APACHE SPARK

Different ways to create Spark RDD

Spark RDD can be created in several ways using Scala & Pyspark languages, for example, It can be created by using sparkContext.parallelize(), from text file, from another RDD, DataFrame, and Dataset.

Resilient Distributed Datasets (RDD) is the fundamental data structure of Spark. RDDs are immutable and fault-tolerant in nature. RDD is just the way of representing Dataset distributed across multiple nodes in a cluster, which can be operated in parallel. RDDs are called resilient because they have the ability to always re-compute an RDD when a node failure.

Let's see how to create an RDD in Apache Spark with examples:

Spark Create RDD from Seq or List (using Parallelize)

RDD's are generally created by parallelized collection i.e. by taking an existing collection from driver program (scala, python e.t.c) and passing it to SparkContext's parallelize() method. This method is used only for testing but not in realtime as the entire data will reside on one node which is not ideal for production.

Spark(Scala) Transformations

Map Operations

```
//1. map()-  Return a new RDD by applying a function to each element of this RDD

//map(func)

val x = sc.parallelize(Array("b", "a", "c"))

val y = x.map(z => (z,1))

println(x.collect().mkString(", "))

println(y.collect().mkString(", "))
```

```
b, a, c (b,1), (a,1), (c,1) x: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[102] at parallelize at command-509646307872266:3 y: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[103] at map at command-509646307872266:4
```

```
//2. flatmap()- Return a new RDD by first applying a function to all elements of this
RDD, and then flattening the results

//flatMap(func)

val x = sc.parallelize(Array(1,2,3))

val y = x.flatMap(n => Array(n, n*100, 42))

println(x.collect().mkString(", "))

println(y.collect().mkString(", "))
```

```
1, 2, 3 1, 100, 42, 2, 200, 42, 3, 300, 42 x: org.apache.spark.rdd.RDD[Int] =
ParallelCollectionRDD[104] at parallelize at command-509646307872269:3 y:
org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD[105] at flatMap at
command-509646307872269:4
```

```
//3. mapPartitions()- Return a new RDD by applying a function to each partition of
this RDD

//similar to map

//mapPartitions(func)

val parallel = sc.parallelize(1 to 9, 3)

parallel.mapPartitions( x => List(x.next).iterator).collect
```

```
parallel: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[106] at parallelize at
command-509646307872272:3 res34: Array[Int] = Array(1, 4, 7)
```

Set Operations

```
//union()- Return a new RDD containing all items from two original RDDs. Duplicates
are not culled

//union(otherDataset)

//glom() flattens elements on the same partition

val x= sc.parallelize(Array(1,2,3), 2)

val y= sc.parallelize(Array(3,4), 1)

val z= x.union(y)
val zOut= z.glom().collect()

val zout1= z.collect()
```

```
x: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[108] at parallelize at
command-509646307872273:4 y: org.apache.spark.rdd.RDD[Int] =
ParallelCollectionRDD[109] at parallelize at command-509646307872273:5 z:
org.apache.spark.rdd.RDD[Int] = UnionRDD[110] at union at
command-509646307872273:6 zOut: Array[Array[Int]] = Array(Array(1), Array(2, 3),
Array(3, 4)) zout1: Array[Int] = Array(1, 2, 3, 3, 4)
```

```
//distinct()- Return a new RDD containing distinct items from the original RDD
(omitting all duplicates)

//distinct([numPartitions]))

val x= sc.parallelize(Array(1,2,3,3,4))

val y= x.distinct()

println(y.collect().mkString(", "))
```

```
1, 2, 3, 4 x: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[112] at parallelize
at command-509646307872284:3 y: org.apache.spark.rdd.RDD[Int] =
MapPartitionsRDD[115] at distinct at command-509646307872284:4
```

```
//intersection() :  Returns elements that are common b/w both RDDs. Also removed
Duplicates

//intersection(otherDataset)

val x = sc.parallelize(List("lion", "tiger", "tiger", "peacock", "horse"))

val y = sc.parallelize(List("lion", "tiger"))

x.intersection(y).collect()
```

```
x: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[171] at parallelize at
command-3668865374100100:3 y: org.apache.spark.rdd.RDD[String] =
ParallelCollectionRDD[172] at parallelize at command-3668865374100100:4 res54:
Array[String] = Array(lion, tiger)
```

```
//subtract()- Returns only elements that are present in the first RDD

val x = sc.parallelize(List("lion", "tiger", "tiger", "peacock", "horse"))

val y = sc.parallelize(List("lion", "tiger"))

x.subtract(y).collect()
```

```
x: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[179] at parallelize at
command-3668865374100101:3 y: org.apache.spark.rdd.RDD[String] =
ParallelCollectionRDD[180] at parallelize at command-3668865374100101:4 res55:
Array[String] = Array(peacock, horse)
```

```
//cartesian()- Provides cartesian product b/w 2 RDDs

//cartesian(otherDataset)

val x = sc.parallelize(List("lion", "tiger", "tiger", "peacock", "horse"))

val y = sc.parallelize(List("lion", "tiger"))

x.cartesian(y).collect()
```

```
x: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[0] at parallelize at
command-3668865374100102:3 y: org.apache.spark.rdd.RDD[String] =
ParallelCollectionRDD[1] at parallelize at command-3668865374100102:4 res0:
Array[(String, String)] = Array((lion,lion), (lion,tiger), (tiger,lion), (tiger,tiger), (tiger,lion),
(tiger,tiger), (peacock,lion), (peacock,tiger), (horse,lion), (horse,tiger))
```

Other Operators

```
//filter

//filter(func)- Return a new dataset formed by selecting those elements of the source
on which func returns true.

val x = sc.parallelize(Array(1,2,3))

val y = x.filter(n => n%2 == 1)

println(x.collect().mkString(", "))

println(y.collect().mkString(", "))
```

```
1, 2, 3 1, 3 x: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[116] at
```

```
parallelize at command-509646307872268:2 y: org.apache.spark.rdd.RDD[Int] =
MapPartitionsRDD[117] at filter at command-509646307872268:3
```

```scala
//groupby

val x= sc.parallelize(Array("John", "Fred", "Anna", "James"))

val y= x.groupBy(w => w.charAt(0))

println(x.collect().mkString(", "))

println(y.collect().mkString(", "))
```

```
John, Fred, Anna, James (A,CompactBuffer(Anna)), (J,CompactBuffer(John, James)),
(F,CompactBuffer(Fred)) x: org.apache.spark.rdd.RDD[String] =
ParallelCollectionRDD[118] at parallelize at command-509646307872270:2 y:
org.apache.spark.rdd.RDD[(Char, Iterable[String])] = ShuffledRDD[120] at groupBy at
command-509646307872270:3
```

```scala
//toDebugString

val x = sc.parallelize(List("This is a word", "This is another word"), 7)

val y = x.flatMap(line => line.split(" "))

val z = y.map ( word => (word, 1) )

val a = z.reduceByKey ( (x, y) => x + y )

a.toDebugString

a.collect()
```

```
x: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[192] at parallelize at
command-3668865374100103:3 y: org.apache.spark.rdd.RDD[String] =
MapPartitionsRDD[193] at flatMap at command-3668865374100103:4 z:
org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[194] at map at
command-3668865374100103:5 a: org.apache.spark.rdd.RDD[(String, Int)] =
ShuffledRDD[195] at reduceByKey at command-3668865374100103:6 res59:
Array[(String, Int)] = Array((This,2), (is,2), (another,1), (a,1), (word,2))
```

RDD Functions

//groupbykey - When called on a dataset of (K, V) pairs, returns a dataset of (K, Iterable<V>) pairs

//groupByKey([numPartitions])

```
val x= sc.parallelize(Array(('B',5),('B',4),('A',3),('A',2),('A',1)))

val y= x.groupByKey()

println(x.collect().mkString(", "))

println(y.collect().mkString(", "))
```

```
(B,5), (B,4), (A,3), (A,2), (A,1) (A,CompactBuffer(3, 2, 1)), (B,CompactBuffer(5, 4)) x:
org.apache.spark.rdd.RDD[(Char, Int)] = ParallelCollectionRDD[121] at parallelize at
command-509646307872271:2 y: org.apache.spark.rdd.RDD[(Char, Iterable[Int])] =
ShuffledRDD[122] at groupByKey at command-509646307872271:3
```

//keyBy- Create a Pair RDD, forming one pair for each item in the original RDD. The pair's key is calculated from the value via a user-supplied function.

```
val x= sc.parallelize(Array("John", "Fred", "Anna", "James"))

val y= x.keyBy(w => w.charAt(0))

println(x.collect().mkString(", "))

println(y.collect().mkString(", "))
```

```
John, Fred, Anna, James (J,John), (F,Fred), (A,Anna), (J,James) x:
org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[123] at parallelize at
command-509646307872286:3 y: org.apache.spark.rdd.RDD[(Char, String)] =
MapPartitionsRDD[124] at keyBy at command-509646307872286:4
```

```
//join - Return a new RDD containing all pairs of elements having the same key in the
original RDDs

//join(otherDataset, [numPartitions])


val x= sc.parallelize(Seq(("math", 55),("math", 56),("english", 57),("english", 58),
("science", 59),("science", 54)))

val y= sc.parallelize(Seq(("math", 60),("math", 65),("science", 61),("science", 62),
("history", 63),("history", 64)))

val z= x.join(y)

println(z.collect().mkString(", "))

(math,(55,60)), (math,(55,65)), (math,(56,60)), (math,(56,65)), (science,(59,61)),
(science,(59,62)), (science,(54,61)), (science,(54,62)) x:
org.apache.spark.rdd.RDD[(String, Int)] = ParallelCollectionRDD[9] at parallelize at
command-509646307872274:3 y: org.apache.spark.rdd.RDD[(String, Int)] =
ParallelCollectionRDD[10] at parallelize at command-509646307872274:4 z:
org.apache.spark.rdd.RDD[(String, (Int, Int))] = MapPartitionsRDD[13] at join at
command-509646307872274:5


//leftOuterJoin()

val x= sc.parallelize(Seq(("math", 55),("math", 56),("english", 57),("english", 58),
("science", 59),("science", 54)))

val y= sc.parallelize(Seq(("math", 60),("math", 65),("science", 61),("science", 62),
("history", 63),("history", 64)))

val z= x.leftOuterJoin(y)

println(z.collect().mkString(", "))

(math,(55,Some(60))), (math,(55,Some(65))), (math,(56,Some(60))), (math,
(56,Some(65))), (english,(57,None)), (english,(58,None)), (science,(59,Some(61))),
(science,(59,Some(62))), (science,(54,Some(61))), (science,(54,Some(62))) x:
org.apache.spark.rdd.RDD[(String, Int)] = ParallelCollectionRDD[14] at parallelize at
command-3381355437890437:3 y: org.apache.spark.rdd.RDD[(String, Int)] =
ParallelCollectionRDD[15] at parallelize at command-3381355437890437:4 z:
org.apache.spark.rdd.RDD[(String, (Int, Option[Int]))] = MapPartitionsRDD[18] at
leftOuterJoin at command-3381355437890437:5
```

```
//rightOuterJoin()

val x= sc.parallelize(Seq(("math", 55),("math", 56),("english", 57),("english", 58),
("science", 59),("science", 54)))

val y= sc.parallelize(Seq(("math", 60),("math", 65),("science", 61),("science", 62),
("history", 63),("history", 64)))

val z= x.rightOuterJoin(y)

println(z.collect().mkString(", "))
```

```
(math,(Some(55),60)), (math,(Some(55),65)), (math,(Some(56),60)), (math,
(Some(56),65)), (history,(None,63)), (history,(None,64)), (science,(Some(59),61)),
(science,(Some(59),62)), (science,(Some(54),61)), (science,(Some(54),62)) x:
org.apache.spark.rdd.RDD[(String, Int)] = ParallelCollectionRDD[19] at parallelize at
command-3381355437890438:3 y: org.apache.spark.rdd.RDD[(String, Int)] =
ParallelCollectionRDD[20] at parallelize at command-3381355437890438:4 z:
org.apache.spark.rdd.RDD[(String, (Option[Int], Int))] = MapPartitionsRDD[23] at
rightOuterJoin at command-3381355437890438:5
```

```
//cogroup - Multiple Pair RDDs can be combined using cogroup()

//cogroup(otherDataset, [numPartitions])

val x= sc.parallelize(Array(("a", 1), ("b", 2)))

val y= sc.parallelize(Array(("a", 3), ("a", 4), ("b", 5)))

val z= x.cogroup(y)

println(z.collect().mkString(", "))
```

```
(a,(CompactBuffer(1),CompactBuffer(3, 4))), (b,
(CompactBuffer(2),CompactBuffer(5))) x: org.apache.spark.rdd.RDD[(String, Int)] =
ParallelCollectionRDD[0] at parallelize at command-3381355437890434:3 y:
org.apache.spark.rdd.RDD[(String, Int)] = ParallelCollectionRDD[1] at parallelize at
command-3381355437890434:4 z: org.apache.spark.rdd.RDD[(String, (Iterable[Int],
Iterable[Int]))] = MapPartitionsRDD[3] at cogroup at command-3381355437890434:5
```

Sorting

```
//sortByKey()- part of OrderedRDDFunctions that works on Key/Value pairs.

//sortByKey([ascending], [numPartitions])
```

```
val x= sc.parallelize(Seq(("math", 55),("math", 56),("english", 57),("english", 58),
("science", 59),("science", 54)))

x.sortByKey().collect()
```

```
x: org.apache.spark.rdd.RDD[(String, Int)] = ParallelCollectionRDD[24] at parallelize at
command-3381355437890439:2 res6: Array[(String, Int)] = Array((english,57),
(english,58), (math,55), (math,56), (science,59), (science,54))
```

```
//top() & takeOrdered() are actions that return the N elements based on the default
ordering or the Customer ordering provided by us

//takeOrdered(n, [ordering])


val x = sc.parallelize( Seq(10, 4, 5, 3, 11, 2, 6) )

val xout = x.top(5)

val yout = x.takeOrdered(5)

println(xout)

println(yout)
```

```
[I@301a39d4 [I@38c5f3cc x: org.apache.spark.rdd.RDD[Int] =
ParallelCollectionRDD[30] at parallelize at command-3381355437890440:3 xout:
Array[Int] = Array(11, 10, 6, 5, 4) yout: Array[Int] = Array(2, 3, 4, 5, 6)
```

Partitioning

```
//COALESCE - Return a new RDD which is reduced to a smaller number of partitions

//coalesce(numPartitions)

val x= sc.parallelize(Array(1, 2, 3, 4, 5), 3)

val y= x.coalesce(2)

val xOut= x.glom().collect()

val yOut= y.glom().collect()

val xOut1= x.collect()

val yOut2= y.collect()
```

```
x: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[130] at parallelize at command-509646307872285:2 y: org.apache.spark.rdd.RDD[Int] = CoalescedRDD[131] at coalesce at command-509646307872285:3 xOut: Array[Array[Int]] = Array(Array(1), Array(2, 3), Array(4, 5)) yOut: Array[Array[Int]] = Array(Array(1), Array(2, 3, 4, 5)) xOut1: Array[Int] = Array(1, 2, 3, 4, 5) yOut2: Array[Int] = Array(1, 2, 3, 4, 5)
```

```
//zip - Return a new RDD containing pairs whose key is the item in the original RDD, and whose value is that item's corresponding element (same partition, same index) in a second RDD

// zip(otherRDD)


val x= sc.parallelize(Array(1,2,3))

val y= x.map(n=>n*n)

val z= x.zip(y)

println(z.collect().mkString(", "))
```

```
(1,1), (2,4), (3,9) x: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[138] at parallelize at command-752734678252690:4 y: org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD[139] at map at command-752734678252690:5 z: org.apache.spark.rdd.RDD[(Int, Int)] = ZippedPartitionsRDD2[140] at zip at command-752734678252690:6
```

Actions

```
// getnumpartitions - Return the number of partitions in RDD

val x= sc.parallelize(Array(1,2,3), 2)

val y= x.partitions.size

val xOut= x.glom().collect()

println(y)
```

```
2 x: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[141] at parallelize at command-752734678252692:3 y: Int = 2 xOut: Array[Array[Int]] = Array(Array(1), Array(2, 3))
```

```
//collect - Return all items in the RDD to the driver in a single list

val x= sc.parallelize(Array(1,2,3), 2)

val y= x.collect()

val xOut= x.glom().collect()

println(y)
```

```
[I@78e0ec97 x: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[143] at
parallelize at command-752734678252691:3 y: Array[Int] = Array(1, 2, 3) xOut:
Array[Array[Int]] = Array(Array(1), Array(2, 3))
```

```
//reduce - Aggregate all the elements of the RDD by applying a user function
pairwise to elements and partial results, and returns a result to the driver

//reduce(func)

val x= sc.parallelize(Array(1,2,3,4))

val y= x.reduce((a,b) => a+b)

println(x.collect.mkString(", "))

println(y)
```

```
1, 2, 3, 4 10 x: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[145] at
parallelize at command-2599402715378481:3 y: Int = 10
```

```
//max - Return the maximum item in the RDD

val x= sc.parallelize(Array(2,4,1))

val y= x.max

println(x.collect().mkString(", "))

println(y)
```

```
2, 4, 1 4 x: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[147] at parallelize
at command-2599402715378483:3 y: Int = 4
```

```
//sum - Return the sum of the items in the RDD
```

```
val x= sc.parallelize(Array(2,4,1))

val y= x.sum

println(x.collect().mkString(", "))

println(y)
```

```
2, 4, 1 7.0 x: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[148] at parallelize
at command-2599402715378484:3 y: Double = 7.0
```

```
//mean - Return the mean of the items in the RDD

val x= sc.parallelize(Array(2,4,1))

val y= x.mean

println(x.collect().mkString(", "))

println(y)
```

```
2, 4, 1 2.3333333333333335 x: org.apache.spark.rdd.RDD[Int] =
ParallelCollectionRDD[150] at parallelize at command-2599402715378485:3 y:
Double = 2.3333333333333335
```

```
//stddev- Return the standard deviation of the items in the RDD

val x= sc.parallelize(Array(2,4,1))

val y= x.stdev

println(x.collect().mkString(", "))

println(y)
```

```
2, 4, 1 1.247219128924647 x: org.apache.spark.rdd.RDD[Int] =
ParallelCollectionRDD[153] at parallelize at command-2599402715378486:3 y:
Double = 1.247219128924647
```

```
//countByKey- Return a map of keys and counts of their occurrences in the RDD

val x= sc.parallelize(Array(('J',"James"),('F',"Fred"),('A',"Anna"),('J',"John")))

val y= x.countByKey()

println(y)
```

```
Map(A -> 1, J -> 2, F -> 1) x: org.apache.spark.rdd.RDD[(Char, String)] =
ParallelCollectionRDD[156] at parallelize at command-2599402715378487:3 y:
scala.collection.Map[Char,Long] = Map(A -> 1, J -> 2, F -> 1)
```