SL-01 5.2 键值对RDD



5.2.1 键值对RDD的创建

5.2.2 常用的键值对RDD转换操作

5.2.3 一个综合实例

SL-02 5.2.1 键值对RDD的创建



1. 第一种创建方式：从文件中加载

可以采用多种方式创建键值对RDD，其中一种主要方式是使用map()函数来实现

```
>>> lines = sc.textFile("file:///usr/local/spark/mycode/pairrdd/word.txt")
>>> pairRDD = lines.flatMap(lambda line:line.split(" ")).map(lambda word:(word,1))

>>> pairRDD.foreach(print)
('I', 1)
('love', 1)
('Hadoop', 1)
......
```

SL-03 5.2.1 键值对RDD的创建



2. 第二种创建方式：通过并行集合（列表）创建RDD

```
>>> list = ["Hadoop","Spark","Hive","Spark"]
>>> rdd = sc.parallelize(list)
>>> pairRDD = rdd.map(lambda word:(word,1))

>>> pairRDD.foreach(print)
(Hadoop,1)
(Spark,1)
(Hive,1)
(Spark,1)
```

SL-04 5.2.2 常用的键值对RDD转换操作

1. reduceByKey(func)
2. groupByKey()
3. keys()
4. values()
5. sortByKey()
6. mapValues(func)
7. join()
8. combineByKey()

SL-05 5.2.2 常用的键值对RDD转换操作



1. reduceByKey(func)

reduceByKey(func)的功能是，使用func函数合并具有相同键的值

(Hadoop,1)
(Spark,1)
(Hive,1)
(Spark,1)

```
>>> pairRDD = sc.parallelize([("Hadoop",1),("Spark",1),("Hive",1),("Spark",1)])
>>> pairRDD.reduceByKey(lambda a,b:a+b).foreach(print)
('Spark', 2)
('Hive', 1)
('Hadoop', 1)
```

SL-06 5.2.2 常用的键值对RDD转换操作



2. groupByKey()

groupByKey()的功能是，对具有相同键的值进行分组

比如，对四个键值对("spark",1)、("spark",2)、("hadoop",3)和("hadoop",5)，采用groupByKey()后得到的结果是：("spark",(1,2))和("hadoop",(3,5))

```
>>> list = [("spark",1),("spark",2),("hadoop",3),("hadoop",5)]
>>> pairRDD = sc.parallelize(list)
>>> pairRDD.groupByKey()
PythonRDD[27] at RDD at PythonRDD.scala:48
>>> pairRDD.groupByKey().foreach(print)
('hadoop', <pyspark.resultiterable.ResultIterable object at 0x7f2c1093ecf8>)
('spark', <pyspark.resultiterable.ResultIterable object at 0x7f2c1093ecf8>)
```

SL-07 5.2.2 常用的键值对RDD转换操作



- **reduceByKey和groupByKey的区别**

  - reduceByKey用于对每个key对应的多个value进行merge操作，最重要的是它能够在本地先进行merge操作，并且merge操作可以通过函数自定义

  - groupByKey也是对每个key进行操作，但只生成一个sequence，groupByKey本身不能自定义函数，
    需要先用groupByKey生成RDD，然后才能对此RDD通过map进行自定义函数操作

SL-08 5.2.2 常用的键值对RDD转换操作



😊 reduceByKey和groupByKey的区别

```
>>> words = ["one", "two", "two", "three", "three", "three"]
>>> wordPairsRDD = sc.parallelize(words).map(lambda word:(word, 1))
>>> wordCountsWithReduce = wordPairsRDD.reduceByKey(lambda a,b:a+b)
>>> wordCountsWithReduce.foreach(print)
('one', 1)
('two', 2)
('three', 3)
>>> wordCountsWithGroup = wordPairsRDD.groupByKey(). ₩
... map(lambda t:(t[0],sum(t[1])))
>>> wordCountsWithGroup.foreach(print)
('two', 2)
('three', 3)
('one', 1)
```

上面得到的wordCountsWithReduce和wordCountsWithGroup是完全一样的，但是，它们的内部运算过程是不同的

SL-09 5.2.2 常用的键值对RDD转换操作



3. keys()

keys只会把Pair RDD中的key返回形成一个新的RDD

(Hadoop,1)
(Spark,1)
(Hive,1)
(Spark,1)

```
>>> list = [("Hadoop",1),("Spark",1),("Hive",1),("Spark",1)]
>>> pairRDD = sc.parallelize(list)
>>> pairRDD.keys().foreach(print)
Hadoop
Spark
Hive
Spark
```

SL-10 5.2.2 常用的键值对RDD转换操作



4. values()

values只会把Pair RDD中的value返回形成一个新的RDD。

(Hadoop,1)
(Spark,1)
(Hive,1)
(Spark,1)

```
>>> list = [("Hadoop",1),("Spark",1),("Hive",1),("Spark",1)]
>>> pairRDD = sc.parallelize(list)
>>> pairRDD.values().foreach(print)
1
1
1
1
```

SL-11 5.2.2 常用的键值对RDD转换操作



5. sortByKey()

sortByKey()的功能是返回一个根据键排序的RDD

(Hadoop,1)　(Spark,1)　(Hive,1)　(Spark,1)

```
>>> list = [("Hadoop",1),("Spark",1),("Hive",1),("Spark",1)]
>>> pairRDD = sc.parallelize(list)
>>> pairRDD.foreach(print)
('Hadoop', 1)
('Spark', 1)
('Hive', 1)
('Spark', 1)
>>> pairRDD.sortByKey().foreach(print)
('Hadoop', 1)
('Hive', 1)
('Spark', 1)
('Spark', 1)
```

SL-12 5.2.2 常用的键值对RDD转换操作



😊 sortByKey()和sortBy()

```
>>> d1 = sc.parallelize([("c",8),("b",25),("c",17),("a",42), ₩
... ("b",4),("d",9),("e",17),("c",2),("f",29),("g",21),("b",9)])
>>> print(d1.reduceByKey(lambda a,b:a+b).sortByKey(False).collect())
[('g', 21), ('f', 29), ('e', 17), ('d', 9), ('c', 27), ('b', 38), ('a', 42)]
```

```
>>> d1 = sc.parallelize([("c",8),("b",25),("c",17),("a",42), ₩
... ("b",4),("d",9),("e",17),("c",2),("f",29),("g",21),("b",9)])
>>> d1.reduceByKey(lambda a,b:a+b).sortBy(lambda x:x,False).collect()
[('g', 21), ('f', 29), ('e', 17), ('d', 9), ('c', 27), ('b', 38), ('a', 42)]
>>> d1.reduceByKey(lambda a,b:a+b).sortBy(lambda x:x[0],False).collect()
[('g', 21), ('f', 29), ('e', 17), ('d', 9), ('c', 27), ('b', 38), ('a', 42)]
>>> d1.reduceByKey(lambda a,b:a+b).sortBy(lambda x:x[1],False).collect()
[('a', 42), ('b', 38), ('f', 29), ('c', 27), ('g', 21), ('e', 17), ('d', 9)]
```

SL-13 5.2.2 常用的键值对RDD转换操作



6. mapValues(func)

对键值对RDD中的每个value都应用一个函数，但是，key不会发生变化

(Hadoop,1)
(Spark,1)
(Hive,1)
(Spark,1)

```
>>> list = [("Hadoop",1),("Spark",1),("Hive",1),("Spark",1)]
>>> pairRDD = sc.parallelize(list)
>>> pairRDD1 = pairRDD.mapValues(lambda x:x+1)
>>> pairRDD1.foreach(print)
('Hadoop', 2)
('Spark', 2)
('Hive', 2)
('Spark', 2)
```

SL-14 5.2.2 常用的键值对RDD转换操作



7. join()

join就表示内连接。
对于内连接，对于给定的两个输入数据集(K,V1)和(K,V2)，只有在两个数据集中都存在的key才会被输出，
最终得到一个(K,(V1,V2))类型的数据集。

```
>>> pairRDD1 = sc.parallelize([("spark",1),("spark",2),("hadoop",3),("hadoop",5)])
>>> pairRDD2 = sc.parallelize([("spark","fast")])
>>> pairRDD3 = pairRDD1.join(pairRDD2)
>>> pairRDD3.foreach(print)
('spark', (1, 'fast'))
('spark', (2, 'fast'))
```

SL-15 5.2.2 常用的键值对RDD转换操作



8. combineByKey

combineByKey(createCombiner,mergeValue,mergeCombiners,partitioner,mapSideCombine)中的各个参数的含义如下。

（1）createCombiner：在第一次遇到key时创建组合器函数，将RDD数据集中的V类型值转换成C类型值（V => C）；

（2）mergeValue：合并值函数，表达的是同一个分区内的计算规则，再次遇到相同的key时，将createCombiner的C类型值与这次传入的V类型值合并成一个C类型值（C,V）=>C;

（3）mergeCombiners：合并组合器函数，表达的是不同分区之间的计算规则，将C类型值两两合并成一个C类型值;

（4）partitioner：使用已有的或自定义的分区函数，默认是HashPartitioner;

（5）mapSideCombine：是否在map端进行Combine操作，默认为true。

SL-16 5.2.2 常用的键值对RDD转换操作



下面通过一个实例来解释如何使用combineByKey操作。假设有一些销售数据，数据采用键值对的形式，即<公司,当月收入>，要求使用combineByKey操作求出每个公司的总收入和每月平均收入，并保存在本地文件中。

```
from pyspark import SparkConf, SparkContext
if __name__ == '__main__':
    conf = SparkConf().setMaster("local[*]").setAppName("My App")
    sc = SparkContext(conf = conf)
    data = sc.parallelize([("company-1", 88), ("company-1", 96), ("company-1", 85), ₩
                           ("company-2", 94), ("company-2", 86), ("company-2", 74), ₩
                           ("company-3", 86), ("company-3", 88), ("company-3", 92)], 3)
    res = data.combineByKey( ₩
        lambda income: (income, 1), ₩
        lambda acc, income: (acc[0] + income, acc[1] + 1), ₩
        lambda acc1, acc2: (acc1[0] + acc2[0], acc1[1] + acc2[1])). ₩
        map(lambda x: (x[0], x[1][0], x[1][0] / float(x[1][1])))
    res.repartition(1) ₩
        .saveAsTextFile("file:///usr/local/spark/mycode/rdd/combineresult")
    sc.stop()
```

SL-17 5.2.2 常用的键值对RDD转换操作



下面解释代码的执行过程。
data = sc.parallelize()用来创建一个RDD，即data，data中的每个元素都是(key,value)键值对的形式，比如("company-1",88)和("company-1",96)。
res = data.combineByKey()语句用来计算得到每个公司的总收入和平均收入，
combineByKey()函数中使用了3个参数，
即createCombiner、mergeValue和mergeCombiners，
另外两个参数（partitioner和mapSideCombine）都采用默认值。
为了让代码中combineByKey()的参数值和参数名称之间的对应关系更加清晰，表5-3列出了二者的对应关系。

参数名称	参数值
createCombiner	lambda income:(income,1)
mergeValue	lambdaacc,income:(acc[0]+income, acc[1]+1)
mergeCombiners	lambda x:(x[0],x[1][0],x[1][0]/float(x[1][1]))

SL-18 5.2.2 常用的键值对RDD转换操作



在执行data.combineByKey()时，
首先，系统取出data中的第一个RDD元素，即("company-1",88)，
key是"company-1"，
这个key是第一次遇到，因此Spark会为这个key创建一个组合器函数createCombiner，负责把value从V类型值转换成C类型值。
这里createCombiner的值是一个Lambda表达式（或者称为匿名函数），即lambda income:(income,1)，系统会把"company-1"这个key对应的value赋值给income，也就是把88赋值给income，然后执行函数体部分，把income转换成一个元组(income,1)。
因此，88会被转换成(88,1)，从V类型值变成C类型值。然后，系统取出data中的第2个RDD元素，即("company-1",96)，key是"company-1"，这是第2次遇到相同的key，因此，系统会使用mergeValue所提供的合并值函数，将createCombiner的C类型值与这次传入的V类型值合并成一个C类型值。

SL-19 5.2.2 常用的键值对RDD转换操作



这里mergeValue参数所对应的值是一个Lambda表达式，
即lambda acc,income:(acc[0]+income, acc[1]+1)，
也就是使用这个Lambda表达式作为合并值函数，系统会把("company-1",96)中的96这个V类型值赋值给income，
把之前已经得到的(88,1)这个C类型值赋值给acc，然后执行函数体部分，
其中，acc[0]+income语句会把88和96相加，acc[1]+1语句会把(88,1)中的1增加1，得到新的C类型值(184,2)。
实际上，C类型值(184,2)中，184就是company-1这个公司两个月的收入总和，2就是表示两个月。通过这种方式，下次再扫描到一个key为"company-1"的键值对时，又会把该公司的收入累加进来，最终得到"company-1"对应的C类型值(*m*,*n*)，其中，*m*表示总收入，*n*表示月份总数，用*m*除以*n*就可以得出该公司的每月平均收入。
同理，当扫描到的RDD元素的key是"company-2"或者key是"company-3"时，系统也会执行类似上述的过程。这样，就可以得到每个公司对应的C类型值(*m*,*n*)。

SL-20 5.2.2 常用的键值对RDD转换操作



由于RDD元素被分成了多个分区，在实际应用中，多个分区可能位于不同的机器上，
因此，需要根据mergeCombiners提供的函数，对不同分区的统计结果进行汇总。
这里mergeCombiners 的值是一个Lambda表达式，即lambda acc1,acc2:(acc1[0]+acc2[0],acc1[1]+acc2[1]),
该函数的功能是把两个C类型值进行合并，得到一个C类型值。比如，
假设在一个分区里，key为"company-1"对应的统计结果是一个C类型值($m1$,$n1$),
在另一个分区里key为"company-1"对应的统计结果是一个C类型值($m2$,$n2$),
则acc1取值为($m1$,$n1$)，acc2取值为($m2$,$n2$)，acc1[0]+acc2[0]就是$m1+m2$，acc1[1]+acc2[1]就是$n1+n2$,
最终得到一个合并后的C类型值($m1+m2$,$n1+n2$)。

SL21 5.2.2 常用的键值对RDD转换操作



map(lambda x:(x[0],x[1][0],x[1][0]/float(x[1][1])))语句用来求出每个公司的总收入和平均收入。

输入到map()的每个RDD元素是类似于("company-1",(432,5))的形式，这时，x[0]就是"company-1"，x[1]就是(432,5)，x[1][0]就是432，x[1][1]就是5，因此，(x[0],x[1][0],x[1][0]/float(x[1][1]))的结果就是类似("company-1", 432, 86.4)这种形式。

最后，res.repartition(1)语句用来把RDD从3个分区变成一个分区，这样就可以保证所有生成的结果都保存到同一个文件中（即part-00000）。

SL-22 5.2.2 常用的键值对RDD转换操作



执行如下命令运行该程序:

```
$cd /usr/local/spark/mycode/rdd
$ /usr/local/spark/bin/spark-submit Combine.py
```

执行后，在“file:///usr/local/spark/mycode/rdd/combineresult”目录下可以看到part-00000文件和_SUCCESS文件（该文件可以不用考虑），part-00000文件里面包含的结果如下:

```
('company-3', 266, 88.66666666666667)
('company-1', 269, 89.66666666666667)
('company-2', 254, 84.66666666666667)
```

在Combine.py中，如果没有使用res.repartition(1)把RDD从3个分区变成1个分区，则res这个RDD还是3个分区，那么执行后在“file:///usr/local/spark/mycode/rdd/combineresult”目录下会看到part-00000、part-00001、part-00002这3个文件和_SUCCESS文件，其中，part-00000文件中包含("company-3",266,88.666667)，part-00001文件中包含("company-1",269,89.666667)，part-00002文件中包含("company-2",254,84.666667)。

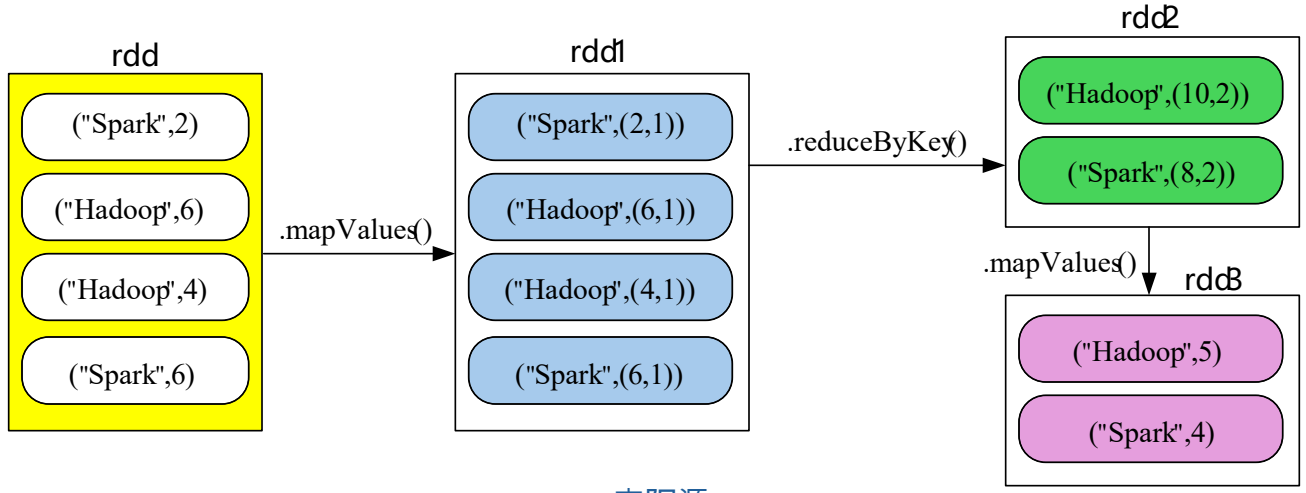
SL-78 5.2.3 一个综合实例



一个综合实例

题目：给定一组键值对("spark",2),("hadoop",6),("hadoop",4),("spark",6)，键值对的key表示图书名称，value表示某天图书销量，请计算每个键对应的平均值，也就是计算每种图书的每天平均销量。

```
>>> rdd = sc.parallelize([("spark",2),("hadoop",6),("hadoop",4),("spark",6)])
>>> rdd.mapValues(lambda x:(x,1)).⩩
... reduceByKey(lambda x,y:(x[0]+y[0],x[1]+y[1])).⩩
... mapValues(lambda x:x[0]/x[1]).collect()
[('hadoop', 5.0), ('spark', 4.0)]
```



SL-79 5.2.3 一个综合实例



如图5-20所示，rdd = sc.parallelize()执行后，生成一个名称为rdd的RDD，里面包含了4个RDD元素，即("Spark",2)、("Hadoop",6)、("Hadoop",4)、("Spark",6)。

rdd.mapValues(lambda x:(x,1))会把rdd中的每个元素依次取出来，并对该元素的value使用lambda x:(x,1)这个匿名函数进行转换。

比如，扫描到("Spark",2)这个元素时，就会把该元素的value（也就是2），转换成一个元组(2,1)，转换后得到的("Spark",(2,1))放入新的RDD（假设为rdd1）。

同理，("Hadoop",6)、("Hadoop",4)、("Spark",6)也会被分别转换成("Hadoop",(6,1))、("Hadoop",(4,1))、("Spark",(6,1))，放入到rdd1中。

SL-80 5.2.3 一个综合实例



reduceByKey(lambda x,y:(x[0]+y[0],x[1]+y[1]))会对rdd1中相同的key所对应的所有value进行聚合运算。比如，("Hadoop",(6,1))和("Hadoop",(4,1))，这两个键值对具有相同的key，
因此，reduceByKey()操作首先得到"Hadoop"这个key对应的value-list，即((6,1),(4,1))；然后，使用匿名函数lambda x,y:(x[0]+y[0],x[1]+y[1])对这个value-list进行聚合运算；
这时，Spark会把(6,1)赋值给参数x，把(4,1)赋值给参数y，因此，x._1和y._1分别是6和4，x._2和y._2都是1，(x[0]+y[0],x[1]+y[1])就是(10,2)。
如果value-list还有更多的元素，比如，假设value-list是((6,1),(4,1),(3,1))，那么，刚才计算得到的(10,2)就会作为新的x，(3,1)作为新的y，继续执行计算。
reduceByKey()操作结束后得到的新的RDD（假设为rdd2）中包含了两个元素，分别是("Hadoop",(10,2))和("Spark",(8,2))。

SL-81 5.2.3 一个综合实例



mapValues(lambda x:x[0]/x[1])操作会对rdd2中的每个元素的value执行变换，比如，当扫描到RDD中的第1个元素("Hadoop",(10,2))时，会对该元素的value（即(10,2)）进行变换。
这时，x[0]是10，x[1]是2，x[0]/x[1]就是5。
因此，经过变换后得到的结果("Hadoop",5)就会被放入新的RDD（假设为rdd3）。
最终，执行结果的RDD中就包含了("Hadoop",5)和("Spark",4)。



File Edit View Navigate Code Refactor Run Tools VCS Window Help

Ch05RDD 〉 p116_key_value_pair.py

p116_key_value_pair

Project

Ch05RDD ~/PycharmProjects/Ch05RDD

line_for_each.py
main.py
p93_parallel.py
p109_example.py
p112_differece_groupBy_reduceBy.
p113_sortBy.py
p116_key_value_pair.py
psedo_dd.py
External Libraries
Scratches and Consoles

p113_sortBy.py | p116_key_value_pair.py

```
1  from pyspark import SparkConf, SparkContext
2  if __name__=='__main__':
3      conf=SparkConf().setMaster("local").setAppName("My App")
4      sc=SparkContext(conf=conf)
5      list=[('Spark',2),('Hadoop',6),('Hadoop',4),('Spark',6)]
6      rdd=sc.parallelize(list)
7      rddmV=rdd.mapValues(lambda x:(x,1))
8      rddmV.foreach(print)
9      rddrBK=rddmV.reduceByKey(lambda x,y:(x[0]+y[0],x[1]+y[1]))
10     rddmV2=rddrBK.mapValues(lambda x:x[0]/x[1]).collect()
11     print(rddmV2)
12     sc.stop()
```

if __name__=='__main__'

Run: p116_key_value_pair

```
/home/hadoop/anaconda3/envs/nanyang/bin/python /home/hadoop/PycharmProjects/Ch05RDD/p116_key_value_pai
('Spark', (2, 1))
('Hadoop', (6, 1))
('Hadoop', (4, 1))
('Spark', (6, 1))
[('Spark', 4.0), ('Hadoop', 5.0)]
```

Version Control | Run | Python Packages | TODO | Python Console | Problems | Terminal | Services

PEP 8: W292 no newline at end of file 12:14 LF UTF-8 4 spaces