Commonly used transformations:

| **TRANSFORMATION & PURPOSE** | **EXAMPLE & RESULT** |
| --- | --- |
| filter(func) **Purpose:** return elements which gives true when “func” applied. | scala> val filtered = df.filter($”age” < 18) // returns people who are under 18  scala> filtered.show()  +---+------+  |age|  name|  +---+------+  | 16|Zeynep|  | 16| Murat|  | 12| Murat|  +---+------+ |
| map(func) **Purpose:** return new RDD by applying function on each element | scala> val rdd = sc.parallelized(List(1,2,3,4,5,6))  scala> val halfed = rdd.map(\_/2)  scala> halfed.collect()  res10: Array[Int] = Array(0, 1, 1, 2, 2, 3) |
| flatMap(func) **Purpose:** Similar to map except func returns a sequence rather than a value | scala> val textFile = spark.read.textFile(“text.txt”)  scala> val words = textFile.flatMap(str=>str.split(“ “))  scala> words.collect() **Result:** words.collect()  res12: Array[String] = Array(#, Apache, Spark, "", Spark, is, a, fast, and, general, cluster, computing, system, for, Big, Data., It, provides, high-level, APIs, in, Scala,, Java,, Python,, and, R,, and, an, optimized, engine, that, supports, general, computation, graphs, for, data, analysis., It, also, supports, a, rich, set, of, hig |
| reduceByKey(func,[numTasks]) **Purpose:** Aggregates values of a key by func. numTasks is optional to specify number of reduce tasks | scala> val ages = df.select(“age”).rdd.map(age=>(age,1))  scala> val ageCount = ages.reduceByKey(\_+\_)  scala> ageCount.collect()  res14: Array[(org.apache.spark.sql.Row, Int)] = Array(([21],4), ([12],1), ([16],2), ([42],1), ([22],1), ([23],2)) |
| groupByKey([numTasks]) **Purpose:** make groups by key values. Converts (K,V) values to (K, Iterable<V>) | scala> val countAge = ageCount.map{case (age, count) => (count, age)}  scala> countAge.groupByKey().collect()  res15: Array[(Int, Iterable[org.apache.spark.sql.Row])] = Array((4,CompactBuffer([21])), (1,CompactBuffer([12], [42], [22])), (2,CompactBuffer([16], [23]))) |
| distinct([numTasks]) **Purpose:** Removes duplicates | scala> var myList = sc.parallelize(List(“A”,”B”,”C”,”D”,”A”,”B”,”C”,”D”))  scala> var distinctMyList = myList.distinct()  scala> distinctMyList.collect()  res16: Array[String] = Array(A, B, C, D) |

Commonly used set operations:

| **TRANSFORMATION AND PURPOSE** | **EXAMPLE AND RESULT** |
| --- | --- |
| union() **Purpose:** set union | scala> var set1 = sc.parallelize(List("a","b"))  set1: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[45] at parallelize at <console>:30  scala> var set2 = sc.parallelize(List("c","d"))  set2: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[46] at parallelize at <console>:30  scala> var set3 = set1.union(set2)  set3: org.apache.spark.rdd.RDD[String] = UnionRDD[47] at union at <console>:34  scala> set3.collect()  res17: Array[String] = Array(a, b, c, d) |
| intersection() **Purpose:** set intersection | scala> var set1 = sc.parallelize(List("a","b","c"))  set1: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[48] at parallelize at <console>:30  scala> var set2 = sc.parallelize(List("c","d"))  set2: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[49] at parallelize at <console>:30  scala> var set3 = set1.intersection(set2)  set3: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[55] at intersection at <console>:34  scala> set3.collect()  res18: Array[String] = Array(c) |
| cartesian() **Purpose:** cartesian product | scala> var set1 = sc.parallelize(List("a","b"))  set1: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[56] at parallelize at <console>:30  scala> var set2 = sc.parallelize(List("c","d"))  set2: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[57] at parallelize at <console>:30  scala> var set3 = set1.cartesian(set2)  set3: org.apache.spark.rdd.RDD[(String, String)] = CartesianRDD[58] at cartesian at <console>:34  scala> set3.collect()  res19: Array[(String, String)] = Array((a,c), (a,d), (b,c), (b,d)) |
| subtract() **Purpose:** subtract set operation | scala> var set1 = sc.parallelize(List("a","b","c"))  set1: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[59] at parallelize at <console>:30  scala> var set2 = sc.parallelize(List("c","d"))  set2: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[60] at parallelize at <console>:30  scala> var set3 = set1.subtract(set2)  set3: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[64] at subtract at <console>:34  scala> set3.collect()  res20: Array[String] = Array(a, b) |
| join(RDD,[numTasks]) **Purpose:** Join 2 (K,V) sets | scala> val personJob = sc.parallelize(Seq(("Ahmet","Manav"),("Mehmet","Kasap"),("Ayse","Ogretmen")))  personJob: org.apache.spark.rdd.RDD[(String, String)] = ParallelCollectionRDD[70] at parallelize at <console>:30  scala> val personSal = sc.parallelize(Seq(("Ahmet",3000),("Mehmet",5000),("Ayse",3500)))  personSal: org.apache.spark.rdd.RDD[(String, Int)] = ParallelCollectionRDD[71] at parallelize at <console>:30  scala> val joined = personJob.join(personSal)  joined: org.apache.spark.rdd.RDD[(String, (String, Int))] = MapPartitionsRDD[74] at join at <console>:34  scala> joined.collect()  res22: Array[(String, (String, Int))] = Array((Ayse,(Ogretmen,3500)), (Mehmet,(Kasap,5000)), (Ahmet,(Manav,3000))) |
| cogroup(RDD,[numTasks]) **Purpose:** To convert (K,V) to (K,Iterable<V>) | scala> personJob.cogroup(personSal).collect()  res23: Array[(String, (Iterable[String], Iterable[Int]))] = Array((Ayse,(CompactBuffer(Ogretmen),CompactBuffer(3500))), (Mehmet,(CompactBuffer(Kasap),CompactBuffer(5000))), (Ahmet,(CompactBuffer(Manav),CompactBuffer(3000)))) |

Commonly used actions:

| **ACTION & PURPOSE** | **EXAMPLE & RESULT** |
| --- | --- |
| count() **Purpose:** get the number of data elements in the RDD | scala> val rdd = sc.parallelize(list(‘A’,’B’,’c’)) scala> rdd.count() **Result:** long = 3 |
| collect() **Purpose:** get all the data elements in an RDD as an array | See above examples |
| reduce(func) **Purpose:** Aggregate data elements in RDD in to one value by using “func” which is a binary operation | scala> val rdd = sc.parallelize(List("A","B","C"))  rdd: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[78] at parallelize at <console>:30  scala> rdd.reduce(\_+\_)  res26: String = ABC |
| take (n) **Purpose:** : takes n element | scala> rdd.take(2)  res27: Array[String] = Array(A, B) |
| foreach(func) **Purpose:** executes function for each element in rdd. | scala> val rdd =sc.parallelize(List(1,2,3,4,5))  rdd: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[79] at parallelize at <console>:30  scala> rdd.foreach(x=>println("%s \* 10 = %s".format(x,x\*10)))  1 \* 10 = 10  4 \* 10 = 40  5 \* 10 = 50  2 \* 10 = 20  3 \* 10 = 30 |
| first() **Purpose:** retrieves the first data element in RDD. Similar to take(1) | scala> val rdd = sc.parallelize(list(1,2,3,4)) scala> rdd.first() **Result:** Int = 1 |
| saveAsTextFile(path) **Purpose:** Writes the content of RDD to a text file or a set of text files to local file system/ HDFS | scala> val textFile = sc.textFile("/usr/local/spark/README.md")  textFile: org.apache.spark.rdd.RDD[String] = /usr/local/spark/README.md MapPartitionsRDD[93] at textFile at <console>:30  scala> textFile.filter(\_.contains("spark")).saveAsTextFile("/usr/local/spark/filtered\_README.md") |