Working with Key, Value Transformation

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.

**aggregateByKey()**

Works as Fold() but with more options of function on partitions and function on output of the partitions.

*>>> a = sc.parallelize(list("aaabbbcccddabcd"))*

*b = a.map(lambda x:(x,1))*

*>>> b.glom().collect()*

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

*>>> c = b.aggregateByKey("inital\_value",lambda s,d:"%s + %s" %(s,str(d)),lambda s1,s2 :"{%s + %s}" %(s1,s2),)*

*>>> c.collect()*

*[('a', '{inital\_value + 1 + 1 + 1 + inital\_value + 1}'), ('c', '{inital\_value + 1 + inital\_value + 1 + 1 + 1}'), ('b', '{inital\_value + 1 + 1 + 1 + inital\_value + 1}'), ('d', 'inital\_value + 1 + 1 + 1')]*

**collectAsMap**()

Return the key-value pairs in this RDD to the master as a dictionary.

*>>> m = sc.parallelize([(1, 2), (3, 4)]).collectAsMap()*

*>>> m[1]*

*2*

*>>> m[3]*

*4*

**countByKey**()

Count the number of elements for each key, and return the result to the master as a dictionary.

*>>> rdd = sc.parallelize([("a", 1), ("b", 1), ("a", 1)])*

*>>> rdd.countByKey()*

*defaultdict(<type 'int'>, {'a': 2, 'b': 1})*

**fullOuterJoin()**

Perform a right outer join

*>>> x = sc.parallelize([("a", 1), ("b", 4)])*

*>>> y = sc.parallelize([("a", 2), ("c", 8)])*

*>>> x.fullOuterJoin(y).collect()*

*[('a', (1, 2)), ('c', (None, 8)), ('b', (4, None))]*

**groupByKey()**

groups the (k,v) pair with key values.

*>>> x = sc.parallelize([("a", 1), ("b", 1), ("a", 1)])*

*>>> x.groupByKey().map(lambda x:(x[0],list(x[1]))).collect()*

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

*>>>*

**Join()**

Return an RDD containing all pair of elements with matching keys.

*>>> x = sc.parallelize([("a", 1), ("b", 4)])*

*>>> y = sc.parallelize([("a", 2), ("a", 3)])*

*>>> x.join(y).collect()*

*[('a', (1, 2)), ('a', (1, 3))]*

**keyBy**()

Create a tuples of the elements in this RDD by applying given function.

*>>> x = sc.parallelize(range(10))*

*>>> x.keyBy(lambda x: x\*x).collect()*

*[(0, 0), (1, 1), (4, 2), (9, 3), (16, 4), (25, 5), (36, 6), (49, 7), (64, 8), (81, 9)]*

*>>>*

**Keys()**

Return an RDD with the keys of each tuple.

*>>> x = sc.parallelize(range(10)).keyBy(lambda x : x\*x).keys()*

*>>> x.collect()*

*[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]*

>>>

**Lookup()**

Returns a value for a key. More efficient when used on sorted data.

*>>> x = sc.parallelize(range(1000)).keyBy(lambda x: x\*x)*

*>>> x.lookup(4)*

*[2]*

*>>>*

**reduceByKey()**

Merge the values for each key using a associative reduce function.

*>>> from operator import add*

*>>> rdd = sc.parallelize([("a", 1), ("b", 1), ("a", 1)])*

*>>> rdd.reduceByKey(add).collect()*

*[('a', 2), ('b', 1)]*

*>>>*

**subtractByKey()**

Return each (key, value) pair in self that has no pair with matching key in other.

*>>> x = sc.parallelize([("a", 1), ("b", 4), ("b", 5), ("a", 2)])*

*>>> y = sc.parallelize([("a", 3), ("c", None)])*

*>>> x.subtractByKey(y).collect()*

*[('b', 4), ('b', 5)]*

**Zip()**

Zips this RDD with another one, returning key-value pairs with the first element in each RDD second element in each RDD

*>>> x = sc.parallelize(range(0,5))*

*>>> y = sc.parallelize(range(1000, 1005))*

*>>> x.zip(y).collect()*

*[(0, 1000), (1, 1001), (2, 1002), (3, 1003), (4, 1004)]*

*>>>*

**Values()**

Return an RDD with the values of each tuple.

*>>> m = sc.parallelize([(1, 2), (3, 4)]).values()*

*>>> m.collect()*

*[2, 4]*

*>>>*

**mapValues()**

Pass each value in the key-value pair RDD through a map function without changing the keys

*>>> x = sc.parallelize([("a", ["apple", "banana", "lemon"]), ("b", ["grapes"])])*

*>>> x.mapValues(lambda x:len(x)).collect()*

*[('a', 3), ('b', 1)]*

*>>>*