

Spark shell 是一个特别适合快速开发 Spark 程序的工具。即使你对 Scala 不熟悉，仍然可以使用这个工具快速应用 Scala 操作 Spark。Spark shell 使得用户可以和 Spark 集群交互，提交查询，这便于调试，也便于初学者使用 Spark。Spark shell 是非常方便的，因为它很大程度上基于 Scala REPL(Scala 交互式 shell，即 Scala 解释器)，并继承了 Scala REPL(读取-求值-打印-循环)(Read-Evaluate-Print-Loop)的所有功能。运行 spark-shell，则会运行 spark-submit，spark-shell 其实是对 spark-submit 的一层封装。

下面是 Spark shell 的运行原理图

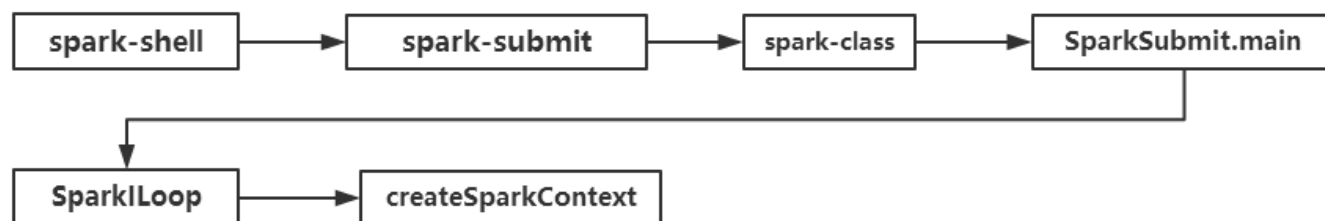


图 1: Spark shell 的运行原理图

RDD 有两种类型的操作，分别是 Transformation（返回一个新的 RDD）和 Action（返回 values）。

- Transformation：根据已有 RDD 创建新的 RDD 数据集 build
  - 1) map(func): 对调用 map 的 RDD 数据集中的每个 element 都使用 func，然后返回一个新的 RDD，这个返回的数据集是分布式的数据集。
  - 2) filter(func) : 对调用 filter 的 RDD 数据集中的每个元素都使用 func，然后返回一个包含使 func 为 true 的元素构成的 RDD。
  - 3) flatMap(func): 和 map 很像，但是 flatMap 生成的是多个结果。
  - 4) mapPartitions(func): 和 map 很像，但是 map 是每个 element，而 mapPartitions 是每个 partition。
  - 5) mapPartitionsWithSplit(func): 和 mapPartitions 很像，但是 func 作用的是其中一个 split 上，所以 func 中应该有 index。
  - 6) sample(withReplacement,faction,seed): 抽样。
  - 7) union(otherDataset): 返回一个新的 dataset，包含源 dataset 和给定 dataset 的元素的集合。
  - 8) distinct([numTasks]): 返回一个新的 dataset，这个 dataset 含有的是源 dataset 中的 distinct 的 element。
  - 9) groupByKey(numTasks): 返回 (K,Seq[V])，也就是 Hadoop 中 reduce 函数接受的 key-valuelist。

- 
- 10) `reduceByKey(func,[numTasks])`: 就是用一个给定的 `reduce func` 再作用在 `groupByKey` 产生的 `(K,Seq[V])`, 比如求和, 求平均数。
  - 11) `sortByKey([ascending],[numTasks])`: 按照 `key` 来进行排序, 是升序还是降序, `ascending` 是 `boolean` 类型。
  - Action: 在 RDD 数据集运行计算后, 返回一个值或者将结果写入外部存储
    - 1) `reduce(func)`: 就是聚集, 但是传入的函数是两个参数输入返回一个值, 这个函数必须是满足交换律和结合律的。
    - 2) `collect()`: 一般在 `filter` 或者足够小的结果的时候, 再用 `collect` 封装返回一个数组。
    - 3) `count()`: 返回的是 `dataset` 中的 `element` 的个数。
    - 4) `first()`: 返回的是 `dataset` 中的第一个元素。
    - 5) `take(n)`: 返回前 `n` 个 `elements`。
    - 6) `takeSample(withReplacement, num, seed)`: 抽样返回一个 `dataset` 中的 `num` 个元素, 随机种子 `seed`。
    - 7) `saveAsTextFile (path)`: 把 `dataset` 写到一个 `textfile` 中, 或者 `HDFS`, 或者 `HDFS` 支持的文件系统中, `Spark` 把每条记录都转换为一行记录, 然后写到 `file` 中。
    - 8) `saveAsSequenceFile(path)`: 只能用在 `key-value` 对上, 然后生成 `SequenceFile` 写到本地或者 `Hadoop` 文件系统。
    - 9) `countByKey()`: 返回的是 `key` 对应的个数的一个 `map`, 作用于一个 `RDD`。
    - 10) `foreach(func)`: 对 `dataset` 中的每个元素都使用 `func`。

#### 获取实验测试数据

```
wget -P ./spark5 file_ip:60000/allfiles/spark5/orders
wget -P ./spark5 file_ip:60000/allfiles/spark5/order_items

wget file_ip:60000/allfiles/spark3/wordcount/buyer_favorite
wget file_ip:60000/allfiels/spark3/distinct/buyer_favorite
wget -P ./sort file_ip:60000/allfiles/spark3/sort/goods_visit

wget -P ./join file_ip:60000/allfiles/spark3/join/orders
wget -P ./join file_ip:60000/allfiles/spark3/join/order_items

wget -P ./avg file_ip:60000/allfiles/spark3/avg/goods
wget -P ./avg file_ip:60000/allfiles/spark3/avg/goods_visit
```

## 测试 Shell 操作

```
$ hdfs dfs -mkdir /spark5
$ hdfs dfs -put /data/spark5/orders /spark5
$ hdfs dfs -put /data/spark5/order_items /spark5
$ spark-shell
scala> val sqlContext = new org.apache.spark.sql.SQLContext(sc)
scala> import sqlContext.implicits._
scala> case class Orders(order_id:String,order_number:String,buyer_id
    :String,create_dt:String)
scala> val dforders = sc.textFile("/myspark5/orders").map(_.split('\t
    ')).map(line=>Orders(line(0),line(1),line(2),line(3))).toDF()
scala> dforders.registerTempTable("orders")
scala> sqlContext.sql("show tables").map(t=>"tableName is:"+t(0)).
    collect().foreach(println)
scala> sqlContext.sql("select order_id,buyer_id from orders").collect
scala>
scala> import org.apache.spark.sql._
scala> import org.apache.spark.sql.types._
scala> val rddorder_items = sc.textFile("/myspark5/order_items")
scala> val roworder_items = rddorder_items.map(_.split("\t")).map( p
    =>Row(p(0),p(1),p(2) ) )
scala> val schemaorder_items = "item_id order_id goods_id"
scala> val schema = StructType(schemaorder_items.split(" ").map(
    fieldName=>StructField(fieldName,StringType,true)) )
scala> val dforder_items = sqlContext.applySchema(roworder_items,
    schema)
scala> dforder_items.registerTempTable("order_items")
scala>
scala> sqlContext.sql("show tables").map(t=>"tableName is:"+t(0)).
    collect().foreach(println)
scala> sqlContext.sql("select order_id,goods_id from order_items ").
    collect
scala>
scala> sqlContext.sql("select orders.buyer_id, order_items.goods_id
    from order_items join orders on order_items.order_id=orders.
    order_id ").collect
```

```
scala>
scala> exit
```

一些测试过程中的效果截图：

```
order_items orders
zhangyu@bd2a17a1c41a:~/datas/spark/spark5$ cd ..
zhangyu@bd2a17a1c41a:~/datas/spark$ ls
avg buyer_favorite get_data.sh join sort spark5
zhangyu@bd2a17a1c41a:~/datas/spark$ vi buyer_favorite
zhangyu@bd2a17a1c41a:~/datas/spark$ hdfs dfs -mkdir -p /spark/wordcount
zhangyu@bd2a17a1c41a:~/datas/spark$ hdfs dfs -put buyer_favorite /spark/wordcount
zhangyu@bd2a17a1c41a:~/datas/spark$ spark-shell
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/home/zhangyu/spark/jars/slf4j-log4j12-1.7.16.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/hadoop/share/hadoop/common/lib/slf4j-log4j12-1.7.25.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
2019-07-12 03:37:32,176 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
Spark context Web UI available at http://master:4040
Spark context available as 'sc' (master = local[*], app id = local-1562902674231).
Spark session available as 'spark'.
Welcome to
Failed
Running
July 2019
Version 2.4.3
Mon 15 Tue 16 Wed 17 Thu 18 Fr
Using Scala version 2.11.12 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_212)
Type in expressions to have them evaluated.
Type :help for more information.
scala>
[0] 0: bash- 1: bash+ "bd2a17a1c41a" 11:39 12-7月-19
```

```
zhangyu@bd2a17a1c41a:~/datas/spark$ hdfs dfs -ls -R /spark/
drwxr-xr-x - zhangyu supergroup 0 2019-07-12 03:49 /spark/avg
-rw-r--r-- 3 zhangyu supergroup 208799 2019-07-12 03:49 /spark/avg/goods
-rw-r--r-- 3 zhangyu supergroup 82421 2019-07-12 03:49 /spark/avg/goods_visit
drwxr-xr-x - zhangyu supergroup 0 2019-07-12 03:48 /spark/distinct
drwxr-xr-x - zhangyu supergroup 0 2019-07-12 03:50 /spark/join
-rw-r--r-- 3 zhangyu supergroup 328 2019-07-12 03:50 /spark/join/order_items
-rw-r--r-- 3 zhangyu supergroup 460 2019-07-12 03:50 /spark/join/orders
drwxr-xr-x - zhangyu supergroup 0 2019-07-12 03:49 /spark/sort
-rw-r--r-- 3 zhangyu supergroup 104 2019-07-12 03:49 /spark/sort/goods_visit
drwxr-xr-x - zhangyu supergroup 0 2019-07-12 03:50 /spark/spark5
-rw-r--r-- 3 zhangyu supergroup 315 2019-07-12 03:50 /spark/spark5/order_items
-rw-r--r-- 3 zhangyu supergroup 460 2019-07-12 03:50 /spark/spark5/orders
drwxr-xr-x - zhangyu supergroup 0 2019-07-12 03:37 /spark/wordcount
-rw-r--r-- 3 zhangyu supergroup 1019 2019-07-12 03:37 /spark/wordcount/buyer_favorite
zhangyu@bd2a17a1c41a:~/datas/spark$
```

buyer\_favorite

用户id	商品id	收藏日期
10181	1000481	2010-04-04 16:54:31
20001	1001597	2010-04-07 15:07:52
20001	1001560	2010-04-07 15:08:27

20042	1001368	2010-04-08 08:20:30
20067	1002061	2010-04-08 16:45:33
20056	1003289	2010-04-12 10:50:55

```
scala> rdd.map(line => (line.split('\t')(0),1)).reduceByKey(_+_).collect
res4: Array[(String, Int)] = Array((20042,1), (20055,1), (20064,1), (20054,6), (20001,2), (10181,1),
(20067,1), (20056,12), (20076,5))

scala> rdd.map(line => (line.split('\t')(1),1)).reduceByKey(_+_).collect
res5: Array[(String, Int)] = Array((1003055,1), (1003103,2), (1001679,1), (1010675,1), (1003064,1),
(1002061,1), (1002429,1), (1003101,1), (1002427,1), (1003066,2), (1010183,1), (1000481,1), (1010178,
1), (1003290,1), (1003326,1), (1002420,2), (1003292,1), (1001368,1), (1001560,1), (1001597,1), (1003
289,1), (1002422,2), (1003094,1), (1003100,3))

scala>
```

Spark 实现 WordCount，比 Hadoop 要简单的多

```
scala> val rdd = sc.textFile("hdfs://ldcluster:8020/spark/wordcount/buyer_favorite")
rdd: org.apache.spark.rdd.RDD[String] = hdfs://ldcluster:8020/spark/wordcount/buyer_favorite MapPart
itionsRDD[1] at textFile at <console>:24

scala> rdd.count
res2: Long = 30

scala> rdd.map(line => line.split('\t')(1)).distinct.collect
res3: Array[String] = Array(1003055, 1003103, 1001679, 1010675, 1003064, 1002061, 1002429, 1003101,
1002427, 1003066, 1010183, 1000481, 1010178, 1003290, 1003326, 1002420, 1003292, 1001368, 1001560, 1
001597, 1003289, 1002422, 1003094, 1003100)

scala>
```

Spark 实现去重，查看哪些商品被收藏

goods\_visit

商品ID	点击次数
1010037	100
1010102	100
1010152	97
1010178	96
1010280	104

```
scala> val rdd = sc.textFile("hdfs://ldcluster:8020/spark/sort/goods_visit")
rdd: org.apache.spark.rdd.RDD[String] = hdfs://ldcluster:8020/spark/sort/goods_visit MapPartitionsRD
D[1] at textFile at <console>:24

scala> rdd.map(line => (line.split('\t')(1).toInt, line.split('\t')(0))).sortByKey(true).collect
res0: Array[(Int, String)] = Array((96,1010178), (96,1010603), (97,1010152), (97,1010637), (100,1010
037), (100,1010102), (103,1010320), (104,1010280), (104,1010510))

scala>
```

Spark 对用户记录进行排序，实现按购买数升序排列。



orders

订单ID	订单号	用户ID	下单日期
52304	111215052630	176474	2011-12-15 04:58:21
52303	111215052629	178350	2011-12-15 04:45:31
52302	111215052628	172296	2011-12-15 03:12:23
52301	111215052627	178348	2011-12-15 02:37:32
52300	111215052626	174893	2011-12-15 02:18:56

order\_items

明细ID	订单ID	商品ID
252578	52293	1016840
252579	52293	1014040
252580	52294	1014200
252581	52294	1001012
252582	52294	1022245

```
scala> val rdd11 = rdd1.map(line => (line.split('\t')(0), line.split('\t')(2)))
rdd11: org.apache.spark.rdd.RDD[(String, String)] = MapPartitionsRDD[10] at map at <console>:25

scala> val rdd22 = rdd2.map(line => (line.split('\t')(1), line.split('\t')(2)))
rdd22: org.apache.spark.rdd.RDD[(String, String)] = MapPartitionsRDD[11] at map at <console>:25

scala> rdd1.collect
res1: Array[String] = Array(52304, 111215052630, 176474, 2011-12-15 04:58:21, 52303, 111215052629, 178350, 2011-12-15 04:45:31, 52302, 111215052628, 172296, 2011-12-15 03:12:23, 52301, 111215052627, 178348, 2011-12-15 02:37:32, 52300, 111215052626, 174893, 2011-12-15 02:18:56, 52299, 111215052625, 169471, 2011-12-15 01:33:46, 52298, 111215052624, 178345, 2011-12-15 01:04:41, 52297, 111215052623, 176369, 2011-12-15 01:02:20, 52296, 111215052622, 178343, 2011-12-15 00:38:02, 52295, 111215052621, 178342, 2011-12-15 00:18:43)

scala> rdd11.collect
res2: Array[(String, String)] = Array((52304,176474), (52303,178350), (52302,172296), (52301,178348), (52300,174893), (52299,169471), (52298,178345), (52297,176369), (52296,178343), (52295,178342))

scala> rdd22.collect
res3: Array[(String, String)] = Array((52293,1016840), (52293,1014040), (52294,1014200), (52294,1001012), (52294,1022245), (52294,1014724), (52294,1010731), (52295,1023399), (52295,1016840), (52296,1021134), (52296,1021133), (52295,1021840), (52295,1014040), (52296,1014040), (52296,1019043))

scala> var rddjoin = rdd11 join rdd22
rddjoin: org.apache.spark.rdd.RDD[(String, (String, String))] = MapPartitionsRDD[14] at join at <console>:27

scala> rddjoin.collect
res4: Array[(String, (String, String))] = Array((52296,(178343,1021134)), (52296,(178343,1021133)), (52296,(178343,1014040)), (52296,(178343,1019043)), (52295,(178342,1023399)), (52295,(178342,1016840)), (52295,(178342,1021840)), (52295,(178342,1014040)))

scala>
```

Spark 实现两张表的 Join 操作