08: Spark on Zeppelin – convert DataFrames to RDD[Row] and RDD[Row] to DataFrame

Posted on September 8, 2018

Pre-requisite: Docker is installed on your machine for Mac OS X (E.g. $ brew cask install docker) or Windows 10. Docker interview Q&As. This extends setting up Apache Zeppelin Notebook.

Important: It is not a best practice to mutate values or to use RDD directly as opposed to using Dataframes.

300+ Java Interview FAQs

300+ Java FAQs 🔥

16+ Java Key Areas Q&As

150+ Java Architect FAQs

80+ Java Code Quality Q&As

150+ Java Coding Q&As

300+ Big Data Interview FAQs

300+ Big Data FAQs 🔥

Tutorials - Big Data

TUT - ⏯ Starting Big Data

TUT - Starting Spark & Scala

Use of groupBy operation on RDD is also discouraged as it causes wide shuffling. There are alternative solutions like using "withColumn" on Dataframe to add a new column, UDF functions, window functions, explode function, etc to achieve the desired results more efficiently. There will be edge cases where use of the RDDs will give you more flexibility to achieve the desired outcome. The examples shown below is for the illustration purpose only.

Step 1: Pull this from the docker hub, and build the image with the following command.

```
$ docker pull apache/zeppelin:0.7.3
```

You can verify the image with the "docker images" command.

Step 2: Run the container with the above image.

```
$ docker run --rm -it -p 8080:8080 apache/zeppelin
```

Step 3: Open Zeppelin notebook via a web browser "http:localhost:8080". Create a note book with "spark" as a default interpreter.

1. Dataframe to RDD and back to Dataframe

```
%spark


import org.apache.spark.sql.types._
import org.apache.spark.sql.Row
import org.apache.spark.rdd.RDD

val schema = StructType(
```

TUT - Starting with Python
TUT - Kafka
TUT - Pig
TUT - Apache Storm
TUT - Spark Scala on Zeppelin
TUT - Cloudera
TUT - Cloudera on Docker
TUT - File Formats
TUT - Spark on Docker
TUT - Flume
TUT - Hadoop (HDFS)
TUT - HBase (NoSQL)
TUT - Hive (SQL)
TUT - Hadoop & Spark
TUT - MapReduce
TUT - Spark and Scala
TUT - Spark & Java
TUT - PySpark on Databricks
TUT - Zookeeper

```
  List(
    StructField("id", IntegerType, true),
    StructField("name", StringType, true),
    StructField("location", StringType, true),
    StructField("salary", DoubleType, true)
  )
)

val employees = Seq(
    Row(1, "John", "USA", 50000.0),
    Row(2, "Peter", "AU",60000.0),
    Row(3, "Sam", "AU", 60000.0),
    Row(4, "Susan", "USA", 50000.0),
    Row(5, "David", "USA", 70000.0),
    Row(6, "Elliot", "AU", 50000.0)
)

val employee_df = spark.createDataFrame(
  spark.sparkContext.parallelize(employees),
  schema
)

employee_df.show()

val rdd  = employee_df.rdd
                    .groupBy(row => row(2)) //gr
                    .flatMap(x => {         //fl
                        //x is a Tuple, x._1 is
                        // filter salary 60k or
                        for (row <- x._2 if row.
                    })

val df2 = sqlContext.createDataFrame(rdd, schema)
df2.show()

```

Output:

```
import org.apache.spark.sql.types._
import org.apache.spark.sql.Row
import org.apache.spark.rdd.RDD
schema: org.apache.spark.sql.types.StructType = S
employees: Seq[org.apache.spark.sql.Row] = List([1
employee_df: org.apache.spark.sql.DataFrame = [id
+---+------+--------+-------+
| id|  name|location| salary|
+---+------+--------+-------+
|  1|  John|     USA|50000.0|
|  2| Peter|      AU|60000.0|
|  3|   Sam|      AU|60000.0|
|  4| Susan|     USA|50000.0|
|  5| David|     USA|70000.0|
```

```
|  6|Elliot|      AU|50000.0|
+---+------+--------+-------+

rdd: org.apache.spark.rdd.RDD[org.apache.spark.sq
df2: org.apache.spark.sql.DataFrame = [id: int, na
+---+-----+--------+-------+
| id| name|location| salary|
+---+-----+--------+-------+
|  2|Peter|      AU|60000.0|
|  3|  Sam|      AU|60000.0|
|  5|David|     USA|70000.0|
+---+-----+--------+-------+
```

After groupBy you will get

```
rdd: org.apache.spark.rdd.RDD[(Any, Iterable[org.ap
```

If you want to output the RDD with grouped values, you can

```
rdd.collect().foreach(println)
```

which outputs following grouped by locations AU and USA :

```
(AU,CompactBuffer([2,Peter,AU,60000.0], [3,Sam,AU,6
(USA,CompactBuffer([1,John,USA,50000.0], [4,Susan,U

```

Before performing flatMap, you can transform the values via a number of mapValues function that transforms only the values.

2. Dataframe to RDD and back to Dataframe

The following example adds new Rows with bonus (i.e. 10% of the salary) if the salary is <= 50K.

```
%spark


import org.apache.spark.sql.types._
import org.apache.spark.sql.Row
import org.apache.spark.rdd.RDD
import scala.collection._

val schema = StructType(
  List(
    StructField("id", IntegerType, true),
    StructField("name", StringType, true),
    StructField("location", StringType, true),
    StructField("salary", DoubleType, true)
  )
)

val employees = Seq(
    Row(1, "John", "USA", 50000.0),
    Row(2, "Peter", "AU",60000.0),
    Row(3, "Sam", "AU", 60000.0),
    Row(4, "Susan", "USA", 50000.0),
    Row(5, "David", "USA", 70000.0),
    Row(6, "Elliot", "AU", 50000.0)
)

val employee_df = spark.createDataFrame(
  spark.sparkContext.parallelize(employees),
  schema
)

employee_df.show()

val rdd  = employee_df.rdd
                      .groupBy(row => row(2)) //gr
                      .flatMap(x => {         //fl
                          var newList = mutable.Mu

                          for (row <- x._2){
                              newList += row
                              if(row.getDouble(3) 
                                   //Give 10% bonu
                                   val newRow = Ro
                                   newList += newR
                              }
                          }

                          newList
                      })

val df2 = sqlContext.createDataFrame(rdd, schema)
df2.show()

```

Output:

```
import org.apache.spark.sql.types._
import org.apache.spark.sql.Row
import org.apache.spark.rdd.RDD
import scala.collection._
schema: org.apache.spark.sql.types.StructType = St
employees: Seq[org.apache.spark.sql.Row] = List([1
employee_df: org.apache.spark.sql.DataFrame = [id
+---+------+--------+-------+
| id|  name|location| salary|
+---+------+--------+-------+
|  1|  John|     USA|50000.0|
|  2| Peter|      AU|60000.0|
|  3|   Sam|      AU|60000.0|
|  4| Susan|     USA|50000.0|
|  5| David|     USA|70000.0|
|  6|Elliot|      AU|50000.0|
+---+------+--------+-------+
rdd: org.apache.spark.rdd.RDD[org.apache.spark.sq
df2: org.apache.spark.sql.DataFrame = [id: int, na
+---+------+--------+-------+
| id|  name|location| salary|
+---+------+--------+-------+
|  2| Peter|      AU|60000.0|
|  3|   Sam|      AU|60000.0|
|  6|Elliot|      AU|50000.0|
|  6|Elliot|      AU| 5000.0|
|  1|  John|     USA|50000.0|
|  1|  John|     USA| 5000.0|
|  4| Susan|     USA|50000.0|
|  4| Susan|     USA| 5000.0|
|  5| David|     USA|70000.0|
+---+------+--------+-------+

```

‹ 07: Spark on Zeppelin – window functions in Scala

09: Spark on Zeppelin – convert DataFrames to RDD and RDD to DataFrame ›

Disclaimer