---

toc: true title: 《从1到100深入学习Flink》—— Standalone Session Cluster 启动流程深度分析之 Task Manager 启动 date: 2019-02-02 tags:

- Flink
- 大数据
- 流式计算

前言

在文章《《从1到100深入学习Flink》—— Standalone Session Cluster 启动流程深度分析之 Job Manager 启动》中我们分析了 Standalone 模式下 Job Manager 的启动流程，查看了源码的实现步骤，这篇文章我们来查看下 Standalone 模式下 Task Manager 的启动流程，并跟着我一起来查看源码的实现过程！

入口类 TaskManagerRunner

在文章《《从1到100深入学习Flink》—— standalonesession 模式启动流程》中我们其实就找到了 Task Manager 的启动入口类为 `org.apache.flink.runtime.taskexecutor.TaskManagerRunner`。我们先来看下 main 方法如下：

```
public static void main(String[] args) throws Exception {
    // 启动检查和日志记录
    EnvironmentInformation.logEnvironmentInfo(LOG, "TaskManager",
args);
    SignalHandler.register(LOG);
    JvmShutdownSafeguard.installAsShutdownHook(LOG);

    long maxOpenFileHandles =
EnvironmentInformation.getOpenFileHandlesLimit();

    ...

    //解析参数 args
    ParameterTool parameterTool = ParameterTool.fromArgs(args);
    //获取到配置文件的目录
    final String configDir = parameterTool.get("configDir");
    //根据配置文件目录地址加载里面所有的配置
    final Configuration configuration =
GlobalConfiguration.loadConfiguration(configDir);

    try {
        //文件系统初始化
        FileSystem.initialize(configuration);
    } catch (IOException e) {}

    //安装相关的安全配置
    SecurityUtils.install(new
SecurityConfiguration(configuration));

    try {
        SecurityUtils.getInstalledContext().runSecured(new
Callable<Void>() {
            @Override
            public Void call() throws Exception {
                //最主要的方法
                runTaskManager(configuration,
ResourceID.generate());
                return null;
            }
        });
    } catch (Throwable t) {
        LOG.error("TaskManager initialization failed.", t);
        System.exit(STARTUP_FAILURE_RETURN_CODE);
    }
}
```

这个 main 方法里面的大部分功能是和之前我们看的 Job manager 启动类是类似的，唯一不同的就是这里是 `runTaskManager()` 最重要了。

下面我们就来看看这个方法：

runTaskManager() 方法

这个方法的代码如下：

```
public static void runTaskManager(Configuration configuration,
ResourceID resourceId) throws Exception {
    //根据传进来的配置和资源 id 构建了 TaskManagerRunner 实例
    final TaskManagerRunner taskManagerRunner = new
TaskManagerRunner(configuration, resourceId);
    //执行 TaskManagerRunner 中的 start 方法
    taskManagerRunner.start();
}
```

1、创建 TaskManagerRunner 实例（传入配置和 resourceId）

在这个构造器里面有个 startTaskManager 方法：

```
taskManager = startTaskManager(
    this.configuration,this.resourceId,rpcService,highAvailabilityS
ervices,
heartbeatServices,metricRegistry,blobCacheService,executor,false,th
is);
```

在 startTaskManager 方法里面根据配置来获取参数并创建 TaskExecutor 实例：

```
InetAddress remoteAddress =
InetAddress.getByName(rpcService.getAddress());

TaskManagerServicesConfiguration taskManagerServicesConfiguration =
    TaskManagerServicesConfiguration.fromConfiguration(
        configuration,remoteAddress,localCommunicationOnly);

TaskManagerServices taskManagerServices =
TaskManagerServices.fromConfiguration(

taskManagerServicesConfiguration,resourceID,rpcService.getExecutor(
),
    EnvironmentInformation.getSizeOfFreeHeapMemoryWithDefrag(),
    EnvironmentInformation.getMaxJvmHeapMemory());

TaskManagerMetricGroup taskManagerMetricGroup =
MetricUtils.instantiateTaskManagerMetricGroup(
    metricRegistry,taskManagerServices.getTaskManagerLocation(),
    taskManagerServices.getNetworkEnvironment());

TaskManagerConfiguration taskManagerConfiguration =
TaskManagerConfiguration.fromConfiguration(configuration);

return new TaskExecutor(

rpcService,taskManagerConfiguration,highAvailabilityServices,taskMa
nagerServices,

heartbeatServices,taskManagerMetricGroup,blobCacheService,executorS
ervice,fatalErrorHandler);
```

2、start 方法如下：

```
private final TaskExecutor taskManager;

public void start() throws Exception {
    taskManager.start();
}
```

TaskExecutor start 方法

内部调用了 TaskExecutor 的 start 方法:

```
public void start() throws Exception {

    //1、调用父类的 start 方法
    super.start();

    //2、连接资源管理器
    try {
        startRegistrationTimeout();
        resourceManagerLeaderRetriever.start(new
ResourceManagerLeaderListener());
    } catch (Exception e) {
        onFatalError(e);
    }

    //3、tell the task slot table who's responsible for the task
slot actions
    taskSlotTable.start(new SlotActionsImpl());

    //4、启动 job leader 服务
    jobLeaderService.start(getAddress(), getRpcService(),
haServices, new JobLeaderListenerImpl());

    //5、文件缓存
    fileCache = new
FileCache(taskManagerConfiguration.getTmpDirectories(),
blobCacheService.getPermanentBlobService());
}
```

1、super.start()

这个 start 方法是重写了 RpcEndpoint 中的 start 方法，但是为什么内部还要再调用父类的 start 方法呢？我们查看一下源代码如下：

```
protected final RpcServer rpcServer;

/**
 * Starts the rpc endpoint. This tells the underlying rpc server
that the rpc endpoint is ready
 * to process remote procedure calls.
 *
 * <p>IMPORTANT: Whenever you override this method, call the parent
implementation to enable
 * rpc processing. It is advised to make the parent call last.
 *
 * @throws Exception indicating that something went wrong while
starting the RPC endpoint
 */
public void start() throws Exception {
    rpcServer.start();
}
```

父类方法里面调用 RpcServer 的 start 方法。注意看该方法的注释，大概意思是：启动 rpc 的 endpoint，告诉底层的 rpc 服务器，rpc endpoint 已经准备好处理远程过程的调用了。无论何时重写此方法，都需要调用父类实现来启动 rpc 处理，建议将父调用放在最后！

2、连接资源管理器

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

这里先创建了 ResourceManagerLeaderListener 实例，该类中实现了 LeaderRetrievalListener 接口的 notifyLeaderAddress 方法，一旦有新的 leader 选举时，LeaderRetrievalService 将调用此方法。

重写后的 notifyLeaderAddress 方法里面会通知新的资源管理器 leader。如果连接资源管理器出现出现错误的话，则会抛出严重的异常！

3、taskSlotTable.start(new SlotActionsImpl())

先创建 SlotActionsImpl 实例，该类实现了 SlotActions，重写了其中的 freeSlot 和 timeoutSlot 方法，在 freeSlot 方法中根据给定的 allocation id 释放任务 slot。

在该方法里面根据 allocation id 来从 taskSlotTable 中找到对应的 JobID，然后根据 allocation id 释放 slot。具体代码实现如下：

```
private void freeSlotInternal(AllocationID allocationId, Throwable
cause) {
    //根据 allocationId 找到对应的 TaskSlot 的所属 JobID
    final JobID jobId = taskSlotTable.getOwningJob(allocationId);
    //根据 allocationId 释放对应的
    final int slotIndex = taskSlotTable.freeSlot(allocationId,
cause);


}


public int freeSlot(AllocationID allocationId, Throwable cause)
throws SlotNotFoundException {
    TaskSlot taskSlot = getTaskSlot(allocationId);
    if (taskSlot != null) {
        final JobID jobId = taskSlot.getJobId();
        //标志这个 slot 是 free 的
        if (taskSlot.markFree()) {
            //移除 allocation id 和 task slot 映射
            allocationIDTaskSlotMap.remove(allocationId);

            // unregister a potential timeout
            timerService.unregisterTimeout(allocationId);

            Set<AllocationID> slots = slotsPerJob.get(jobId);

            if (slots == null) {
                throw new IllegalStateException("There are no more
slots allocated for the job " + jobId +
                        ". This indicates a programming bug.");
            }
            //移除 allocationId
            slots.remove(allocationId);

            if (slots.isEmpty()) {
                slotsPerJob.remove(jobId);
            }
        }
    }
}
```

然后执行 taskSlotTable.start() 方法。

4、启动 job leader 服务

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

这里先获取到 Job Manager 的地址、RPC 服务、HA 服务、并且新建 JobLeaderListenerImpl 实例，在新建的实例中建立 JobManager 连接：

```
public void jobManagerGainedLeadership(
    final JobID jobId,
    final JobMasterGateway jobManagerGateway,
    final JMTMRegistrationSuccess registrationMessage) {
    runAsync(
        () ->
            //建立 JobManager 连接
            establishJobManagerConnection(
                jobId,
                jobManagerGateway,
                registrationMessage));
}
```

在 establishJobManagerConnection 方法中

```
private void establishJobManagerConnection(JobID jobId, final
JobMasterGateway jobMasterGateway, JMTMRegistrationSuccess
registrationSuccess) {
    // Remove pending reconnection if there is one.
    //如果存在挂起的连接，移除
    reconnectingJobManagerTable.remove(jobId);
    //如果jobManagerTable存在该jobId
    if (jobManagerTable.contains(jobId)) {
        JobManagerConnection oldJobManagerConnection =
jobManagerTable.get(jobId);

        if
(Objects.equals(oldJobManagerConnection.getJobMasterId(),
jobMasterGateway.getFencingToken())) {
            //之前已经有 job manager 连接
            log.debug("Ignore JobManager gained leadership message
for {} because we are already connected to it.",
jobMasterGateway.getFencingToken());
            return;
        } else {
            closeJobManagerConnection(jobId, new Exception("Found
new job leader for job id " + jobId + '.'));
```

```
            }
        }


        log.info("Establish JobManager connection for job {}.", jobId);
        //建立新的连接
        ResourceID jobManagerResourceID =
registrationSuccess.getResourceID();
        JobManagerConnection newJobManagerConnection =
associateWithJobManager(
                jobId,
                jobManagerResourceID,
                jobMasterGateway);

        checkArgument(jobManagerTable.add(newJobManagerConnection),
"Adding new JobManagerConnection failed.");

        //monitor the job manager as heartbeat target
        jobManagerHeartbeatManager.monitorTarget(jobManagerResourceID,
new HeartbeatTarget<AccumulatorReport>() {
            @Override
            public void receiveHeartbeat(ResourceID resourceID,
AccumulatorReport payload) {
                jobMasterGateway.heartbeatFromTaskManager(resourceID,
payload);
            }

            @Override
            public void requestHeartbeat(ResourceID resourceID,
AccumulatorReport payload) {
                // request heartbeat will never be called on the task
manager side
            }
        });

        if (taskSlotTable.getActiveSlots(jobId).hasNext()) {
            // We still have active slots, which infers that tm has
just lost connection with
            // previous jm. Thus try to report tasks execution status
for possible reconcile.
            reportTasksExecutionStatus(jobId);
        }

        offerSlotsToJobManager(jobId);
    }
```

5、FileCache

```
fileCache = new

FileCache(taskManagerConfiguration.getTmpDirectories(),
blobCacheService.getPermanentBlobService());

public FileCache(String[] tempDirectories, PermanentBlobService
blobService) throws IOException {
    this (tempDirectories, blobService,
Executors.newScheduledThreadPool(10,
            new ExecutorThreadFactory("flink-file-cache")), 5000);
}
```

新建一个 FileCache 的实例，当部署任务时，FileCache 用于为已注册的缓存文件创建本地文件。

出现异常

如果上面的 start 方法出现任何异常报错的话，那么都会向上抛出异常，最后会在 TaskManagerRunner 中捕获异常

总结

本文分析了 `org.apache.flink.runtime.taskexecutor.TaskManagerRunner` 类的源码，该类是以 Standalone 模式下启动 Task Manager 时的入口类，分析了整个类的 main 方法执行流程。