06: Spark on Zeppelin – RDD operation zipWithIndex

Posted on August 25, 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.

Q. Why do we need zipWithIndex?
A. In database world there are various instances where we want to assign a unique sequence number to the records in a table. It may be a challenge in a distributed environment like Hadoop as we have to make sure we don't come across duplicate sequence numbers for data stored in multiple nodes. Before

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

using **zipWIthIndex** we could run distinct() and sort() functions to assign indices the way we want.

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: The input file to read “employees.txt” in the $(pwd)/seed.

```
00~Information About employees
01~1~John~USA~100000.00
01~2~Peter~Australia~200000.00
01~3~Sam~USA~76000.00
01~4~Daniel~France~86000.00
01~5~Simon~Australia~96000.00
01~6~Roseanne~France~156000.00
99~6 records

```

Step 3: Run the container with the above image.

```
$ docker run --rm -it -p 8080:8080 -v "$(pwd)/seed

```

Note: $(pwd)/seed – is the folder where the **employees.txt** input file will be placed on the host system, and will be synchronized with the container path “/zeppelin/seed”.

You can inspect the container files/logs with the following commands in a separate terminal window:

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

Get the container id with:

```
$ docker ps

```

sh to the container with:

```
$ docker exec -it <container id> /bin/bash

```

Step 4: Open Zeppelin notebook via a web browser “http:localhost:8080”. Create a note book with “spark” as a default interpreter.

The following code gets rid of the header & footer record types with values “00” and “99” respectively, and extracts out the employee details as shown below:

```
%spark

val lines_rdd = sc.textFile("file:///zeppelin/see

val cached_rdd = lines_rdd.cache()
val total_records = cached_rdd.count()

case class Employee (id: Integer, name: String, l

val employee_df = cached_rdd.zipWithIndex()
                  .filter(_._2 > 0)
                  .filter(_._2 < (total_records - 1
                  .map(_._1.split("~"))
                  .map(s => Employee(s(1).toInt, s(
                  .toDF()

employee_df.show(false)

```

Output:

```
+---+--------+---------+--------+
|id |name    |location |salary  |
+---+--------+---------+--------+
|1  |John    |USA      |100000.0|
|2  |Peter   |Australia|200000.0|
|3  |Sam     |USA      |76000.0 |
|4  |Daniel  |France   |86000.0 |
|5  |Simon   |Australia|96000.0 |
|6  |Roseanne|France   |156000.0|
+---+--------+---------+--------+
```

Getting the bottom n records

Getting the top “N” records is straightforward (E.g. df.limit(2)). How do we get the bottom “N” records?

```
%spark

val lines_rdd = sc.textFile("file:///zeppelin/seed

val cached_rdd = lines_rdd.cache()
val total_records = cached_rdd.count()

case class Employee (id: Integer, name: String, lo

val startIndex: Long =  (total_records - 2) - 2  /
                                                 //

val employee_df = cached_rdd.zipWithIndex()
                .filter(_._2 > 0)
                .filter(_._2 < (total_records - 1)
                .filter{case (_, index) => index >
                .map(_._1.split("~"))
                .map(s => Employee(s(1).toInt, s(2
                .toDF()

employee_df.show(false)
```

Output:

```
+---+--------+---------+--------+
|id |name    |location |salary  |
+---+--------+---------+--------+
|5  |Simon   |Australia|96000.0 |
|6  |Roseanne|France   |156000.0|
+---+--------+---------+--------+
```

```
7
```

Note: Since Spark 1.6 there is a function called monotonically_increasing_id(), which generates a new column with unique 64-bit monotonic index for each row, but it isn't consequential.

‹ 5 curl vs wget interview Q&As

07: Spark on Zeppelin – window functions in Scala ›

Disclaimer