**Flink TaskManager**[](#flink-taskmanager)

* 1、Flink TaskManager启动过程
* 2、源码分析-启动（Start）
* 3、源码分析-执行task

**1、Flink TaskManager启动过程**[](#1flink-taskmanager)

**进入Flink的安装目录**[](#flink)

/usr/local/Cellar/apache-flink/1.10.0/libexec/libexec

**打开flink-deamon.sh,如下：**[](#flink-deamonsh)

**case** $DAEMON **in**

**(**taskexecutor**)**

CLASS\_TO\_RUN**=**org**.**apache**.**flink**.**runtime**.**taskexecutor**.**TaskManagerRunner

**;;**

**入口类**[](#_1)

TM的启动入口类为：org.apache.flink.runtime.taskexecutor.TaskManagerRunner

**2、源码分析-启动（Start）**[](#2-start)

*//This class is the executable entry point for the task manager in yarn or standalone mode.It constructs the related components (network, I/O manager, memory manager, RPC service, HA service)and starts them.*

TaskManagerRunner**.**java

**public** **static** **void** **main(**String**[]** args**)** **throws** Exception **{**

*// startup checks and logging*

EnvironmentInformation**.**logEnvironmentInfo**(**LOG**,** "TaskManager"**,** args**);**

SignalHandler**.**register**(**LOG**);**

JvmShutdownSafeguard**.**installAsShutdownHook**(**LOG**);**

**long** maxOpenFileHandles **=** EnvironmentInformation**.**getOpenFileHandlesLimit**();**

**if** **(**maxOpenFileHandles **!=** **-**1L**)** **{**

LOG**.**info**(**"Maximum number of open file descriptors is {}."**,** maxOpenFileHandles**);**

**}** **else** **{**

LOG**.**info**(**"Cannot determine the maximum number of open file descriptors"**);**

**}**

*//ResourceID.generate()会生成一个类似UUID*

★ runTaskManagerSecurely**(**args**,** ResourceID**.**generate**());**

**}**

**-----**》

**public** **static** **void** **runTaskManagerSecurely(**String**[]** args**,** ResourceID resourceID**)** **{**

**final** Configuration configuration **=** loadConfiguration**(**args**);**

FileSystem**.**initialize**(**configuration**,** PluginUtils**.**createPluginManagerFromRootFolder**(**configuration**));**

SecurityUtils**.**install**(new** SecurityConfiguration**(**configuration**));**

★ SecurityUtils**.**getInstalledContext**().**runSecured**(()** **->** **{**

runTaskManager**(**configuration**,** resourceID**);**

**return** **null;**

**});**

**}**

**-----**》

**public** **static** **void** **runTaskManager(**Configuration configuration**,** ResourceID resourceId**){**

*//构造TaskManagerRunner实例*

**final** TaskManagerRunner taskManagerRunner **=** **new** TaskManagerRunner**(**configuration**,** resourceId**);**

*//启动TaskManagerRunner实例*

taskManagerRunner**.**start**();**

**}**

**-----**》

**public** **void** **start()** **throws** Exception **{**

taskManager**.**start**();**

**}**

启动RPC for TM rpcService

RpcEndpoint**.**java

**public** **final** **void** **start()** **{**

rpcServer**.**start**();**

**}**

**----->**

*//Internal method which is called by the RpcService implementation to start the RpcEndpoint.*

**public** **final** **void** **internalCallOnStart()** **throws** Exception **{**

validateRunsInMainThread**();**

isRunning **=** **true;**

onStart**();**

**}**

TaskExecutor负责执行多个Task：TaskExecutor implementation. The task executor is responsible for the execution of multiple Task.

从上面的这些代码中可以看到虽然TaskManagerRunner是入口类，但是主要执行操作的是TaskExecutor类，注释里面也说明了这点。

TaskExecutor**.**java

@Override

**public** **void** **onStart()** **throws** Exception **{**

★ startTaskExecutorServices**();**

startRegistrationTimeout**();***//启动注册taskslot ，超时等*

**}**

taskSlotTable中包含一个timeService(用于超时分配slot)和一个taskSlots（TaskSlot的数组

下面启动TaskExecutor：TaskExecutor类的构造方法(**主要完成一些service的初始化**)及start方法执行完毕后，TaskManager启动完成后

**private** **void** **startTaskExecutorServices()** **throws** Exception **{**

*// start by connecting to the ResourceManager*

resourceManagerLeaderRetriever**.**start**(new** ResourceManagerLeaderListener**());**

*// tell the task slot table who's responsible for the task slot actions*

taskSlotTable**.**start**(new** SlotActionsImpl**(),** getMainThreadExecutor**());**

*// start the job leader service*

jobLeaderService**.**start**(**getAddress**(),** getRpcService**(),** haServices**,** **new** JobLeaderListenerImpl**());**

fileCache **=** **new** FileCache**(**taskManagerConfiguration**.**getTmpDirectories**(),** blobCacheService**.**getPermanentBlobService**());**

**}**

**3、源码分析-执行task**[](#3-task)

在上一篇：***Flink JobManager讲解*** SchedulingUtils.java的scheduleEager()中对调用：

**for** **(**Execution execution **:** executionsToDeploy**)** **{**

★ execution**.**deploy**();**

**}**

Execution对应的就是具体任务一次执行，在createDepolymentDescroptor()方法中转换为具体的物理部署描述。将IntermediateResultPartition转化为ResultPartition，ExecutionEdge转为InputChannelDepolymentDescriptor。调用submitTask方法实际调用的是RpcTaskManagerGateWay的方法。

deploy(): 这个方法会直接向TM提交这个task任务, 其具体逻辑如下：

Execution**.**java

**public** **void** **deploy()** **throws** JobException **{**

assertRunningInJobMasterMainThread**();**

**final** LogicalSlot slot **=** assignedResource**;**

ExecutionState previous **=** **this.**state**;**

**if** **(this.**state **!=** DEPLOYING**)** **{**

slot**.**releaseSlot**(new** FlinkException**(**"Actual state of execution " **+** **this** **+** " (" **+** state **+** ") does not match expected state DEPLOYING."**));**

**return;**

**}**

**final** TaskDeploymentDescriptor deployment **=** TaskDeploymentDescriptorFactory

**.**fromExecutionVertex**(**vertex**,** attemptNumber**)**

★★ **.**createDeploymentDescriptor**(**

slot**.**getAllocationId**(),**

slot**.**getPhysicalSlotNumber**(),**

taskRestore**,**

producedPartitions**.**values**());**

taskRestore **=** **null;**

**final** TaskManagerGateway taskManagerGateway **=** slot**.**getTaskManagerGateway**();**

**final** ComponentMainThreadExecutor jobMasterMainThreadExecutor **=**

vertex**.**getExecutionGraph**().**getJobMasterMainThreadExecutor**();**

★★ CompletableFuture**.**supplyAsync**(()** **->** taskManagerGateway**.**submitTask**(**deployment**,** rpcTimeout**),** executor**);**

**}**

*//RPCTaskManagerGateWay.java*

@Override

**public** CompletableFuture**<**Acknowledge**>** **submitTask(**TaskDeploymentDescriptor tdd**,** Time timeout**)** **{**

**return** taskExecutorGateway**.**submitTask**(**tdd**,** jobMasterId**,** timeout**);**

**}**

上面都是JobMaster(JobManager)的逻辑，通过taskExecutorGateway发送给TaskManager的TaskExecutor.submitTask()方法

TaskExecutorGateway**.**java

CompletableFuture**<**Acknowledge**>** **submitTask(**

TaskDeploymentDescriptor tdd**,**

Time timeout**);**

CompletableFuture**<**Acknowledge**>** **submitTask(**

TaskDeploymentDescriptor tdd**,**

JobMasterId jobMasterId**,**

@RpcTimeout Time timeout**);**

最后调用TaskExecutor的submitTask()具体如下：

TaskExecutor**.**java

*// 这个方法是提供给JobManager提交任务的*

@Override

**public** CompletableFuture**<**Acknowledge**>** **submitTask(**

TaskDeploymentDescriptor tdd**,**

JobMasterId jobMasterId**,**

Time timeout**)** **{**

*// tdd:A task deployment descriptor contains all the information*

*// necessary to deploy a task on a task manager.*

**final** JobID jobId **=** tdd**.**getJobId**();**

**final** JobManagerConnection jobManagerConnection **=** jobManagerTable**.**get**(**jobId**);**

**if** **(**jobManagerConnection **==** **null)** **{**

**final** String message **=** "Could not submit task because there is no JobManager " **+**

"associated for the job " **+** jobId **+** '.'**;**

log**.**debug**(**message**);**

**throw** **new** TaskSubmissionException**(**message**);**

**}**

**if** **(!**Objects**.**equals**(**jobManagerConnection**.**getJobMasterId**(),** jobMasterId**))** **{**

**final** String message **=** "Rejecting the task submission because the job manager leader id " **+**

jobMasterId **+** " does not match the expected job manager leader id " **+**

jobManagerConnection**.**getJobMasterId**()** **+** '.'**;**

log**.**debug**(**message**);**

**throw** **new** TaskSubmissionException**(**message**);**

**}**

*//分配执行资源*

**if** **(!**taskSlotTable**.**tryMarkSlotActive**(**jobId**,** tdd**.**getAllocationId**()))** **{**

**final** String message **=** "No task slot allocated for job ID " **+** jobId **+**

" and allocation ID " **+** tdd**.**getAllocationId**()** **+** '.'**;**

log**.**debug**(**message**);**

**throw** **new** TaskSubmissionException**(**message**);**

**}**

*// re-integrate offloaded data:重新加载卸载的数据*

tdd**.**loadBigData**(**blobCacheService**.**getPermanentBlobService**());**

*// deserialize the pre-serialized information*

**final** JobInformation jobInformation**;**

**final** TaskInformation taskInformation**;**

*//获取JobInformation、taskInformation*

jobInformation **=** tdd**.**getSerializedJobInformation**().**deserializeValue**(**getClass**().**getClassLoader**());**

taskInformation **=** tdd**.**getSerializedTaskInformation**().**deserializeValue**(**getClass**().**getClassLoader**());**

**if** **(!**jobId**.**equals**(**jobInformation**.**getJobId**()))** **{**

**throw** **new** TaskSubmissionException**(**

"Inconsistent job ID information inside TaskDeploymentDescriptor (" **+**

tdd**.**getJobId**()** **+** " vs. " **+** jobInformation**.**getJobId**()** **+** ")"**);**

**}**

TaskMetricGroup taskMetricGroup **=** taskManagerMetricGroup**.**addTaskForJob**(**

jobInformation**.**getJobId**(),**

jobInformation**.**getJobName**(),**

taskInformation**.**getJobVertexId**(),**

tdd**.**getExecutionAttemptId**(),**

taskInformation**.**getTaskName**(),**

tdd**.**getSubtaskIndex**(),**

tdd**.**getAttemptNumber**());**

InputSplitProvider inputSplitProvider **=** **new** RpcInputSplitProvider**(**

jobManagerConnection**.**getJobManagerGateway**(),**

taskInformation**.**getJobVertexId**(),**

tdd**.**getExecutionAttemptId**(),**

taskManagerConfiguration**.**getTimeout**());**

TaskManagerActions taskManagerActions **=** jobManagerConnection**.**getTaskManagerActions**();**

CheckpointResponder checkpointResponder **=** jobManagerConnection**.**getCheckpointResponder**();**

GlobalAggregateManager aggregateManager **=** jobManagerConnection**.**getGlobalAggregateManager**();**

LibraryCacheManager libraryCache **=** jobManagerConnection**.**getLibraryCacheManager**();**

ResultPartitionConsumableNotifier resultPartitionConsumableNotifier **=** jobManagerConnection**.**getResultPartitionConsumableNotifier**();**

PartitionProducerStateChecker partitionStateChecker **=** jobManagerConnection**.**getPartitionStateChecker**();**

**final** TaskLocalStateStore localStateStore **=** localStateStoresManager**.**localStateStoreForSubtask**(**

jobId**,**

tdd**.**getAllocationId**(),**

taskInformation**.**getJobVertexId**(),**

tdd**.**getSubtaskIndex**());**

**final** JobManagerTaskRestore taskRestore **=** tdd**.**getTaskRestore**();**

**final** TaskStateManager taskStateManager **=** **new** TaskStateManagerImpl**(**

jobId**,**

tdd**.**getExecutionAttemptId**(),**

localStateStore**,**

taskRestore**,**

checkpointResponder**);**

MemoryManager memoryManager**;**

memoryManager **=** taskSlotTable**.**getTaskMemoryManager**(**tdd**.**

*//新建一个task*

Task task **=** **new** Task**(**

jobInformation**,**

taskInformation**,**

tdd**.**getExecutionAttemptId**(),**

tdd**.**getAllocationId**(),**

tdd**.**getSubtaskIndex**(),**

tdd**.**getAttemptNumber**(),**

tdd**.**getProducedPartitions**(),**

tdd**.**getInputGates**(),**

tdd**.**getTargetSlotNumber**(),**

memoryManager**,**

taskExecutorServices**.**getIOManager**(),**

taskExecutorServices**.**getShuffleEnvironment**(),**

taskExecutorServices**.**getKvStateService**(),**

taskExecutorServices**.**getBroadcastVariableManager**(),**

taskExecutorServices**.**getTaskEventDispatcher**(),**

taskStateManager**,**

taskManagerActions**,**

inputSplitProvider**,**

checkpointResponder**,**

aggregateManager**,**

blobCacheService**,**

libraryCache**,**

fileCache**,**

taskManagerConfiguration**,**

taskMetricGroup**,**

resultPartitionConsumableNotifier**,**

partitionStateChecker**,**

getRpcService**().**getExecutor**());**

taskMetricGroup**.**gauge**(**MetricNames**.**IS\_BACKPRESSURED**,** task**::**isBackPressured**);**

log**.**info**(**"Received task {}."**,** task**.**getTaskInfo**().**getTaskNameWithSubtasks**());**

**boolean** taskAdded**;**

taskAdded **=** taskSlotTable**.**addTask**(**task**);**

**if** **(**taskAdded**)** **{**

*//启动任务*

★ task**.**startTaskThread**();**

setupResultPartitionBookkeeping**(**

tdd**.**getJobId**(),**

tdd**.**getProducedPartitions**(),**

task**.**getTerminationFuture**());**

**return** CompletableFuture**.**completedFuture**(**Acknowledge**.**get**());**

**}** **else** **{**

**final** String message **=** "TaskManager already contains a task for id " **+**

task**.**getExecutionId**()** **+** '.'**;**

log**.**debug**(**message**);**

**throw** **new** TaskSubmissionException**(**message**);**

**}**

**}**

Task**.**java

*/\*\**

*\* Starts the task's thread.*

*\*/*

**public** **void** **startTaskThread()** **{**

executingThread**.**start**();**

**}**

Task是一个线程:task代表一个在TaskManager中的并行子任务，task中已经保存了一个Flink Operator(包括用户自定义的UDF)并且会执行它,task并不会知道他们和另外的task之间的关系，他们也没有办法分辨自己是第一次执行task还尝试重复执行的（这些信息只有JobManager知道），所有的task仅仅关心自己的运行代码，task 配置、下游游需要消费的intermdeiate results的ID和自己上游的produce

*//The Task represents one execution of a parallel subtask on a TaskManager. A Task wraps a Flink operator (which may be a user function) and runs it, providing all services necessary for example to consume input data,produce its results (intermediate result partitions) and communicate with the JobManager.*

*//Tasks have no knowledge about how they relate to other tasks, or whether they are the first attempt to execute the task, or a repeated attempt. All of that is only known to the JobManager. All the task knows are its own runnable code, the task's configuration, and the IDs of the intermediate results to consume and produce (if any).*