# 1. 上课约定须知

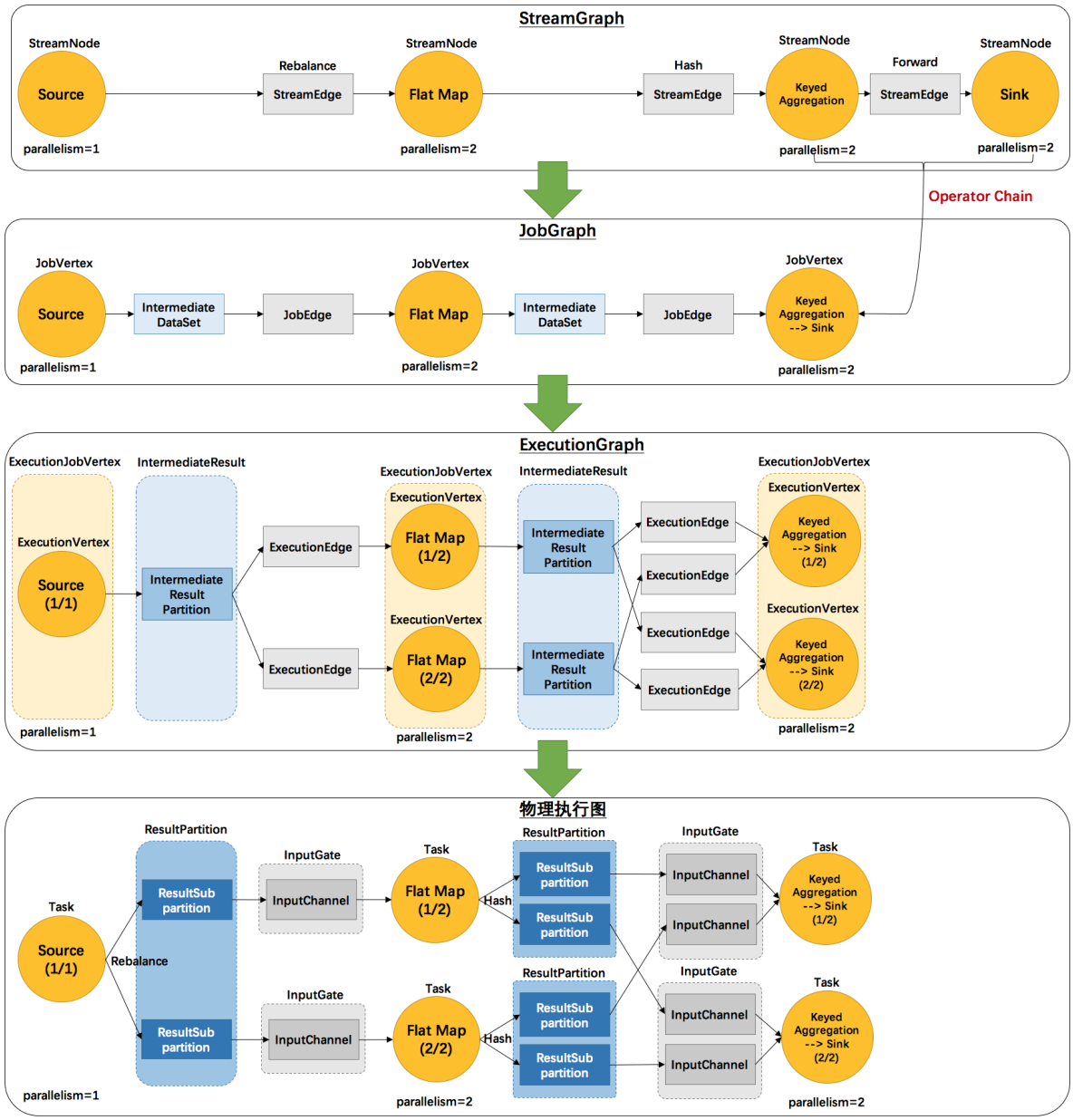
课程主题：Flink源码解析 -- 第四次课
上课时间：20:00 - 23:00
课件休息：21:30 左右 休息10分钟
课前签到：如果能听见音乐，能看到画面，请在直播间扣 666 签到

# 2. 上次内容总结

今天的课程是 Flink 源码剖析 的第四次：主要内容是 Flink 应用程序的提交，下次课是应用程序执行的一些核心细节。包含 Graph 的处理，最终生成 物理执行图，以及申请 Slot 到发布 Task 到 TaskExecutor 上运行。其中后半部分，关于 Slot 的申请并未讲到。这次搞定！

```
1、Flink 编程套路总结
2、Flink 提交执行脚本分析
3、Flink CliFronted 提交应用程序源码剖析
4、ExecutionEnvironment 源码解析
5、Job 提交流程源码分析
6、StreamGraph 构建和提交源码解析
7、JobGragh 构建和提交源码解析
8、ExecutionGragh 构建和提交源码解析
```

最好理解这张图：




# 3. 本次内容大纲

今天主要讲解的是，当 FLink 应用程序提交之后，应用程序到底是怎么执行的，以及执行过程中的，一些核心细节。

```
1、Flink 集群服务端处理 JobSubmit
2、Slot 管理（申请）源码解析
3、Task 提交到 TaskExecutor 执行源码剖析
4、StreamTask 执行
5、MailBox 线程模型剖析
6、Task 数据传输源码剖析
```

State管理 Checkpoint流程

# 4. 详细课堂内容

第一次 第二次： Flink 启动了一些必要的服务组件，当 client 提交job 进来的时候，这些 组件都参与作用

```
1、主节点： JobManager
    WebMonitorEndpoint： 维护了一个 netty 服务端，client 通过 RestClient 提交job（JobSubmitHandler）
    ResourceManager： 资源集群的主节点
    Dispatcher： job的调度执行

2、从节点： TaskManager
    TaskExecutor： 提供计算资源，注册给 ResourceManager，维持心跳，执行 JobMaster发送给他的要执行的 Task
```

# 4.1. WebMonitorEndpoint 处理 RestClient 的 JobSubmit 请求

最终处理这个请求： JobSubmitHandler 来处理！

核心入口：

```
// JobManager 服务端处理入口
JobSubmitHandler.handleRequest();

    // 恢复得到 JobGraph
    JobGraph jobGraph = loadJobGraph(requestBody, nameToFile);

    // 通过 Dispatcher 提交 JobGraph
    Dispatcher.submitJob(jobGraph, timeout)
```

Dispatcher 的提交执行逻辑：

```
Dispatcher.persistAndRunJob()

    // 保存 JobGraph 在 ZK 上
    jobGraphWriter.putJobGraph(jobGraph);

    // 提交 JobGraph 执行
    Dispatcher.runJob(jobGraph);

        // 第一件事：主要的事情，是创建 JobMaster
        Dispatcher.createJobManagerRunner(jobGraph);
            // 初始化 new JobManagerRunnerImpl
            new JobManagerRunnerImpl()
                // 初始化 JobMaster，把 JobGraph 转变成 ExecutionGraph 当前整个job到底需要多少资源
                new JobMaster(...)
                    // 创建 DefaultScheduler
                    this.schedulerNG = createScheduler(jobManagerJobMetricGroup);
                        schedulerNGFactory.createInstance(...)
                            new DefaultScheduler()
                                super()
```

```
this.executionGraph=createAndRestoreExecutionGraph()
                                        // 重点：创建 ExecutionGraph
                                        newExecutionGraph =
createExecutionGraph()
                    // 然后跳转到 JobMaster 的 onStart() 方法。
                    onStart()

        // 第二件事：启动 JobMaster
        Dispatcher.startJobManagerRunner()
            jobManagerRunner.start();
                leaderElectionService.start(this);
                    ZooKeeperLeaderElectionService.isLeader()

JobManagerRunnerImpl.grantLeadership(issuedLeaderSessionID);

verifyJobSchedulingStatusAndStartJobManager(leaderSessionID);
                                startJobMaster(leaderSessionId);
```

接上面的继续：

```
startJobMaster(leaderSessionId);

    runningJobsRegistry.setJobRunning(jobGraph.getJobID());
    jobMasterService.start(new JobMasterId(leaderSessionId));

        // 内部主要做两件事：
        // startJobMasterServices();
        // resetAndStartScheduler();
        startJobExecution(newJobMasterId), RpcUtils.INF_TIMEOUT);

            // 第一件事：跑起来 JobMaster 相关的服务，主要是注册 和 心跳
            startJobMasterServices();

                startHeartbeatServices();
                slotPool.start(getFencingToken(), getAddress(),
getMainThreadExecutor());
                scheduler.start(getMainThreadExecutor());
                reconnectToResourceManager(new FlinkException("..."));

                    closeResourceManagerConnection(cause);
                    tryConnectToResourceManager();

                        connectToResourceManager();

                            resourceManagerConnection = new
ResourceManagerConnection()
                            resourceManagerConnection.start();

                                createNewRegistration();
                                newRegistration.startRegistration();

                                    register(...)
                                        invokeRegistration(...);
                                            gateway.registerJobManager(...)
                                                // 完成注册
                                                registerJobMasterInternal()
```

```
                // 监听 ResourceManager
                resourceManagerLeadRetriever.start(new
ResourceManagerLeaderListener());

            // 第二件事：开始申请 Slot，并且部署 Task
            resetAndStartScheduler();

                JobMaster.startScheduling()

                    schedulerNG.startScheduling();

                        // 启动所有的服务协调组
                        startAllOperatorCoordinators();
                        // 开始调度
                        startSchedulingInternal();

                            prepareExecutionGraphForNgScheduling();
                            schedulingStrategy.startScheduling();

                                allocateSlotsAndDeploy(...)
```

接着继续：

```
allocateSlotsAndDeploy(...)

    DefaultScheduler.allocateSlotsAndDeploy(executionVertexDeploymentOptions);

        // 申请 Slot
        allocateSlots(executionVertexDeploymentOptions);

        // 部署 Task 运行
        waitForAllSlotsAndDeploy(deploymentHandles);
```

## 4.2. Slot 管理（申请和释放）源码解析

核心入口：allocateSlots(executionVertexDeploymentOptions);

大体上，分为四个大步骤：

```
1、JobMaster 发送请求申请 slot
2、ResourceManager 接收到请求，执行 slot请求处理
3、TaskManager 处理 ResourceManager 发送过来的 Slot 请求
4、JobMaster 接收到 TaskManager 发送过来的 Slot 申请处理结果
```

对应的一些组件：

```
1、ResourceManager 管理所有的 TaskManager（TaskExecutor）
2、TaskExecutor 中关于资源的管理，使用 slot的抽象：
    slot 的状态管理
    专门有一个 做 slot 管理的 SlotManagerImpl
3、JobMaster 申请slot, 管理组件：SlotPool
    slot共享！ 既然有 slot 共享的概念，如果要执行一个 Task,其实就可以先尝试从 SlotPool 中申请，如果申请不到，则再向 ResourceManager 申请
```

接下来看 TaskManager 的 slot 管理：

```
// JobMaster 发送请求申请 slot
DefaultScheduler.allocateSlots();
DefaultExecutionSlotAllocator.allocateSlotsFor();
NormalSlotProviderStrategy.allocateSlot();
SchedulerImpl.allocateSlot();
SchedulerImpl.allocateSlotInternal();
SchedulerImpl.internalAllocateSlot();
SchedulerImpl.allocateSingleSlot();
SchedulerImpl.requestNewAllocatedSlot();
SlotPoolImpl.requestNewAllocatedBatchSlot();
SlotPoolImpl.requestNewAllocatedSlotInternal();
SlotPoolImpl.requestSlotFromResourceManager();

// ResourceManager 接收到请求，执行 slot请求处理
ResourceManager.requestSlot();
SlotManagerImpl.registerSlotRequest();
SlotManagerImpl.internalRequestSlot();
SlotManagerImpl.allocateSlot();
TaskExecutorGateway.requestSlot();

// TaskManager 处理 ResourceManager 发送过来的 Slot 请求
TaskExecutor.requestSlot();
TaskExecutor.offerSlotsToJobManager();
TaskExecutor.internalOfferSlotsToJobManager();
JobMasterGateway.offerSlots();

// JobMaster 接收到 TaskManager 发送过来的 Slot 申请处理结果
JobMaster.offerSlots();
SlotPoolImpl.offerSlots
```

## 4.3. Task 部署和提交

核心入口是：

```
waitForAllSlotsAndDeploy(deploymentHandles);
```

详细流程：

```
DefaultScheduler.waitForAllSlotsAndDeploy();
DefaultScheduler.deployAll();
DefaultScheduler.deployOrHandleError();
DefaultScheduler.deployTaskSafe();
DefaultExecutionVertexOperations.deploy();
ExecutionVertex.deploy();
Execution.deploy();
TaskDeploymentDescriptorFactory.createDeploymentDescriptor();
RpcTaskManagerGateway.submitTask();
TaskExecutor.submitTask();
```

# 5. 本次课程总结

本次课程主要讲解的是：Task 的 Slot 申请和 Task 的部署运行。