

# 08

## Spark

- Spark简介
- Spark架构
- RDD
- Spark SQL
- Spark Streaming
- MLlib



# Spark简介

---

- Spark最初由美国加州伯克利大学（UC Berkeley）的AMP实验室于2009年开发，是基于内存计算的大数据并行计算框架，可用于构建大型的、低延迟的数据分析应用程序
- 2013年Spark加入Apache孵化器项目后发展迅猛，如今已成为Apache软件基金会最重要的三大分布式计算系统开源项目之一（Hadoop、Spark、Storm）
- Spark在2014年打破了Hadoop保持的基准排序纪录
  - Spark/206个节点/23分钟/100TB数据
  - Hadoop/2000个节点/72分钟/100TB数据
  - Spark用十分之一的计算资源，获得了比Hadoop快3倍的速度

# Spark简介

---

Spark主要特点:

- 运行速度快: 使用DAG执行引擎以支持循环数据流与内存计算
- 容易使用: 支持使用Scala、Java、Python和R语言进行编程, 可以通过Spark Shell进行交互式编程
- 通用性: Spark提供了完整而强大的技术栈, 包括SQL查询、流式计算、机器学习和图算法组件
- 运行模式多样: 可运行于独立的集群模式中, 可运行于Hadoop中, 也可运行于Amazon EC2等云环境中, 并且可以访问HDFS、Cassandra、HBase、Hive等多种数据源

# Scala简介

---

Scala是一门现代的多范式编程语言，运行于Java平台（JVM，Java 虚拟机），并兼容现有的Java程序

Scala的特性：

- Scala具备强大的并发性，支持函数式编程，可以更好地支持分布式系统
- Scala语法简洁，能提供优雅的API

Scala兼容Java，运行速度快，且能融合到Hadoop生态圈中

Scala是Spark的主要编程语言，但Spark还支持Java、Python、R作为编程语言

Scala的优势是提供了REPL（Read-Eval-Print Loop，交互式解释器），提高程序开发效率

# Spark与Hadoop的对比

---

Hadoop存在的缺点:

- 表达能力有限
- 磁盘IO开销大
- 延迟高
  - 任务之间的衔接涉及IO开销
  - 在前一个任务执行完成之前, 其他任务就无法开始, 难以胜任复杂、多阶段的计算任务

# Spark与Hadoop的对比

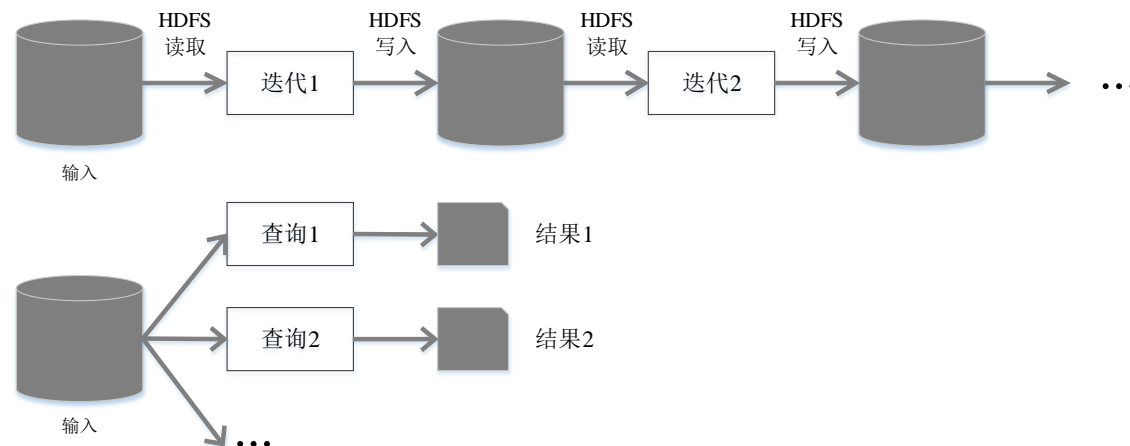
---

Spark在借鉴Hadoop MapReduce优点的同时，很好地解决了MapReduce所面临的问题

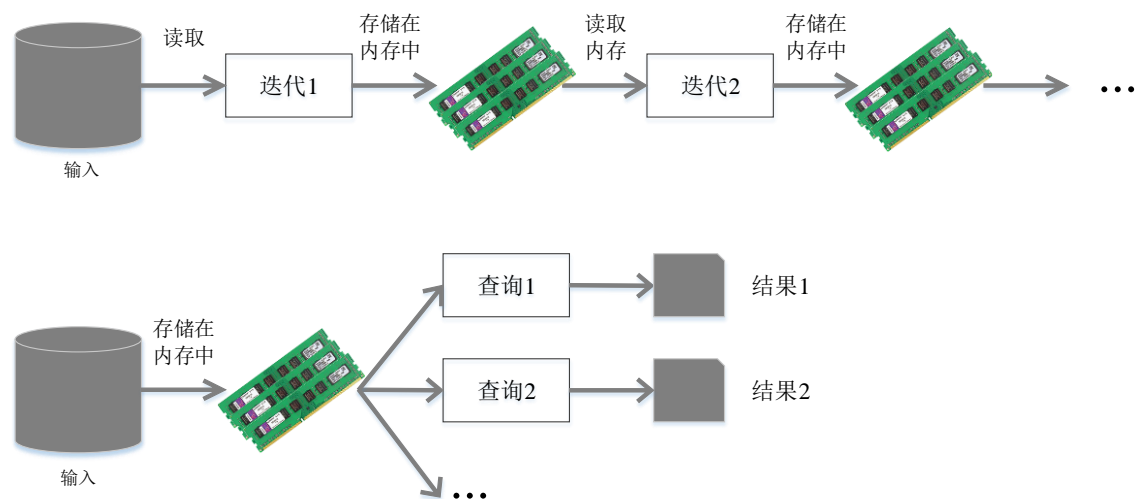
相比于Hadoop MapReduce，Spark主要具有如下优点：

- Spark的计算模式也属于MapReduce，但不局限于Map和Reduce操作，还提供了多种数据集操作类型，编程模型比Hadoop MapReduce更灵活
- Spark提供了内存计算，可将中间结果放到内存中，对于迭代运算效率更高
- Spark基于DAG的任务调度执行机制，要优于Hadoop MapReduce的迭代执行机制

# Spark与Hadoop的对比



(a) Hadoop MapReduce执行流程

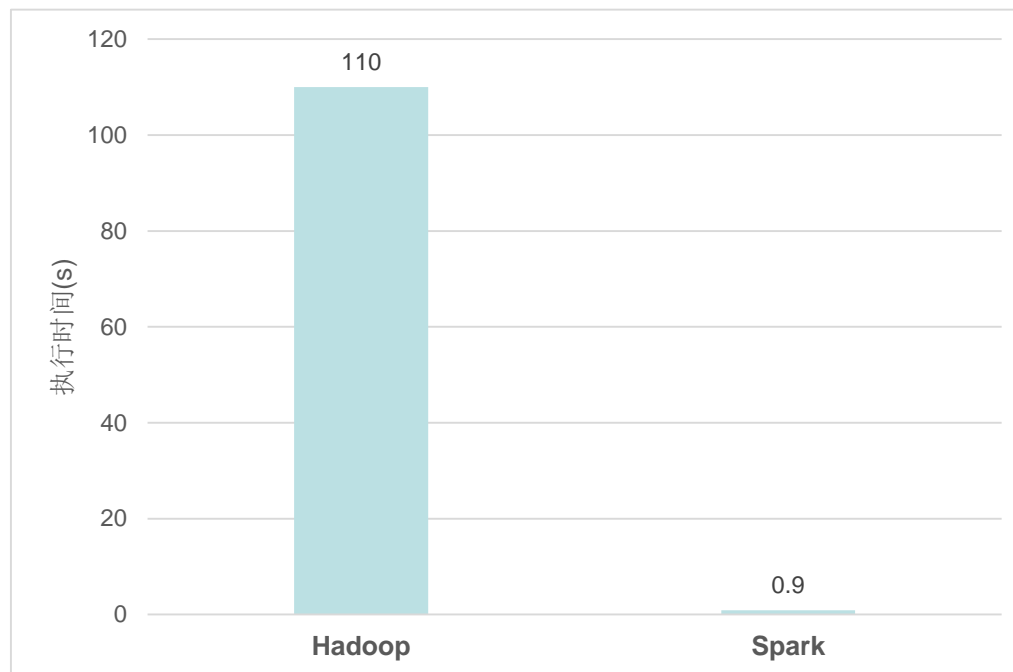


(b) Spark执行流程

# Spark与Hadoop的对比

---

Hadoop与Spark执行逻辑回归的时间对比





# Spark生态系统

---

在实际应用中，大数据处理主要包括以下三个类型：

- 复杂的批量数据处理：通常时间跨度在数十分钟到数小时之间
- 基于历史数据的交互式查询：通常时间跨度在数十秒到数分钟之间
- 基于实时数据流的数据处理：通常时间跨度在数百毫秒到数秒之间

当同时存在以上三种场景时，就需要同时部署三种不同的软件

- 比如: MapReduce / Impala (Hive) / Storm

这样做难免会带来一些问题：

- 不同场景之间输入输出数据无法做到无缝共享，通常需要进行数据格式的转换
- 不同的软件需要不同的开发和维护团队，带来了较高的使用成本
- 比较难以对同一个集群中的各个系统进行统一的资源协调和分配

# Spark生态系统

---

- Spark的设计遵循 “一个软件栈满足不同应用场景” 的理念，逐渐形成了一套完整的生态系统
- 既能够提供内存计算框架，也可以支持SQL即席查询、实时流式计算、机器学习和图计算等
- Spark可以部署在资源管理器YARN之上，提供一站式的大数据解决方案
- 因此，Spark所提供的生态系统足以应对上述三种场景，即同时支持批处理、交互式查询和流数据处理

# Spark生态系统

Spark生态系统已经成为伯克利数据分析软件栈BDAS（Berkeley Data Analytics Stack）的重要组成部分

Access and Interfaces	Spark Streaming	BlinkDB	GraphX	MLBase
		Spark SQL		MLlib
Processing Engine	Spark Core			
Storage	Tachyon			
	HDFS, S3			
Resource Virtualization	Mesos		Hadoop Yarn	

BDAS架构

Spark的生态系统主要包含了Spark Core、Spark SQL、Spark Streaming、MLlib和GraphX等组件

# Spark生态系统

---

应用场景	时间跨度	其他框架	Spark生态系统中的组件
复杂的批量数据处理	小时级	MapReduce、Hive	Spark
基于历史数据的交互式查询	分钟级、秒级	Impala 、 Dremel 、 Drill	Spark SQL
基于实时数据流的数据处理	毫秒、秒级	Storm、S4	Spark Streaming
基于历史数据的数据挖掘	-	Mahout	MLlib
图结构数据的处理	-	Pregel、Hama	GraphX

# Spark运行架构

---

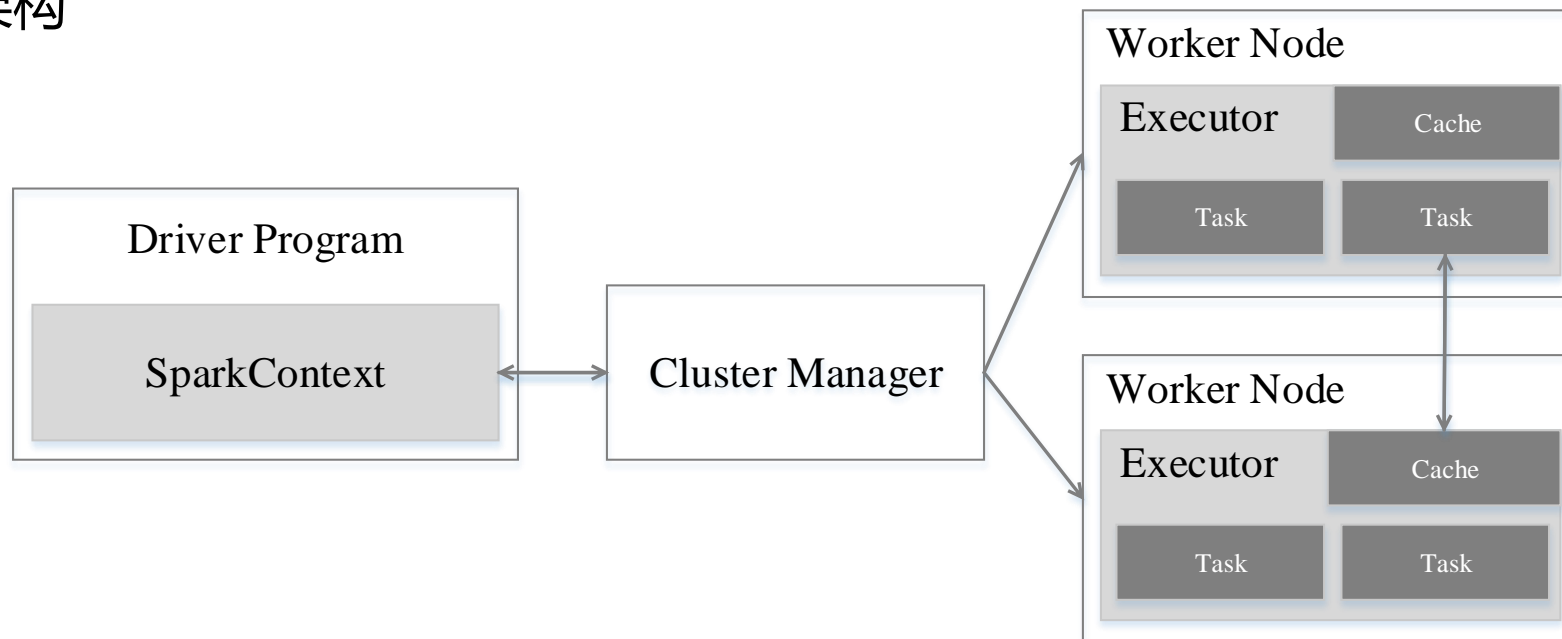
## 基本概念

- RDD: 是Resilient Distributed Dataset (弹性分布式数据集) 的简称, 是分布式内存的一个抽象概念, 提供了一种高度受限的共享内存模型
- DAG: 是Directed Acyclic Graph (有向无环图) 的简称, 反映RDD之间的依赖关系
- Executor: 是运行在工作节点 (WorkerNode) 的一个进程, 负责运行Task
- Application: 用户编写的Spark应用程序
- Task: 运行在Executor上的工作单元
- Job: 一个Job包含多个RDD及作用于相应RDD上的各种操作
- Stage: 是Job的基本调度单位, 一个Job会分为多组Task, 每组Task被称为Stage, 或者也被称为TaskSet, 代表了一组关联的、相互之间没有Shuffle依赖关系的任务组成的任务集

# 架构设计

- Spark运行架构包括集群资源管理器（Cluster Manager）、运行作业任务的工作节点（Worker Node）、每个应用的任务控制节点（Driver）和每个工作节点上负责具体任务的执行进程（Executor）
- 资源管理器可以自带或Mesos或YARN

## Spark运行架构



# 架构设计

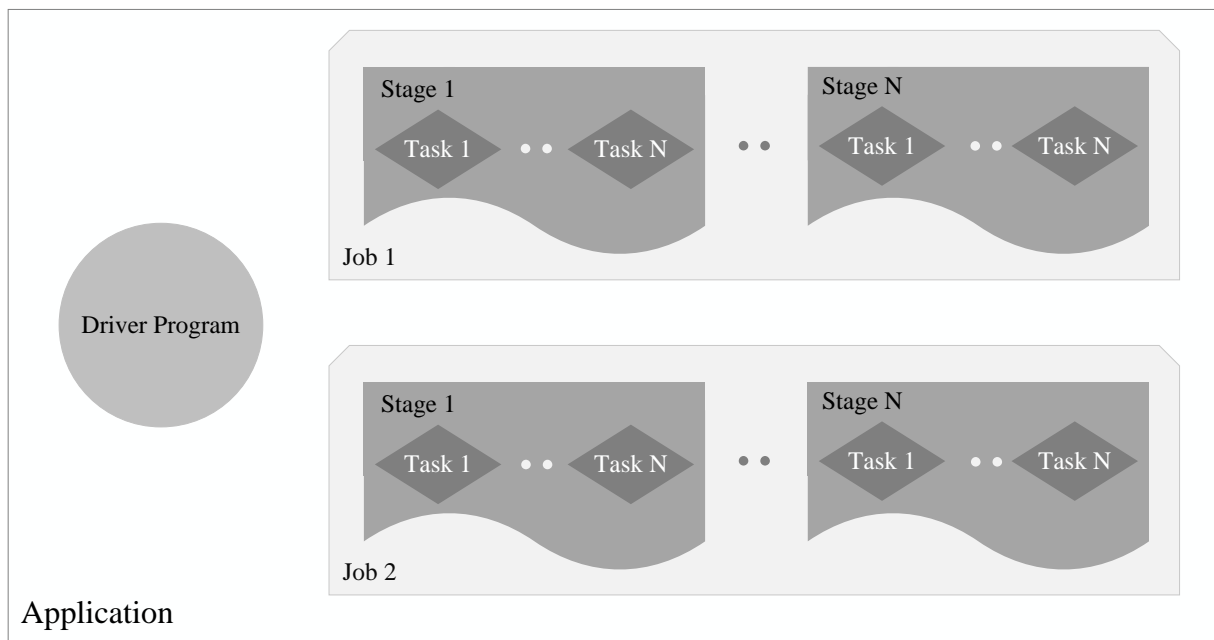
---

与Hadoop MapReduce计算框架相比，Spark所采用的Executor有两个优点：

- 一是利用多线程来执行具体的任务，减少任务的启动开销
- 二是Executor中有一个BlockManager存储模块，会将内存和磁盘共同作为存储设备，有效减少IO开销

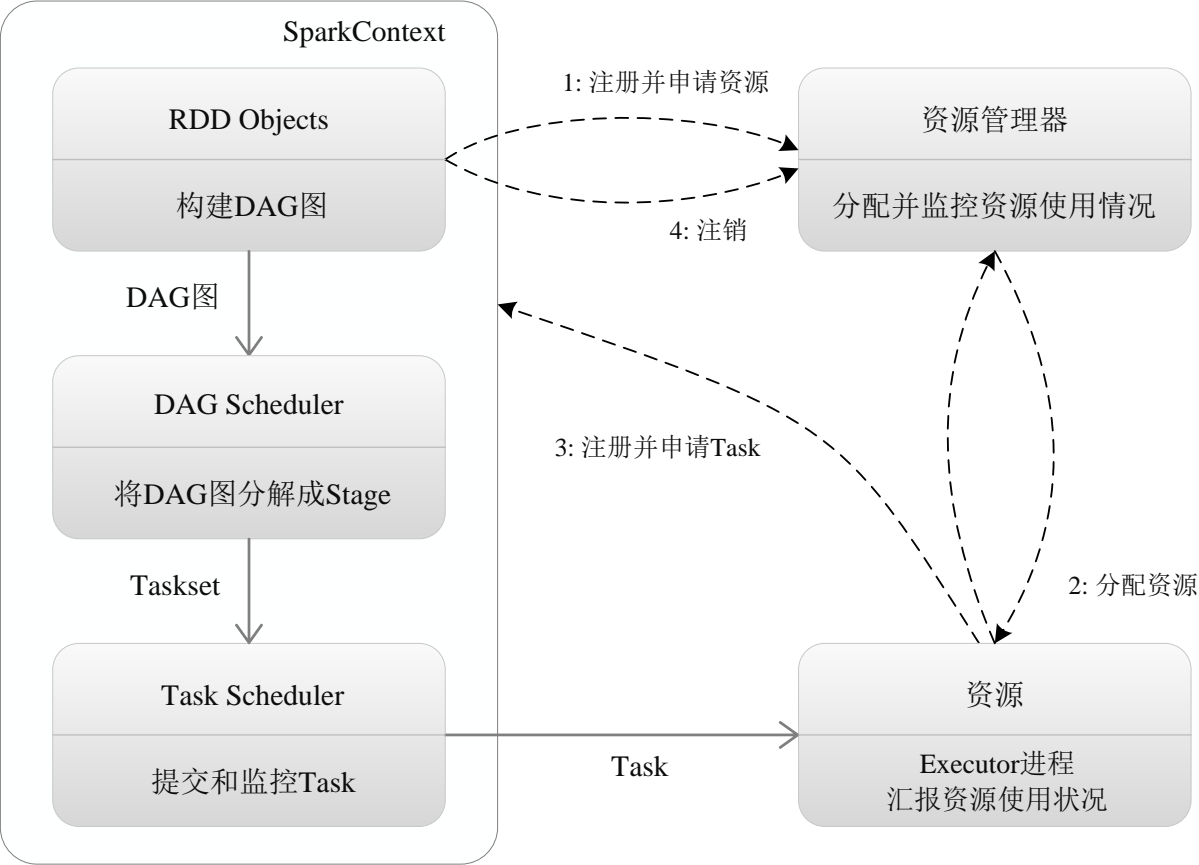
# 架构设计

- 一个Application由一个Driver和若干个Job构成，一个Job由多个Stage构成，一个Stage由多个没有Shuffle关系的Task组成
- 当执行一个Application时，Driver会向集群管理器申请资源，启动Executor，并向Executor发送应用程序代码和文件，然后在Executor上执行Task，运行结束后，执行结果会返回给Driver，或者写到HDFS或者其他数据库中

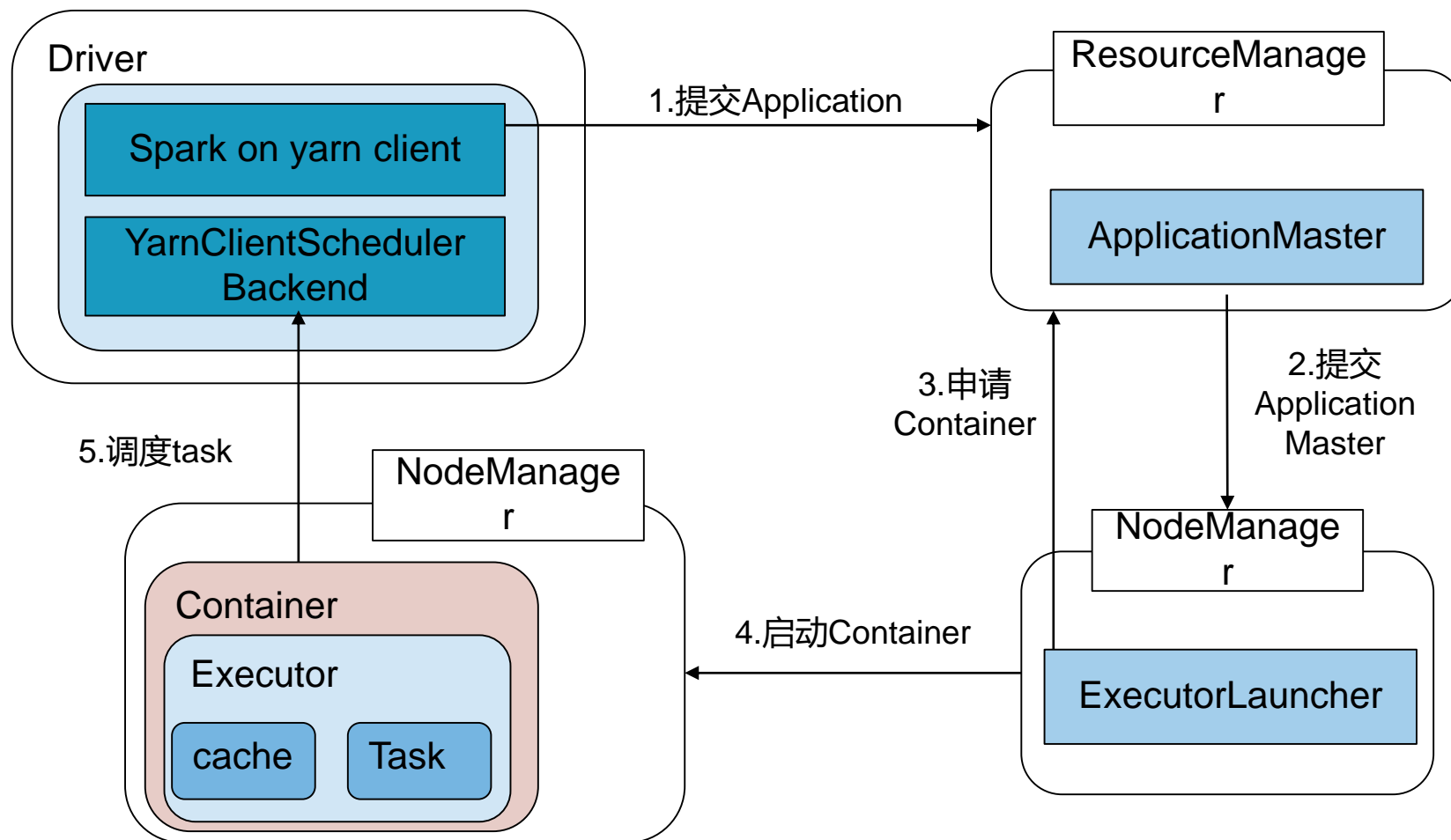




# Spark运行基本流程



# Spark on Yarn-client的运行流程



# Spark运行基本流程

---

Spark运行架构具有以下特点：

- (1) 每个Application都有自己专属的Executor进程，并且该进程在Application运行期间一直驻留。Executor进程以多线程的方式运行Task
- (2) Spark运行过程与资源管理器无关，只要能够获取Executor进程并保持通信即可
- (3) Executor上有一个BlockManager存储模块，用于存储中间结果
- (4) Task采用了数据本地性和推测执行等优化机制

# RDD (Resilient Distributed DataSet)弹性分布式数据集

---

## 设计背景

- 许多迭代式算法（比如机器学习、图算法等）和交互式数据挖掘工具，共同之处是，不同计算阶段之间会重用中间结果
- 目前的MapReduce框架都是把中间结果写入到HDFS中，带来了大量的数据复制、磁盘IO和序列化开销
- RDD就是为了满足这种需求而出现的，它提供了一个抽象的数据架构，我们不必担心底层数据的分布式特性，只需将具体的应用逻辑表达为一系列转换处理，不同RDD之间的转换操作形成依赖关系，可以实现管道化，避免中间数据存储

# RDD概念

---

- 一个RDD就是一个分布式对象集合，本质上是一个只读的分区记录集合，每个RDD可分成多个分区，每个分区就是一个数据集片段，并且一个RDD的不同分区可以被保存到集群中不同的节点上，从而可以在集群中的不同节点上进行并行计算
- RDD提供了一种高度受限的共享内存模型，即RDD是只读的记录分区的集合，不能直接修改，只能基于稳定的物理存储中的数据集创建RDD，或者通过在其它RDD上执行确定的转换操作（如map、join和group by）而创建得到新的RDD

# RDD运行原理

---

- RDD提供了一组丰富的操作以支持常见的数据运算，分为“动作”（Action）和“转换”（Transformation）两种类型
- RDD提供的转换接口都非常简单，都是类似map、filter、groupBy、join等粗粒度的数据转换操作，而不是针对某个数据项的细粒度修改
- 表面上RDD的功能很受限、不够强大，实际上RDD已经被实践证明可以高效地表达许多框架的编程模型（比如MapReduce、SQL、Pregel）
- Spark用Scala语言实现了RDD的API，程序员可以通过调用API实现对RDD的各种操作

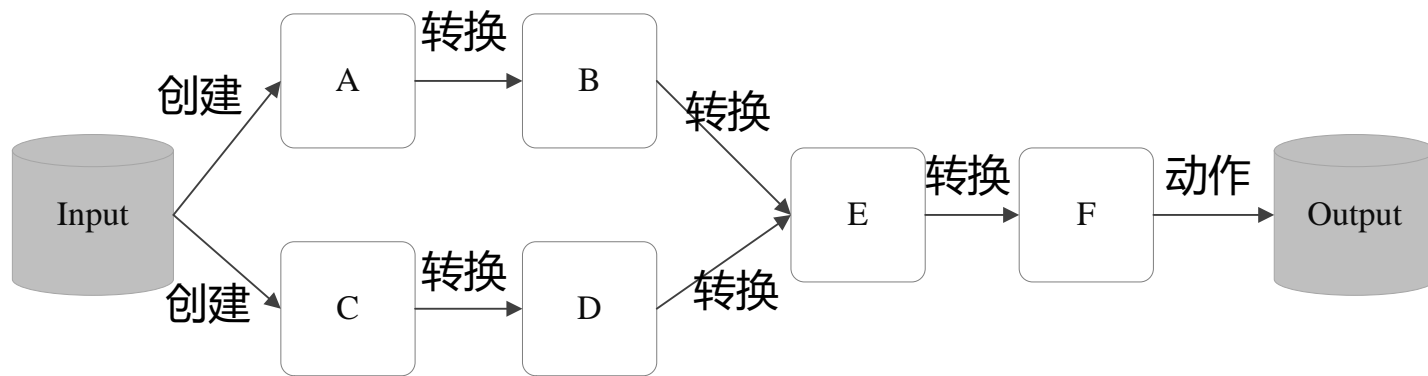
# RDD运行原理

RDD执行过程如下：

- RDD读入外部数据源进行创建
- RDD经过一系列的转换（Transformation）操作，每一次都会产生不同的RDD，供给下一个转换操作使用
- 最后一个RDD经过“动作”操作进行转换，并输出到外部数据源

这一系列处理称为一个Lineage（血缘关系），即DAG拓扑排序的结果

优点：惰性调用、管道化、避免同步等待、不需要保存中间结果、每次操作变得简单



# RDD特性

---

Spark采用RDD以后能够实现高效计算的原因主要在于：

(1) 高效的容错性

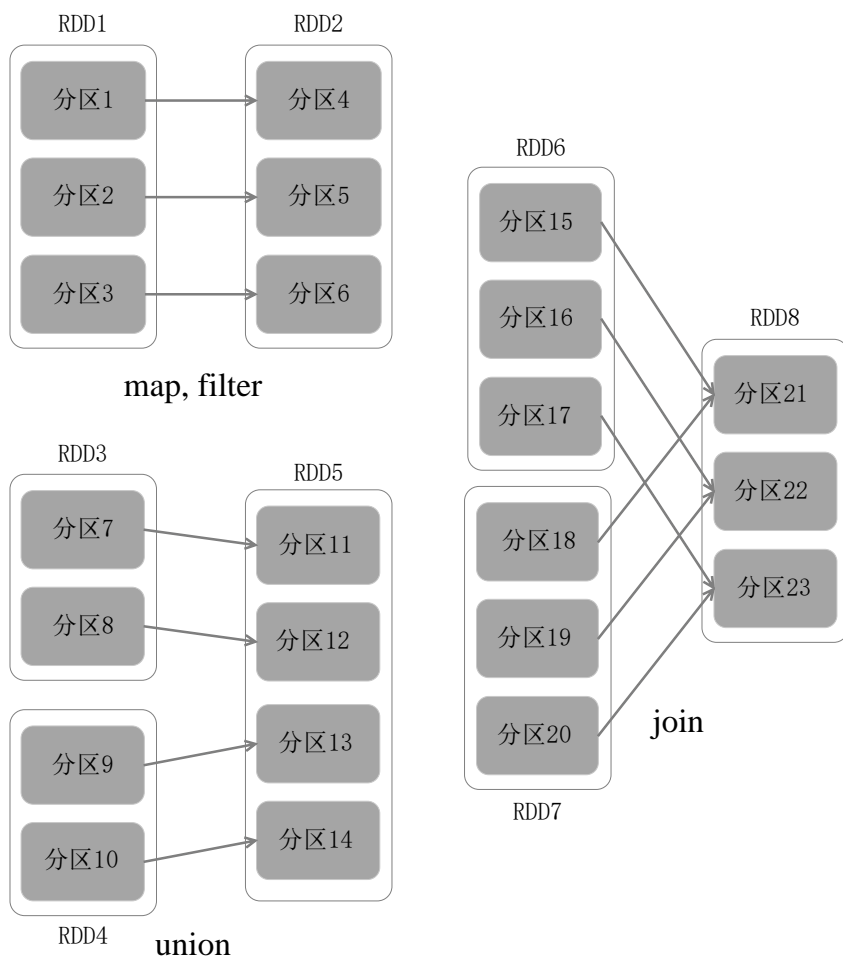
- 现有容错机制：数据复制或者记录日志
- RDD：血缘关系、重新计算丢失分区、无需回滚系统、重算过程在不同节点之间并行、只记录粗粒度的操作

(2) 中间结果持久化到内存，数据在内存中的多个RDD操作之间进行传递，避免了不必要的读写磁盘开销

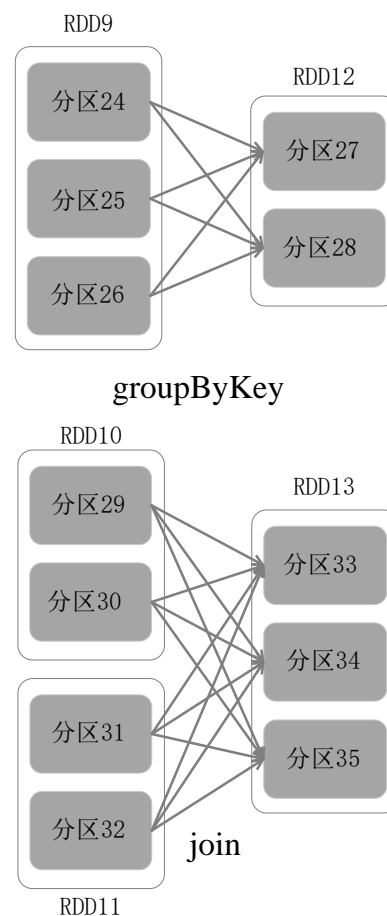
(3) 存放的数据可以是Java对象，避免了不必要的对象序列化和反序列化



# RDD之间的依赖关系



(a)窄依赖



(b)宽依赖

- 窄依赖表现为一个父RDD的分区对应于一个子RDD的分区或多个父RDD的分区对应于一个子RDD的分区
- 宽依赖则表现为存在一个父RDD的一个分区对应一个子RDD的多个分区

# Stage的划分

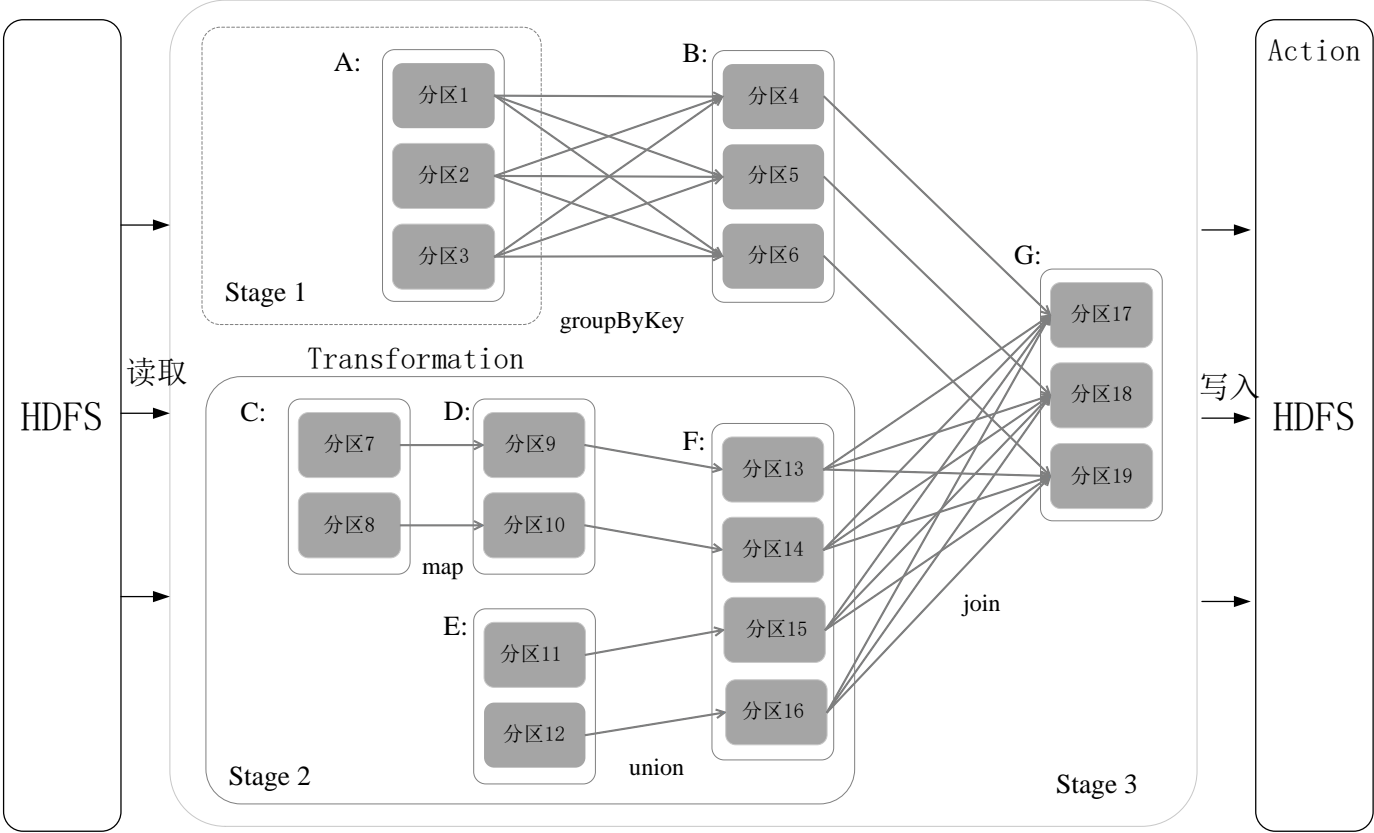
---

Spark通过分析各个RDD的依赖关系生成了DAG，再通过分析各个RDD中的分区之间的依赖关系来决定如何划分Stage，具体划分方法是：

- 在DAG中进行反向解析，遇到宽依赖就断开
- 遇到窄依赖就把当前的RDD加入到Stage中
- 将窄依赖尽量划分在同一个Stage中，可以实现流水线计算

# Stage的划分

下图被分成三个Stage，在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



## 流水线操作实例

分区7通过map操作生成的分区9，可以不用等待分区8到分区10这个map操作的计算结束，而是继续进行union操作，得到分区13，这样流水线执行大大提高了计算的效率

# Stage的划分

---

Stage的类型包括两种：ShuffleMapStage和ResultStage，具体如下：

(1) ShuffleMapStage：不是最终的Stage，在它之后还有其他Stage，所以，它的输出一定需要经过Shuffle过程，并作为后续Stage的输入；这种Stage是以Shuffle为输出边界，其输入边界可以是外部获取数据，也可以是另一个ShuffleMapStage的输出，其输出可以是另一个Stage的开始；在一个Job里可能有该类型的Stage，也可能没有该类型Stage；

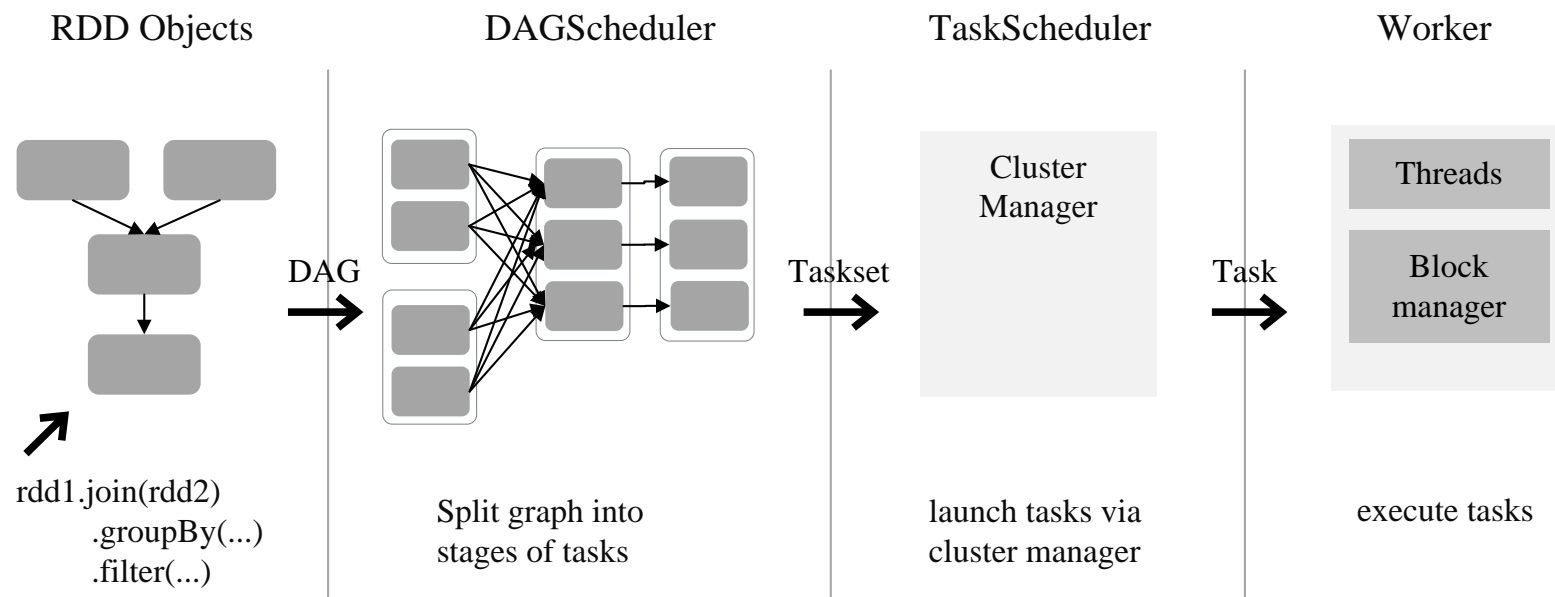
(2) ResultStage：最终的Stage，没有输出，而是直接产生结果或存储。这种Stage是直接输出结果，其输入边界可以是外部获取数据，也可以是另一个ShuffleMapStage的输出。在一个Job里必定有该类型Stage。

因此，一个Job含有一个或多个Stage，其中至少含有一个ResultStage。

# RDD运行过程

RDD在Spark架构中的运行过程：

- (1) 创建RDD对象；
- (2) SparkContext负责计算RDD之间的依赖关系，构建DAG；
- (3) DAGScheduler负责把DAG图分解成多个Stage，每个Stage中包含了多个Task，每个Task会被TaskScheduler分发给各个WorkerNode上的Executor去执行。



# RDD的算子

---

- **Transformation**

- Transformation是通过转换从一个或多个RDD生成新的RDD,该操作是lazy的，当调用action算子，才发起job。
- 典型算子：map、flatMap、filter、reduceByKey等。

- **Action**

- 当代码调用该类型算子时，立即启动job。
- 典型算子：take、count、saveAsTextFile等。

# Transformation

---

Transformation API	说明
filter(func)	筛选出满足函数func的元素，并返回一个新的数据集
map(func)	将每个元素传递到函数func中，并将结果返回为一个新的数据集
flatMap(func)	与map()相似，但每个输入元素都可以映射到0或多个输出结果
groupByKey()	应用于(K,V)键值对的数据集时，返回一个新的(K, Iterable<V>)形式的数据集
reduceByKey(func)	应用于(K,V)键值对的数据集时，返回一个新的(K, V)形式的数据集，其中的每个值是将每个key传递到函数func中进行聚合

# Action

---

Action API	说明
count()	返回数据集中的元素个数
collect()	以数组的形式返回数据集中的所有元素
first()	返回数据集中的第一个元素
take(n)	以数组的形式返回数据集中的前n个元素
reduce(func)	通过函数func（输入两个参数并返回一个值）聚合数据集中的元素
foreach(func)	将数据集中的每个元素传递到函数func中运行