# 1 Spark

## 1.1 SQLContext

### 1.1.1 SparkContext，SQLContext，SparkSession的区别

SparkContext是Spark 1.\*中的Spark SQL的入口，必须先创建SparkContext。再用SparkContext创建SQLContext。

图1（先创建SparkContext，再用SparkContext创建SQLContext）



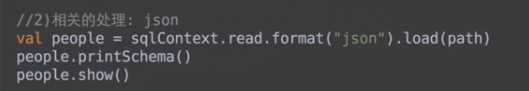
SparkSession是Spark2.\*中的入口。

图2



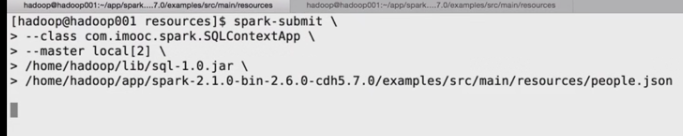
### 1.1.2 SQLContext读取JSON文件

图1



## 1.2 服务器上运行Spark SQL的jar包

图1

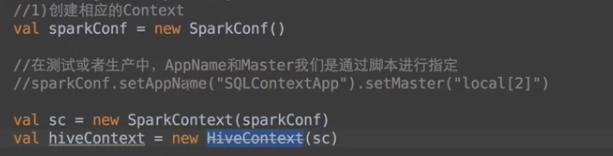


## 1.3 HiveContext

SparkContext是Spark 1.\*中的Spark SQL的入口，必须先创建SparkContext。再用SparkContext创建HiveContext。

（注意：使用HiveContext是不需要搭建Hive环境的）

图1



### 1.3.1 HiveContext显示table的数据

图1



## 1.4 SparkSession

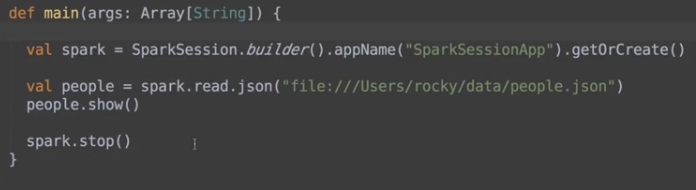
SparkSession是Spark2.\*中的入口。

图1



### 1.4.1 SparkSession读取JSON文件

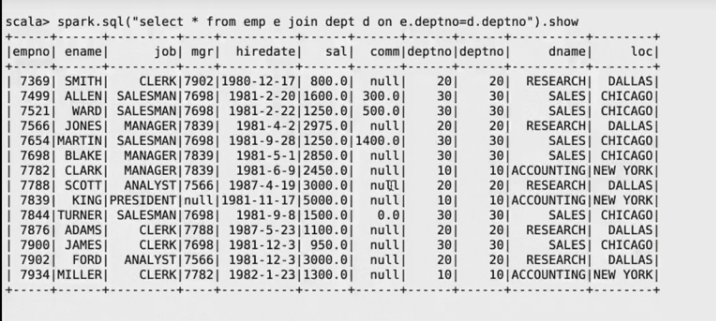
图1



## 1.5 Spark SQL相比Hive的速度

Spark SQL比Hive SQL的执行速度快很多。

图1



## 1.6 Spark thriftserver

启动thriftserver，然后通过客户端（beeline/code）去连接该thriftserver。

图1



图2



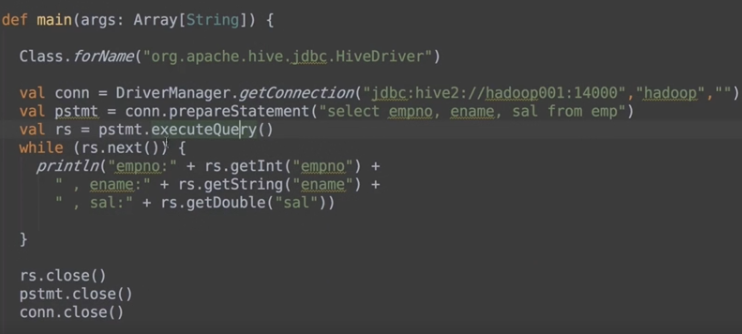
### 1.6.1 thriftserver和spark-shell/spark-sql的区别

1.每一次启动spark-shell或spark-sql都会启动一个Spark Application。

2.启动thriftserver时会启动一个Spark Application，但后面无论多少个客户端连接，永远都是一个Spark Application。

### 1.6.2 JDBC访问thriftserver

图1



## 1.7 DataFrame和DataSet

DataSet是一个分布式数据集，DataFrame是一个以列（列名，列类型，列值）的形式构成的分布式数据集。

图1



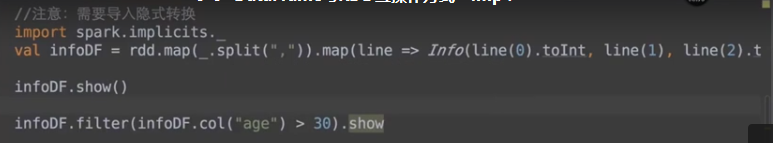
图2（DataFrame的基本API）



### 1.7.1 反射的方式根据RDD获取DataFrame

这种方式必须预先知道schema。

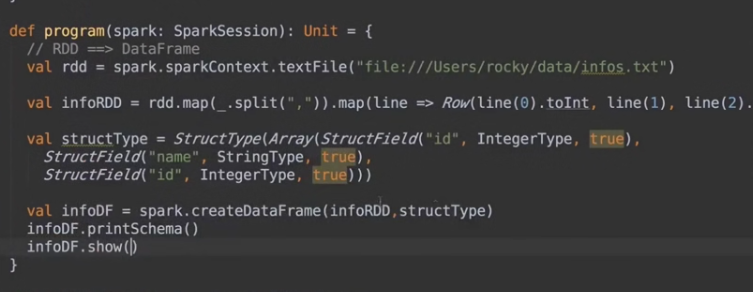
图1



### 1.7.2 structType的方式根据RDD获取DataFrame

这种方式不需要预先知道schema。

图1



## 1.8 Spark运行模式

### 1.8.1 local

local一般用来开发调试Spark应用程序。

### 1.8.2 Standlone

Standlone使用Spark自带的资源管理和调度器，运行Spark集群，Spark集群使用Master/Slave结构。

### 1.8.3 Apache Mesos

使用著名的Mesos资源管理框架，运行Spark集群。这种模式将资源管理交给Mesos，Spark只负责任务调度和计算。

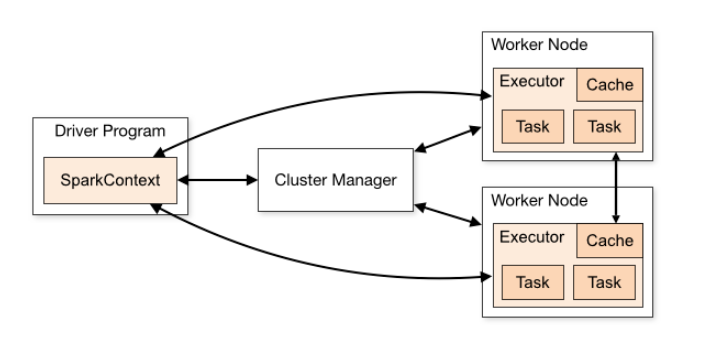
### 1.8.4 Hadoop YARN

使用 Hadoop的YARN资源管理器，运行Spark集群。这种模式将资源管理交给YARN，Spark只负责任务调度和计算。

## 1.9 Spark组件架构

原文链接：<https://www.cnblogs.com/miqi1992/p/5621268.html>

图1



Driver Program：每个Spark应用（Spark Application）都由一个驱动程序（Driver Program）来发起集群的所有并行操作。

Cluster Manager：资源管理器的主节点，在Standlone模式下是Master节点，在YARN模式下是ResourceManager。

Worker Node：资源管理器的从节点，在Standlone模式下是Slave节点，在YARN模式下是NodeManager，负责启动Executor。

Executor：执行器，在Worker Node上启动，负责计算任务（Task）。

SparkContext：整个Spark应用（Spark Application）的上下文，负责控制应用的生命周期。

# 2 Spark Streaming

Spark Streaming是一个实时流处理框架。

## 2.1 实时流处理

### 2.1.1 为什么需要实时流处理

比如对于需求：统计慕课网主站上每个课程的地域信息（由IP可以得到地域信息）。

对于1小时，10分钟，1分钟级别需求或许都可以采用离线计算（mapreduce）的方式，但对于秒级别的需求只能用实时流处理，因为离线计算可能光启动进程就已经超过1秒了。

### 2.1.2 什么时候需要进行实时流处理

1.时效性高：比如，秒级别的需求

2.数据量大

### 2.1.3 实时流处理的特点

1.实时：时效性高

2.流式：数据连绵不断（像滔滔江水连绵不绝），7\*24小时连续运作

### 2.1.4 实时流计算和离线计算的区别

数据来源：

1. 离线计算的数据来源于HDFS上的历史数据，数据量很大。
2. 实时流计算的数据来源于消息队列（比如kafka），数据量一般是实时新增/修改的某一笔增量数据，增量数据很小。

处理过程：

1. 离线计算用mapreduce处理数据。
2. Spark实时流计算用Spark（DStream/SS）处理数据。

处理速度：

1. 离线慢，实时块。

进程：

1. 离线计算map进程和reduce进程用完销毁。
2. Spark Streaming 7\*24小时不停运作。

### 2.1.5 流行的实时流框架

1.Apache Storm：完全实时，来一笔数据就计算。

2.Apache Spark Streaming：Spark Streaming是Spark的一个子模块，不是完全实时，定时间（比如每2分钟，每5分钟）处理N笔数据。

3.IBM Stream。

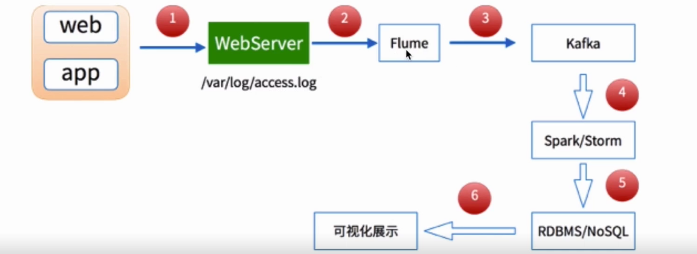
4.Kafka Streaming。

5.Yahoo! S4。

6.Apache Flink。

### 2.1.6 实时流架构

图1



1. 前端的请求到达web服务器（Web Server或Ngnix）后，会产生日志/var/log/access.log。
2. 通过分布式日志收集框架Flume收集web服务器的日志/var/log/access.log。
3. Flume收集日志后，丢到Kafka中。

（注意：其实Flume可以直接丢进Spark/Storm，但高峰期如果直接丢进Spark/Storm可以会导致Spark/Storm崩掉，所以Kakfa起到了缓冲的作用）

1. Spark/Storm处理完数据后，将结果写到关系型数据库（RDBMS）或NoSQL数据库中。
2. 通过可视化展示来观察统计数据是否有问题。

## 2.2 Spark提交任务

### 2.2.1 spark-submit

spark-submit需要指定jar包，适合生产。

### 2.2.2 spark-shell

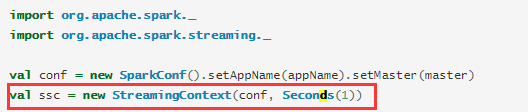
spark-shell可以直接写scala语句，适合开发和测试。

## 2.3 SparkContext

SparkContext是Spark上下文。

Seconds：是批处理间隔，单位是秒。

图1



## 2.4 StreamingContext

StreamingContext是Spark Streaming上下文，由SparkContext创建。

图1

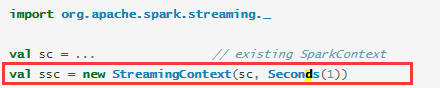
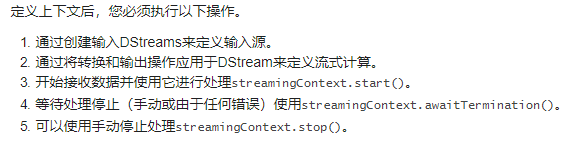


图2（创建StreamingContext后进行的操作）



## 2.5 DStream

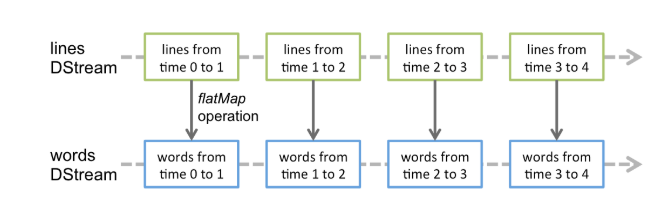
DStream（Discretized Stream，离散的数据流）表示连续的数据流。DStream由连续的RDD构成，每个RDD是特定时间间隔的数据集合。

（注意：DStream的任何操作都会转换成底层RDD上的操作）

图1



图2



### 2.5.1 DStream的变换（transform）

INPUT DStream可以进行转换，从而修改成不一样的INPUT DStream。

#### 2.5.1.1 map

map：将DStream（所有行）的RDD（一个batch interval的某几行）的每个元素传递给函数func来返回一个新的DStream。

（换句话说，map是一个元素转换成一个元素）

图1



#### 2.5.1.2 flatmap

flatmap：与map类型，但DStream的RDD的每个元素传递给函数func来返回0个或多个输出项。

（换句话说，flatmap是一个元素转换成多个元素）

图1



#### 2.5.1.3 updateStateByKey

updateStateByKey：调用该方法的DStream是以(K,V)出现的，让每个key维护一个state，并指定一个更新函数取更新该key的state。

1. 首先，要定义一个state，可以是任意的数据类型。
2. 其次，要定义一个state更新函数，这个函数表示如何根据“旧state和新值”来更新“新state”。

（注意：如果state更新函数返回none，则该key对应的state就会被删除）

#### 2.5.1.4 窗口操作

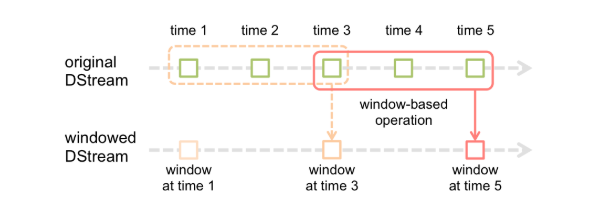
窗口操作设计2个参数：

1. 窗口长度：窗口的持续时间
2. 滑动间隔：执行窗口操作的间隔时间

（每隔xxx秒统计前xxx秒的结果，第一个是滑动间隔，第二个是窗口长度）

（注意：窗口长度和滑动间隔都必须是batch interval的整数倍，否则会报错）

图1



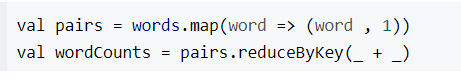
#### 2.5.1.5 transform

transform：将该DStream中的每个RDD都进行一个变换，形成一个新的RDD。

#### 2.5.1.6 reduceByKey

reduceByKey：调用该方法的DStream是以(K,V)出现的，K相同则合并，返回一个(K,V)形式的DStream。K为原来的K，V根据传入的func计算得到。

图1



### 2.5.2 DStream的输出操作

输出操作可以将DStream的数据推送到外部系统，比如：数据库或文件系统。

#### 2.5.2.1 foreachRDD

foreachRDD：对于该DStream中的每个RDD执行函数func的操作。

（注意：foreachRDD是对于该DStream中的每个RDD；foreachPartition是对于该RDD中的每个分区（因为RDD是batch interval时间间隔内的数据集，数据量可能会很大，所以RDD内部会有分区）；foreach是对于该分区内的每条记录）

图1



### 2.5.3 DStream和RDD的区别

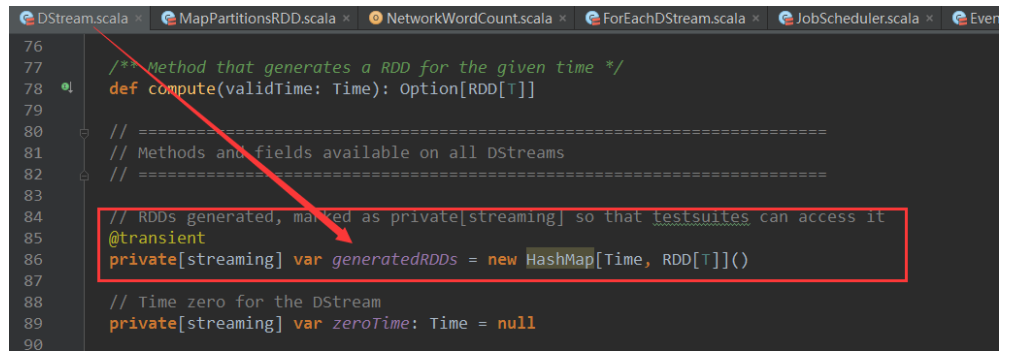
DStream由连续的RDD构成，每个RDD是特定时间间隔的数据集合。

图1



图2

（从源码角度，1个DStream中维护了一个HashMap，key是时间Time，Value是1个RDD）



### 2.5.4 DStream的分类

DStream主要分为3大类：

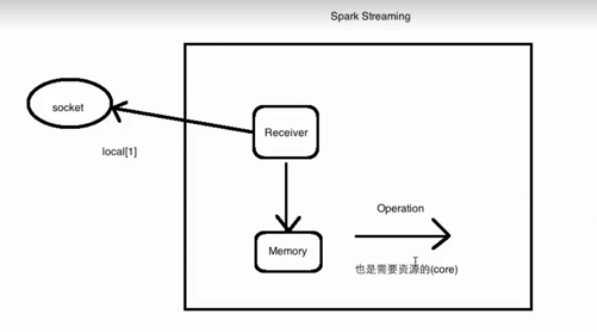
1. Input DStream
2. Transformed DStream
3. Output DStream

## 2.6 Receiver

每个DStream（除了文件流）都与Receiver相关联，Receiver会从源接收INPUT DStream，并将它存储在Spark的内存中，然后进行处理。

（注意：如果使用基于Receiver的INPUT DStream，不要使用local或local[1]来运行，因为这个单一的线程会被用来运行Receiver，没有剩余的线程来处理接收到的INPUT DStream，所以local[n]的n必须>Receiver的数量）

图1



### 2.6.1 两类内置的源

1.基本来源：StreamingContext API中直接提供的源。比如：文件系统和套接字连接。

2.高级来源：从额外的实用程序获取。比如：Kafka，Flume，Kinesis等。

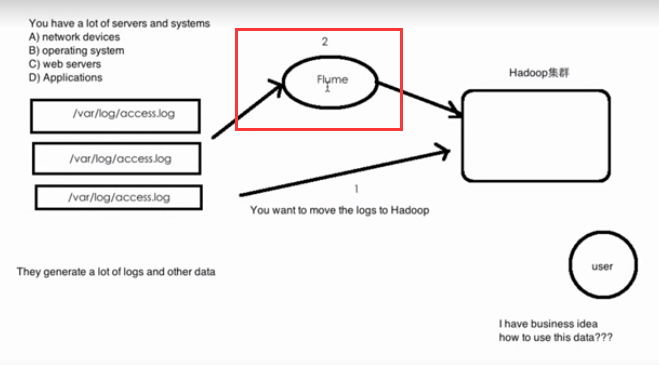
# 3 Flume

Flume是一个分布式日志收集框架。

## 3.1 为什么需要Flume

我们需要从日志中收集有价值的信息，因为日志数据量很大，所以需要将日志文件从web服务器收集到Hadoop集群中进行计算，这个收集过程Flume可以完成。

图1



## 3.2 Flume架构

每一个agent的组成包括：

Source：数据来源

Channel：数据缓存池，当Channel满了才将数据写到Sink中。

Sink：数据目的地

图1（单个agent）

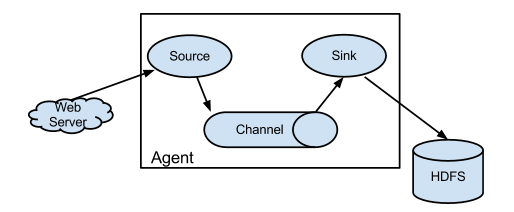


图2（多个agent）

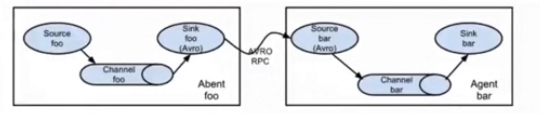


图3（多个agent）

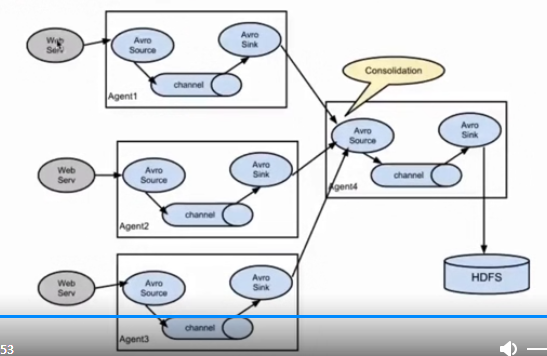


图4（多个Channel，多个Sink）

