Job submit -> 事件经由状态机StateMachineFactory进行各种状态转换

1. RMAppManager (RMAppEventType.START) ->

2.RMAppImpl(RMStateStoreEventType.STORE\_APP) ->

3.RMStateStore(RMAppEventType.APP\_NEW\_SAVED) ->

4.RMAppImpl(SchedulerEventType.APP\_ADDED) ->

5.CapacityScheduler[handle()方法APP\_ADDED](RMAppEventType.APP\_ACCEPTED) ->

6.RMAppImpl(RMAppAttemptEventType.START) ->

7.RMAppAttemptImpl(SchedulerEventType.APP\_ATTEMPT\_ADDED) ->

\*8.CapacityScheduler[handle()方法APP\_ATTEMPT\_ADDED -> LeafQueue的addApplicationAttempt方法将应用添加到

activeApplications列表](RMAppAttemptEventType.ATTEMPT\_ADDED)

\*\*\*

处理activeApplications集合中的application分析详见【CapacityScheduler异步调度线程AsyncScheduleThread详细分析】

\*\*\* ->

9. RMAppAttemptImpl

\*\*\*代码分析

核心代码

Allocation amContainerAllocation = appAttempt.scheduler.allocate(appAttempt.applicationAttemptId,

Collections.*singletonList* (request), ***EMPTY\_CONTAINER\_RELEASE\_LIST***, **null**, **null**); ->

CapacityScheduler ：allocate()方法被调用!分配资源的核心方法! ->

ContainersAndNMTokensAllocation allocation = pullNewlyAllocatedContainersAndNMTokens(); ->

//第8步中的异步调度线程会为各node分配新的Container(LeafQueue的assignContainer方法中分配的),将Containers放到 、、 、、 //newlyAllocatedContainers 集合中,执行到此处遍历这个集合

**for** (Iterator<RMContainer> i = newlyAllocatedContainers.iterator (); i.hasNext () ;) {

RMContainer rmContainer = i.next ();

...

}

\*\*\*

(RMContainerEventType.ACQUIRED) - >

10.RMAppImpl(RMAppEventType.APP\_RUNNING\_ON\_NODE)[更新RMApp的ranNodes,即运行的节点] ->

## CapacityScheduler异步调度线程AsyncScheduleThread详细分析

易混点:CapacityScheduler中的allocate方法时AM向RM请求资源,获得application的

FiCaSchedulerApp则为申请App申请Containers

CapacityScheduler调度器启动时内部启动了一个异步调度线程AsyncScheduleThread,会不断的轮询处理CapacityScheduler中的节点并将Container分配到各节点下

----------------------------------------------------------------------------------------------------------------------------

调用AllocateContainersToNode () ->

\*分析:最终调用LeafQueue中的assignContainers(Resource clusterResource, FiCaSchedulerNode node)方法 ->

方法中会轮询activeApplications,处理各个application

// Try to assign containers to applications in order

**for** (FiCaSchedulerApp application : activeApplications) {

**If** (***LOG***.isDebugEnabled ()) {

***LOG***.debug ("pre-assignContainers for application " + application.getApplicationId ());

application.showRequests ();

}

. . .

// Try to schedule

CSAssignment assignment = assignContainersOnNode (clusterResource, node, application, priority, **null**);

. . .

->

assignContainersOnNode(),资源类型为ResourceRequest.ANY ->

AssignOffSwitchContainers () ;->

assignContainer() 核心方法-> 核心代码 ->

// Inform the application 通知application

RMContainer allocatedContainer = application.allocate (type, node, priority, request, container); ->

// Create RMContainer

RMContainer rmContainer = **new** RMContainerImpl(container, **this**.getApplicationAttemptId(), node.getNodeID(),

appSchedulingInfo.getUser(), **this**.rmContext);

// Add it to allContainers list.

\* newlyAllocatedContainers.add (rmContainer);\*

liveContainers.put (container.getId (), rmContainer);

// Inform the container 通知container,发送Start事件

rmContainer.handle (**new** RMContainerEvent (container.getId (), RMContainerEventType.***START***)); ->

----------------------------------------------------------------------------------------------------------------------------

RMContainerImpl (RMAppAttemptEventType.***CONTAINER\_ALLOCATED***) ->

RMAppAttemptImpl 核心代码 ->

// Acquire the AM container from the scheduler.再次调用CapacityScheduler的allocate方法,从调度处理线程

//获得Container. 从资源调度器接收am的容器分配，如果amContainer的数量为0,则状态为RMAppAttemptState.SCHEDULED，否则 //appAttempt.setMasterContainer(amContainerAllocation.getContainers().get(0));将第一个Container分配给amMaster,并存储状态信息 //以便故障恢复[1]，状态为RMAppAttemptState.ALLOCATED\_SAVING

Allocation amContainerAllocation = appAttempt.scheduler.allocate (appAttempt.applicationAttemptId,

***EMPTY\_CONTAINER\_REQUEST\_LIST***, ***EMPTY\_CONTAINER\_RELEASE\_LIST***, **null**, **null**);

...

// The node set in NMTokenSecrentManager is used for marking whether the

// NMToken has been issued for this node to the AM.

// When AM container was allocated to RM itself, the node which allocates

// this AM container was marked as the NMToken already sent. Thus,

// clear this node set so that the following allocate requests from AM are

// able to retrieve the corresponding NMToken.

appAttempt.rmContext.getNMTokenSecretManager().clearNodeSetForAttempt(appAttempt.applicationAttemptId);

appAttempt.getSubmissionContext().setResource(appAttempt.getMasterContainer().getResource());

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* appAttempt.storeAttempt ();\* -> rmContext.getStateStore ().storeNewApplicationAttempt (**this**) ->

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

RMStateStore的storeNewApplicationAttempt(RMAppAttempt app)方法中发出事件RMStateStoreEventType.***STORE\_APP\_ATTEMPT***) ->

RMStateStore中回调方法StoreAppAttemptTransition被触发,核心代码 ->

//将AppAttempt存储起来

store.storeApplicationAttemptStateInternal (attemptState.getAttemptId (), attemptStateData);

/\*\*

\* In (@link handleStoreEvent}, this method is called to notify the

\* application attempt that new attempt is stored in state store

\* **@param** appAttempt attempt that has been saved

\*/

store.notifyDoneStoringApplicationAttempt (attemptState.getAttemptId (), **null**);发送事件 RMAppAttemptEventType.***ATTEMPT\_NEW\_SAVED*** ->

触发RMAppAttemptImpl的回调方法AttempStoredTransition,方法中启动Attemp –>

@Override

**public** **void** transition(RMAppAttemptImpl appAttempt, RMAppAttemptEvent event) {

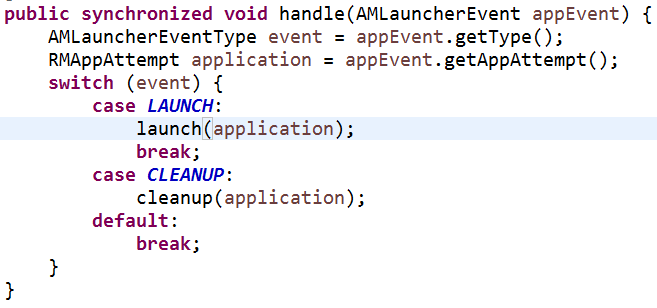
appAttempt.checkAttemptStoreError (event);

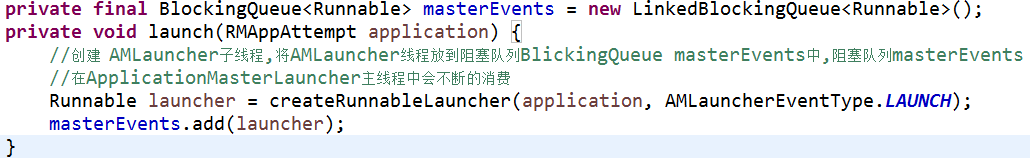
//发出事件AMLauncherEventType.LAUNCH事件

appAttempt.launchAttempt ();

}->

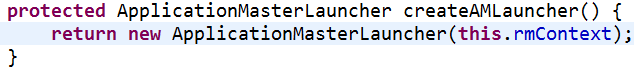
ApplicationMasterLauncher中的handle方法

 ->

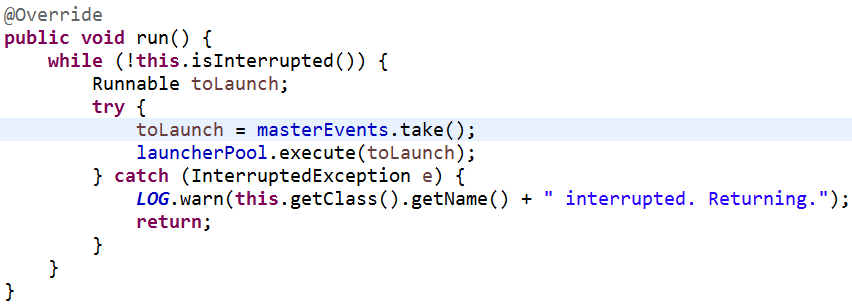


* ApplicationMasterLauncher中的执行线程类LauncherThread在主类启动时会跟随启动, ApplicationMasterLauncher

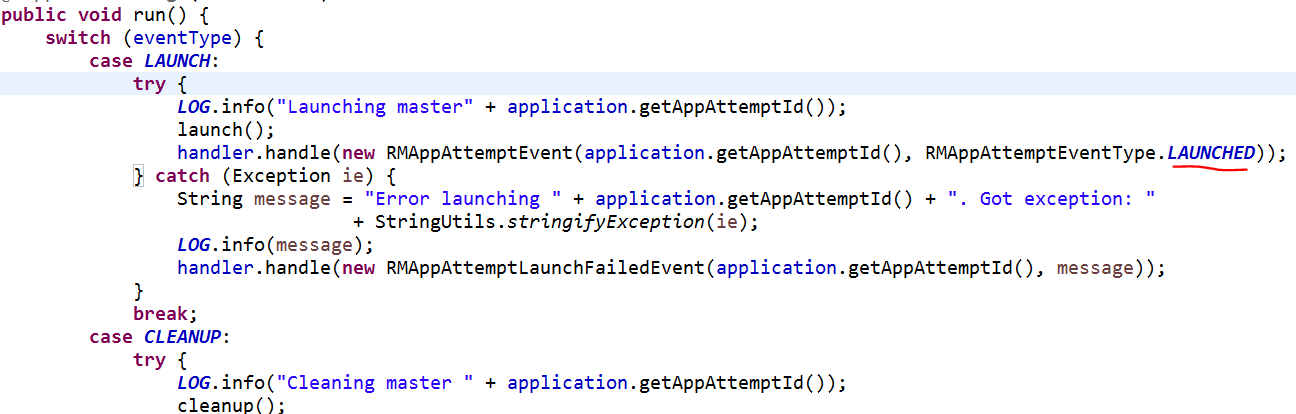
类在RM初始化启动的时候会启动,RM中代码：



LauncherThread核心代码(即消费masterEvents)：



* AMLauncher线程类:



--------------------------------------------------------------------------------------------------------------------------

Launch();方法即执行启动container,执行脚本

分析 ：

[1].Connect();根据masterContainerId创建[ContainerManagementProtocol](eclipse-javadoc:%E2%98%82=hadoop-2.5.0-cdh5.2.0/src%5C/hadoop-yarn-project%5C/hadoop-yarn%5C/hadoop-yarn-server%5C/hadoop-yarn-server-resourcemanager%5C/src%5C/main%5C/java%3Corg.apache.hadoop.yarn.server.resourcemanager.amlauncher%7BAMLauncher.java%E2%98%83AMLauncher~getContainerMgrProxy~QContainerId;%E2%98%82org.apache.hadoop.yarn.api.ContainerManagementProtocol)协议的RPC proxy(具体实例:ContainerManagerImpl),与container所 在node通信(即AM与NM通信,代表着上边的RM与AM通信完毕 ?)

[2].构建ContainerLaunchContext container启动的参数上下文及StartContainerRequest,启动Container

核心代码: StartContainersResponse response = containerMgrProxy.startContainers (allRequests);

* 调用ContainerManagerImpl中的startContainers方法

核心代码 : startContainerInternal (nmTokenIdentifier, containerTokenIdentifier, request);

* dispatcher.getEventHandler().handle(**new** ApplicationInitEvent(applicationID, appAcls));

发出ApplicationEventType.***INIT\_APPLICATION***事件

* ApplicationImpl回调方法AppInitTransition(LogHandlerEventType.***APPLICATION\_STARTED***))

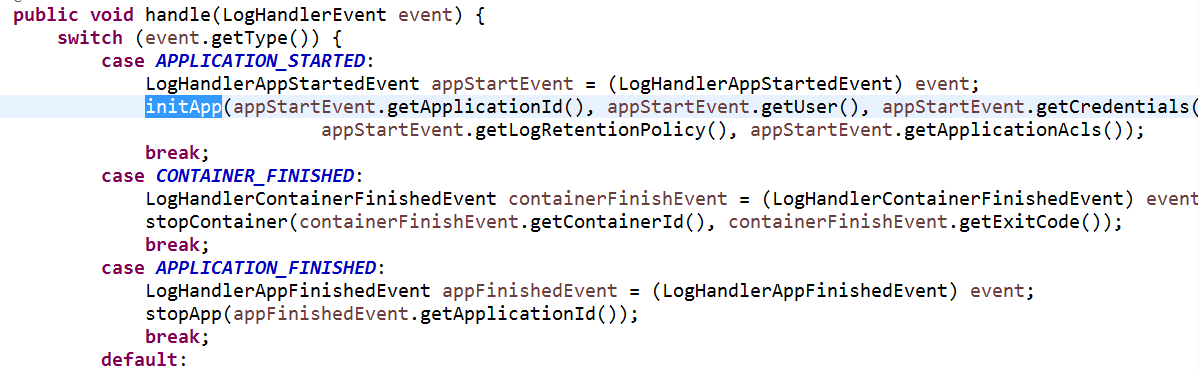
// Inform the logAggregator ---> LogAggregationService类 aggregator:聚合器

app.dispatcher.getEventHandler ().handle (

**new** LogHandlerAppStartedEvent (app.appId, app.user, app.credentials,

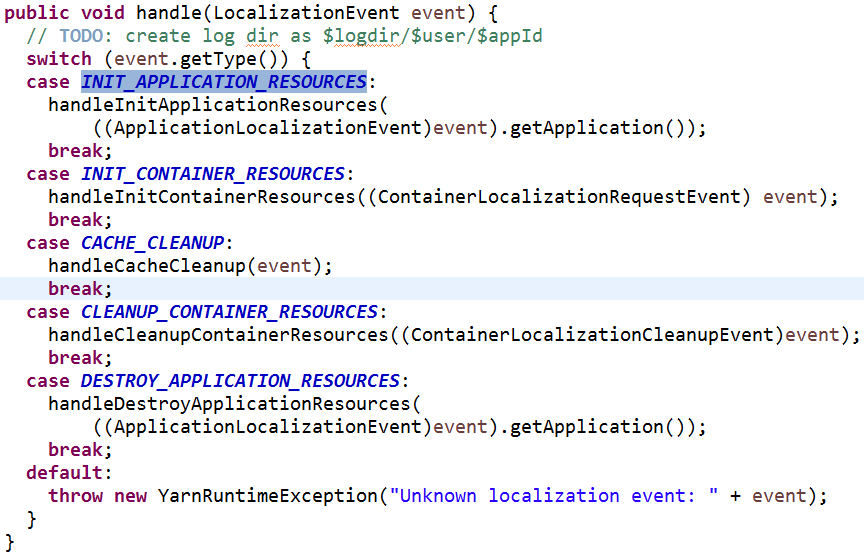
ContainerLogsRetentionPolicy.***ALL\_CONTAINERS***, app.applicationACLs));

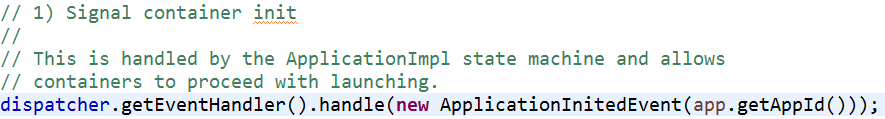
* LogAggregationService的handle方法 :

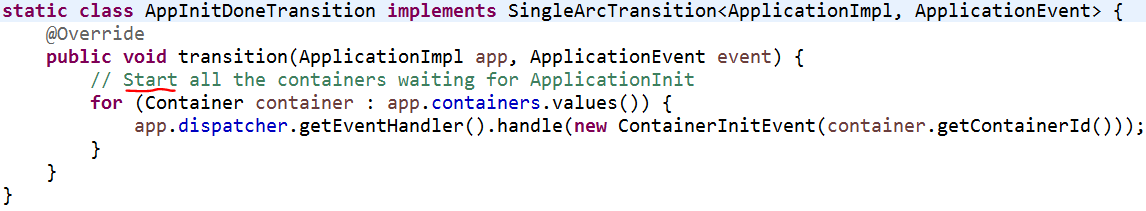


发出事件: ApplicationEventType.***APPLICATION\_LOG\_HANDLING\_INITED***

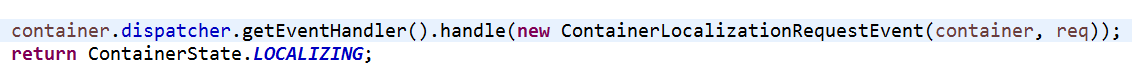
* ApplicationImpl的回调方法AppLogInitDoneTransition(LocalizationEventType.INIT\_APPLICATION\_RESOURCES)
* ResourceLocalizationService的handle方法



* 
* ApplicationImpl的回调方法AppInitDoneTransition(app状态由：ApplicationState.INITING -> ApplicationState.RUNNING)

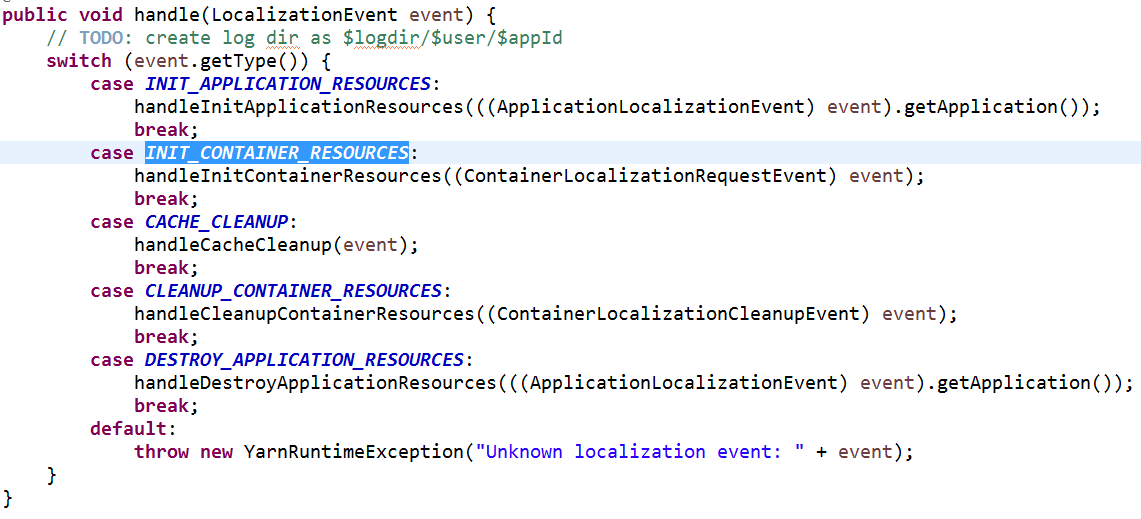


* ContainerImpl的回调方法RequestResourcesTransition



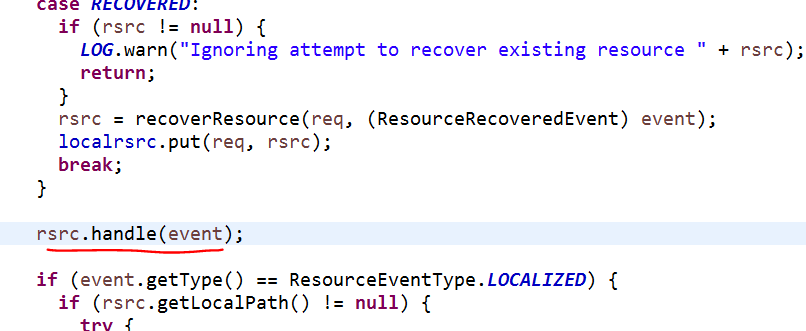
发出事件LocalizationEventType.***INIT\_CONTAINER\_RESOURCES***

* ResourceLocalizationService的handle方法

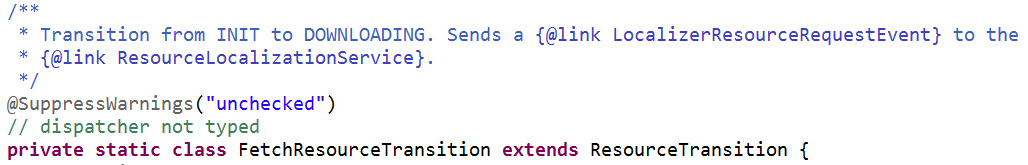


(上边状态转换就进入过此handle,处理的事件类型为第一种INIT\_APPLICATION\_RESOURCES)

* LocalResourcesTrackerImpl的handle方法



* LocalizedResource的回调方法FetchResourceTransition



发出事件 : LocalizerEventType.REQUEST\_RESOURCE\_LOCALIZATION

* ResourceLocalizationService$LocalizerTracker

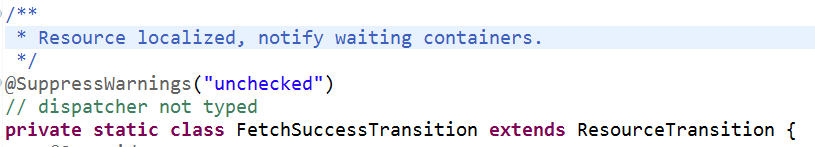


ResourceLocalizationService$PublicLocalizer.run(),等待下载完成,下载成功之后调用

publicRsrc.handle (**new** ResourceLocalizedEvent (key, local, FileUtil.*getDU* (**new** File (local.toUri ()))));

发出事件: ResourceEventType.LOCALIZED

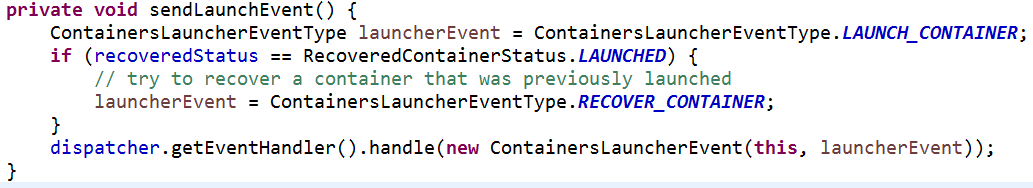
* LocalizedResource的回调方法FetchSuccessTransition



发出事件:ContainerEventType.RESOURCE\_LOCALIZED

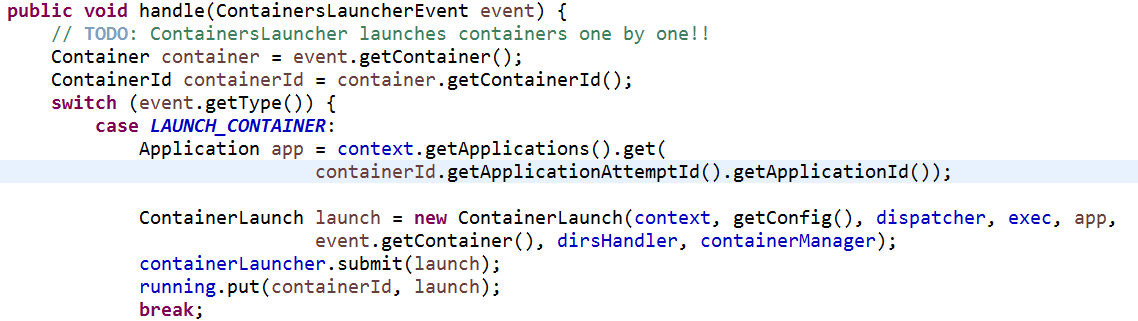
* ContainerImpl的回调方法LocalizedTransition

核心代码: container.sendLaunchEvent();



发出事件:ContainersLauncherEventType.LAUNCH\_CONTAINER(启动Container)

* ContainersLauncher的handle方法



创建ContainerLaunch线程,启动等后续操作在线程中执行

// exec是在org.apache.hadoop.yarn.server.nodemanager.NodeManager.serviceInit(Configuration)实例化的，默认为 //DefaultContainerExecutor，由参数yarn.nodemanager.container-executor.class配置

* ContainerLaunch的call方法

核心代码：

...

// LaunchContainer is a blocking call. We are here almost means the

// container is launched, so send out the event.将container放入ContainerMonitor中监控,状态由Localized->Running

dispatcher.getEventHandler().handle(**new** ContainerEvent(containerID, ContainerEventType.***CONTAINER\_LAUNCHED***));

...

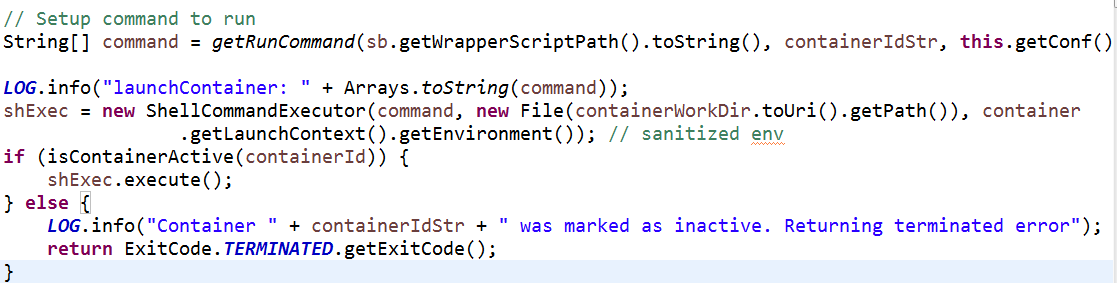
exec.activateContainer (containerID, pidFilePath);

ret = exec.launchContainer (container, nmPrivateContainerScriptPath, nmPrivateTokensPath, user,

appIdStr, containerWorkDir, localDirs, logDirs);

* DefaultContainerExecutor的launchContainer方法

核心代码：

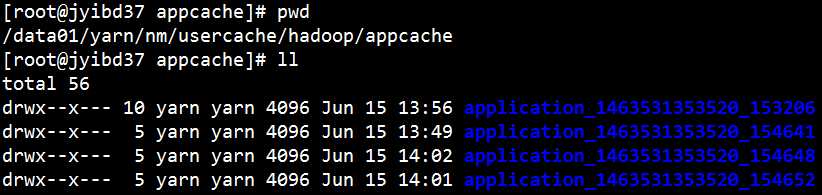


\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* 启动applicationMaster的shell脚本脚本位置位于：${yarn.nodemanager.local-dirs}(${hadoop.tmp.dir}/nm-local- \* \* dir)/usercache/${user.name}/appcache/${Application名}/${Container名}/default\_container\_executor.sh \*

\* 该脚本实际调用launch\_container.sh脚本 \*

\* 例如:我现在集群jyibd37节点上： \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*  \*

\* 启动脚本： \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

启动AM ---> **MRAppMaster(在某处已经启起来了。没找到。。。)**

* 启动YarnChild,执行具体任务(MapTask or ReduceTask)

--------------------------------------------------------------------------------------------------------------------------

发出事件:RMAppAttemptEventType.LAUNCHED

* RMAppAttemptImpl的回调方法AMLaunchedTransition

将Attempt存储起来,并将其放入AMLivelinessMonitor(RM启动时会启动此服务)中,持续的和RM发送心跳监控此AM是否还活着,并监控此任务Attempt,

执行超时做一些后续处理.