1. 上课约定须知

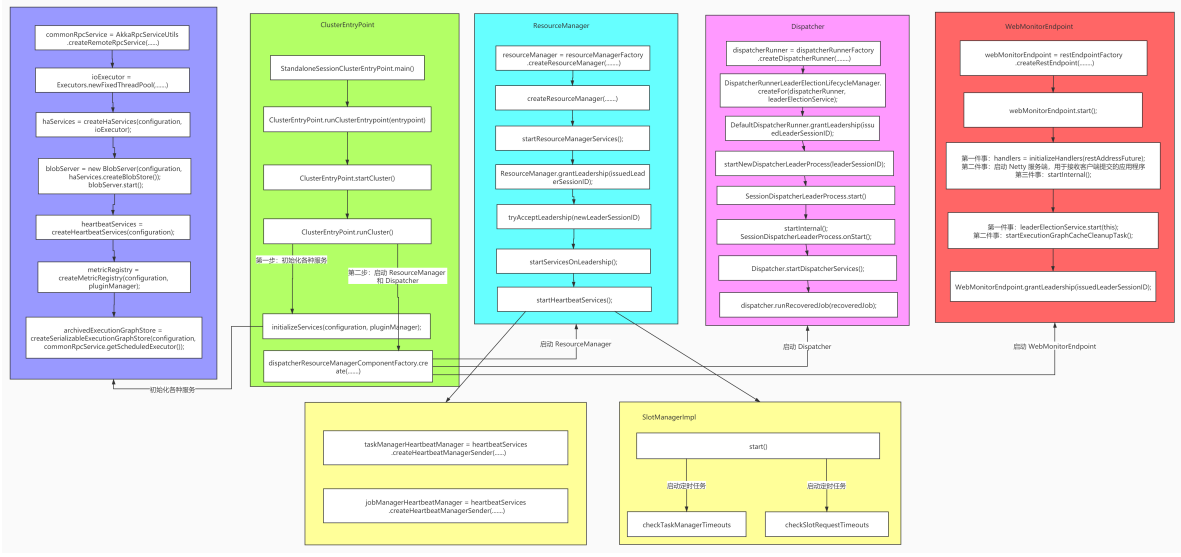
课程主题：Flink源码解析 -- 第二次课
上课时间：20:00 - 23:00
课件休息：21:30 左右 休息10分钟
课前签到：如果能听见音乐，能看到画面，请在直播间扣 666 签到

2. 上次内容总结

上次课程是 Flink 源码的第一次课程，主讲讲解的是 Flink 源码阅读的一些基础准备：Flink的Rpc 和 集群启动脚本分析，然后重点讲解 JobManager 的启动过程的源码分析，主要内容是：

```
1、Flink RPC 详解
2、Flink 集群启动脚本分析
3、Flink 主节点 JobManager 启动分析
```

其中关于 Flink 集群的 JobManager 启动源码分析，主要包括：



3. 本次内容大纲

今天主要的内容：

```
1、Flink TaskManager 启动源码分析
2、TaskManager/TaskExecutor 注册和心跳
3、Flink 编程套路总结
4、Flink 应用程序提交脚本分析
5、CliFronted 源码分析
```

4. 详细课堂内容

4.1. Flink TaskManager 启动源码分析

TaskManager 是 Flink 的 worker 节点，它负责 Flink 中本机 slot 资源的管理以及具体 task 的执行。

TaskManager 上的基本资源单位是 slot，一个作业的 task 最终会部署在一个 TM 的 slot 上运行，TM 会负责维护本地的 slot 资源列表，并来与 Flink Master 和 JobManager 通信。

根据以上的脚本启动分析：TaskManager 的启动主类： TaskManagerRunner

```
TaskManagerRunner.main()
    runTaskManagerSecurely(args, ResourceID.generate());

        # 加载配置
        Configuration configuration = loadConfiguration(args);

        # 启动 TaskManager
        runTaskManagerSecurely(configuration, resourceID);

            # 启动 TaskManager
            runTaskManager(configuration, resourceID, pluginManager);

                # 第一步：构建 TaskManagerRunner 实例
                taskManagerRunner = new TaskManagerRunner(...);

                    # 初始化一个线程池
                    this.executor = Executors.newScheduledThreadPool(....)

                    # 获取高可用模式
                    highAvailabilityServices = HighAvailabilityServicesUtils
                       .createHighAvailabilityServices(...)

                    # 创建 RPC 服务
                    rpcService = createRpcService(configuration,
                         highAvailabilityServices);

                    # 创建心跳服务
                    heartbeatServices =
HeartbeatServices.fromConfiguration(conf);
```

```
                    # 创建 BlobCacheService
                    blobCacheService = new BlobCacheService(....)

                    # 创建 TaskManager： 实际上返回的是： TaskExecutor
                    taskManager = startTaskManager(.....)

                        # 第一件大事：初始化 TaskManagerServices
                        taskManagerServices =
TaskManagerServices.fromConfiguration(...)

                            # 初始化 TaskEventDispatcher
                            taskEventDispatcher = new TaskEventDispatcher();
                            # 初始化 IOManagerASync
                            ioManager = new IOManagerAsync(...)
                            # 初始化 NettyShuffleEnvironment
                            shuffleEnvironment = createShuffleEnvironment(...)
                            # 初始化 KVStageService
                            kvStateService =
KvStateService.fromConfiguration(...)
                            # 初始化 BroadCastVariableManager
                            broadcastVariableManager = new
BroadcastVariableManager();
                            # 初始化 TaskSlotTable
                            taskSlotTable = createTaskSlotTable(...)
                            # 初始化 DefaultJobTable
                            jobTable = DefaultJobTable.create();
                            # 初始化 JobLeaderService
                            jobLeaderService = new DefaultJobLeaderService(....)
                            # 初始化 TaskStateManager
                            taskStateManager = new
TaskExecutorLocalStateStoresManager()
                            # 初始化 LibraryCacheManager
                            libraryCacheManager = new BlobLibraryCacheManager()
                            # 返回 TaskManagerServices
                            return new TaskManagerServices(....)

                        # 第二件大事： 初始化一个 TaskExecutor
                        # TaskExecutor本身是一个 RpcEndpoint，构造方法完了之后，会调
用：onStart
                        return new TaskExecutor(.....)

                            # 初始化心跳管理器：jobManagerHeartbeatManager
                            this.jobManagerHeartbeatManager =
                            createJobManagerHeartbeatManager(heartbeatServices,
                               resourceId);

                            # 初始化心跳管理器：resourceManagerHeartbeatManager
                            this.resourceManagerHeartbeatManager =

createResourceManagerHeartbeatManager(heartbeatServices,
                             resourceId);

                            # 转到 TaskExecutor 的 onStart() 方法
                            TaskExecutor.onStart();

                                # 第一步：开启相关服务
                                startTaskExecutorServices();
```

```
                                    # 第一件事：监控 ResourceManager
                                    resourceManagerLeaderRetriever.start(....);

                                    # 第二件事：启动 TaskSlotTable 服务
                                    taskSlotTable.start(...);

                                    # 第三件事：监控 JobMaster
                                    jobLeaderService.start(...)

                                    # 第四件事：启动 FileCache 服务
                                    fileCache = new FileCache(....)

                                # 第二步：开始注册超时检查： 5min
                                startRegistrationTimeout();

            # 第二步：启动 TaskManagerRunner
            taskManagerRunner.start();
```

TaskManagerRunner 的启动大致分为三类比较重要的：

```
1、一些基础服务
2、TaskManagerService
3、TaskExecutor
```

Flink 集群主从架构： JobManager TaskManager

```
Flink:  ResourceManager + TaskExecutor（负责slot的管理 + 负责task的执行）
YARN:   ResourceManager + NodeManager
    ResourceManager的内部有一个：ApplicationMaster
```

不管是主节点： JobManager 还是 从节点 TaskManager , 除了关于资源的管理和调度意外，还需要一些其他的服务

Flink 的代码中，有大量的异步编程的代码。！

```
CompletableFuture，大量提交的请求的执行，和回调的执行等等都是由线程池来搞定的！
```

```
future.xxx(() -> xxxxx(),  exceutor)
```

当一个 Flink Job 运行的时候，同样有主控程序，也有任务程序：

```
JobMaster  +  StreamTask
Driver  + Executor（Task）  MapReduce： Task进程级别！新版本也改成了线程级别！
```

JobManager ResourceManager JobMaster

```
startJobManager() ---> new JobMaster()
```

如果你把 Job 提交给 YARN 去运行的时候： Per-Job （JobManager）类似于 Standalone 集群，ResourceManager Dispatcher 等都启动在 JobManager(JobMaster) 里面

4.2. TaskManager/TaskExecutor 注册和心跳

总结一句最重要的精髓：TaskManager 是一个逻辑抽象，代表一台服务器，这条服务器的启动，必然会包含一些服务，另外再包含一个 TaskExecutor，存在于TaskManager的内部，真实的帮助 TaskManager 完成各种核心操作：

```
1、提交Task执行
2、申请和释放slot
```

核心入口为：resourceManagerLeaderRetriever.start(....);

具体见源码注释。

4.3. Flink编程套路总结

Flink 底层提供了一个功能完善且复杂的分布式流式计算引擎，但是上层的应用 API 却很简单，简单来说，把整个 Flink 应用程序的编写，抽象成三个方面：

- 执行环境 ExectionEnvironment
- 数据抽象 DataSet DataStream
- 逻辑操作 Source Transformation Sink

所以 Flink 的应用程序在编写的时候，基本是一个简单的统一套路：

```
1、获取执行环境对象
   StreamExecutionEnvironment env =
StreamExecutionEnvironment.getExecutionEnvironment();
2、通过执行环境对象，注册数据源Source，得到数据抽象
   DataStream ds = env.socketTextStream(...)
3、调用数据抽象的各种Transformation执行逻辑计算
   DataStream resultDS = ds.flatMap(...).keyBy(...).sum(...);
4、将各种Transformation执行完毕之后得到的计算结果数据抽象注册 Sink
   resultDS.addSink(...)
5、提交Job执行
   env.execute(...)
```

基本路数，和 Spark 一致，并且，在 Flink-1.12 版本中， DataStream 已经具备高效批处理操作处理了。更加做到了流批处理的统一（API统一）。

4.4. Flink提交脚本解析

当编写好 Flink 的应用程序，正常的提交方式为：打成jar包，通过 flink 命令来进行提交。

flink 命令脚本的底层，是通过 java 命令启动：CliFrontend 类 来启动 JVM 进程执行任务的构造和提交。

4.5. CliFronted 提交分析

当用户把 Flink 应用程序打成 jar 使用 flink run ... 的 shell 命令提交的时候，底层是通过 CliFronted 来处理。底层的逻辑，就是通过反射来调用用户程序的 main() 方法执行。

5. 本次课程总结

本次课程的主要内容：TaskManager 的启动

```
1、Flink TaskManager 启动源码分析
2、TaskManager/TaskExecutor 注册和心跳
3、Flink 编程套路总结
4、Flink 应用程序提交脚本分析
5、CliFronted 源码分析
```

到此为止，把 Flink 的 Standalone 集群启动的流程基本讲完。最终，总结，其实不管是主节点 JobManager 还是 TaskManager 在启动过程中，都是提前启动一些服务来为将来的 Job 提交执行做准备。

大致总结一下：

JobManager 的核心作用：

```
1、启动了 WebMonitorEndppint 来接收客户端的 rest 请求
2、启动了 ResourceManager 来管理集群的所有资源，资源使用 slot 的抽象来进行管理
3、启动了 Dispatcher 来负责调度 Job 执行
```

TaskManager 的核心作用：

```
1、启动 TaskManagerService，其实是启动很多服务组件
2、启动 TaskExecutor，进行资源抽象封装和注册，并维持心跳
```

其实 TaskManager 还有其他重要的作用，比如：

```
1、Slot 资源管理
2、Task 提交和执行
3、Checkppoint 触发执行
```

等这些工作机制的源码分析，会在 Job 提交和执行的时候，再讲！