Spark进阶

Spark核心原理加强

●课程说明

Spark源码从1.x的40w行发展到现在的超过100w行，有1400多位大牛贡献了代码。整个Spark框架源码是一个巨大的工程，

今天的课程会涉及到一小部分底层源码，有一定难度，大家做好心理准备，尽量跟着老师的思路去耐下心观摩欣赏一下大牛的杰作，更重要的是学习阅读源码的步骤、方法和技巧，为以后进阶为大数据高端人才做准备！

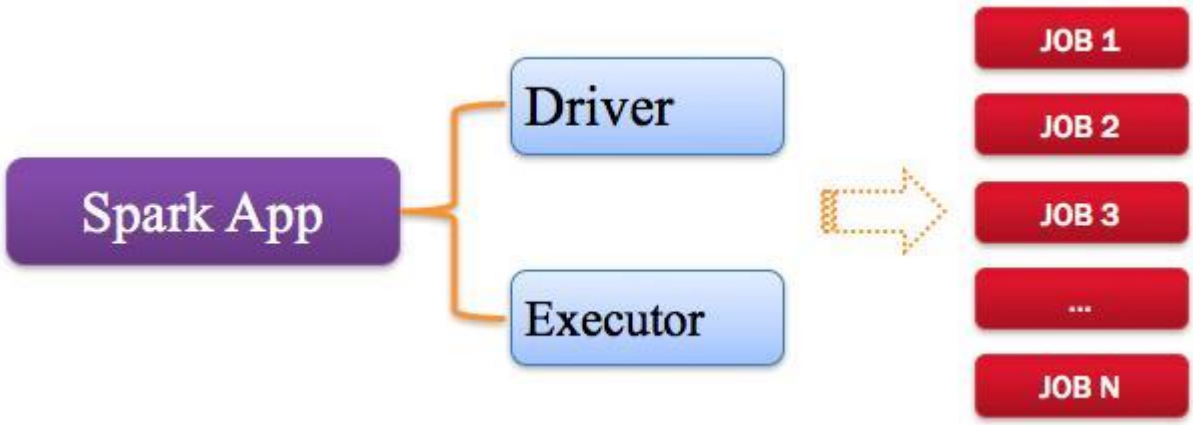
当然内功并不是一日练成的，工作以后还是希望大家能够一直保持对于技术的热情和好奇心，在技术的道路上发展的越来越好！

1. Spark专业术语定义-[掌握]

1.1 Application/App：Spark应用程序

指的是用户编写的Spark应用程序/代码，包含了Driver功能代码和分布在集群中多个节点上运行的Executor代码。

Spark应用程序，由一个或多个作业JOB组成(因为代码中可能会调用多次Action)，如下图所示:



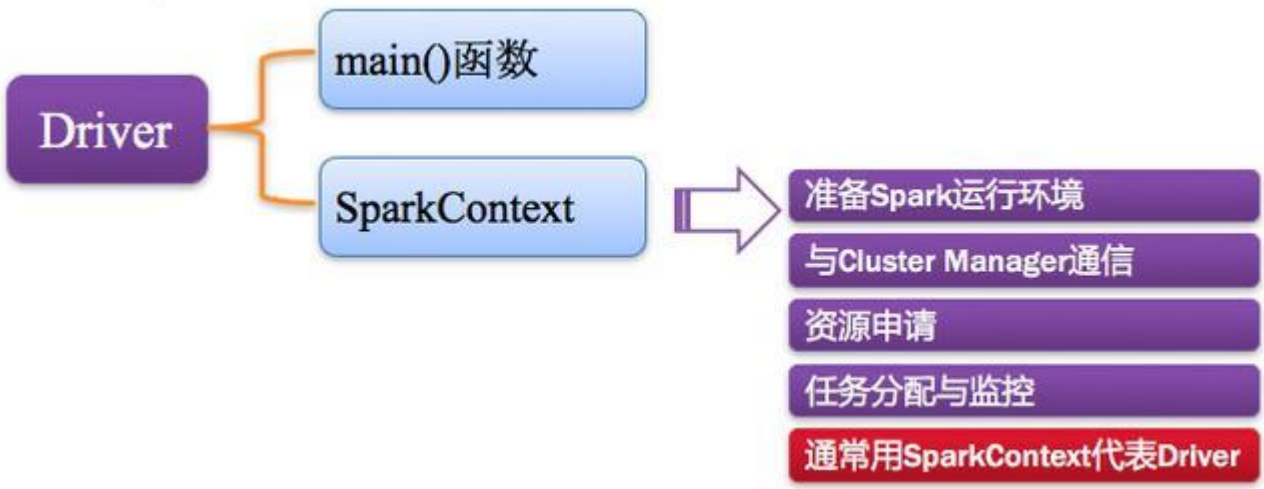
1.2 Driver：驱动程序

Spark中的Driver即运行上述Application的Main()函数并且创建SparkContext，其中创建SparkContext的目的是为了准备Spark应用程序的运行环境。

在Spark中由SparkContext负责和ClusterManager通信，进行资源的申请、任务的分配和监控等;

当Executor部分运行完毕后，Driver负责将SparkContext关闭。

通常SparkContext代表Driver，如下图所示:



1.3 Cluster Manager：资源管理器

指的是在集群上获取资源的外部服务，常用的有：

Standalone，Spark原生的资源管理器，由Master负责资源的分配;

Haddop Yarn，由Yarn中的ResourcesManager负责资源的分配;

Messos，由Messos中的Messos Master负责资源管理，

如下图所示:

1.4 Worker：计算节点

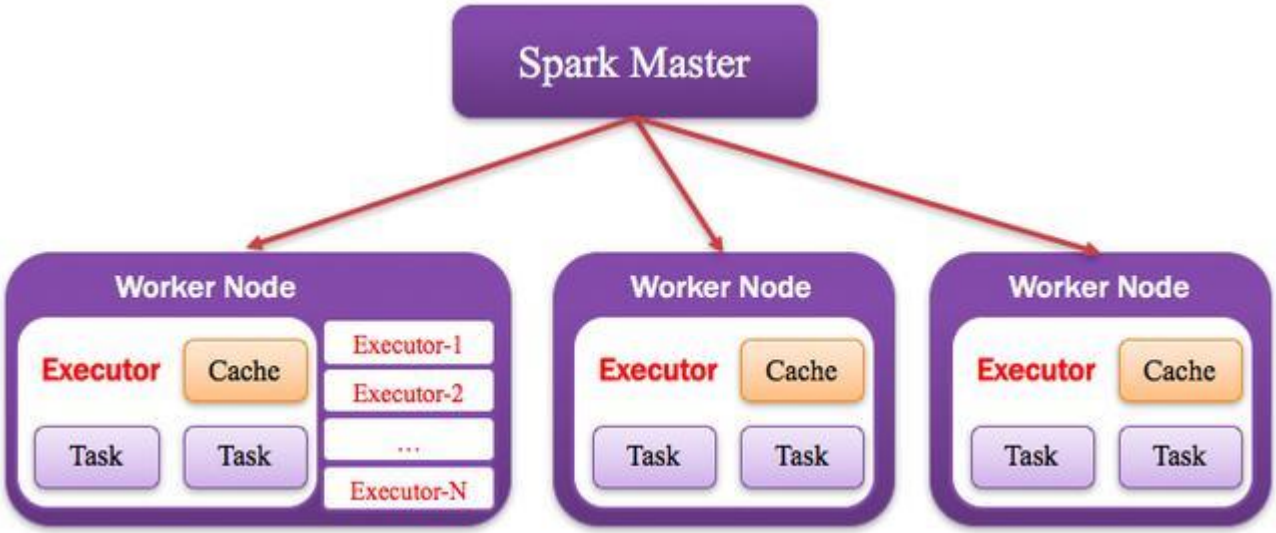
集群中任何可以运行Application代码的节点，类似于Yarn中的NodeManager节点。

在Standalone模式中指的就是通过Slave文件配置的Worker节点，

在Spark on Yarn模式中指的就是NodeManager节点，

在Spark on Messos模式中指的就是Messos Slave节点，

如下图所示:

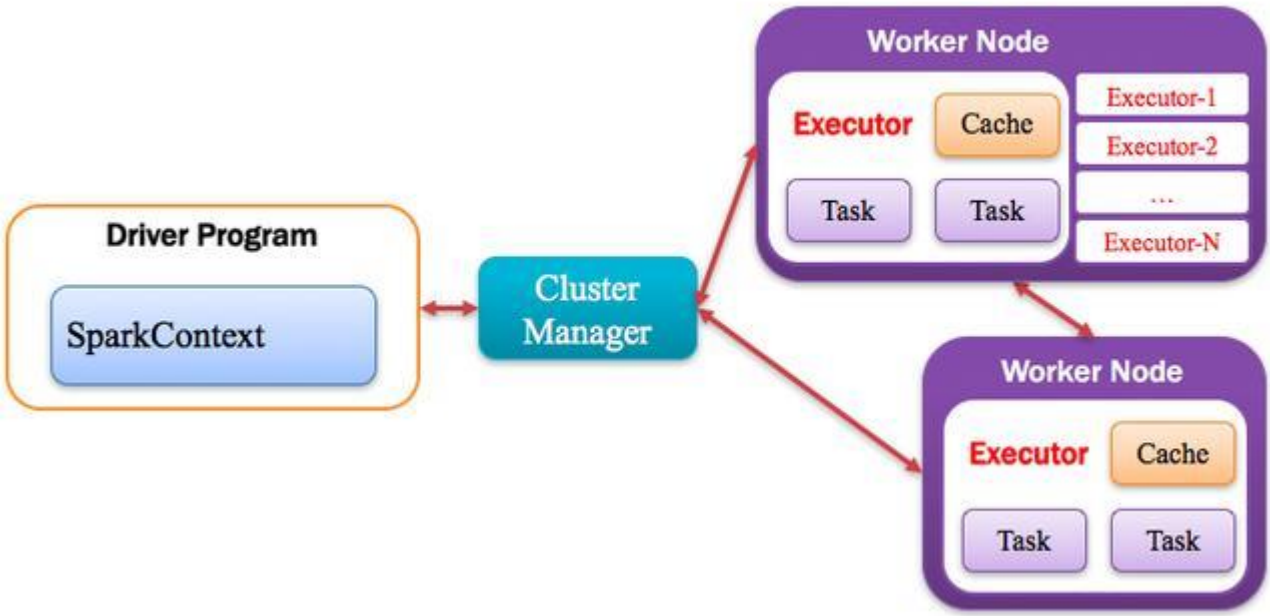


1.5 Executor：执行器

Application运行在Worker节点上的一个进程，该进程负责运行Task，并且负责将数据存在内存或者磁盘上，

每个Application都有各自独立的一批Executor，

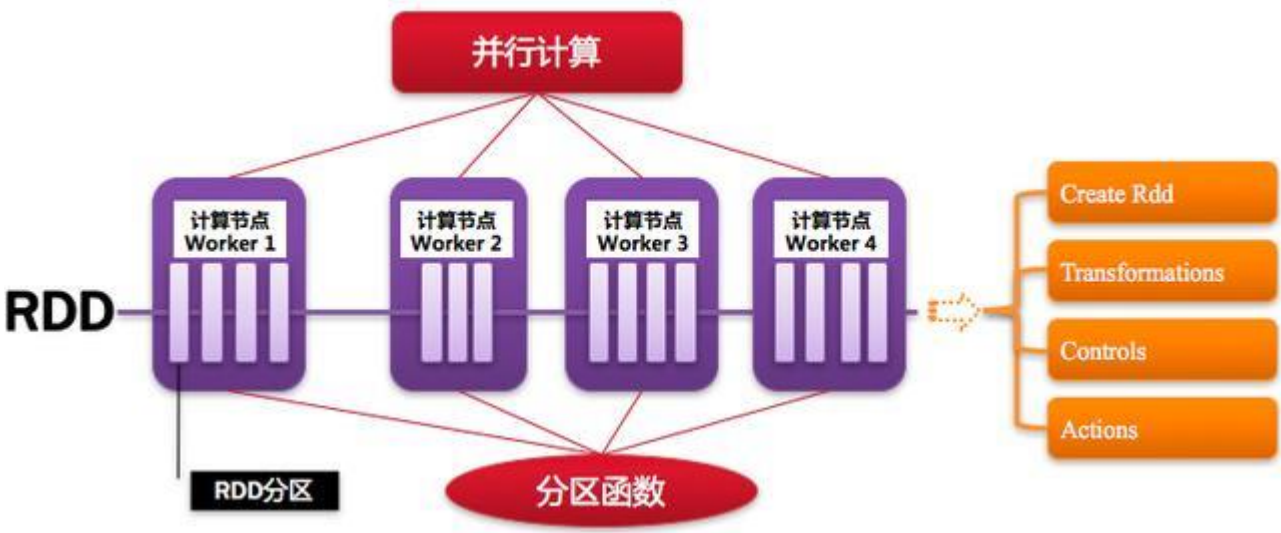
如下图所示:



1.6 RDD：弹性分布式数据集

Resillient Distributed Dataset，Spark的基本计算单元，可以通过一系列算子进行操作(主要有Transformation和Action操作)，

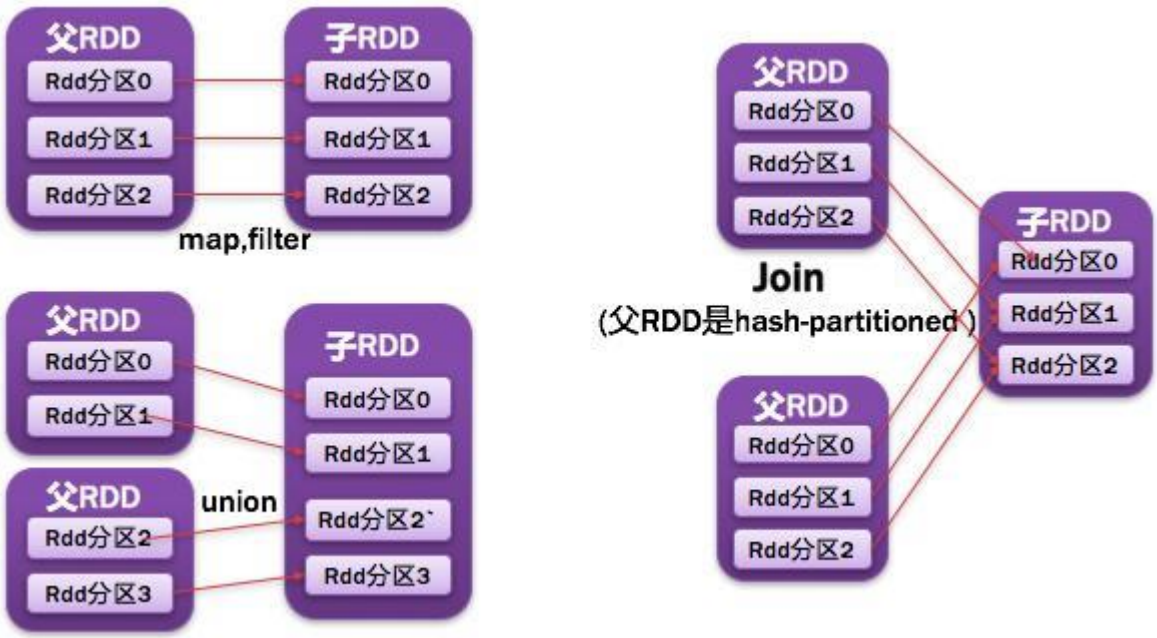
如下图所示:



1.7 NarrowDependency窄依赖

父RDD一个分区被子RDD的一个分区所依赖;

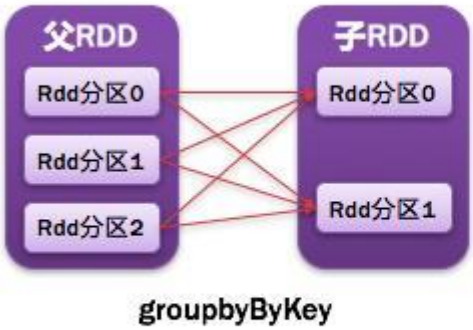
如图所示:



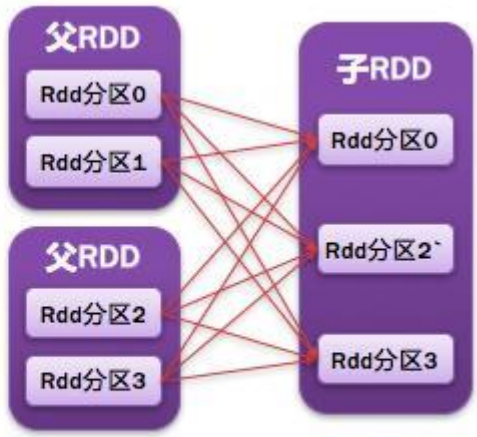
1.8 ShuffleDependency宽依赖

父RDD的一个分区都被子RDD多个分区所使用/依赖

如图所示:



groupbyByKey



Join
(父RDD不是hash-partitioned)

●常见的窄依赖有：

map、filter、union、mapPartitions、mapValues、join、笛卡尔积

●常见的宽依赖有：

groupByKey、partitionBy、reduceByKey、join

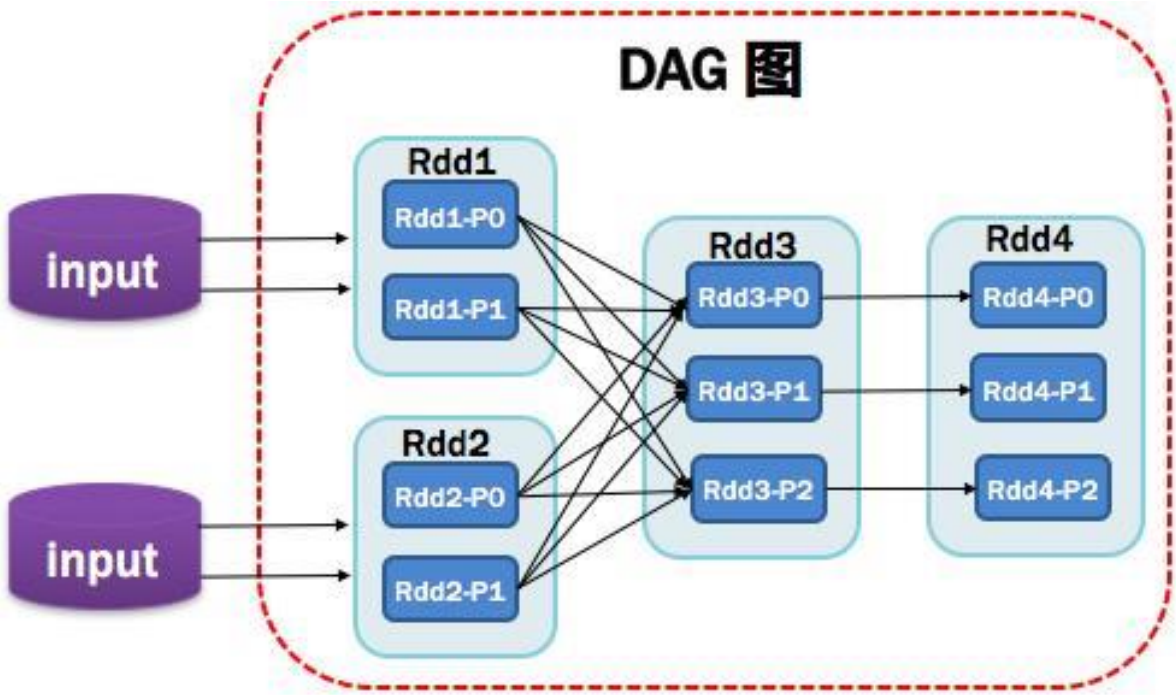
1.9 DAG：有向无环图

Directed Acycle graph，反应RDD之间的依赖关系，

DAG其实就是一个JOB(会根据依赖关系被划分成多个Stage)

注意:一个Spark程序会有1~n个DAG,调用一次Action就会形成一个DAG

如图所示:



1.10 DAGScheduler：有向无环图调度器

基于DAG划分Stage 并以TaskSet的形式提交Stage给TaskScheduler;

负责将作业拆分成不同阶段的具有依赖关系的多批任务;

最重要的任务之一就是：计算作业和任务的依赖关系，制定调度逻辑。

在SparkContext初始化的过程中被实例化，一个SparkContext对应创建一个DAGScheduler。

●总结：DAGScheduler完成以下工作:

1. DAGScheduler划分Stage(TaskSet)，记录哪个RDD或者 Stage 输出被物化(缓存)，通常在一个复杂的shuffle之后，通常物化一下(cache、persist)，方便之后的计算。
2. 重新提交出错/失败的Stage(shuffle输出丢失的stage/stage内部计算出错)
3. 将 Taskset 传给底层调度器
   - ✓ spark-cluster TaskScheduler
   - ✓ yarn-cluster YarnClusterScheduler
   - ✓ yarn-client YarnClientClusterScheduler

1.11 TaskScheduler：任务调度器

将Taskset提交给worker(集群)运行并回报结果;

负责每个具体任务的实际物理调度。

如图所示:

●总结：TaskScheduler完成以下工作:

1. 为每一个TaskSet构建一个TaskSetManager 实例管理这个TaskSet 的生命周期
2. 数据本地性决定每个Task最佳位置(移动计算比移动数据更划算)
3. 提交 taskset(一组task) 到集群运行并监控
4. 推测执行，碰到 straggle(计算缓慢) 任务需要放到别的节点上重试
5. 重新提交Shuffle输出丢失的Stage给DAGScheduler

1.12 Job：作业

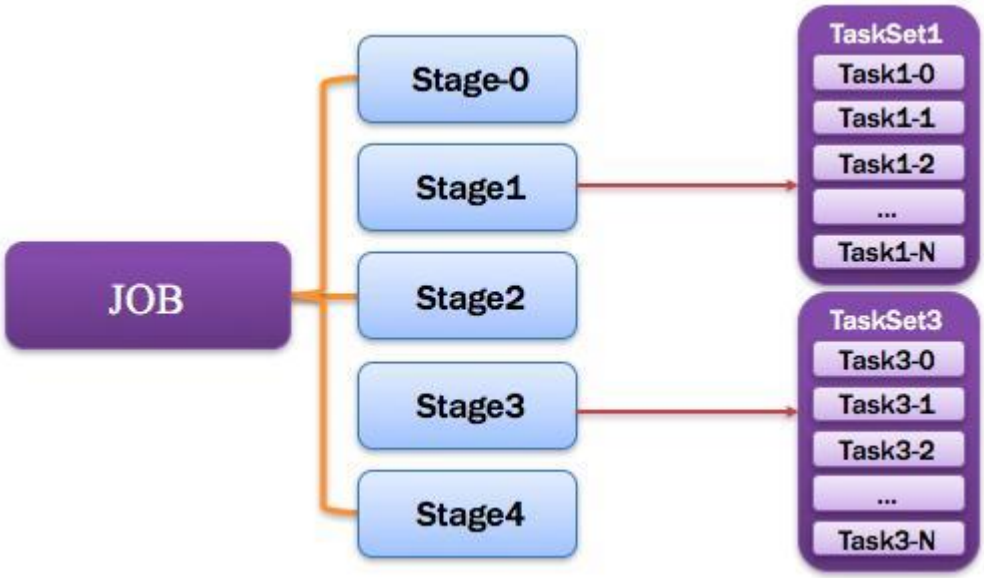
由一个或多个调度阶段Stage所组成的一次计算作业;

包含多个Task组成的并行计算，往往由Spark Action催生，

一个JOB包含多个RDD及作用于相应RDD上的各种Operation。

一个DAG其实就是一个Job

如图所示:



1.13 Stage：阶段

一个任务集对应的调度阶段;

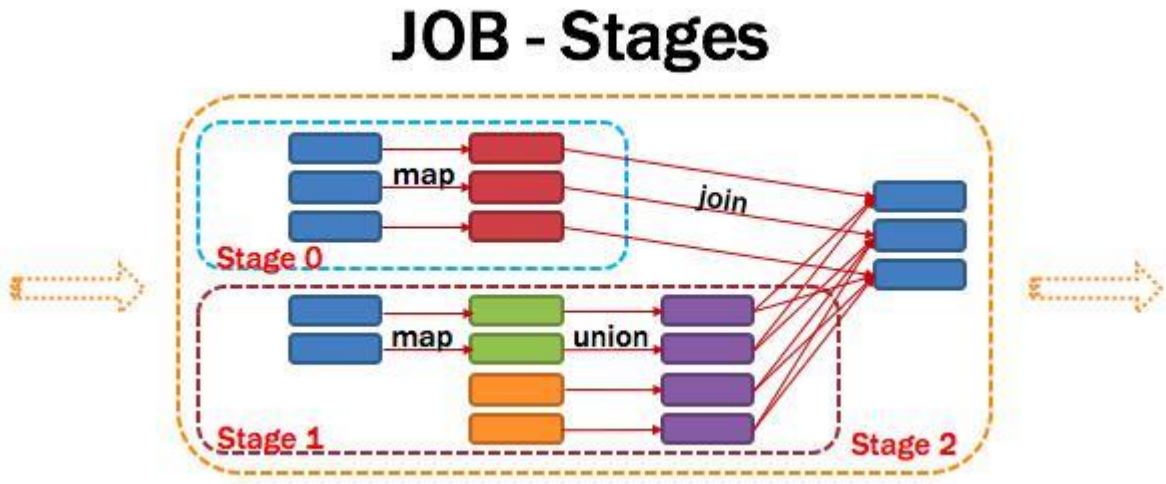
一个Stage ==> 一个TaskSet

DAG会根据shuffle/宽依赖划分Stage(也就是TaskSet)

每个Job会被拆分很多组TaskSet，每组任务被称为Stage，也可称TaskSet，一个作业分为多个阶段;

Stage分成两种类型ShuffleMapStage、ResultStage。

如图所示:

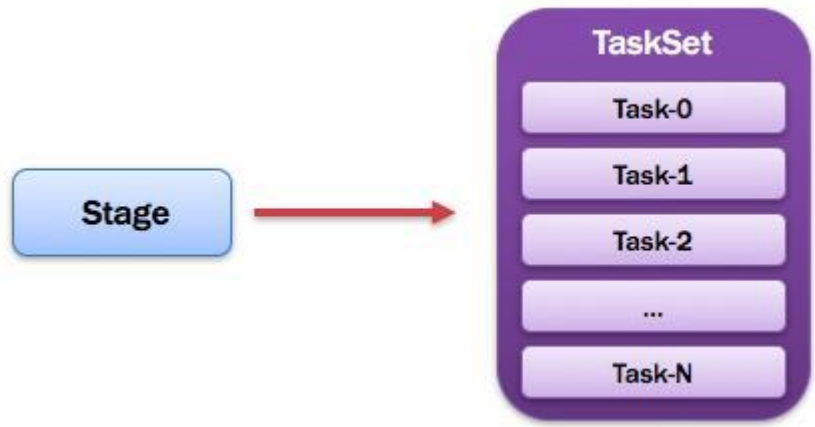


1.14 TaskSet：任务集

由一组关联的，但相互之间没有Shuffle依赖关系的任务所组成的任务集。

也就是一个Stage一个TaskSet

如图所示:



注意:

1)一个Stage创建一个TaskSet;

2)Stage的每个Rdd分区会创建一个Task,多个Task封装成TaskSet

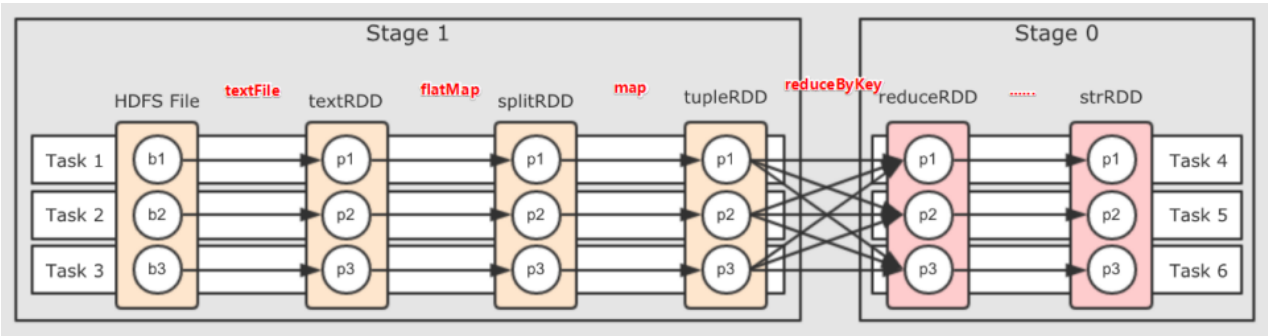
1.15 Task：任务

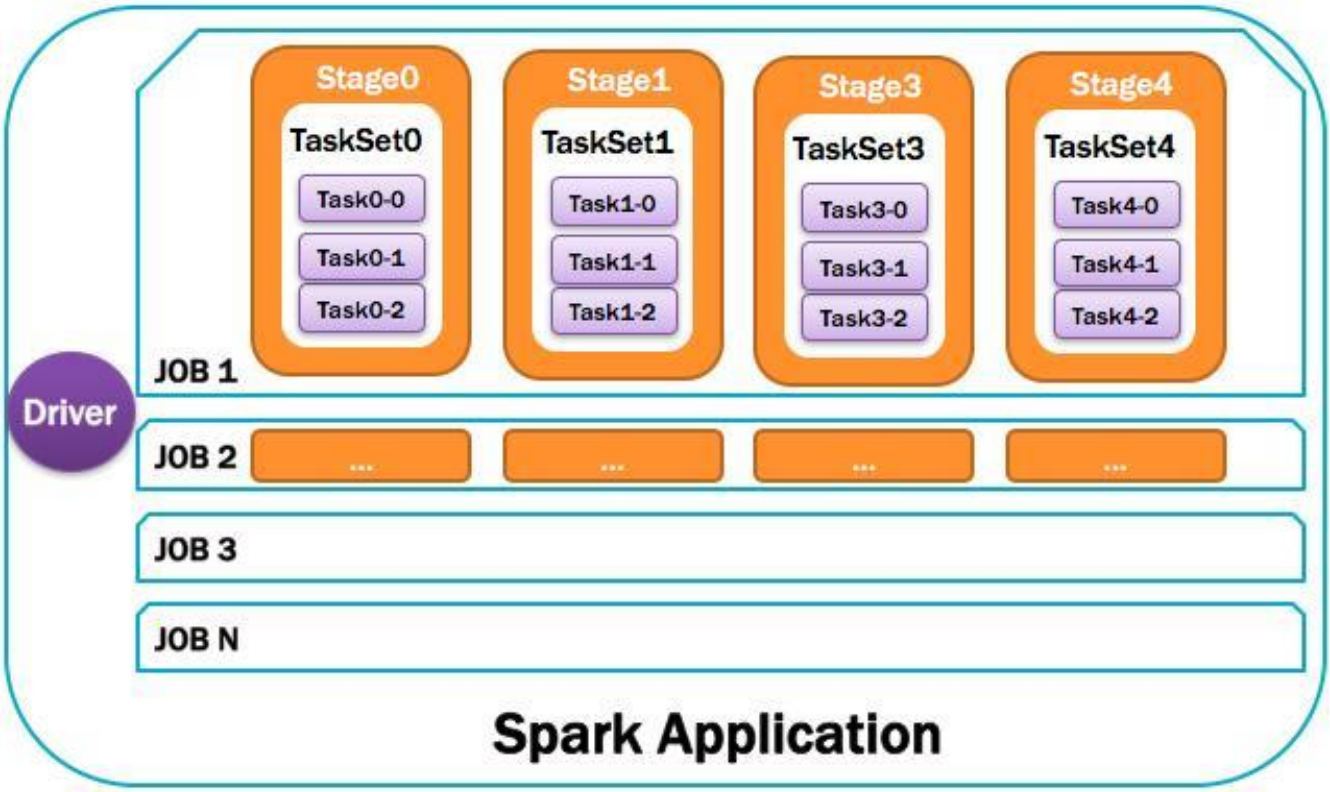
被送到某个Executor上的工作任务;单个分区数据集上的最小处理流程单元。

如图所示:


TaskSet
Task-0
Task-1
Task-2
...
Task-N

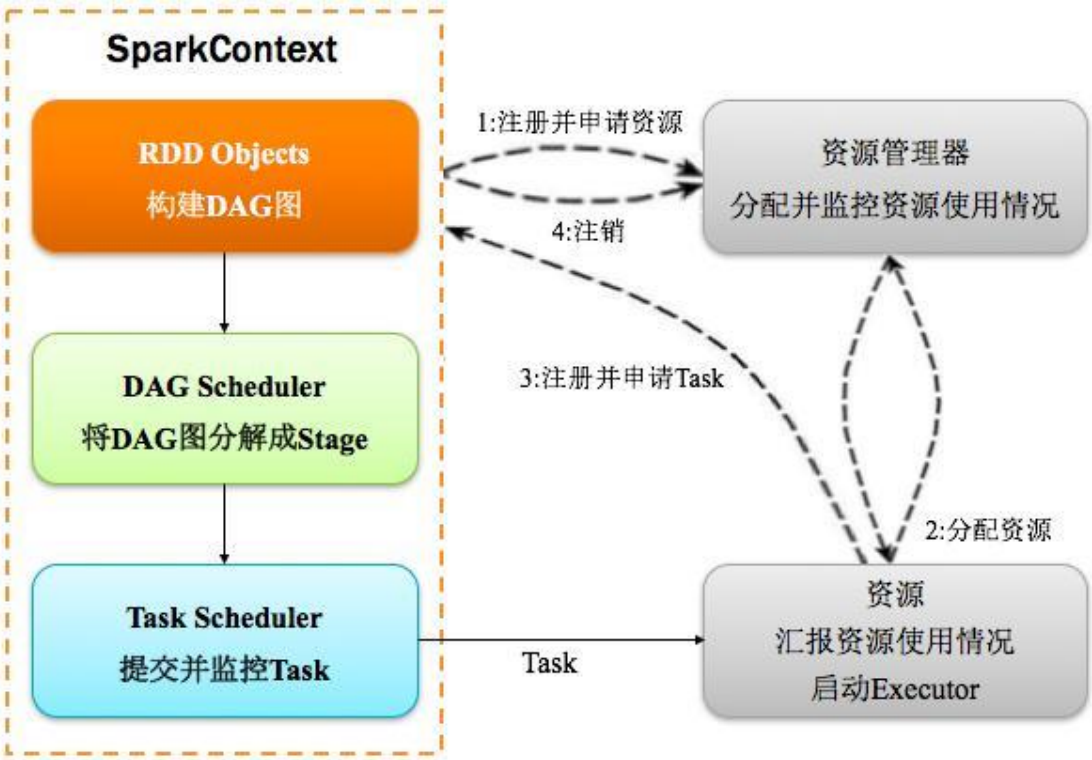
1.16 整体图示

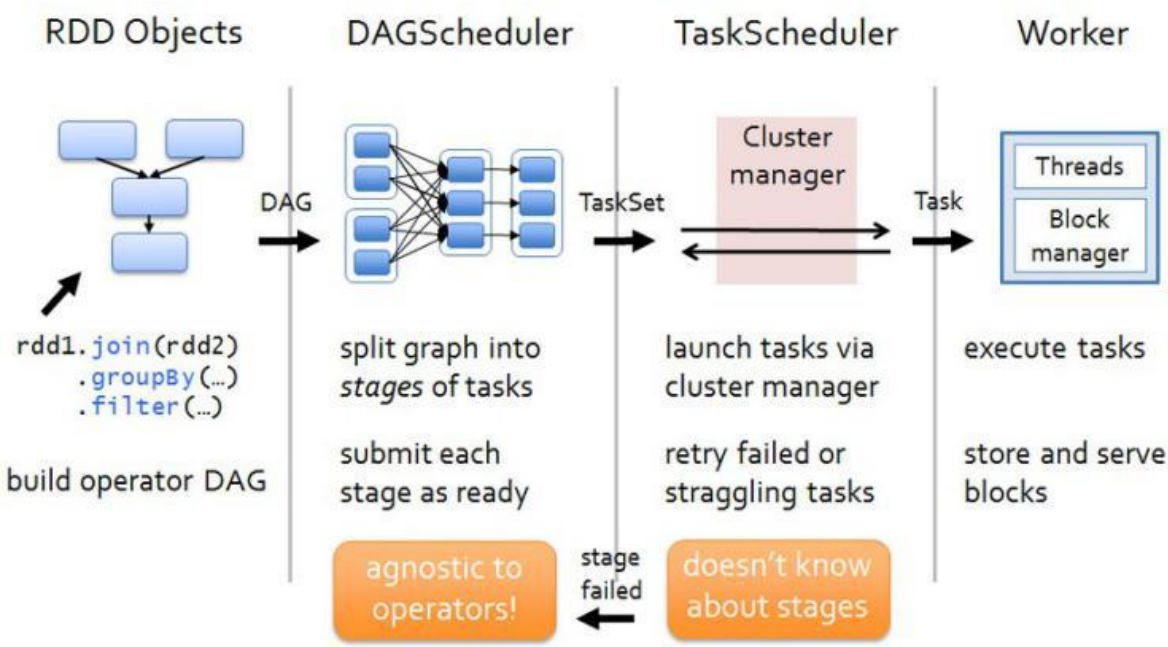

Stage 1
Stage 0
HDFS File
textFile
textRDD
flatMap
splitRDD
map
tupleRDD
reduceByKey
reduceRDD
.....
strRDD
Task 1
Task 2
Task 3
Task 4
Task 5
Task 6
b1
b2
b3
p1
p2
p3


Stage0
TaskSet0
Task0-0
Task0-1
Task0-2
Stage1
TaskSet1
Task1-0
Task1-1
Task1-2
Stage3
TaskSet3
Task3-0
Task3-1
Task3-2
Stage4
TaskSet4
Task4-0
Task4-1
Task4-2
JOB 1
Driver
JOB 2
...
...
...
...
JOB 3
JOB N
Spark Application

2. Spark运行流程-[掌握]

2.1 计算流程




RDD Objects
DAGScheduler
TaskScheduler
Worker
DAG
TaskSet
Task
Cluster manager
Threads
Block manager
rdd1.join(rdd2)
.groupBy(…)
.filter(…)
build operator DAG
split graph into stages of tasks
submit each stage as ready
agnostic to operators!
launch tasks via cluster manager
retry failed or straggling tasks
doesn't know about stages
stage failed
execute tasks
store and serve blocks

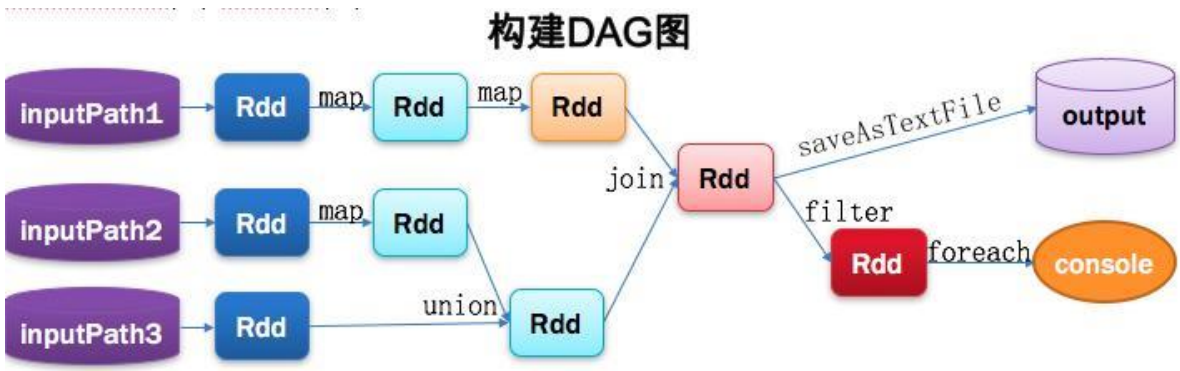

构建Spark Application运行环境
SparkContext向资源管理器注册
SparkContext向资源管理器申请运行Executor
资源管理器分配Executor
资源管理器启动Executor
Executor发送心跳至资源管理器
SparkContext构建成DAG图
将DAG图分解成Stage(TaskSet)
把Stage(TaskSet)发送给TaskScheduler
Executor向SparkContext申请Task
TaskScheduler将Task发放给Executor运行
同时SparkContext将应用程序代码发放给Executor
Task在Executor上运行，运行完毕释放所有资源

2.2 从代码构建DAG图

Spark program
Val lines1 = sc.textFile(inputPath1).map(...).map(...)
Val lines2 = sc.textFile(inputPath2).map(...)
Val lines3 = sc.textFile(inputPath3)
Val dtinone1 = lines2.union(lines3)
Val dtinone = lines1.join(dtinone1)
dtinone.saveAsTextFile(...)
dtinone.filter(...).foreach(...)

Spark的计算发生在RDD的Action操作，而对Action之前的所有Transformation，Spark只是记录下RDD生成的轨迹，而不会触发真正的计算。

Spark内核会在需要计算发生的时刻绘制一张关于计算路径的有向无环图，也就是DAG。



2.3 将DAG划分为Stage核心算法

Application多个job多个Stage：

Spark Application中可以因为不同的Action触发众多的job，一个Application中可以有很多的job，每个job是由一个或者多个Stage构成的，后面的Stage依赖于前面的Stage，也就是说只有前面依赖的Stage计算完毕后，后面的Stage才会运行。

划分依据：

Stage划分的依据就是宽依赖，何时产生宽依赖，reduceByKey, groupByKey等算子，会导致

宽依赖的产生。

核心算法：(回溯算法)

从后往前回溯/反向解析，遇到窄依赖加入本stage，遇见宽依赖进行Stage切分。

Spark内核会从触发Action操作的那个RDD开始从后往前推，

首先会为最后一个RDD创建一个stage，

然后继续倒推，如果发现对某个RDD是宽依赖，那么就会将宽依赖的那个RDD创建一个新的stage，那个RDD就是新的stage的最后一个RDD。

然后依次类推，继续倒推，根据窄依赖或者宽依赖进行stage的划分，

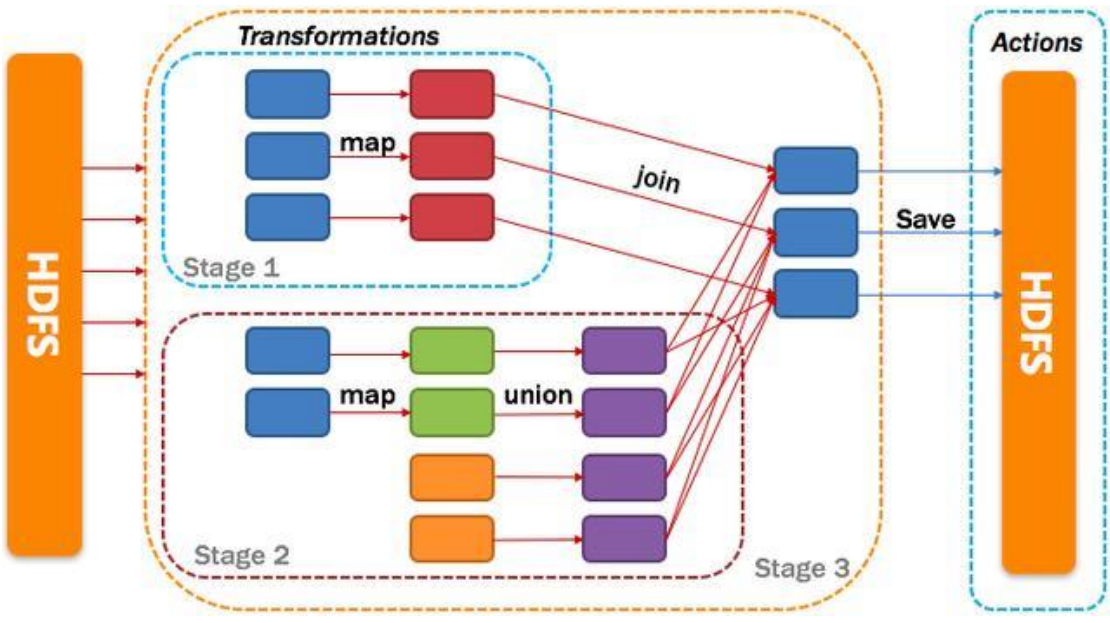
直到所有的RDD全部遍历完成为止。

2.4 将DAG划分为Stage剖析

从HDFS中读入数据生成3个不同的RDD，通过一系列transformation操作后再将计算结果保存回HDFS。

可以看到这个DAG中只有join操作是一个宽依赖，Spark内核会以此为边界将其前后划分成不同的Stage.

同时我们可以注意到，在图中Stage2中，从map到union都是窄依赖，这两步操作可以形成一个流水线操作，通过map操作生成的partition可以不用等待整个RDD计算结束，而是继续进行union操作，这样大大提高了计算的效率。



2.5 提交Stages

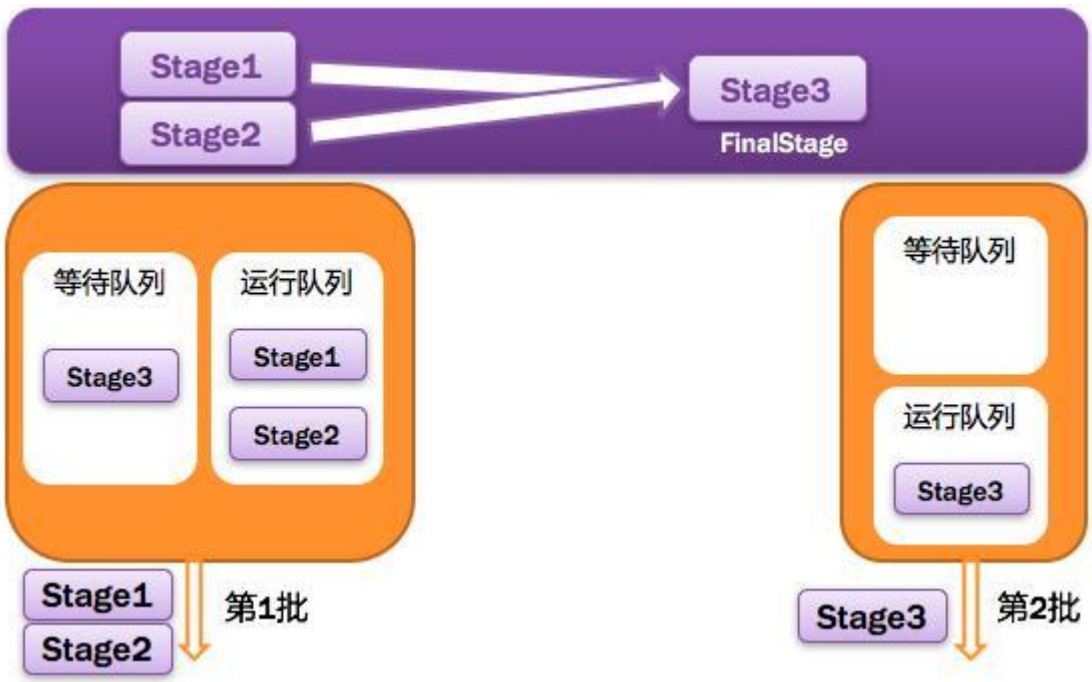
调度阶段的提交，最终会被转换成一个任务集的提交，

DAGScheduler通过TaskScheduler接口提交任务集，

这个任务集最终会触发TaskScheduler构建一个TaskSetManager的实例来管理这个任务集的生命周期，

对于DAGScheduler来说，提交调度阶段的工作到此就完成了。

而TaskScheduler的具体实现则会在得到计算资源的时候，进一步通过TaskSetManager调度具体的任务到对应的Executor节点上进行运算。



2.6 监控Job、Task、Executor

- DAGScheduler监控Job与Task：

要保证相互依赖的作业调度阶段能够得到顺利的调度执行，DAGScheduler需要监控当前作业调度阶段乃至任务的完成情况。

这通过对外暴露一系列的回调函数来实现的，对于TaskScheduler来说，这些回调函数主要包括任务的开始结束失败、任务集的失败，DAGScheduler根据这些任务的生命周期信息进一步维护作业和调度阶段的状态信息。

- DAGScheduler监控Executor的生命状态：

TaskScheduler通过回调函数通知DAGScheduler具体的Executor的生命状态，如果某一个Executor崩溃了，则对应的调度阶段任务集的ShuffleMapTask的输出结果也将标志为不可用，这将导致对应任务集状态的变更，进而重新执行相关计算任务，以获取丢失的相关数据。

2.7 获取任务执行结果

- 结果DAGScheduler：

一个具体的任务在Executor中执行完毕后，其结果需要以某种形式返回给DAGScheduler，根据任务类型的不同，任务结果的返回方式也不同。

- 两种结果，中间结果与最终结果：

对于FinalStage所对应的任务，返回给DAGScheduler的是运算结果本身，而对于中间调度阶段对应的任务ShuffleMapTask，返回给DAGScheduler的是一个MapStatus里的相关存储信息，而非结果本身，这些存储位置信息将作为下一个调度阶段的任务获取输入数据的依据。

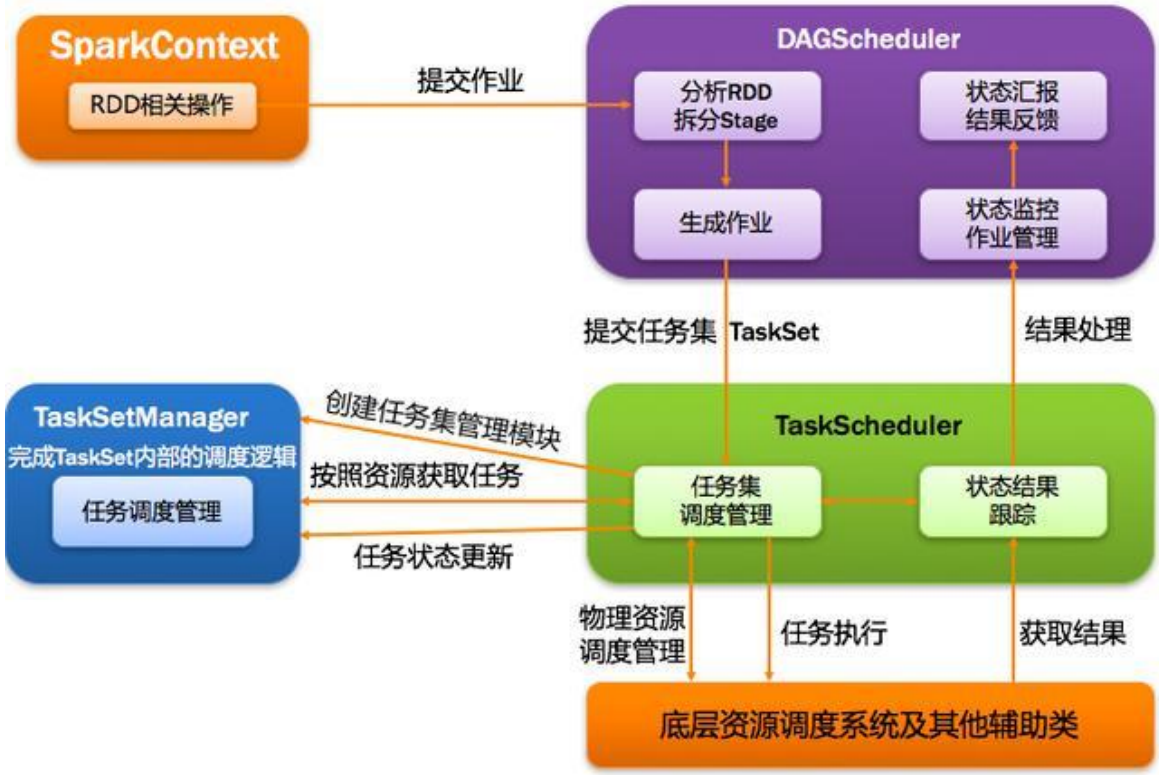
- 两种类型，DirectTaskResult与IndirectTaskResult：

根据任务结果大小的不同，ResultTask返回的结果又分为两类，

如果结果足够小，则直接放在DirectTaskResult对象内中，

如果超过特定尺寸则在Executor端会将DirectTaskResult先序列化，再把序列化的结果作为一个数据块存放在BlockManager中，然后将BlockManager返回的BlockID放在IndirectTaskResult对象中返回给TaskScheduler，TaskScheduler进而调用TaskResultGetter将IndirectTaskResult中的BlockID取出并通过BlockManager最终取得对应的DirectTaskResult。

2.8 任务调度总体诠释

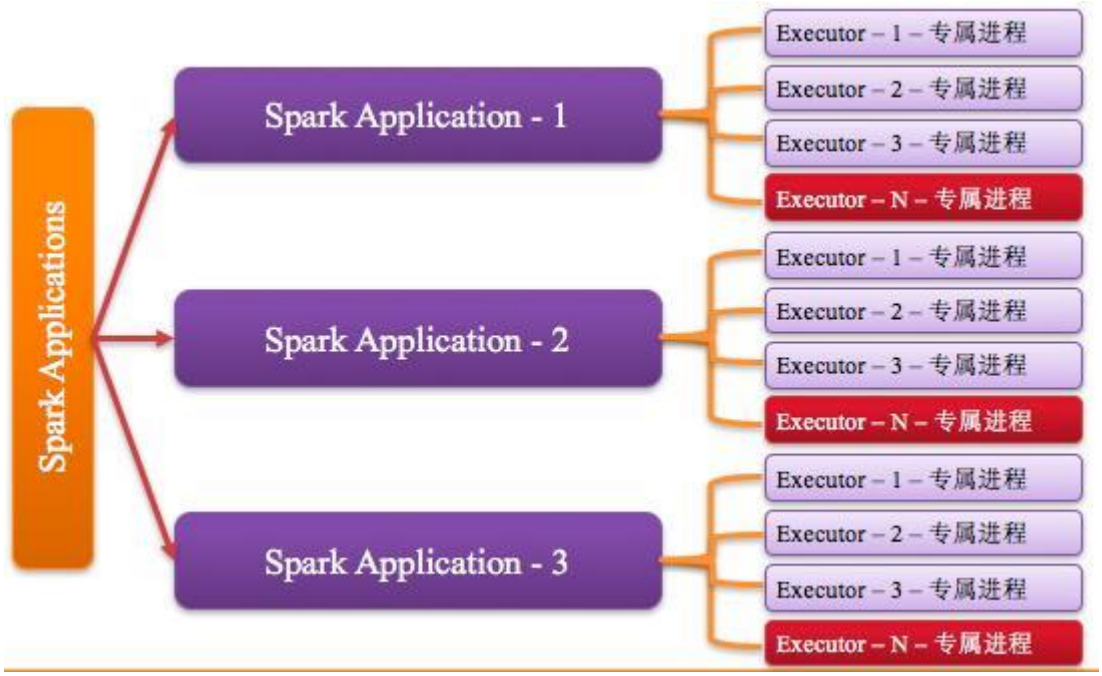


3. Spark运行架构特点-[了解]

3.1 Executor进程专属

每个Application获取专属的executor进程，该进程在Application期间一直驻留，并以多线程方式运行tasks。

Spark Application不能跨应用程序共享数据，除非将数据写入到外部存储系统。如图所示:



3.2 支持多种资源管理器

Spark与资源管理器无关，只要能够获取executor进程，并能保持相互通信就可以了，

Spark支持资源管理器包含： Standalone、On Mesos、On YARN、Or On EC2。

如图所示:

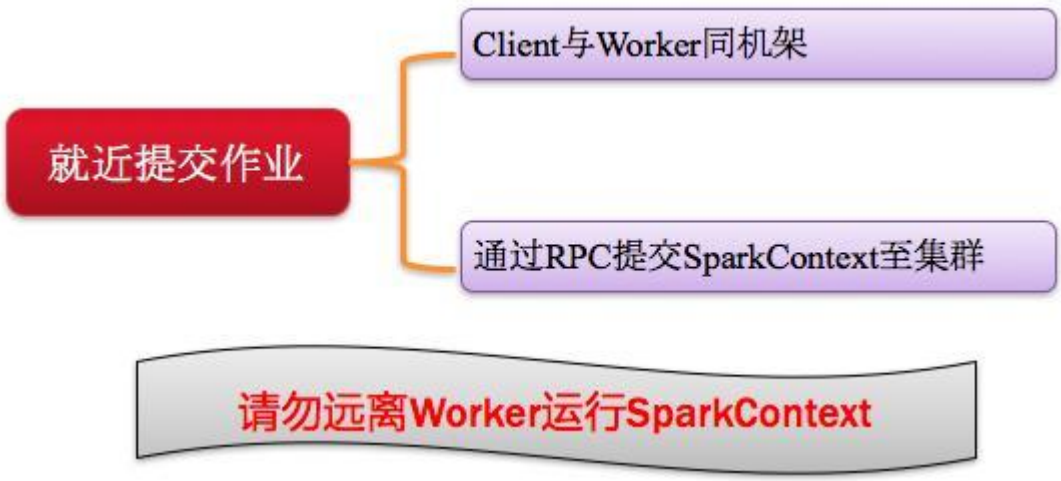
3.3 Job提交就近原则

提交SparkContext的Client应该靠近Worker节点(运行Executor的节点)，最好是在同一个Rack(机架)里，

因为Spark Application运行过程中SparkContext和Executor之间有大量的信息交换;

如果想在远程集群中运行，最好使用RPC将SparkContext提交给集群，不要远离Worker运行SparkContext。

如图所示:

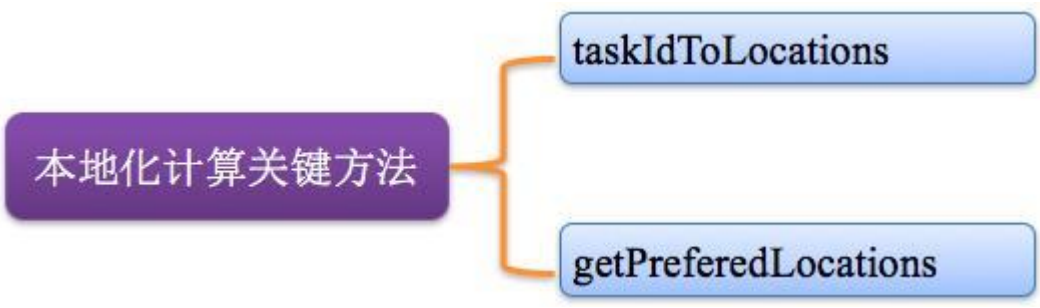


3.4 移动程序而非移动数据的原则执行

Task采用了数据本地性和推测执行的优化机制。

关键方法：taskIdToLocations、getPreferedLocations。

如图所示:



4. Spark部署模式

Spark支持多种集群管理器（Cluster Manager），主要为：

1. Standalone：独立集群模式，Spark原生的简单集群管理器，自带完整的服务，可单独部署到一个集群中，无需依赖任何其他资源管理系统，使用Standalone可以很方便地搭建一个集群；

2. Apache Mesos：一个强大的分布式资源管理框架，它允许多种不同的框架部署在其上，包括yarn；

3. Hadoop YARN：统一的资源管理机制，在上面可以运行多套计算框架，如map reduce、storm等，根据driver在集群中的位置不同，分为yarn client和yarn cluster。

实际上，除了上述这些通用的集群管理器外，Spark内部也提供了一些方便用户测试和学习的简单集群部署模式。由于在实际工厂环境下使用的绝大多数的集群管理器是Hadoop YARN，因此我们关注的重点是Hadoop YARN模式下的Spark集群部署。

Spark的运行模式取决于传递给SparkContext的MASTER环境变量的值，个别模式还需要辅助的程序接口来配合使用，目前支持的Master字符串及URL包括：

表2-1 Spark运行模式配置

Master URL	Meaning
local	在本地运行，只有一个工作进程，无并行计算能力。
local[K]	在本地运行，有K个工作进程，通常设置K为机器的CPU核心数量。
local[*]	在本地运行，工作进程数量等于机器的CPU核心数量。
spark://HOST:PORT	以Standalone模式运行，这是Spark自身提供的集群运行模式，默认端口号: 7077。详细文档见:Spark standalone cluster。
mesos://HOST:PORT	在Mesos集群上运行，Driver进程和Worker进程运行在Mesos集群上，部署模式必须使用固定值:--deploy-mode cluster。详细文档见:MesosClusterDispatcher.

yarn-client	在Yarn集群上运行，Driver进程在本地，Executor进程在Yarn集群上，部署模式必须使用固定值:--deploy-mode client。Yarn集群地址必须在HADOOP_CONF_DIR or YARN_CONF_DIR变量里定义。
yarn-cluster	在Yarn集群上运行，Driver进程在Yarn集群上，Work进程也在Yarn集群上，部署模式必须使用固定值:--deploy-mode cluster。Yarn集群地址必须在HADOOP_CONF_DIR or YARN_CONF_DIR变量里定义。

用户在提交任务给Spark处理时，以下两个参数共同决定了Spark的运行方式。

· –master MASTER_URL ：决定了Spark任务提交给哪种集群处理。

· –deploy-mode DEPLOY_MODE：决定了Driver的运行方式，可选值为Client或者Cluster。

集群有四个重要组成部分，分别是：

1) Driver：是一个进程，我们编写的Spark应用程序就运行在Driver上，由Driver进程执行；

2) Master(RM)：是一个进程，主要负责资源的调度和分配，并进行集群的监控等职责；

3) Worker(NM)：是一个进程，一个Worker运行在集群中的一台服务器上，主要负责两个职责，一个是用自己的内存存储RDD的某个或某些partition；另一个是启动其他进程和线程（Executor），对RDD上的partition进行并行的处理和计算。

4) Executor：是一个进程，一个Worker上可以运行多个Executor，Executor通过启动多个线程（task）来执行对RDD的partition进行并行计算，也就是执行我们对RDD定义的例如map、flatMap、reduce等算子操作。

4.1 YARN Client模式-Driver运行在客户端

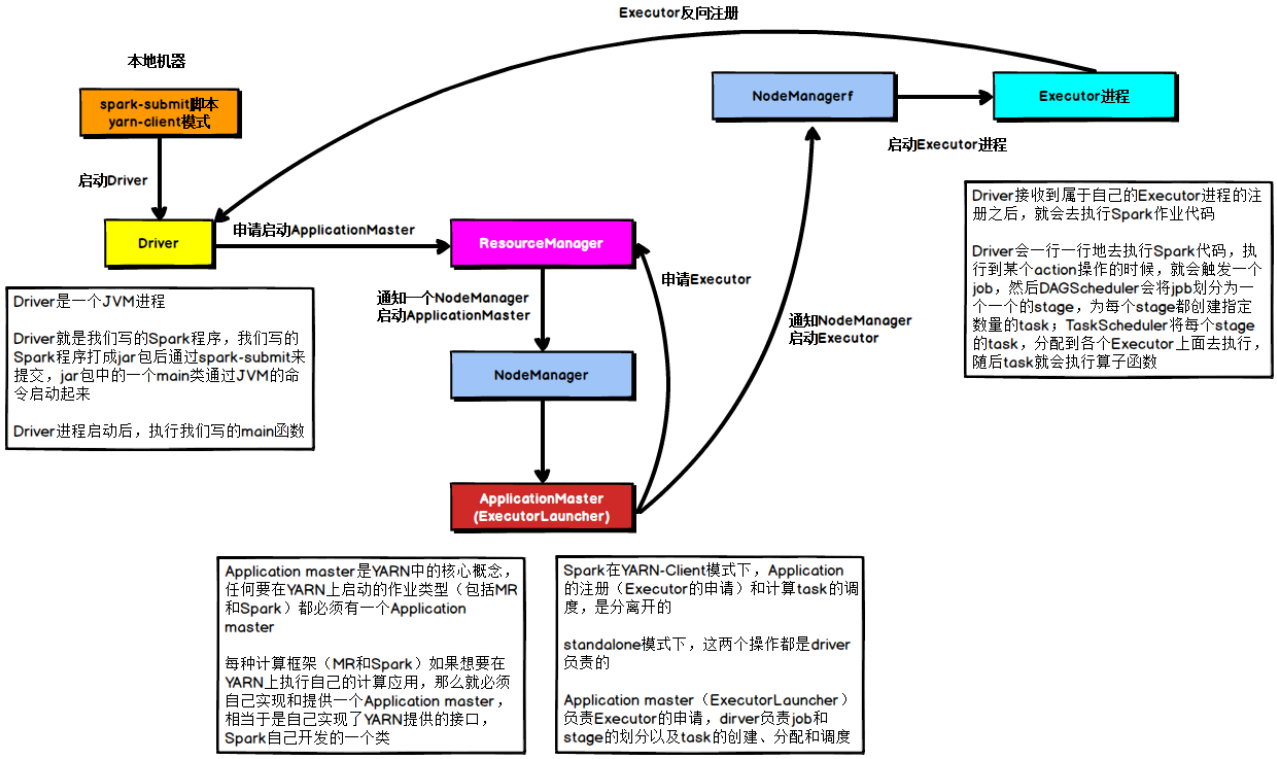
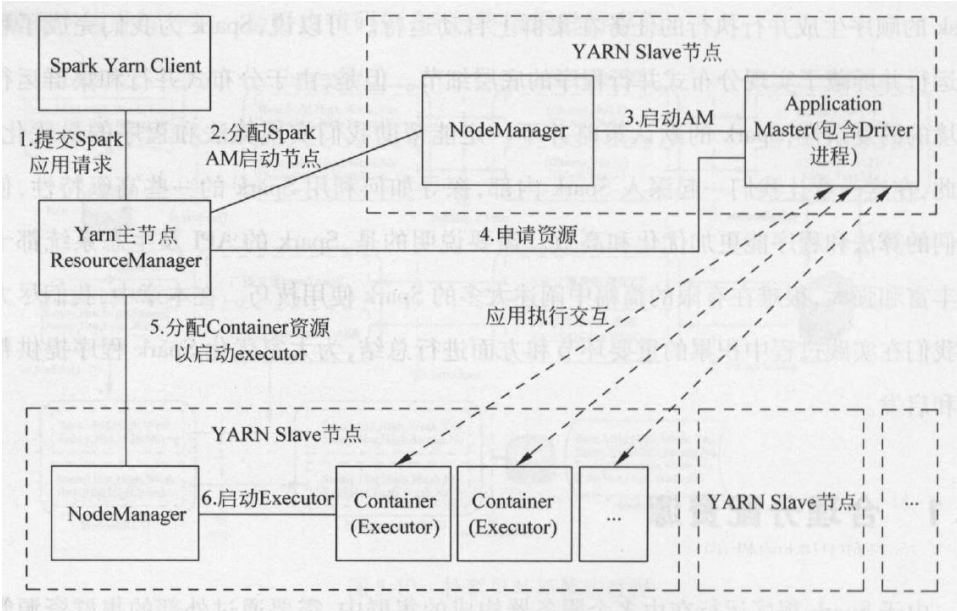


图2-3 YARN Client模式

在YARN Client模式下，Driver在任务提交的本地机器上运行，Driver启动后会和ResourceManager通讯申请启动ApplicationMaster，随后ResourceManager分配container，在合适的NodeManager上启动ApplicationMaster，此时的ApplicationMaster的功能相当于一个ExecutorLaucher，只负责向ResourceManager申请Executor内存。

ResourceManager接到ApplicationMaster的资源申请后会分配container，然后ApplicationMaster在资源分配指定的NodeManager上启动Executor进程，Executor进程启动后会向Driver反向注册，Executor全部注册完成后Driver开始执行main函数，之后执行到Action算子时，触发一个job，并根据宽依赖开始划分stage，每个stage生成对应的taskSet，之后将task分发到各个Executor上执行。

4.2 YARN Cluster模式-Driver运行在YARN



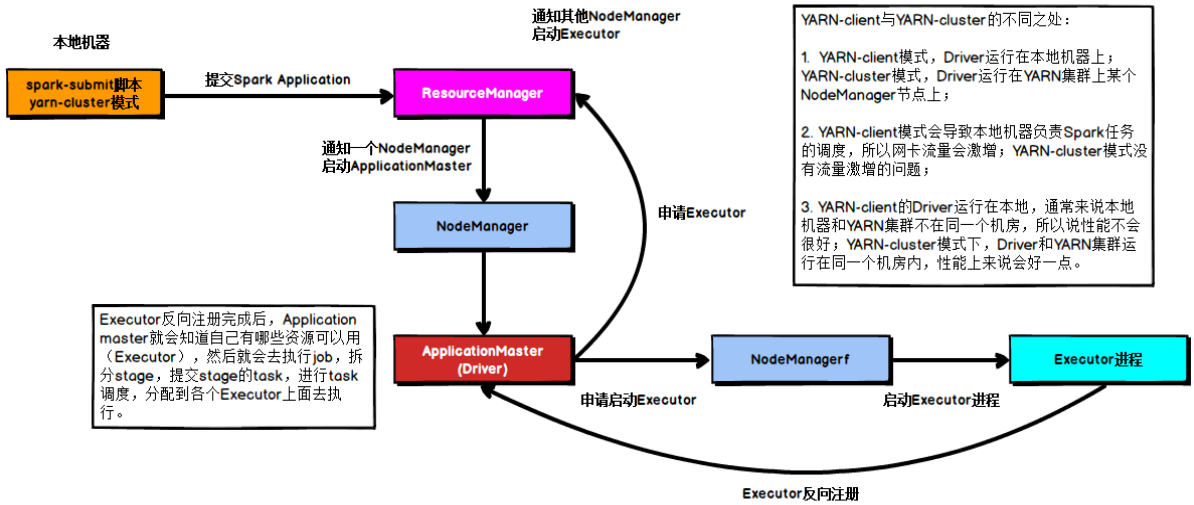


图2-4 YARN Cluster模式

在YARN Cluster模式下，任务提交后会和ResourceManager通讯申请启动ApplicationMaster，随后ResourceManager分配container，在合适的NodeManager上启动ApplicationMaster，此时的ApplicationMaster就是Driver。

Driver启动后向ResourceManager申请Executor内存，ResourceManager接到ApplicationMaster的资源申请后会分配container，然后在合适的NodeManager上启动Executor进程，Executor进程启动后会向Driver反向注册，Executor全部注册完成后Driver开始执行main函数，之后执行到Action算子时，触发一个job，并根据宽依赖开始划分stage，每个stage生成对应的taskSet，之后将task分发到各个Executor上执行。