1. 上课约定须知
2. 上次内容总结
3. 本次内容大纲
4. 详细课堂内容
    4. 1. Flink RPC 详解
    4. 2. Flink 集群启动脚本分析
    4. 3. Flink 主节点 JobManager 启动分析
    4. 4. Flink 从节点 TaskManager 启动分析
5. 本次课程总结
6. 本次课程作业

1. 上课约定须知

课程主题：Flink 源码解析 -- 第一次课（集群启动）
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

上一次课程是 ClickHouse 进阶内容，主要讲解两个方面的知识：

```
1、OLAP 分析引擎的架构设计技术点头脑风暴
2、ClickHouse 的 MergeTree 表引擎底层原理
    分区规则
    分区合并规则
    数据压缩规则
    一级索引
    二级索引
    数据标记
    数据读写
3、ClickHouse 应用案例（去重，求join）
    1、HLL算法 近似去重
    2、BitMap 操作精确去重
    3、BitMap 求and or xor等操作
```

3. 本次内容大纲

今天是 Flink 源码的第一次课程，主要讲解的是 Flink 的集群启动，主要内容分为以下四点：

```
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
4、Flink 从节点 TaskManager 启动分析
```

4. 详细课堂内容

4.1. Flink RPC 详解

学到现在，看了很多技术组件的源码，大致总结一下各大组件 RPC 实现：

技术组件	RPC 实现
HDFS	Netty
HBase	HBase-2.x 以前：NIO + ProtoBuf HBase-2.x 以后：Netty
ZooKeeper	BIO + NIO + Netty
Spark	Spark-1.x 基于 Akka Spark-2.x 基于 Netty
Flink	Akka + Netty

总结：Flink 的 RPC 实现：基于 Scala 的网络编程库： Akka

几个储备知识：

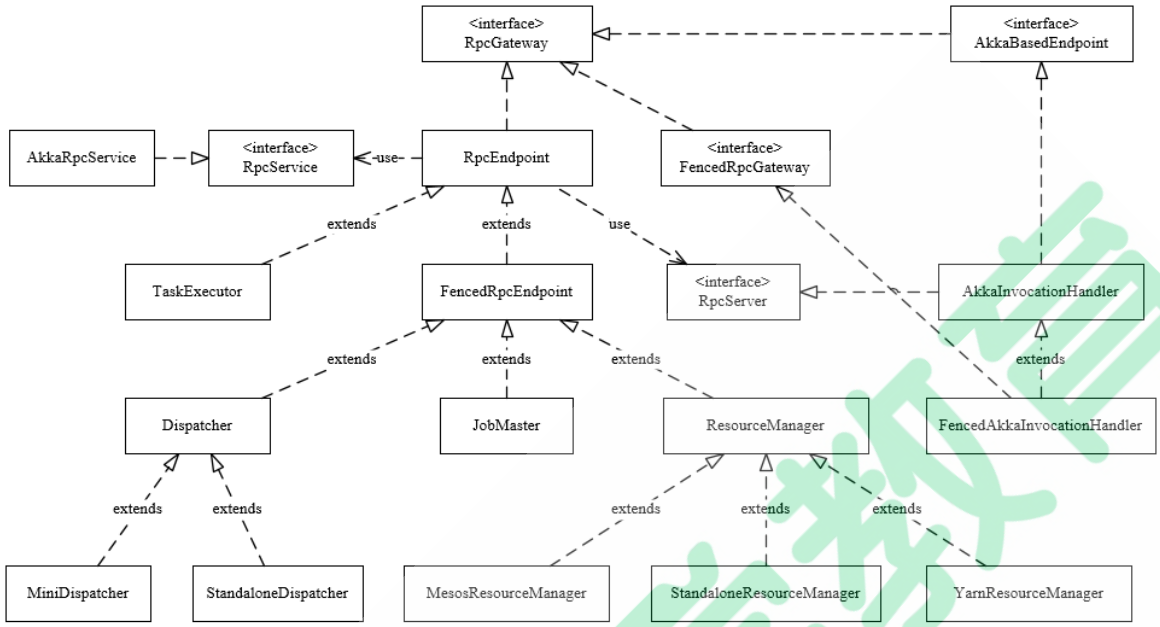
```
1、在 Spark 第一节源码课中，讲解了一个使用 Akka 来模拟实现 YARN 的程序。
2、ActorSystem 和 Actor/ActorRef 的概念，工作方式，通信方式等，注意异步和 mailbox 这两个概念。
```

总结一下：

```
1、ActorSystem 是管理 Actor生命周期的组件， Actor是负责进行通信的组
2、每个 Actor 都有一个 MailBox，别的 Actor 发送给它的消息都首先储存在 MailBox 中，通过这种方式可以实现异步通信。
3、每个 Actor 是单线程的处理方式，不断的从 MailBox 拉取消息执行处理，所以对于 Actor 的消息处理，不适合调用会阻塞的处理方法。
4、Actor 可以改变他自身的状态，可以接收消息，也可以发送消息，还可以生成新的 Actor
5、每一个ActorSystem 和 Actor都在启动的时候会给定一个 name，如果要从ActorSystem中，获取一个 Actor，则通过以下的方式来进行 Actor的获取：
akka.tcp://asname@bigdata02:9527/user/actorname
6、如果一个 Actor 要和另外一个 Actor进行通信，则必须先获取对方 Actor 的 ActorRef 对象，然后通过该对象发送消息即可。
7、通过 tell 发送异步消息，不接收响应，通过 ask 发送异步消息，得到 Future 返回，通过异步回到返回处理结果。
```

Flink 中的 RPC 实现主要在 `flink-runtime` 模块下的 `org.apache.flink.runtime.rpc` 包中，涉及到的最重要的 API 主要是以下这四个：

```
1、RpcGateway     路由，RPC的老祖宗，各种其他RPC组件，都是 RpcGateWay 的子类
2、RpcServer      RpcService 和 RpcEndpoint 之间的粘合层
3、RpcEndpoint    业务逻辑载体，对应的 Actor 的封装
4、RpcService     对应 ActorSystem 的封装
```



最终总结一下：RpcEndpoint 下面有四个比较重要的子类：

```
1、TaskExecutor
2、Dispatcher
3、JobMaster
4、ResourceManager
```

当在任意地方发现要创建这四个组件的任何一个组件的实例对象的时候，创建成功了之后，都会要去执行他的 onStart()，在集群启动的源码分析中，其实这些组件的很多的工作流程，都被放在 onStart() 里面。

4.2. Flink 集群启动脚本分析

Flink 集群的启动脚本在：flink-dist 子项目中，位于 flink-bin 下的 bin 目录：启动脚本为：start-cluster.sh

该脚本会首先调用 config.sh 来获取 masters 和 workers，masters 的信息，是从 conf/masters 配置文件中获取的，workers 是从 conf/workers 配置文件中获取的。然后分别：

```
1、通过 jobmanager.sh 来启动 JobManager
2、通过 taskmanager.sh 来启动 TaskManager
```

他们的内部，都通过 flink-daemon.sh 脚本来启动 JVM 进程，分析 flink-daemon.sh 脚本发现：

```
1、JobManager 的启动代号：standalonesession，实现类是：
StandaloneSessionClusterEntrypoint
2、TaskManager 的启动代号：taskexecutor，实现类是：TaskManagerRunner
```

HDFS集群一样的方式：

```
1、start-all.sh
2、hadoop-daemon.sh start namenode/datanode/zkfc/journalnode
3、java org.apache.hadoop.server.namenode.NameNode
```

4.3. Flink 主节点 JobManager 启动分析

JobManager 是 Flink 集群的主节点，它包含三大重要的组件：

```
1、ResourceManager
    Flink的集群资源管理器，只有一个，关于slot的管理和申请等工作，都由他负责
2、Dispatcher
    负责接收用户提交的 JobGragh，然后启动一个 JobManager， 类似于 YARN 集群中的 AppMaster 角色，类似于 Spark Job 中的 Driver 角色
3、JobManager
    负责一个具体的 Job 的执行，在一个集群中，可能会有多个 JobManager 同时执行，类似于 YARN 集群中的 AppMaster 角色，类似于 Spark Job 中的 Driver 角色
4、WebMonitorEndpoint
    里面维护了很多很多的Handler，如果客户端通过 flink run 的方式来提交一个 job 到 flink 集群，最终，是由 WebMonitorEndpoint 来接收，并且决定使用哪一个 Handler 来执行处理
    submitJob ===> SubmitJobHandler
```

总结一下：

```
Flink 集群的主节点内部运行着：ResourceManager 和 Dispatcher，当 client 提交一个 job 到集群运行的时候（客户端会把该 Job 构建成一个 JobGragh 对象），Dispatcher 负责拉起 JobManager 来负责这个 Job 内部的 Task 的执行，执行Task所需要的资源，JobManager 向 ResourceManager 申请。
```

根据以上的启动脚本分析：JobManager的启动主类：StandaloneSessionClusterEntrypoint

```
# 入口
StandaloneSessionClusterEntrypoint.main()
    ClusterEntrypoint.runClusterEntrypoint(entrypoint);
        clusterEntrypoint.startCluster();
            runCluster(configuration, pluginManager);

                # 第一步：初始化各种服务
                initializeServices(configuration, pluginManager);

                # 创建 DispatcherResourceManagerComponentFactory，初始化各种组件的工厂实例
                createDispatcherResourceManagerComponentFactory(configuration);

                # 创建 集群运行需要的一些组件：Dispatcher， ResourceManager 等
                clusterComponent = dispatcherResourceManagerComponentFactory.create(...)
```

第一步 initializeServices() 中做了很多服务组件的初始化：

```
# 初始化和启动 AkkaRpcService，内部其实包装了一个 ActorSystem
commonRpcService = AkkaRpcServiceUtils.createRemoteRpcService(...)
```

```
# 初始化一个负责 IO 的线程池
ioExecutor = Executors.newFixedThreadPool(...)

# 初始化 HA 服务组件，负责 HA 服务的是：ZooKeeperHaServices
haServices = createHaServices(configuration, ioExecutor);

# 初始化 BlobServer 服务端
blobServer = new BlobServer(configuration, haServices.createBlobStore());
blobServer.start();

# 初始化心跳服务组件，heartbeatServices = HeartbeatServices
heartbeatServices = createHeartbeatServices(configuration);

# 初始化一个用来存储 ExecutionGraph 的 Store，实现是：FileArchivedExecutionGraphStore
archivedExecutionGraphStore = createSerializableExecutionGraphStore(...)
```

第二步 createDispatcherResourceManagerComponentFactory(configuration) 中负责初始化了很多组件的工厂实例：

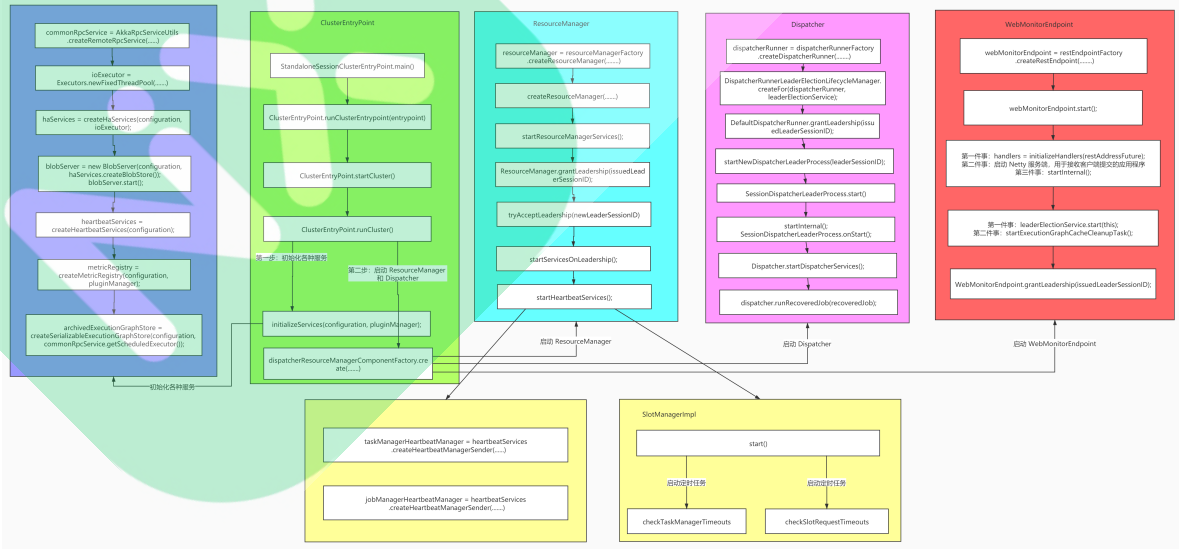
```
1、DispatcherRunnerFactory，默认实现：DefaultDispatcherRunnerFactory
2、ResourceManagerFactory，默认实现：StandaloneResourceManagerFactory
3、RestEndpointFactory，默认实现：SessionRestEndpointFactory
```

其中，DispatcherRunnerFactory 内部也实例化了一个：SessionDispatcherLeaderProcessFactoryFactory 组件

第三步 dispatcherResourceManagerComponentFactory.create(...) 中主要去创建 三个重要的组件：

```
1、DispatcherRunner，实现是：DispatcherRunnerLeaderElectionLifecycleManager
2、ResourceManager，实现是：StandaloneResourceManager
3、WebMonitorEndpoint，实现是：DispatcherRestEndpoint
```

最终总结一下：



4.4. Flink 从节点 TaskManager 启动分析

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);

                # 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
                    taskManager = startTaskManager(.....)

                        # 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
                            return new TaskManagerServices(....)

                        # 初始化一个 TaskExecutor
                        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();
                                startTaskExecutorServices();

                # 启动 TaskManagerRunner
                taskManagerRunner.start();
```

5. 本次课程总结

本次课程，主要讲解集群的启动，在启动过程中，会有各种服务组件的初始化工作

6. 本次课程作业