SPARK GENERAL TRANSFORMATIONS AND ACTIONS

Reference: https://spark.apache.org/docs/1.2.1/api/python/pyspark.html

Step1: Login to the sandbox.

Follow steps 4 and step 5 in the document “Setup\_sandbox.docx” to launch a session.

Step2: Start interactive session.

Run command “pyspark” to start interactive session.

**map()**

Return a new RDD by applying a function to each element of this RDD.

*>>> rdd = sc.parallelize(["b", "a", "c"])*

*>>> rdd.map(lambda x: (x, 1)).collect()*

*[('b', 1), ('a', 1), ('c', 1)]*

*>>>*

**flatMap()**

Return a new RDD by first applying a function to all elements of this RDD, and then flattening the results.

*>>> rdd = sc.parallelize([2, 3, 4])*

*>>> rdd.flatMap(lambda x: [(x, x), (x, x)]).collect()*

*[(2, 2), (2, 2), (3, 3), (3, 3), (4, 4), (4, 4)]*

*>>>*

**filter()**

Return a new RDD containing only the elements that satisfy a predicate.

*>>> rdd = sc.parallelize([1, 2, 3, 4, 5])*

*>>> rdd.filter(lambda x: x % 2 == 0).collect()*

*[2, 4]*

*>>>*

**distinct()**

Return a new RDD containing the distinct elements in this RDD

*>>> sc.parallelize([1, 1, 2, 3]).distinct().collect()*

*[Stage 73:> (0 + 0) / 2]*

*[2, 1, 3]*

*>>>*

**sample(self, withReplacement, fraction, seed=None)**

Return a sampled subset of this RDD

*>>> sc.parallelize(range(0, 100)).sample(False, 0.1, 2).collect()*

*[10, 11, 19, 61, 65, 77]*

**takeSample()**Return a fixed-size sampled subset of this RDD

*>>> rdd = sc.parallelize(range(0, 10))*

*>>> rdd.takeSample(False, 20,)*

*[8, 9, 0, 5, 3, 7, 1, 2, 6, 4]*

*>>> rdd.takeSample(False,3,)*

*[8, 1, 2]*

**union()**

Return the union of this RDD and another one.

*>>> rdd = sc.parallelize([1, 1, 2, 3])*

*>>> rdd1 = sc.parallelize([1,2,3,4,5])*

*>>> rdd.union(rdd1).collect()*

*[1, 1, 2, 3, 1, 2, 3, 4, 5]*

**intersection()**

Return the intersection of this RDD and another one.

*>>> rdd = sc.parallelize([1, 1, 2, 3])*

*>>> rdd1 = sc.parallelize([1,2,3,4,5])*

*>>> rdd1.intersection(rdd).collect()*

*[1, 2, 3]*

*>>>*

**sortByKey(self, ascending=True, numPartitions=None, f)**

Sorts this RDD, which is assumed to consist of (key, value) pairs, with key

*>>> tmp = [('9', 1), ('10', 2), ('1', 3), ('1', 4), ('2', 5)]*

*>>> tmp2 = [(9, 1), (10, 2), (1, 3), (1, 4), (2, 5)]*

*>>> sc.parallelize(tmp).sortByKey(True).collect()*

*[('1', 3), ('1', 4), ('10', 2), ('2', 5), ('9', 1)]*

*>>> sc.parallelize(tmp2).sortByKey(True).collect()*

*[(1, 3), (1, 4), (2, 5), (9, 1), (10, 2)]*

*>>>*

**sortBy()**

Sorts this RDD by the given keyfunc

*>>> tmp2 = [(9, 1), (10, 2), (1, 3), (1, 4), (2, 5)]*

*>>> sc.parallelize(tmp).sortBy(lambda x: x[0]).collect()*

*[('1', 3), ('1', 4), ('10', 2), ('2', 5), ('9', 1)]*

*>>> sc.parallelize(tmp).sortBy(lambda x: x[1]).collect()*

*[('9', 1), ('10', 2), ('1', 3), ('1', 4), ('2', 5)]*

*>>>*

**cartesian()**

Return the Cartesian product of this RDD and another one

*>>> rdd = sc.parallelize([1, 2])*

*>>> rdd.cartesian(rdd).collect()*

*[(1, 1), (1, 2), (2, 1), (2, 2)]*

**foreach()**

Applies a function to all elements of this RDD.

*>>> def f(x): print x*

*...*

*>>> sc.parallelize([1, 2, 3, 4, 5]).foreach(f)*

*1*

*2*

*3*

*4*

*5*

*>>>*

**reduce()**

Reduces the elements of this RDD using the specified commutative and associative binary operator.

*>>> from operator import add*

*>>> sc.parallelize([1, 2, 3, 4, 5]).reduce(add)*

*15*

**max()**

Find the maximum item in this RDD.

*>>> sc.parallelize([1.0, 5.0, 43.0, 10.0]).max()*

*43.0*

**min()**

Find the minimum item in this RDD.

*>>> sc.parallelize([1.0, 5.0, 43.0, 10.0]).min()*

*1.0*

**sum**

Add up the elements in this RDD.

*>>> sc.parallelize([1.0, 2.0, 3.0]).sum()*

*6.0*

**Count()**

Return the number of elements in this RDD

*>>> sc.parallelize([2, 3, 4]).count()*

*3*

**mean(self)**

Compute the mean of this RDD's elements.

*>>> sc.parallelize([1, 2, 3]).mean()*

*2.0*

**variance(self)**

Compute the variance of this RDD's elements.

*>>> sc.parallelize([1, 2, 3]).variance()*

*0.666...*

**stdev(self)**

Compute the standard deviation of this RDD's elements.

*>>> sc.parallelize([1, 2, 3]).stdev()*

*0.816...*

**sampleStdev(self)**

Compute the sample standard deviation of this RDD's elements

*>>> sc.parallelize([1, 2, 3]).sampleStdev()*

*1.0*

**sampleVariance(self)**

Compute the sample variance of this RDD's

*>>> sc.parallelize([1, 2, 3]).sampleVariance()*

*1.0*

**top(self, num)**

Get the top N elements from a RDD.

*>>> sc.parallelize([10, 4, 2, 12, 3]).top(1)*

*[12]*

*>>> sc.parallelize([2, 3, 4, 5, 6], 2).top(2)*

*[6, 5]*

**takeOrdered(self, num, key=None)**

Get the N elements from a RDD ordered in ascending order or as specified by the optional key function.

*>>> sc.parallelize([10, 1, 2, 9, 3, 4, 5, 6, 7]).takeOrdered(6)*

*[1, 2, 3, 4, 5, 6]*

*>>> sc.parallelize([10, 1, 2, 9, 3, 4, 5, 6, 7], 2).takeOrdered(6, key=lambda x: -x)*

*[10, 9, 7, 6, 5, 4]*

**take(self, num)**

Take the first num elements of the RDD.

*>>> sc.parallelize([2, 3, 4, 5, 6]).cache().take(2)*

*[2, 3]*

*>>> sc.parallelize([2, 3, 4, 5, 6]).take(10)*

*[2, 3, 4, 5, 6]*

*>>> sc.parallelize(range(100), 100).filter(lambda x: x > 90).take(3)*

*[91, 92, 93]*

**first(self)**

Return the first element in this RDD.

*>>> sc.parallelize([2, 3, 4]).first()*

*2*