

Java-Success.com

Prepare to fast-track, choose & go places with 800+ Java & Big Data Q&As with lots of code & diagrams.

[Home](#) [Why? ▾](#) [300+ Java FAQs ▾](#) [300+ Big Data FAQs ▾](#) [Courses ▾](#)[👤 Membership ▾](#) [Your Career ▾](#)[Home](#) › [bigdata-success.com](#) › [Tutorials - Big Data](#) › [TUT - Spark Scala on Zeppelin](#) ›

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

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.

```
1 $ docker pull apache/zeppelin:0.7.3
2
```

You can verify the image with the “docker images” command.

Step 2: Run the container with the above image.

```
1 $ docker run --rm -it -p 8080:8080 apache/zeppelin
2
```

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

```
1 %spark
2
3
4 import org.apache.spark.sql.types._
5 import org.apache.spark.sql.Row
6 import org.apache.spark.rdd.RDD
7
8 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

800+ Java Interview Q&As

300+ Core Java Q&As



300+ Enterprise Java Q&As



150+ Java Frameworks Q&As



120+ Companion Tech Q&As



Tutorials - Enterprise Java



```

9   List(
10     StructField("id", IntegerType, true),
11     StructField("name", StringType, true),
12     StructField("location", StringType, true),
13     StructField("salary", DoubleType, true)
14   )
15 )
16
17 val employees = Seq(
18   Row(1, "John", "USA", 50000.0),
19   Row(2, "Peter", "AU", 60000.0),
20   Row(3, "Sam", "AU", 60000.0),
21   Row(4, "Susan", "USA", 50000.0),
22   Row(5, "David", "USA", 70000.0),
23   Row(6, "Elliot", "AU", 50000.0)
24 )
25
26 val employee_df = spark.createDataFrame(
27   spark.sparkContext.parallelize(employees),
28   schema
29 )
30
31 employee_df.show()
32
33 val rdd = employee_df.rdd
34   .groupBy(row => row(2)) //group by location
35   .flatMap(x => {          //flatMap
36     //x is a Tuple, x._1 is location
37     // filter salary 60k or more
38     for (row <- x._2 if row._4 >= 60000.0)
39       yield row
40   })
41 val df2 = sqlContext.createDataFrame(rdd, schema)
42 df2.show()
43

```

Output:

```

1 import org.apache.spark.sql.types._
2 import org.apache.spark.sql.Row
3 import org.apache.spark.rdd.RDD
4 schema: org.apache.spark.sql.types.StructType = StructType(StructField(id,IntegerType,true),StructField(name,StringType,true),StructField(location,StringType,true),StructField(salary,DoubleType,true))
5 employees: Seq[org.apache.spark.sql.Row] = List([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])
6 employee_df: org.apache.spark.sql.DataFrame = [id,name,location,salary]
7 +---+-----+-----+-----+
8 | id| name|location| salary|
9 +---+-----+-----+-----+
10 | 1| John|    USA|50000.0|
11 | 2| Peter|    AU|60000.0|
12 | 3| Sam|    AU|60000.0|
13 | 4| Susan|   USA|50000.0|
14 | 5| David|   USA|70000.0|

```

```

15 | 6|Elliot|      AU|50000.0|
16 | +---+-----+-----+-----+
17 |
18 | rdd: org.apache.spark.rdd.RDD[org.apache.spark.sql.Row] = ...
19 | df2: org.apache.spark.sql.DataFrame = [id: int, name: string, location: string, salary: double]
20 | +---+-----+-----+-----+
21 | id| name|location| salary|
22 | +---+-----+-----+-----+
23 | 2| Peter|      AU|60000.0|
24 | 3|  Sam|      AU|60000.0|
25 | 5|David|      USA|70000.0|
26 | +---+-----+-----+-----+
27 |

```

After **groupBy** you will get

```

1 | rdd: org.apache.spark.rdd.RDD[(Any, Iterable[org.apache.spark.sql.Row])] = ...

```

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

```

1 | rdd.collect().foreach(println)

```

which outputs following grouped by locations AU and USA :

```

1 | (AU, CompactBuffer([2, Peter, AU, 60000.0], [3, Sam, AU, 60000.0]))
2 | (USA, CompactBuffer([1, John, USA, 50000.0], [4, Susan, USA, 70000.0]))
3 |

```

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.

```
1 %spark
2
3
4 import org.apache.spark.sql.types._
5 import org.apache.spark.sql.Row
6 import org.apache.spark.rdd.RDD
7 import scala.collection._
8
9 val schema = StructType(
10   List(
11     StructField("id", IntegerType, true),
12     StructField("name", StringType, true),
13     StructField("location", StringType, true),
14     StructField("salary", DoubleType, true)
15   )
16 )
17
18 val employees = Seq(
19   Row(1, "John", "USA", 50000.0),
20   Row(2, "Peter", "AU", 60000.0),
21   Row(3, "Sam", "AU", 60000.0),
22   Row(4, "Susan", "USA", 50000.0),
23   Row(5, "David", "USA", 70000.0),
24   Row(6, "Elliot", "AU", 50000.0)
25 )
26
27 val employee_df = spark.createDataFrame(
28   spark.sparkContext.parallelize(employees),
29   schema
30 )
31
32 employee_df.show()
33
34 val rdd = employee_df.rdd
35   .groupBy(row => row(2)) //group by location
36   .flatMap(x => {          //flat map to new list
37     var newList = mutable.MutableList[Row]()
38
39     for (row <- x._2){
40       newList += row
41       if(row.getDouble(3) < 60000){
42         //Give 10% bonus
43         val newRow = Row(row(0), row(1), row(2), row(3) * 1.1)
44         newList += newRow
45       }
46     }
47
48     newList
49   })
50
51 val df2 = sqlContext.createDataFrame(rdd, schema)
52 df2.show()
53
```

Output:

```

1 import org.apache.spark.sql.types._
2 import org.apache.spark.sql.Row
3 import org.apache.spark.rdd.RDD
4 import scala.collection._
5 schema: org.apache.spark.sql.types.StructType = StructType(StructField(id,IntegerType,true),StructField(name,StringType,true),StructField(location,StringType,true),StructField(salary,DoubleType,true))
6 employees: Seq[org.apache.spark.sql.Row] = List([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])
7 employee_df: org.apache.spark.sql.DataFrame = [id: int, name: string, location: string, salary: double]
8 +---+-----+-----+-----+
9 | id|  name|location| salary|
10 +---+-----+-----+-----+
11 |  1|  John|     USA|50000.0|
12 |  2| Peter|     AU|60000.0|
13 |  3|   Sam|     AU|60000.0|
14 |  4| Susan|     USA|50000.0|
15 |  5| David|     USA|70000.0|
16 |  6|Elliot|     AU|50000.0|
17 +---+-----+-----+-----+
18 rdd: org.apache.spark.rdd.RDD[org.apache.spark.sql.Row] = org.apache.spark.rdd.RDD[org.apache.spark.sql.Row] = [id: int, name: string, location: string, salary: double]
19 df2: org.apache.spark.sql.DataFrame = [id: int, name: string, location: string, salary: double]
20 +---+-----+-----+-----+
21 | id|  name|location| salary|
22 +---+-----+-----+-----+
23 |  2| Peter|     AU|60000.0|
24 |  3|   Sam|     AU|60000.0|
25 |  6|Elliot|     AU|50000.0|
26 |  6|Elliot|     AU| 5000.0|
27 |  1|  John|     USA|50000.0|
28 |  1|  John|     USA| 5000.0|
29 |  4| Susan|     USA|50000.0|
30 |  4| Susan|     USA| 5000.0|
31 |  5| David|     USA|70000.0|
32 +---+-----+-----+-----+
33

```

◀ 07: Spark on Zeppelin – window functions in Scala

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

Disclaimer

The contents in this Java-Success are copyrighted and from EmpoweringTech pty ltd. The EmpoweringTech pty ltd has the right to correct or enhance the current content without any prior notice. These are general advice only, and one needs to take his/her own circumstances

into consideration. The EmpoweringTech Pty Ltd will not be held liable for any damages caused or alleged to be caused either directly or indirectly by these materials and resources. Any trademarked names or labels used in this blog remain the property of their respective trademark owners. Links to external sites do not imply endorsement of the linked-to sites. [Privacy Policy](#)