Spark Transformations & Actions

Datasets



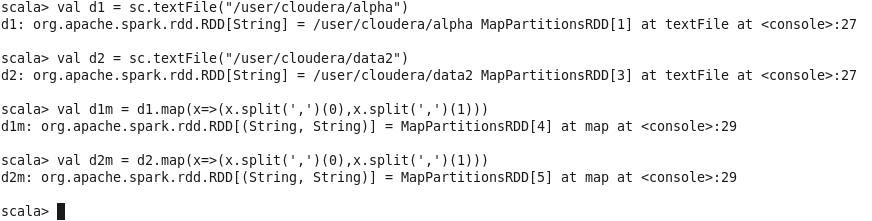


Transformations

**map(func)**

Returns a new distributed dataset, formed by passing each element of the source through a function **func**.

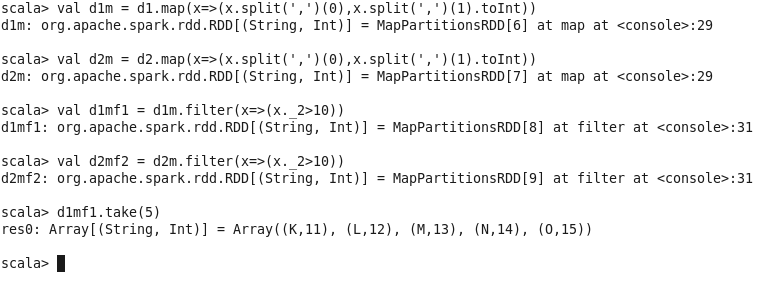
Example – Converted the input datasets which is basic RDD by default, into a pairRDD.



**filter(func)**

Returns a new dataset formed by selecting those elements of the source on which **func** returns true.

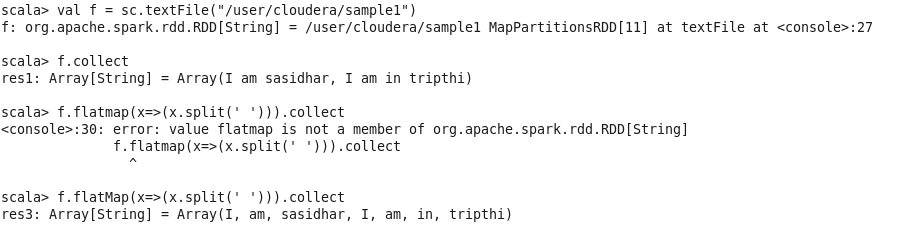
Example – Filtering a pairRDD, which has values greater than 10 and forming a new RDD



**flatMap(func)**

Similar to map, but each input item can be mapped to 0 or more output items (so *func* should return a Seq rather than a single item).

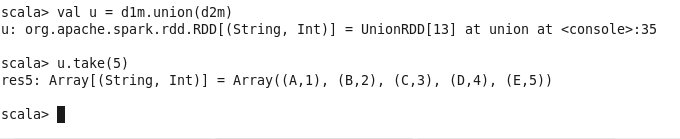
Example – Convert line into words. Mostly used for tokenize the data and transform the newly formed data type into new element.



**union(otherDataset)**

Returns a new dataset that contains the union of the elements in the source dataset and the argument.

Example – combining the data in two RDD’s and forming a new RDD



**intersection(otherDataset)**

Returns a new RDD that contains the intersection of elements in the source dataset and the argument.

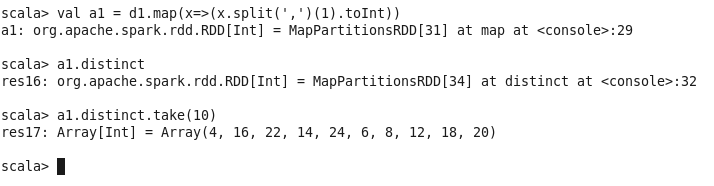
Example – Applied an intersection transformation on two RDD’s. Return value is null since the output is null.



**distinct([numTasks])**

Returns a new dataset that contains the distinct elements of the source dataset.

Note – Distinct is a costly operation as it involves network shuffling, try to avoid as much as you can



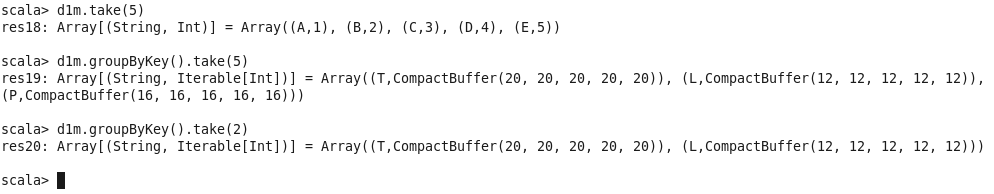
**groupByKey([numTasks])**

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

**Note** − If you are grouping in order to perform an aggregation (such as a sum or average) over each key, using reduceByKey or aggregateByKey will yield much better performance.

Example – grouped the RDD d1m by Key.

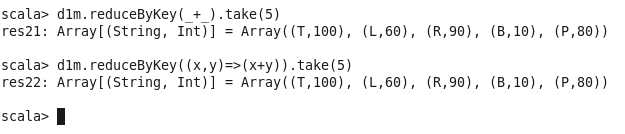
Note – It’s a very costly operation, using this transformation is highly not recommended in cluster mode.



**reduceByKey(func, [numTasks])**

When called on a dataset of (K, V) pairs, returns a dataset of (K, V) pairs where the values for each key are aggregated using the given reduce function *func*, which must be of type (V, V) ⇒ V. Like in groupByKey, the number of reduce tasks is configurable through an optional second argument.

Example – Summed up the values for each key on the pairRDD d1m. Either you can use wildcard functions or you can use usual lambda expressions. Both ways has shown below.



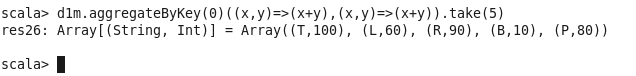
**aggregateByKey(zeroValue)(seqOp, combOp, [numTasks])**

When called on a dataset of (K, V) pairs, returns a dataset of (K, U) pairs where the values for each key are aggregated using the given combine functions and a neutral "zero" value. Allows an aggregated value type that is different from the input value type, while avoiding unnecessary allocations. Like in groupByKey, the number of reduce tasks is configurable through an optional second argument.

Example – Achieved the sum of values for each key using this transformation.

**seqOp** – Acts upon each and every partition.

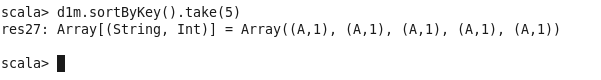
**combOp –** Comes to picture after completion of the seqOp. This will combine the values from each partitioner and displays the final output



**sortByKey([ascending], [numTasks])**

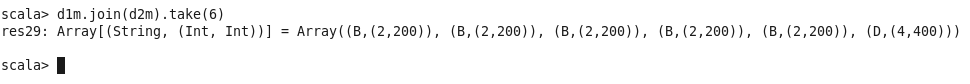
When called on a dataset of (K, V) pairs where K implements Ordered, returns a dataset of (K, V) pairs sorted by keys in ascending or descending order, as specified in the Boolean ascending argument.

Example – Sorted the data in RDD d1m based on the key.

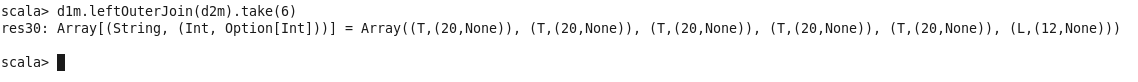


**join(otherDataset, [numTasks])**

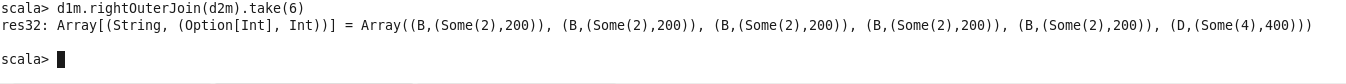
When called on datasets of type (K, V) and (K, W), returns a dataset of (K, (V, W)) pairs with all pairs of elements for each key. Outer joins are supported through leftOuterJoin, rightOuterJoin, and fullOuterJoin.



**leftOuterJoin(otherDataset, [numTasks])**

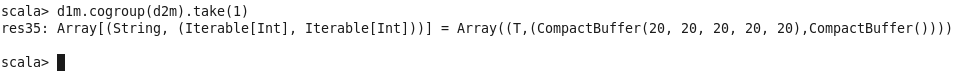
****

**rightOuterJoin(otherDataset, [numTasks])**

****

**cogroup(otherDataset, [numTasks])**

When called on datasets of type (K, V) and (K, W), returns a dataset of (K, (Iterable<V>, Iterable<W>)) tuples. This operation is also called group With.

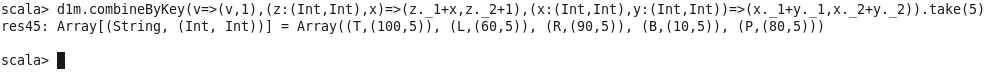
****

**combineByKey(createCombiner, mergeValue, mergeCombiner, [numTasks])**

createCombiner – Is applied only when it sees the key for the first time in a partitioner.

mergeValue – Is applied when its sees the key for more than once in a partitioner.

mergeCombiner – Merge the values of the output from each partitioner.



**Note –** It is always recommended to persist or cache after performing the transformations ending with “ByKey” or Join operations

Actions

Note – Most of the actions are already covered in the transformations section for which you may not find the examples.

**reduce(func)**

Aggregate the elements of the dataset using a function **func** (which takes two arguments and returns one). The function should be commutative and associative so that it can be computed correctly in parallel.

--Reduce is same as reduceByKey. But it is used on basicRDD.

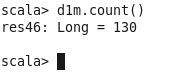
**collect()**

Returns all the elements of the dataset as an array at the driver program. This is usually useful after a filter or other operation that returns a sufficiently small subset of the data.

--Collect is used to collect the data from every node and to display it in driver node console. It is not recommended to use in cluster mode coz you may end up in out of memory issues on driver node.

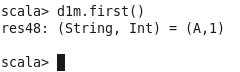
**count()**

Returns the number of elements in the dataset.



**first()**

Returns the first element of the dataset (similar to take (1)).



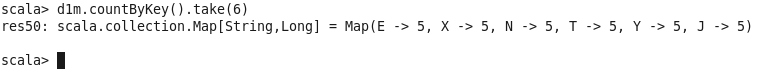
**take(n)**

Returns an array with the first **n** elements of the dataset.

--take is always recommended to use in cluster mode than using collect.

**countByKey()**

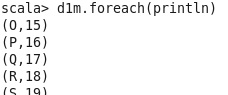
Only available on RDDs of type (K, V). Returns a hashmap of (K, Int) pairs with the count of each key.



**foreach(func)**

Runs a function **func** on each element of the dataset. This is usually, done for side effects such as updating an Accumulator or interacting with external storage systems.

**Note** − modifying variables other than Accumulators outside of the foreach() may result in undefined behavior. See Understanding closures for more details.



**saveAsTextFile(path)**

Writes the elements of the dataset as a text file (or set of text files) in a given directory in the local filesystem, HDFS or any other Hadoop-supported



