Spark分布式计算

美柚@郭厦同

alien.xt.xm@gmail.com

http://github.com/alienxt

目录

Spark是什么？

Apache Spark is a fast and general engine for large-scale data processing.

Apache Spark is an open source cluster computing system that aims to make data analytics fast.

为什么使用Spark？

01

快速的

处理速度快于Hadoop，
10x - 100x倍

02

多语言支持

可以各种语言开发，支持Java, Scala, Python, R

03

通用性

结合了sql，mllib，graphx，streaming库

04

分布式可水平扩展

master/slave结构，master管理worker，master支持ha，worker可扩展

05

Runs Everywhere

数据源丰富
随地可部署

06

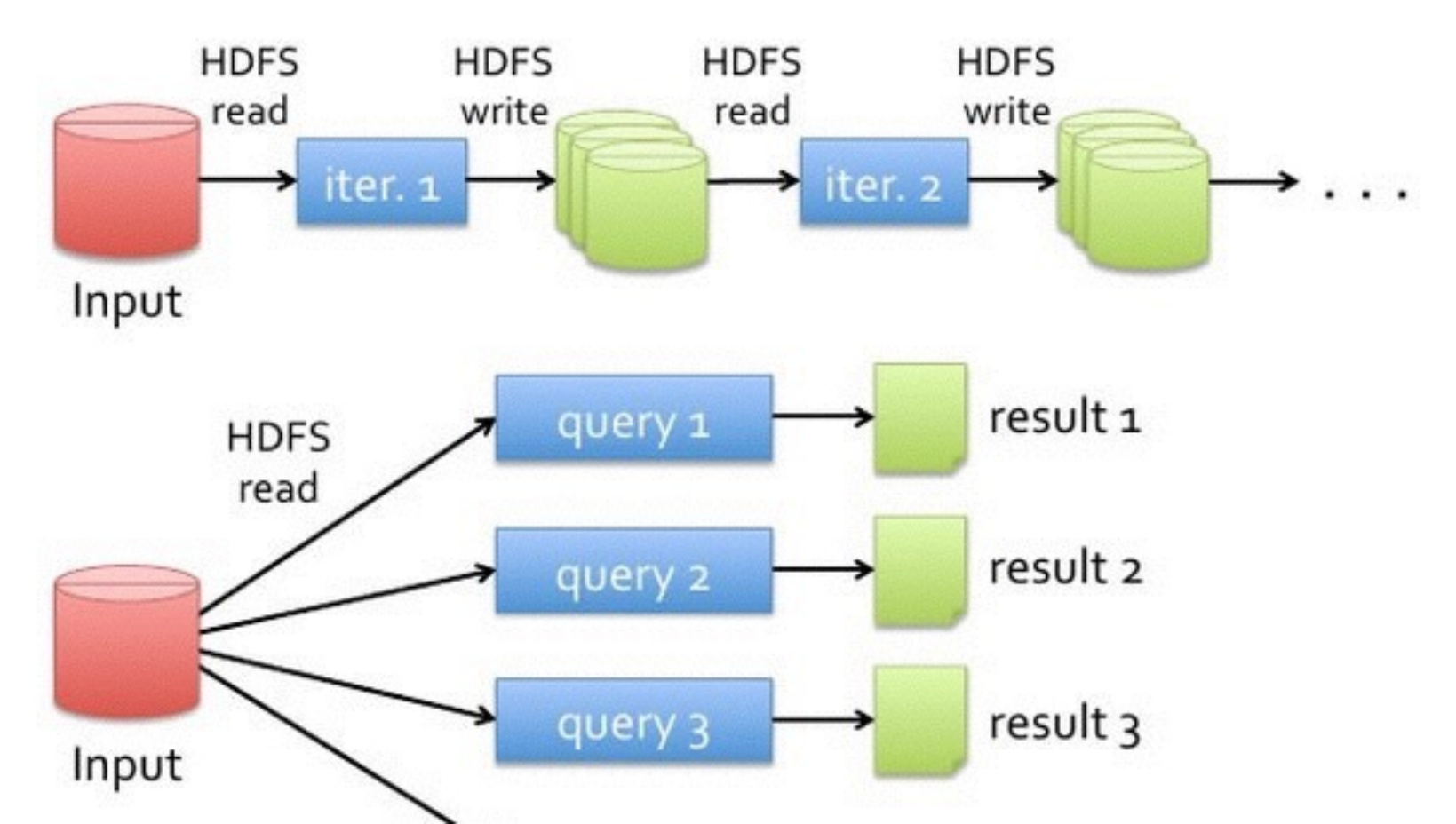
本地模式

Spark有一个本地模式，可以在开发和测试中，完全模拟集群

Spark vs Hadoop

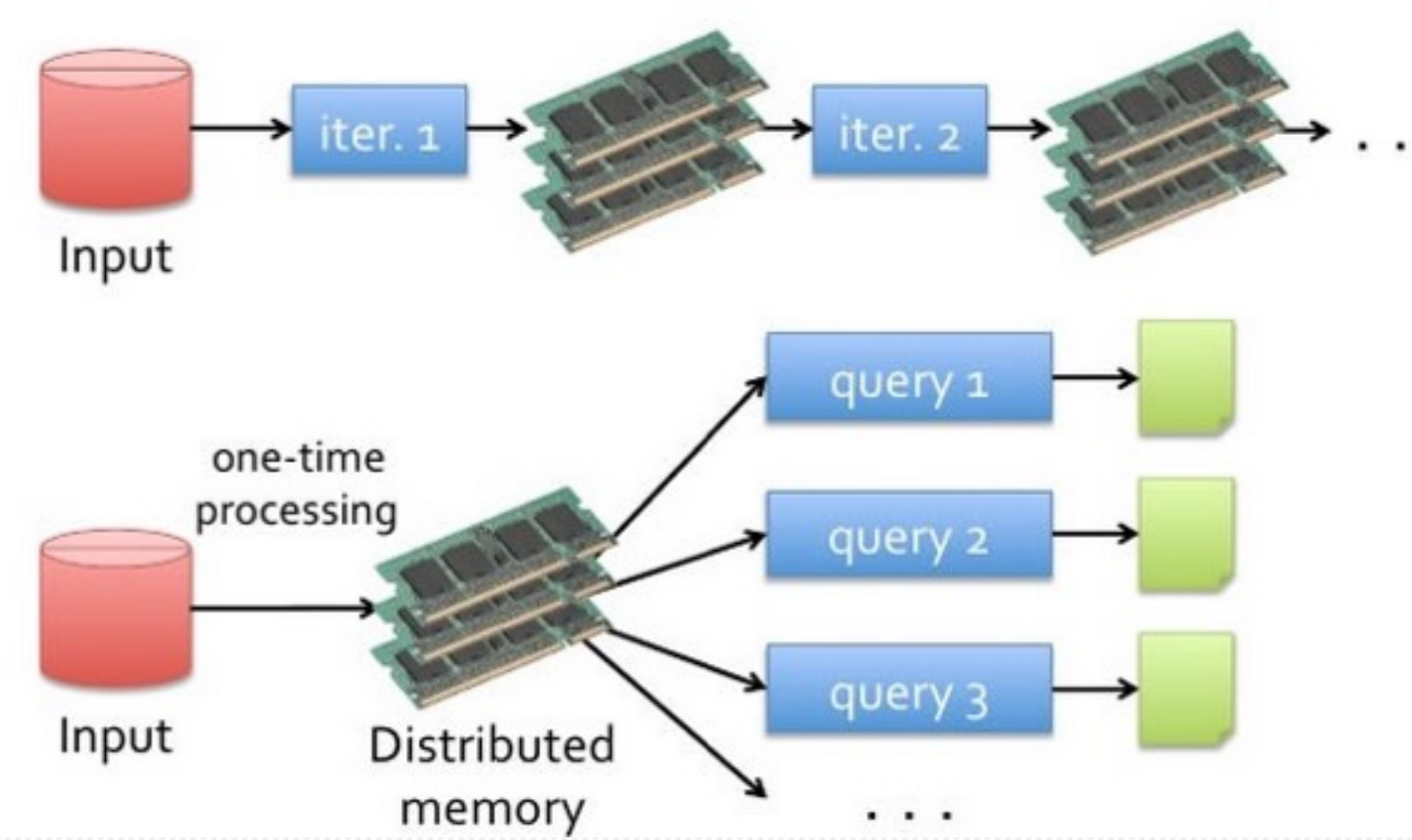
Hadoop缺点

- 计算的中间结果，保存在磁盘，产生大量的io，降低计算速度



Spark为什么快？

- 减少磁盘io
- DAG（统筹性运算）
- 各组件相通



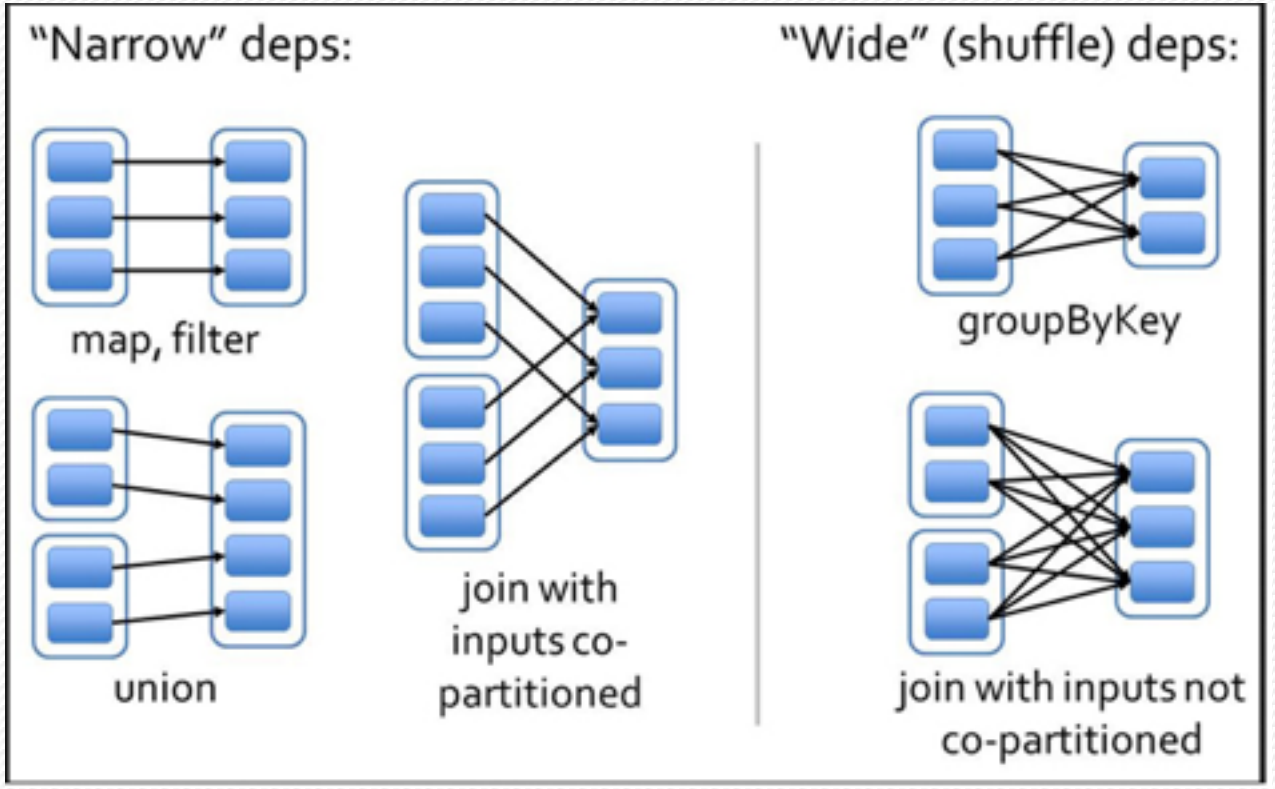
过渡页

Spark核心及基础概念

1. RDD
2. 基础概念
3. 实践

RDD是什么？

RDD：Spark的核心概念是RDD (resilient distributed dataset)，指的是一个只读的，可分区的分布式数据集，这个数据集的全部或部分可以缓存在内存中，在多次计算间重用。



特性：

- 分区性
- 只读性
- 函数性
- 容错性

map(func)	通过将源数据中的每个元素传递给***func***函数来返回一个新的***RDD***
filter(func)	从源数据中选出那些传递给***func***函数后返回***true***的元素构成的新***RDD***
flatMap(func)	与***map***类似，但是每个输入元素可以被映射为***0***或更多的输出元素（所以***func***应该返回一个序列而不是单个元素）
mapPartitions(func)	与***map***类似, 但是在***RDD***的每个分区上分开运算,***mapPartitions()***的输入函数是应用于每个分区
mapPartitionsWithIndex(func)	与***mapPartitions***类似, 但是提供了一个整型参数给函数***func***,代表了分区的索引。所以函数***func*** 的类型必须为 ***(Int, Iterator<T>) => Iterator<U>*** ， 当***RDD***的类型是***RDD[T].***
sample(withReplacement, fraction, seed)	对数据的一部分进行采样,可选的替换参数, 使用一个给定的随机数发生器种子。
union(otherDataset)	返回一个新的数据集，包含了参数与源数据的并集。
intersection(otherDataset)	返回一个新的***RDD***与参数的交集。
distinct([numTasks]))	返回一个新的数据集包含了与源数据集的不同元素
groupByKey([numTasks])	
reduceByKey(func, [numTasks])	
sortByKey([ascending], [numTasks])	
join(otherDataset, [numTasks])	

reduce(func)	聚集，传入的函数是两个参数输入返回一个值，这个函数必须是满足交换律和结合律的
collect()	一般在***filter***或者足够小的结果的时候，再用***collect***封装返回一个数组
count()	返回的是***dataset***中的***element***的个数
first()	返回的是***dataset***中的第一个元素
take(n)	返回前***n***个***elements***，这个士***driverprogram***返回的
takeSample(withReplacement，num，seed)	抽样返回一个***dataset***中的***num***个元素，随机种子***seed***
saveAsTextFile（path）	把***dataset***写到一个***textfile***中，或者***hdfs***，或者***hdfs***支持的文件系统中，***Spark***把每条记录都转换为一行记录，然后写到***file***中
saveAsSequenceFile(path)	只能用在***key-value***对上，然后生成***SequenceFile***写到本地或者***hadoop***文件系统
countByKey()	返回的是***key***对应的个数的一个***map***，作用于一个***RDD***
foreach(func)	对***dataset***中的每个元素都使用***func***

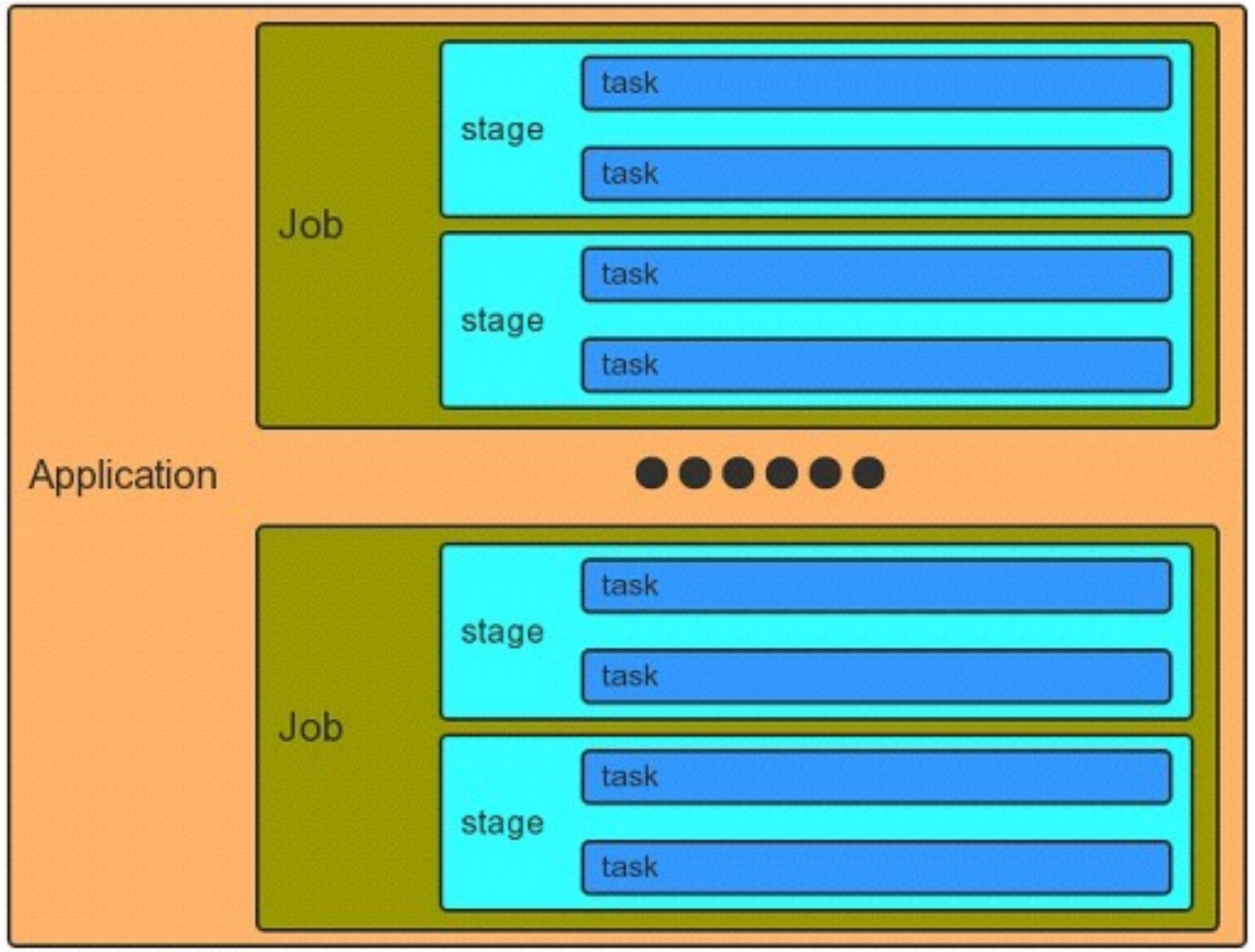
persist()	***persist*方法手工设定*StorageLevel*来满足工程需要的存储级别**
cache()	***RDD*的*cache()*方法其实调用的就是*persist*方法，缓存策略均为*MEMORY_ONLY*；**
unpersist()	**清空缓存**

StorageLevel

https://github.com/apache/spark/blob/v1.6.1/core/src/main/scala/org/apache/spark/storage/StorageLevel.scala

Spark的基本概念

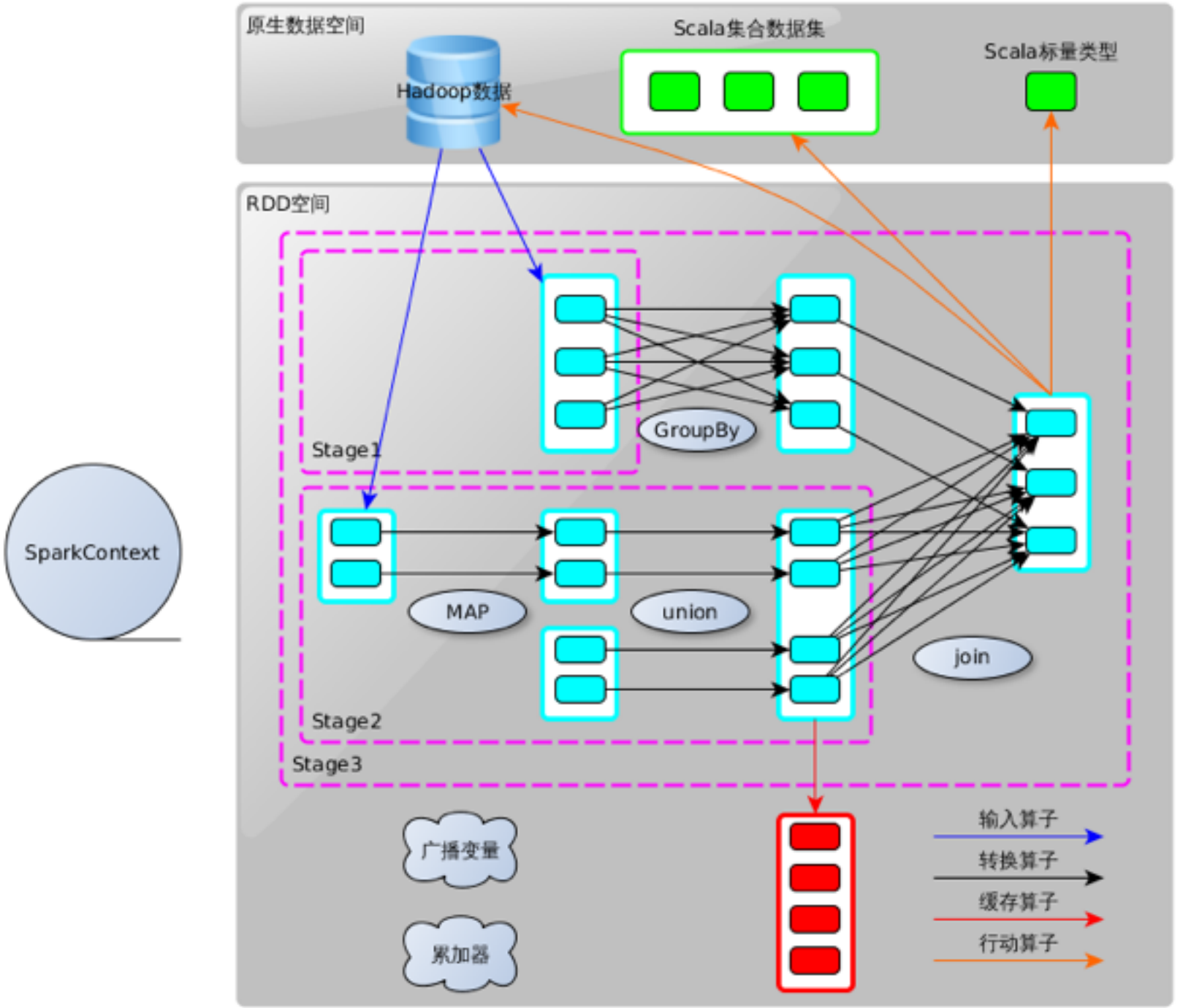
- Application：基于Spark的用户程序，包含了一个driver program和多个executor
- Driver Program：运行application的main函数，并且创建SparkContext
- Executor：application中运行在worker node上的一个进程，该进程负责运行task，并且将数据存在内存或磁盘中，executor是独立的进程
- Cluster Manager：在集群上获取资源的外部服务（如standalone,mesos,yarn,cloud）
- Worker node：集群中负责运行运行executor
- Task：在executor中的工作单元，线程
- Job：包含多个task组成的计算，由action操作催生
- Stage：每个job会拆分成多组的task，每组task称为一个stage，也称为taskset
- RDD：基本计算单元，可以通过一系列的算子进行操作

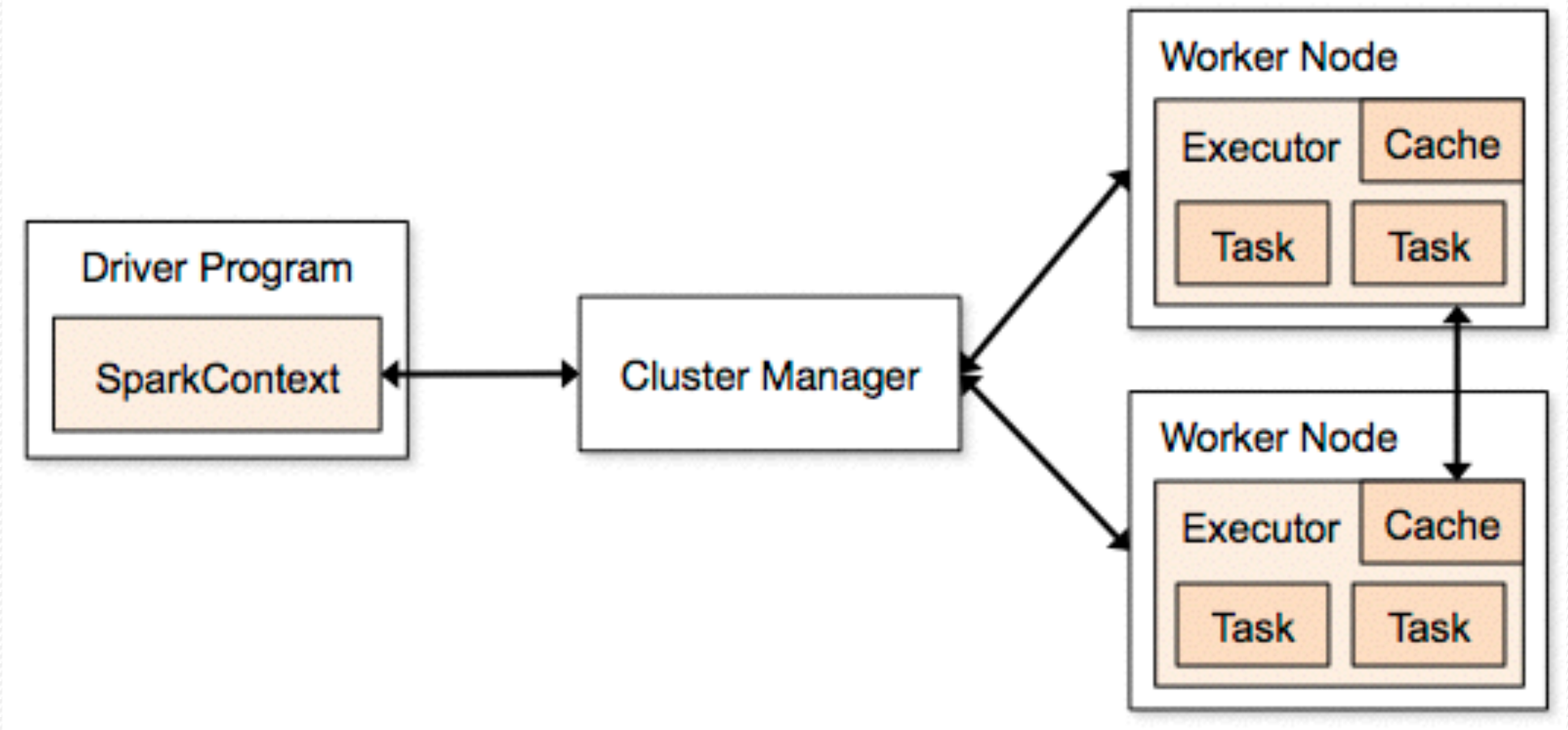

Application
Job
stage
task
task
stage
task
task
Job
stage
task
task
stage
task
task

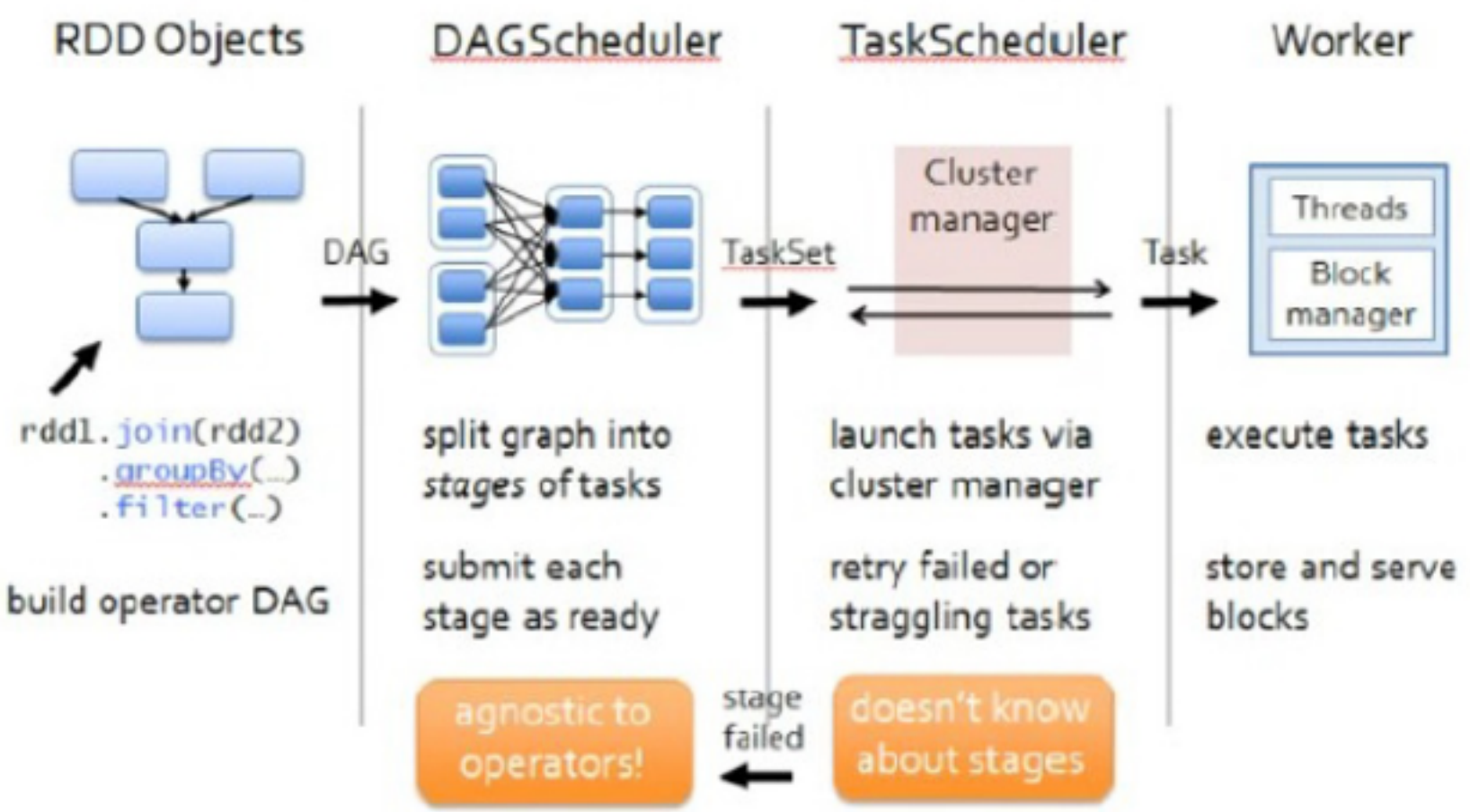
Let's try it.

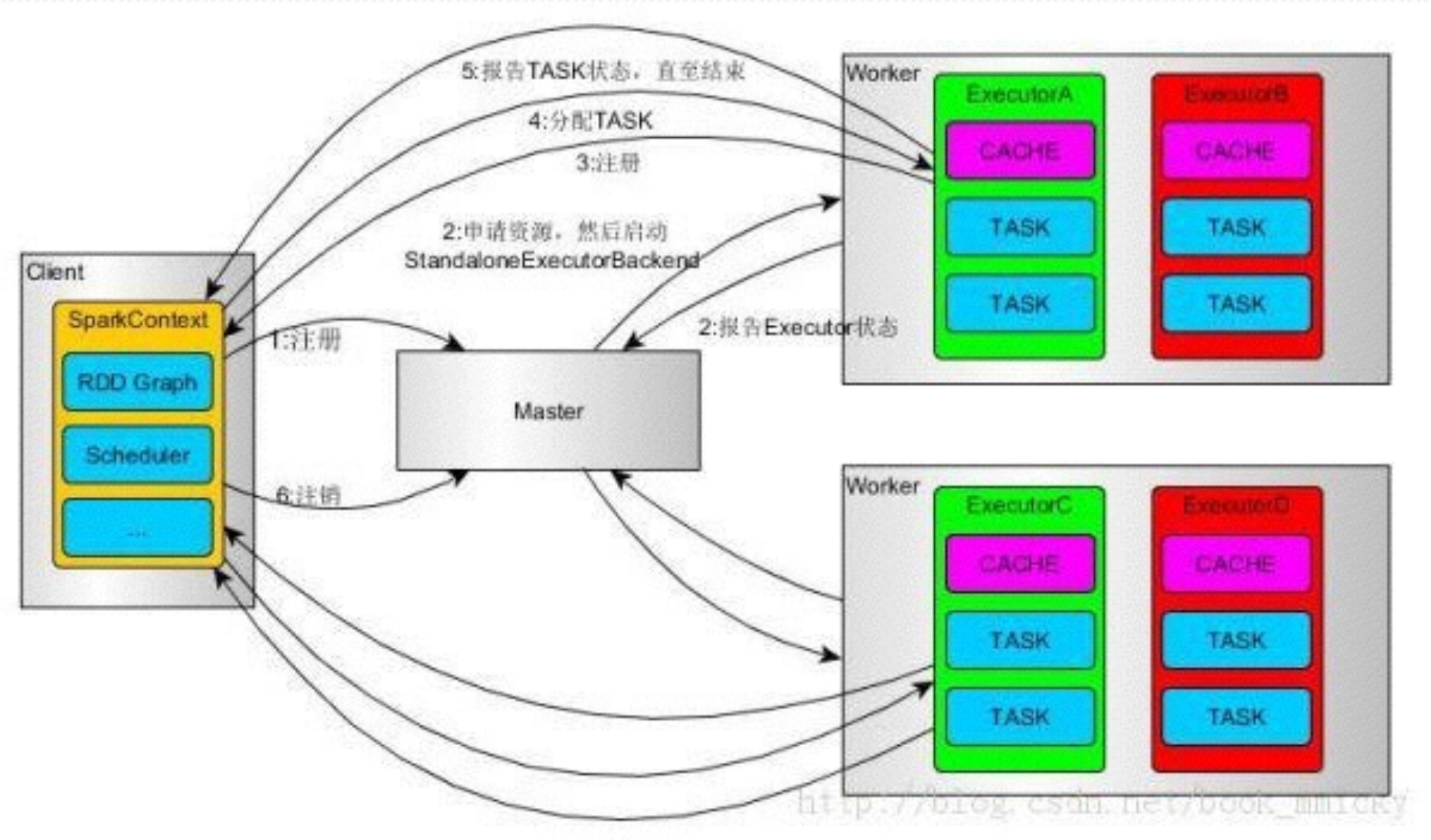
过渡页

Spark运行架构


原生数据空间
Hadoop数据
Scala集合数据集
Scala标量类型
RDD空间
SparkContext
Stage1
GroupBy
MAP
union
join
Stage2
Stage3
广播变量
累加器
输入算子
转换算子
缓存算子
行动算子


Driver Program
SparkContext
Cluster Manager
Worker Node
Executor
Cache
Task
Task
Worker Node
Executor
Cache
Task
Task


RDD Objects
DAGScheduler
TaskScheduler
Worker
DAG
TaskSet
Cluster manager
Task
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


5:报告TASK状态，直至结束
4:分配TASK
3:注册
2:申请资源，然后启动
StandaloneExecutorBackend
2:报告Executor状态
1:注册
6:注销
Client
SparkContext
RDD Graph
Scheduler
...
Master
Worker
ExecutorA
CACHE
TASK
TASK
ExecutorB
CACHE
TASK
TASK
Worker
ExecutorC
CACHE
TASK
TASK
ExecutorD
CACHE
TASK
TASK

过渡页

Spark SQL核心与实践

1. 编程模型DataFrame&DataSets
2. 实践

DataFrame && Datasets

Spark SQL 是 Spark 生态系统里用于处理结构化大数据的模块，它的前身是 Shark。随着 Spark 自身的发展，Spark 团队开始试图放弃 Shark 这个对于 Hive 有太多依赖 (查询优化，语法解析) 的东西，于是才有了 Spark SQL 这个全新的模块

该模块里最重要的概念就是 DataFrame,DataFrame 是基于早期版本中的 SchemaRDD，所以很自然的使用分布式大数据处理的场景。Spark DataFrame 以 RDD 为基础，但是带有 Schema 信息，它类似于传统数据库中的二维表格。

Spark SQL 模块目前支持将多种外部数据源的数据转化为 DataFrame，并像操作 RDD 或者将其注册为临时表的方式处理和分析这些数据。当前支持的数据源有：

- Json
- 文本文件
- RDD
- 关系数据库
- Hive
- Parquet

一旦将 DataFrame 注册成临时表，我们就可以使用类 SQL 的方式操作这些数据，我们将在下文的案例中详细向读者展示如何使用 Spark SQL/DataFrame 提供的 API 完成数据读取，临时表注册，统计分析等步骤。

Let's try it.

过渡页

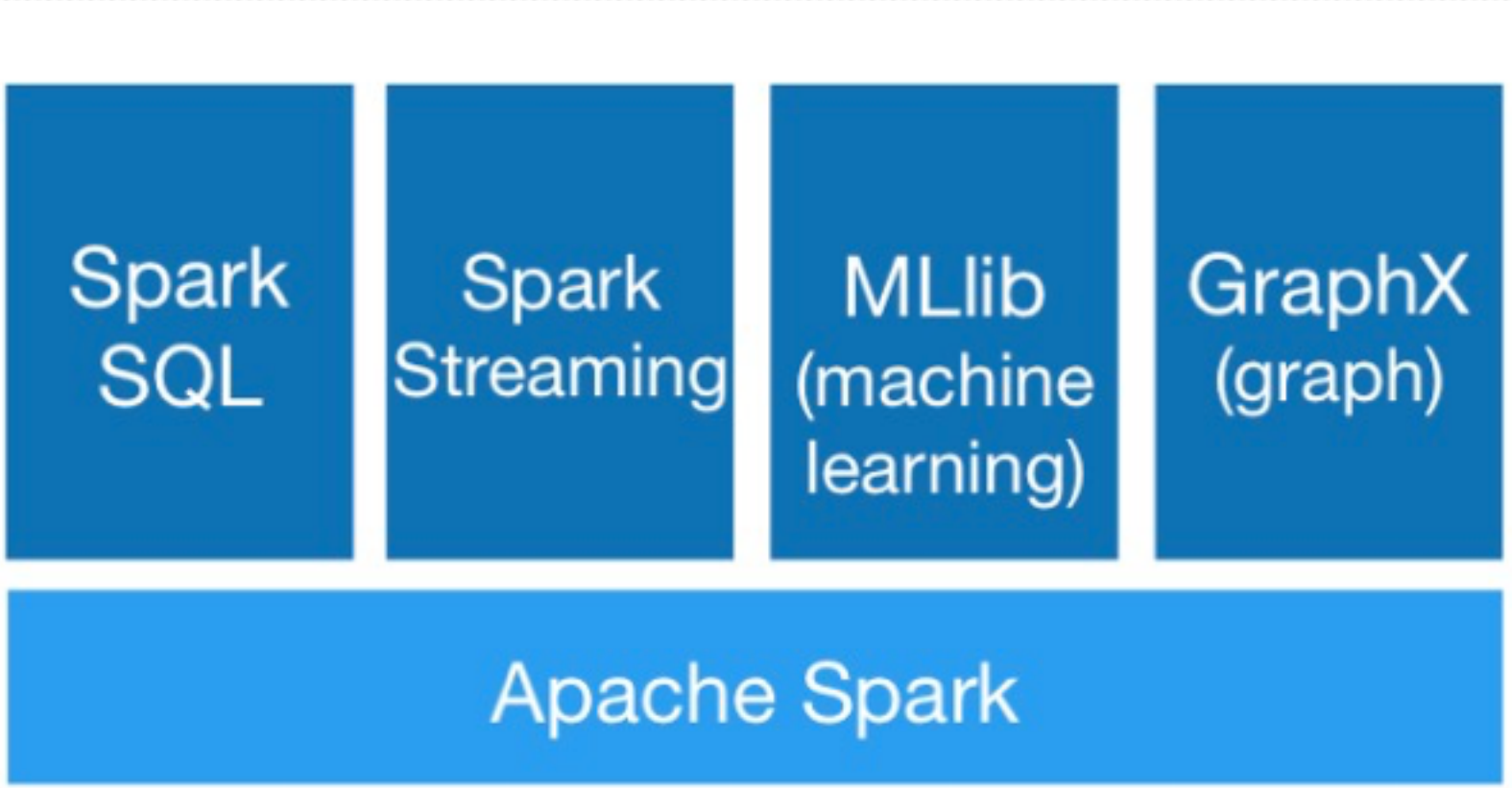
Streaming核心与实践

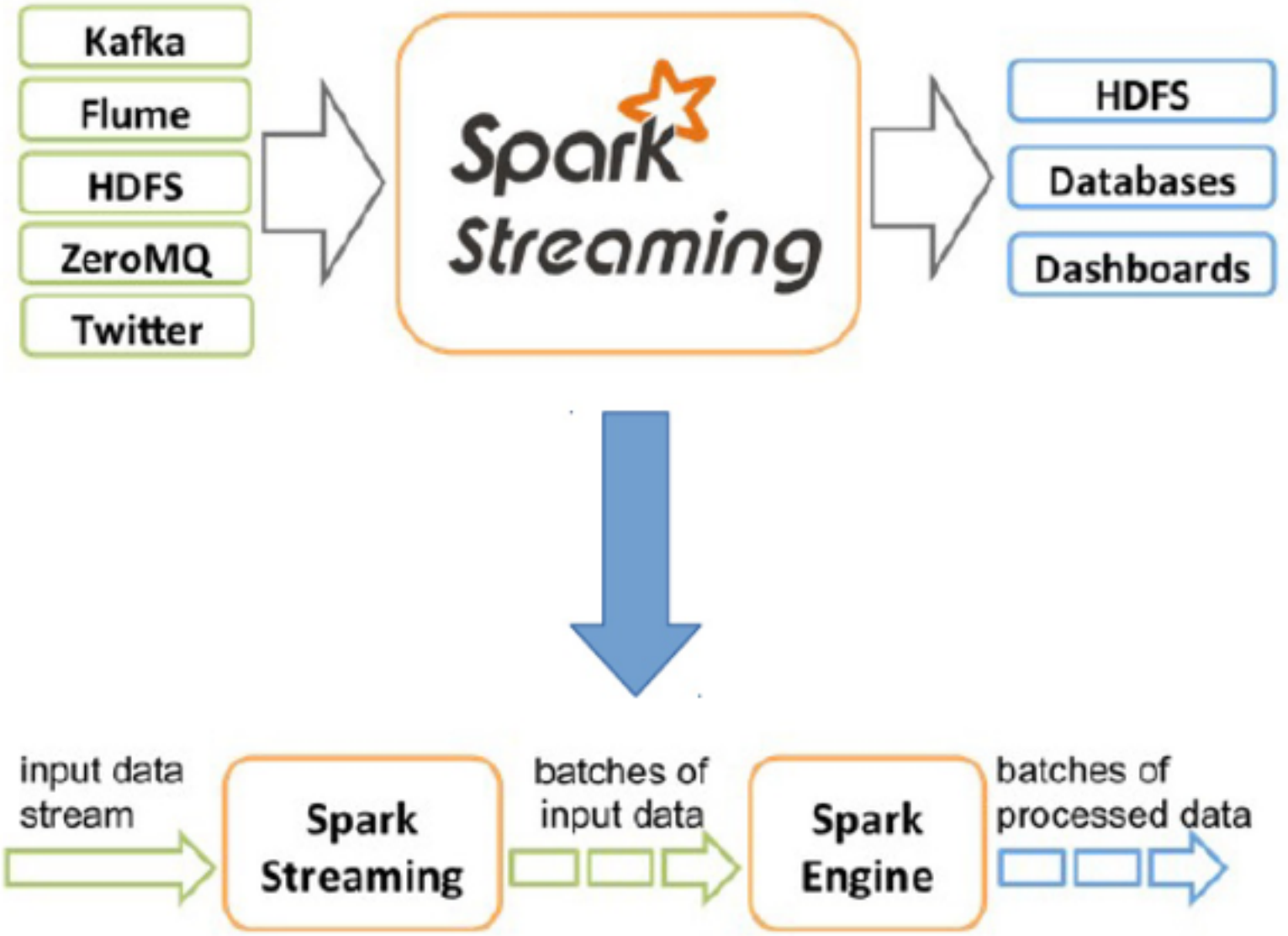
1. 运行原理
2. 运用场景
3. 编程模型DStream
4. 持久化和Checkpointing机制
5. 实践

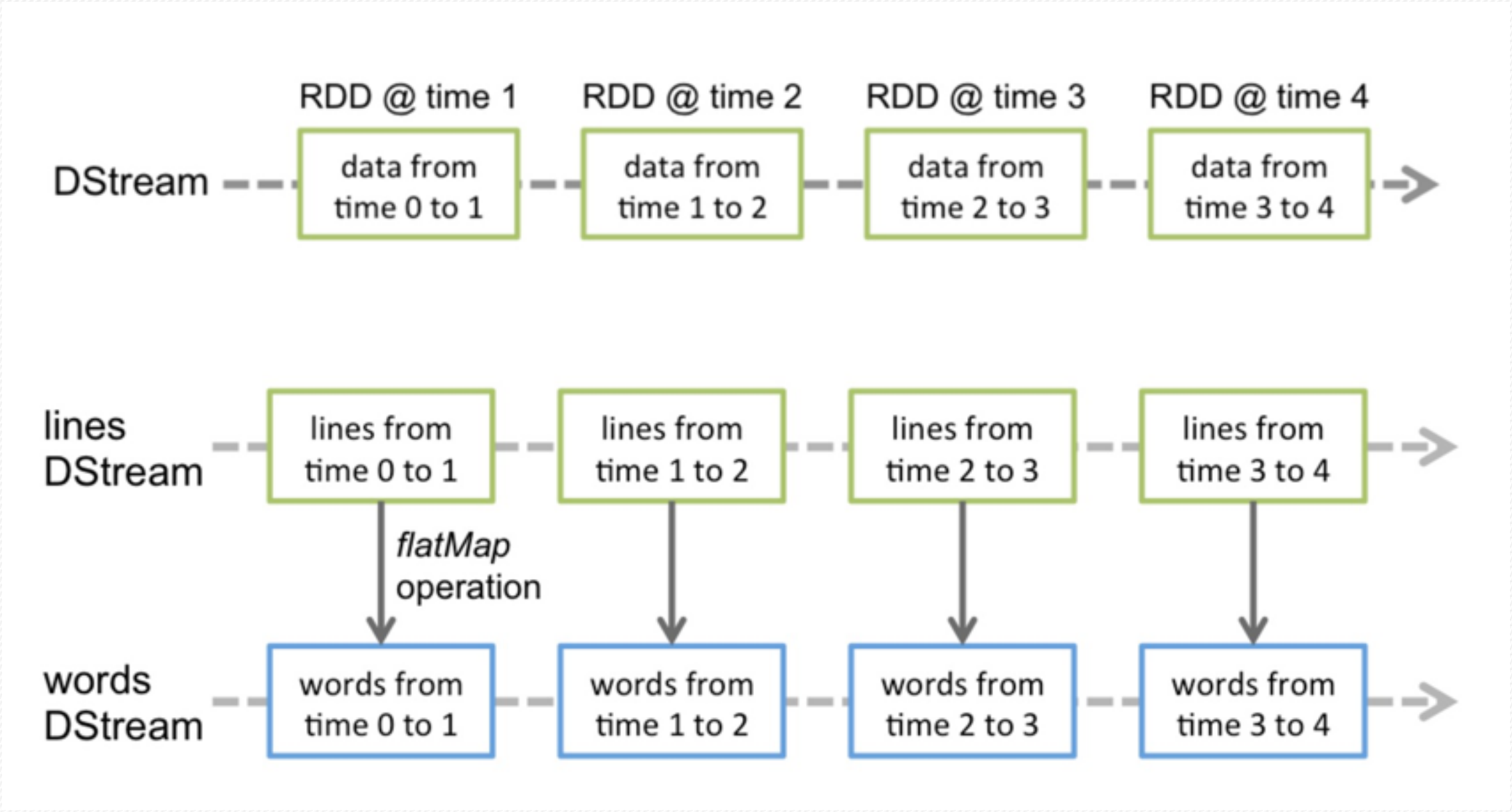
Spark Streaming运行原理

是什么？

- 低延迟
- 横向扩展
- 容错性
- 可结合其他组件




Kafka
Flume
HDFS
ZeroMQ
Twitter
Spark Streaming
HDFS
Databases
Dashboards
input data stream
Spark Streaming
batches of input data
Spark Engine
batches of processed data


RDD @ time 1
RDD @ time 2
RDD @ time 3
RDD @ time 4
DStream
data from time 0 to 1
data from time 1 to 2
data from time 2 to 3
data from time 3 to 4
lines DStream
lines from time 0 to 1
lines from time 1 to 2
lines from time 2 to 3
lines from time 3 to 4
flatMap operation
words DStream
words from time 0 to 1
words from time 1 to 2
words from time 2 to 3
words from time 3 to 4

Streaming的运行场景

- 无状态操作
- 状态操作
  - UpdateStateByKey
- window操作

编程模型DStream

- Discretized Stream是Streaming的基础抽象
- DStream是RDD的系列
- DStream拥有大部分的RDD操作
- 从流输入源创建，从DStream之间transformation而来

Input Sources

- 外部组件

  - Kafka, Flume, ZeroMQ, Akka, Akka Actors, TCP Sockets, HDFS

- 自定义接收器
- 生成RDD系列，作为流

Source	Artifact
Kafka	spark-streaming-kafka_2.10
Flume	spark-streaming-flume_2.10
Kinesis	spark-streaming-kinesis-asl_2.10 [Amazon Software License]
Twitter	spark-streaming-twitter_2.10
ZeroMQ	spark-streaming-zeromq_2.10
MQTT	spark-streaming-mqtt_2.10

Output Operations on DStreams

print()	打印到控制台
foreachRDD(func)	对***Dstream***里面的每个***RDD***执行***func***，保存到外部系统
saveAsObjectFiles(prefix, [suffix])	保存流的计算结果为***SequenceFile,*** 文件名***: "prefix-TIME_IN_MS[.suffix]".***
saveAsTextFiles(prefix, [suffix])	保存流的计算结果为文本文件, 文件名***: "prefix-TIME_IN_MS[.suffix]".***
saveAsHadoopFiles(prefix, [suffix])	保存流的计算结果为***hadoop***文件, 文件名***: "prefix-TIME_IN_MS[.suffix]".***

Transformations on DStreams

map(func)	对每一个元素执行***func***方法
flatMap(func)	类似***map***函数，但是可以***map***到***0+***个输出
filter(func)	过滤
repartition(numPartitions)	增加分区，提高并行度
union(otherStream)	合并两个流
count()	统计元素的个数
reduce(func)	对***RDDs***里面的元素进行聚合操作，***2***个输入参数，***1***个输出参数
countByValue()	针对类型统计，当一个***Dstream***的元素的类型是***K***的时候，调用它会返回一个新的***Dstream***，包含***<K,Long>***键值对，***Long***是每个***K***出现的频率。
reduceByKey(func, [numTasks])	对于一个***(K, V)***类型的***Dstream***，为每个***key***，执行***func***函数，默认是***local***是***2***个线程，***cluster***是***8***个线程，也可以指定***numTasks***
join(otherStream, [numTasks])	把***(K, V)***和***(K, W)***的***Dstream***连接成一个***(K, (V, W))***的新***Dstream***
transform(func)	转换操作，把原来的***RDD***通过***func***转换成一个新的***RDD***
updateStateByKey(func)	针对***key***使用***func***来更新状态和值，可以将***state***该为任何值

Window Operations

window(windowLength, slideInterval)	返回一个根据制定的窗口大小和滑动大小的***dstream***
countByWindow(windowLength, slideInterval)	返回流中窗口滑动的大小技术
reduceByWindow(func, windowLength, slideInterval)	对指定的窗口进行***func***的操作
reduceByKeyAndWindow(func, windowLength, slideInterval, [numTasks])	对流中***kv***类型的窗口，进行***reduceByKey***，可以设置***task***个数
reduceByKeyAndWindow(func, invFunc, windowLength, slideInterval, [numTasks])	***…***
countByValueAndWindow(windowLength, slideInterval, [numTasks])	对流中***kv***类型的窗口，进行***countByValue***，可以设置***task***个数

持久化 & Checkpointing

- 持久化
  - 通过persist()来持久化
  - 默认的持久化级别是MEMORY_ONLY_SER
  - 对于来源于网络的数据源采用MEMORY_AND_DISK_SER_2

- Checkpointing
  - window和stateful操作必须checkpoint
  - ssc.checkpoint指定目录，dstrem.checkpoint指定时间间隔
  - 间隔必须是窗口slide interval的倍数
  - 合理的设置checkponit时间

Let's try it.

THANKS!

alien.xt.xm@gmail.com

http://github.com/alienxt