|  |  |
| --- | --- |
| **Transformation** | **Meaning** |
| **map**(*func*) | Return a new distributed dataset formed by passing each element of the source through a function *func*. |
| **filter**(*func*) | Return a new dataset formed by selecting those elements of the source on which *func*returns true. |
| **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). |
| **mapPartitions**(*func*) | Similar to map, but runs separately on each partition (block) of the RDD, so *func* must be of type Iterator<T> => Iterator<U> when running on an RDD of type T. |
| **mapPartitionsWithIndex**(*func*) | Similar to mapPartitions, but also provides *func* with an integer value representing the index of the partition, so *func* must be of type (Int, Iterator<T>) => Iterator<U> when running on an RDD of type T. |
| **sample**(*withReplacement*, *fraction*, *seed*) | Sample a fraction *fraction* of the data, with or without replacement, using a given random number generator seed. |
| **union**(*otherDataset*) | Return a new dataset that contains the union of the elements in the source dataset and the argument. |
| **intersection**(*otherDataset*) | Return a new RDD that contains the intersection of elements in the source dataset and the argument. |
| **distinct**([*numTasks*])) | Return a new dataset that contains the distinct elements of the source dataset. |
| **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.  **Note:** By default, the level of parallelism in the output depends on the number of partitions of the parent RDD. You can pass an optional numTasks argument to set a different number of tasks. |
| **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. |
| **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 than the input value type, while avoiding unnecessary allocations. Like in groupByKey, the number of reduce tasks is configurable through an optional second argument. |
| **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. |
| **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. |
| **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 groupWith. |
| **cartesian**(*otherDataset*) | When called on datasets of types T and U, returns a dataset of (T, U) pairs (all pairs of elements). |
| **pipe**(*command*, *[envVars]*) | Pipe each partition of the RDD through a shell command, e.g. a Perl or bash script. RDD elements are written to the process's stdin and lines output to its stdout are returned as an RDD of strings. |
| **coalesce**(*numPartitions*) | Decrease the number of partitions in the RDD to numPartitions. Useful for running operations more efficiently after filtering down a large dataset. |
| **repartition**(*numPartitions*) | Reshuffle the data in the RDD randomly to create either more or fewer partitions and balance it across them. This always shuffles all data over the network. |
| **repartitionAndSortWithinPartitions**(*partitioner*) | Repartition the RDD according to the given partitioner and, within each resulting partition, sort records by their keys. This is more efficient than calling repartition and then sorting within each partition because it can push the sorting down into the shuffle machinery. |