Spark 2.0介绍：从RDD API迁移到DataSet API

《Spark 2.0技术预览：更容易、更快速、更智能》文章中简单地介绍了Spark 2.0带来的新技术等。Spark 2.0是Apache Spark的下一个主要版本。此版本在架构抽象、API以及平台的类库方面带来了很大的变化，为该框架明年的发展方向奠定了方向，所以了解Spark 2.0的一些特性对我们能够使用它有着非常重要的作用。本博客将对Spark 2.0进行一序列的介绍（参见Spark 2.0分类），欢迎关注。

RDD迁移到DataSet

DataSet API将RDD和DataFrame两者的优点整合起来，DataSet中的许多API模仿了RDD的API，虽然两者的实现很不一样。所以大多数调用RDD API编写的程序可以很容易地迁移到DataSet API中，下面我将简单地展示几个片段来说明如何将RDD编写的程序迁移到DataSet。

1、加载文件

RDD

```
val rdd = sparkContext.textFile("src/main/resources/data.txt")
```

Dataset

```
val ds = sparkSession.read.text("src/main/resources/data.txt")
```

2、计算总数

RDD

```
rdd.count()
```

Dataset

```
ds.count()
```

3、WordCount实例

RDD

```
val wordsRDD = rdd.flatMap(value => value.split("₩₩s+"))
val wordsPair = wordsRDD.map(word => (word,1))
val wordCount = wordsPair.reduceByKey(_+_)
```

Dataset

```
import sparkSession.implicits._
val wordsDs = ds.flatMap(value => value.split("₩₩s+"))
val wordsPairDs = wordsDs.groupByKey(value => value)
val wordCountDs = wordsPairDs.count()
```

4、缓存(Caching)

RDD

```
rdd.cache()
```

Dataset

```
ds.cache()
```

5、过滤(Filter)

RDD

```
val filteredRDD = wordsRDD.filter(value => value =="hello")
```

Dataset

```
val filteredDS = wordsDs.filter(value => value =="hello")
```

6、Map Partitions

RDD

```
val mapPartitionsRDD = rdd.mapPartitions(iterator =>
      List(iterator.count(value => true)).iterator)
```

Dataset

```
val mapPartitionsDs = ds.mapPartitions(iterator =>
      List(iterator.count(value => true)).iterator)
```

7、reduceByKey

RDD

```
val reduceCountByRDD = wordsPair.reduceByKey(_+_)
```

Dataset

```
val reduceCountByDs = wordsPairDs.mapGroups((key,values) =>(key,values.length))
```

8、RDD和DataSet互相转换

RDD

```
val dsToRDD = ds.rdd
```

Dataset
将RDD转换成DataFrame需要做一些工作，比如需要指定特定的模式。下面展示如何将RDD[String]转换成DataFrame[String]：

```
val rddStringToRowRDD = rdd.map(value => Row(value))
val dfschema = StructType(Array(StructField("value",StringType)))
val rddToDF = sparkSession.createDataFrame(rddStringToRowRDD,dfschema)
val rDDToDataSet = rddToDF.as[String]
```

9、基于Double的操作

RDD

```
val doubleRDD = sparkContext.makeRDD(List(1.0,5.0,8.9,9.0))
val rddSum =doubleRDD.sum()
val rddMean = doubleRDD.mean()
```

Dataset

```
val rowRDD = doubleRDD.map(value => Row.fromSeq(List(value)))
val schema = StructType(Array(StructField("value",DoubleType)))
val doubleDS = sparkSession.createDataFrame(rowRDD,schema)

import org.apache.spark.sql.functions._
doubleDS.agg(sum("value"))
doubleDS.agg(mean("value"))
```

10、Reduce API

RDD

```
val rddReduce = doubleRDD.reduce((a,b) => a +b)
```

```
Dataset

val dsReduce = doubleDS.reduce((row1,row2) =>Row(row1.getDouble(0) + row2.getDouble(0)))
```

上面的代码片段展示了如何将你之前使用RDD API编写的程序转换成DataSet API编写的程序。虽然这里并没有覆盖所有的RDD API，但是通过上面的介绍，你肯定可以将其他RDD API编写的程序转换成DataSet API编写的程序。

完整代码

```
package com.iteblog.spark

import org.apache.spark.sql.types.{DoubleType, StringType, StructField, StructType}
import org.apache.spark.sql.{Row, SparkSession}

/**
  * RDD API to Dataset API
  *
  */
object RDDToDataSet {

  def main(args: Array[String]) {

    val sparkSession = SparkSession.builder.
      master("local")
      .appName("example")
      .getOrCreate()

    val sparkContext = sparkSession.sparkContext

    //read data from text file
    val rdd = sparkContext.textFile("src/main/resources/data.txt")
    val ds = sparkSession.read.text("src/main/resources/data.txt")

    // do count
    println("count ")
    println(rdd.count())
    println(ds.count())
```

```
// wordcount
println(" wordcount ")

val wordsRDD = rdd.flatMap(value => value.split("₩₩s+"))
val wordsPair = wordsRDD.map(word => (word,1))
val wordCount = wordsPair.reduceByKey(_+_)
println(wordCount.collect.toList)

import sparkSession.implicits._
val wordsDs = ds.flatMap(value => value.split("₩₩s+"))
val wordsPairDs = wordsDs.groupByKey(value => value)
val wordCountDs = wordsPairDs.count
wordCountDs.show()

//cache
rdd.cache()
ds.cache()

//filter

val filteredRDD = wordsRDD.filter(value => value =="hello")
println(filteredRDD.collect().toList)

val filteredDS = wordsDs.filter(value => value =="hello")
filteredDS.show()


//map partitions

val mapPartitionsRDD = rdd.mapPartitions(iterator =>
          List(iterator.count(value => true)).iterator)
println(s" the count each partition is ${mapPartitionsRDD.collect().toList}")

val mapPartitionsDs = ds.mapPartitions(iterator =>
          List(iterator.count(value => true)).iterator)
mapPartitionsDs.show()

//converting to each other
val dsToRDD = ds.rdd
println(dsToRDD.collect())

val rddStringToRowRDD = rdd.map(value => Row(value))
val dfschema = StructType(Array(StructField("value",StringType)))
val rddToDF = sparkSession.createDataFrame(rddStringToRowRDD,dfschema)
val rDDToDataSet = rddToDF.as[String]
rDDToDataSet.show()
```

```
    // double based operation

    val doubleRDD = sparkContext.makeRDD(List(1.0,5.0,8.9,9.0))
    val rddSum =doubleRDD.sum()
    val rddMean = doubleRDD.mean()

    println(s"sum is $rddSum")
    println(s"mean is $rddMean")

    val rowRDD = doubleRDD.map(value => Row.fromSeq(List(value)))
    val schema = StructType(Array(StructField("value",DoubleType)))
    val doubleDS = sparkSession.createDataFrame(rowRDD,schema)

    import org.apache.spark.sql.functions._
    doubleDS.agg(sum("value")).show()
    doubleDS.agg(mean("value")).show()

    //reduceByKey API
    val reduceCountByRDD = wordsPair.reduceByKey(_+_)
    val reduceCountByDs = wordsPairDs.mapGroups((key,values) =>(key,values.length))

    println(reduceCountByRDD.collect().toList)
    println(reduceCountByDs.collect().toList)

    //reduce function
    val rddReduce = doubleRDD.reduce((a,b) => a +b)
    val dsReduce = doubleDS.reduce((row1,row2) =>
              Row(row1.getDouble(0) + row2.getDouble(0)))

    println("rdd reduce is " +rddReduce +" dataset reduce "+dsReduce)

  }

}
```