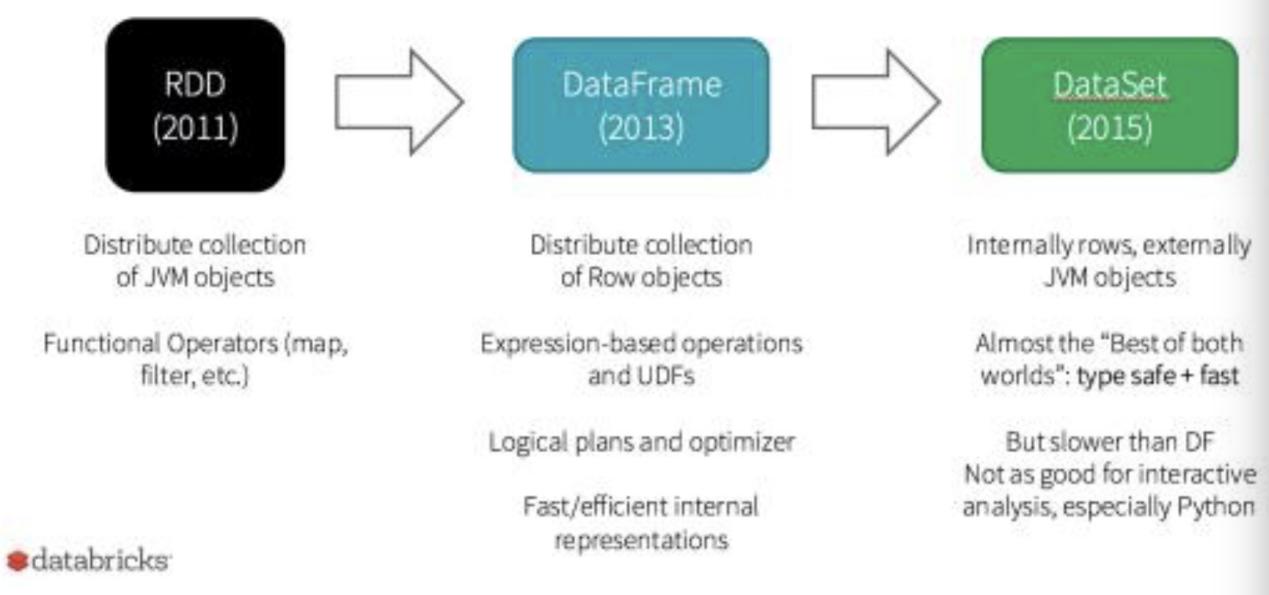
Spark编程模型

Spark是用来实现快速而通用的集群计算的平台，其扩展了MR计算模型而且高效的支持更多的计算模式，包括交互式查询和流处理。Spark使用统一的编程抽象RDD(弹性分布式数据集)，这样全新的模型可以使用户直接控制数据的共享，使得用户可以指定将数据存储到磁盘/内存，其更适合于迭代计算比较多的机器学习和数据挖掘运算。Spark 2.0以后提供三种数据抽象API(RDD,DataFrame和Dataset)，如下图：



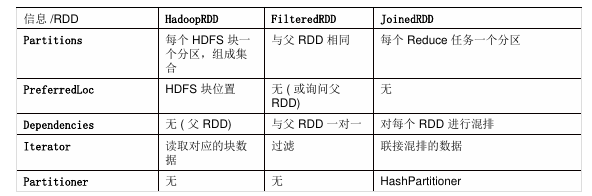
DataFrame和Dataset是基于RDD提供的，而且DataFrame可以当作通用对象Dataset[Row]集合的别名，三种之间可以进行无缝切换从而实现跨函数库的数据能力。在Spark 1.6版本中Dstream是Spark Streaming的编程模型，而Spark 2.0将流计算也统一到DataFrame中去，提出Structured Streaming编程模型，将流数据源映射为类似于数据库中的表。

# 弹性分布式数据集（RDD）

RDD(弹性分布式数据集)，是Spark最核心的概念，其元数据具有以下5部分数据:

* 分区列表(Partition)，数据集的基本组成单位，每个分片都会被一个计算任务处理，并决定并行计算的粒度。用户可以在创建RDD时指定分片个数，如果没有指定则会采用默认值，程序所分配到CPU Core数目
* 每个分区都具有计算函数，Spark的RDD的计算函数是以分片为基本单位， 每个RDD都会实现compute函数，对具体的分片进行计算，从而实现分布式并行计算
* RDD依赖列表，每次转换都会生成新的RDD，其操作会形成类似流水线的前后依赖关系。其依赖关系有两种：窄依赖和宽依赖，窄依赖中每个parent RDD的partition最多被child RDD的一个partition使用；宽依赖是多个child RDD的Partition会被依赖于同一个parent RDD的partition，典型操作有group by及sort by key等，即shuffle操作。
* 数据类型<k,v>的RDD分区器，其决定了如何分区及分区的数量
* 每个分区都有优先位置列表，根据数据本地行快速访问数据的偏好位置，将计算分发到数据所在节点

元数据的示例：



以HadoopRDD为例子，其中分区是一个逻辑概念，在RDD操作中用户可以使用Partitions方法获取其划分的分区数，下面是从HDFS文件中创建

*Scala> val part = sc.textFile(“/data/helloworld”)*

*part: RDD[String]=/data/helloworld MapPartitionsRDD[3] at textFile*

*Scala>part.partitions.size*

*Res1: Int = 2*

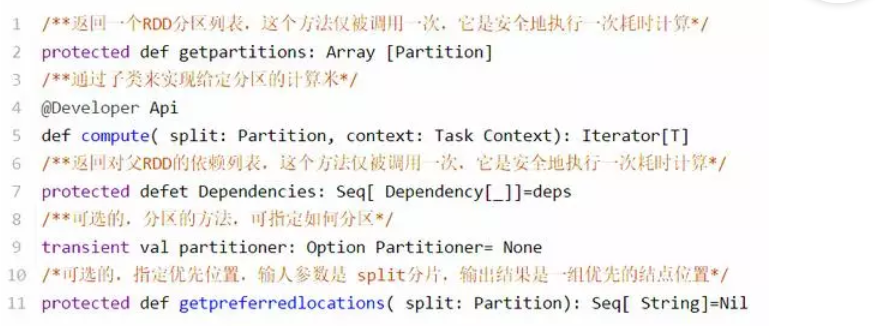
*Scala> val part = sc.textFile(“/data/helloworld”,6) //显示设置分区数*

*part: RDD[String]=/data/helloworld MapPartitionsRDD[3] at textFile*

*Scala>part.partitions.size*

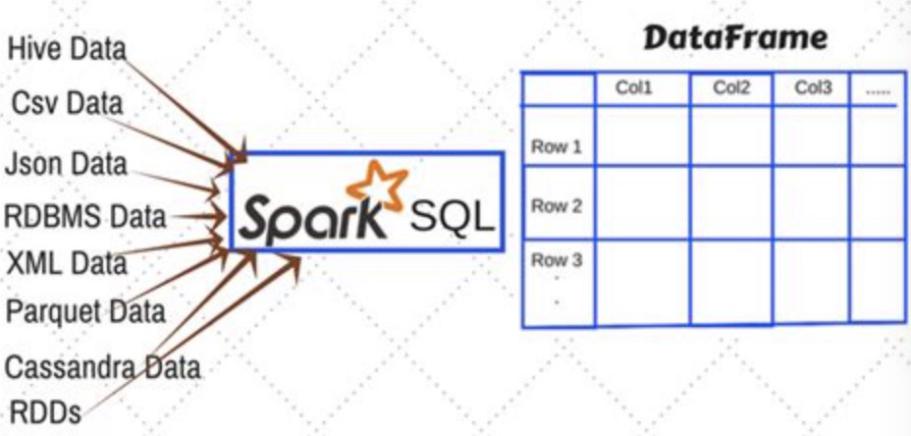
*Res1: Int = 6*

HadoopRDD的核心代码如下：



# DataFrame

DataFrame是数据的不可变分布式集合，但是与RDD不同的是数据都被组织到有名字的列中，就像关系性数据库中的表一样，如下图：



DataFrame的概念如下：

*DataFrame is a distributed collection of data organized into named columns. It is conceptually equivalent to a table in a relational database or a data frame in R/Python, but with richer optimizations under the hood. DataFrames can be constructed from a wide array of sources such as: structured data files, tables in Hive, external databases, or existing RDDs.*

* 分布式数据集
* 类似于关系型数据库中的table
* 支持丰富的操作函数，类似于rdd算子，可以在上面进行sql查询
* 支持丰富的创建方式，包括已有RDD,结构化数据文件，JSON数据集及外部数据库

下面是创建Spark DataFrame示例：

*scala> val df = spark.read.json("/data/resources/people.json")*

*scala> df.printSchema*

*root*

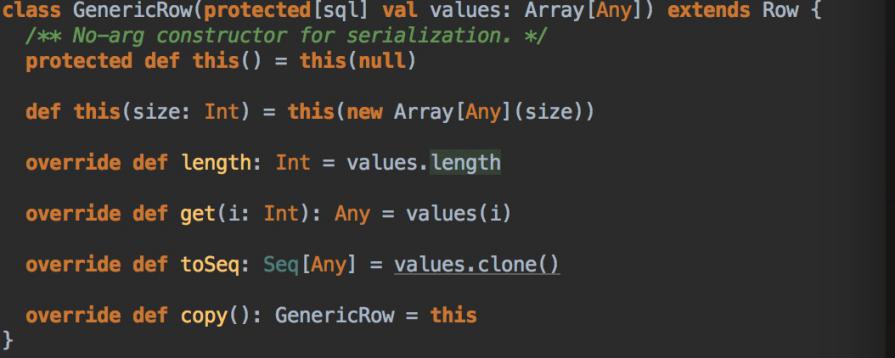
*|-- age: long (nullable = true)*

*|-- name: string (nullable = true)*

*Scala>df.select("name").show()*

DataFrame通过schema定义列及数据类型，其定义：*type DataFrame = Dataset[Row]*

元数据在DataSet中定义，其核心是Row，其类结构如下：



其使用Array来保存数据，列用StructType来描述，在下面介绍。

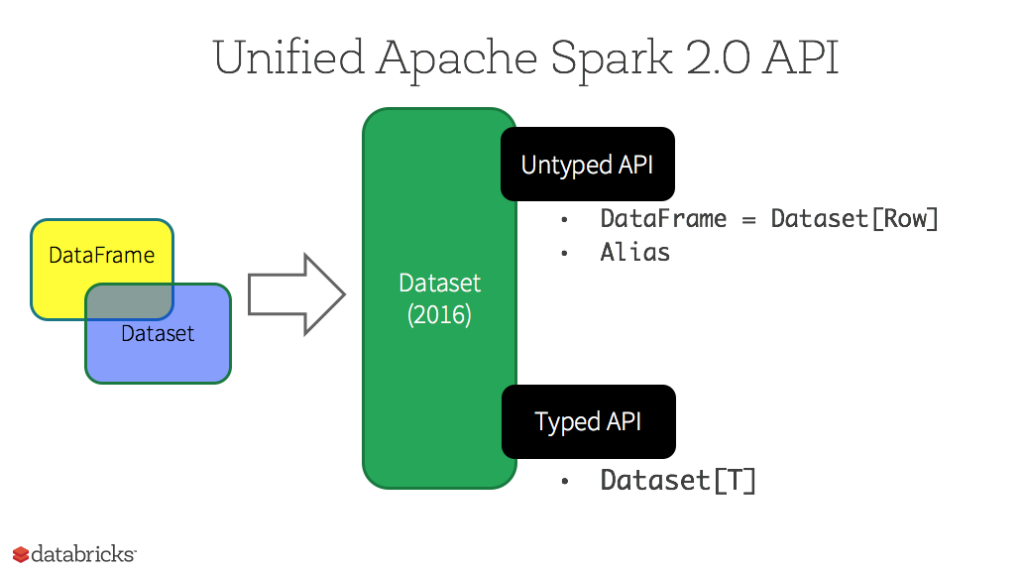
http://litaotao.github.io/spark-dataframe-introduction?s=inner

# DataSet

DataSet是从Spark 1.6开始引入的抽象，在Spark 2.0中成为稳定的api，其官方定义：

*A Dataset is a strongly typed collection of domain-specific objects that can be transformed  
in parallel using functional or relational operations. Each Dataset also has an untyped view  
called a DataFrame, which is a Dataset of Row.*

Dataset是特定域对象中的强类型集合，可以使用函数或者相关操作并进行转换等操作，每个Dataset都有一个成为DataFrame的非类型化视图，这个视图是行的数据集。



下图是RDD/DataFrame/DataSet在类型中的抽象示例

|  |  |  |
| --- | --- | --- |
| RDD | DataFrame | DataSet |
|  |  |  |

与RDD不同的是DataSet是任何对象的集合，而且DataSet总是强类型，其可以使用模式进行优化。DataFrame是特殊的DataSet，其比RDD多表头信息（Schema）。强类型表示JVM对象是case scala class或者java class来, ROW对象是泛型Object。

1. 使用示例如下

*scala> case class People(age : Long, name: String)*

*defined class People*

*scala> val df = spark.read.json("/data/resources/people.json").as[People]*

*scala> peopleDS.printSchema*

*root*

*|-- age: long (nullable = true)*

*|-- name: string (nullable = true)*

对DataSet可以支持的操作select/filter/where/groupby/join等操作，目前最常用的最发是转化成数据表来操作，如下所示：

*scala> peopleDS.createOrReplaceTempView("peoples")*

*scala> sql("select \* from peoples").show(2)*

*+---+-------+*

*|age| name|*

*+---+-------+*

*| 45|Michael|*

*| 30| Andy|*

*+---+-------+*

*only showing top 2 rows*

1. 源码分析

DataSet类定义如下：

*private[sql] object Dataset {  
 def apply[T: Encoder] (sparkSession: SparkSession, logicalPlan: LogicalPlan):*

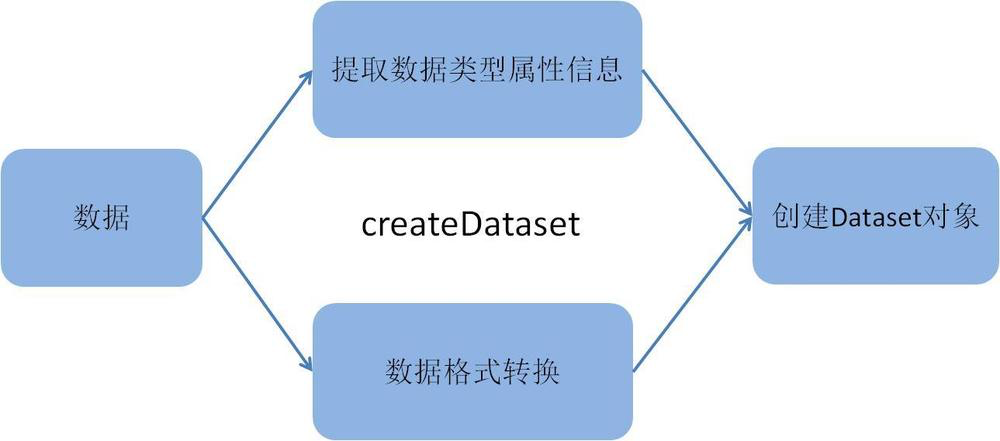
其包括三个参数：

* SparkSession，环境等信息
* LogicalPlan，包含数据和执行逻辑信息
* Encoder，数据结构编码信息，包括序列化，schema和数据type

在SparkSession核心方法createDataSet，其定义如下：

*def createDataset[T : Encoder](data: Seq[T]): Dataset[T] = {  
 val enc = encoderFor[T]  
 val attributes = enc.schema.toAttributes  
 val encoded = data.map(d => enc.toRow(d).copy())  
 val plan = new LocalRelation(attributes, encoded)  
 Dataset[T](self, plan)  
}*

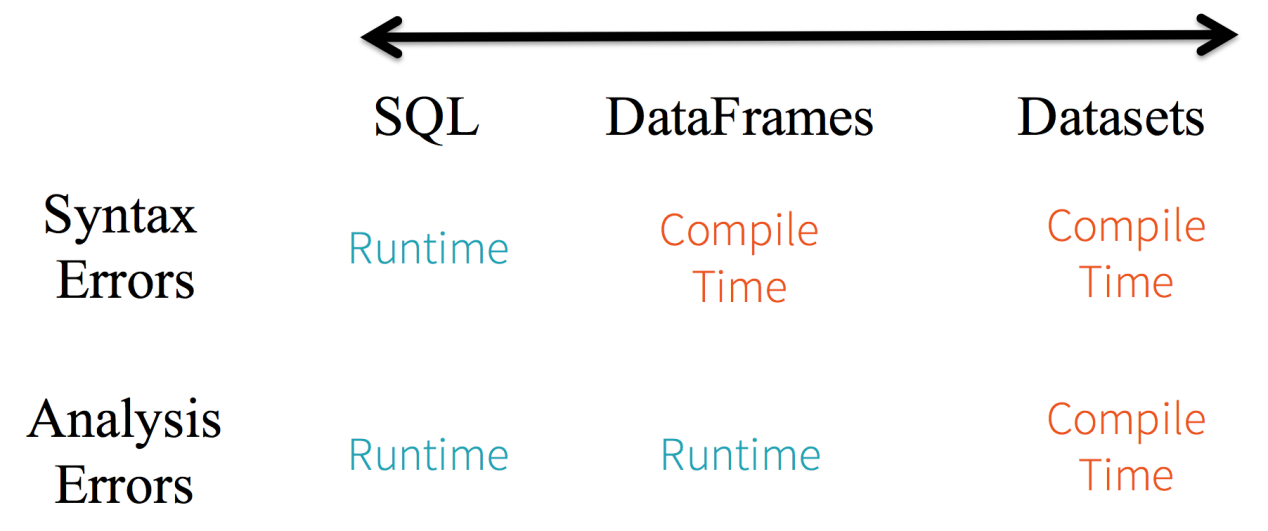
其创建步骤如下图：



函数的实现：

* 将数据传进来后，提取数据类型，通过类型提取schema，并激昂数据转化成spark sql中的ROW
* 将数据和数据属性信息封装成relation对象
* 用Dataset的伴生对象创建Dataset实例，用apply方法来生成对象

在Spark 2.0中使用DataSet和DataFrame统一api会为spark开发者带来很多好处，可以把静态类型和运行时安全看成下图：



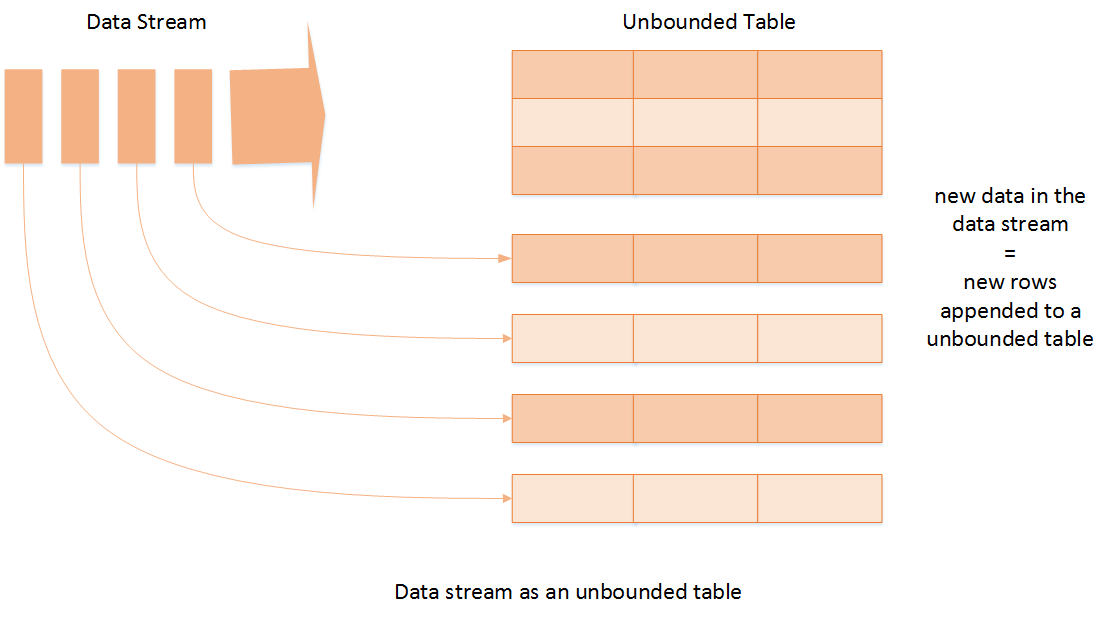
直接使用spark sql 运行时才会发现语法错误，但是使用DataFrame和Dataset在编译时可以捕捉错误，最严格的是Dataset，其用lambda函数和jvm类型表示，所以不匹配的类型参数会在编译时发现，其分析错误也会在编译时发现，可以节省开发者的时间和代价。

Spark可以作为编译器，理解Dataset类型的jvm对象，其使用encoder把特定类型的jvm对象映射成tungsten内部内存表示，提高执行效率。

# Structured Streaming(DStream)

在Spark 1.6中，DStream是spark streaming的编程模型，spark 2.0以后将流计算统一到了DataFrame中，提出Structured Streaming编程模型，将数据源映射为类似于关系数据库中的表，然后将计算结果映射为另外一张表，完全以结构化的方式去操作流式数据。

Structured Streaming最核心的思想就是将实时到达的数据看作是不断追加的“unbound table”，如下图所示：



针对实时数据的查询将会产生一个结果集，每个一段时间（例如1秒），新到来的数据都将添加到unbounded table中。Spark会对新添加的方法进行实时计算并更新结果集

1. 使用示例，创建sparkSession后，读取nc打开端口的数据

*scala> val lines = spark.readStream.*

*| format("socket").*

*| option("host","localhost").*

*| option("port",9999).*

*| load()*

*lines: org.apache.spark.sql.DataFrame = [value: string]*

*scala> val words = lines.as[String].flatMap(\_.split(" "))*

*words: org.apache.spark.sql.Dataset[String] = [value: string]*

*scala> val wordCounts = words.groupBy("value").count()*

*wordCounts: org.apache.spark.sql.DataFrame = [value: string, count: bigint]*

执行流查询如下：

*scala> val query = wordCounts.writeStream.*

*| outputMode("complete").*

*| format("console").*

*| start()*

*query: org.apache.spark.sql.streaming.StreamingQuery = Streaming Query - query-0 [state = ACTIVE]*

*scala> query.awaitTermination()*

*-------------------------------------------*

*Batch: 0*

*----------- --------------------------------*

*+-----+-----+*

*|value|count|*

*+-----+-----+*

*|hello| 1|*

*|spark| 1|*

1. 源码分析

wordCounts.writeStream返回一个DataStreamWriter实例，该实例定义了将实时流查询产生结果输出到外部存储的接口，最后调用awaitTermination等待查询结果。对于DataStreamWriter其执行的核心是TriggerExecutor，周期行执行任务，核心代码如下：

*case class ProcessingTimeExecutor(processingTime: ProcessingTime,*

*clock: Clock = new SystemClock()) extends TriggerExecutor with Logging {*

*private val intervalMs = processingTime.intervalMs*

*require(intervalMs >= 0)*

*override def execute(triggerHandler: () => Boolean): Unit = {*

*while (true) {*

*val triggerTimeMs = clock.getTimeMillis //执行周期*

*val nextTriggerTimeMs = nextBatchTime(triggerTimeMs)*

*val terminated = !triggerHandler()*

*if (intervalMs > 0) {*

*val batchElapsedTimeMs = clock.getTimeMillis - triggerTimeMs*

*if (batchElapsedTimeMs > intervalMs) { //执行batch操作*

*notifyBatchFallingBehind(batchElapsedTimeMs)*

*}*

*clock.waitTillTime(nextTriggerTimeMs)*

*} ......*

在notfiyBatchFallingBehind中调用StreamExecution执行batch处理方法runBatch，具体执行逻辑是将数据源数据基于offset转化成DataFrame，然后获取结果。

https://www.jianshu.com/p/f6acadaa35a2

<http://dblab.xmu.edu.cn/blog/1167/>

http://www.jianshu.com/p/77811ae29fdd

https://www.iteblog.com/archives/1674.html#Dataset

https://edmondfrank.github.io/blog/2017/11/10/shen-ru-li-jie-spark-bian-cheng-mo-xing/

http://dblab.xmu.edu.cn/blog/1167/

http://ytluck.github.io/big-data/my-bigdata-post-33.html

https://www.iteblog.com/archives/1624.html

http://www.cnblogs.com/zhouyf/p/5511636.html

http://www.ituring.com.cn/article/506988

https://www.infoq.cn/article/three-apache-spark-apis-rdds-dataframes-and-datasets