。