Spark 5.2 键值对RDD 강의 스크립트

슬라이드 1: 목차 소개

大家好！今天我们来学习Spark中的键值对RDD(Pair RDD)。这是一个非常重要的概念，也是Spark数据处理中最常用的数据结构之一。

我们今天的课程分为三个部分：

- 首先，我们会学习键值对RDD的创建方法
- 然后，我们会详细讲解常用的键值对RDD转换操作(transformation operations)
- 最后，我们会通过一个综合实例来巩固所学的知识

슬라이드 2: 5.2.1 键值对RDD的创建 - 第一种方式

现在我们来看第一种创建键值对RDD的方式：从文件中加载。

如大家所见，我们可以采用多种方式创建键值对RDD(Pair RDD)，其中一种主要方式是使用map()函数来实现。

让我们看这个例子：

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)

这里的执行过程是这样的：

1. 首先，我们用textFile()方法读取文件，得到一个包含文本行的RDD
2. 然后，我们用flatMap()将每一行分割成单词
3. 最后，我们用map()函数将每个单词转换成键值对的形式，比如("I", 1), ("love", 1), ("Hadoop", 1)

这就是我们最常用的创建键值对RDD的方法。

```
#coding:utf8
from pyspark import SparkConf, SparkContext
if __name__=='__main__':
    conf=SparkConf().setMaster("local").setAppName("5.2.1 pairRDD")
    sc=SparkContext(conf=conf)
    sc.setLogLevel("WARN")
    rdd=sc.textFile('file:///usr/local/spark/mycode/pairrdd/word.txt')
    pairRDD=rdd.flatMap(lambda x:x.split(" ")).map(lambda word:(word,1))
    pairRDD.foreach(print)
    sc.stop()
```

[슬라이드 3 표시] - 键值对RDD的创建 - 第二种方式

现在我们来看第二种创建方式：通过并行集合(parallelize)创建RDD。

list = ["Hadoop","Spark","Hive","Spark"]

rdd = sc.parallelize(list)

pairRDD = rdd.map(lambda word:(word,1))

pairRDD.foreach(print)

这种方式更加直接简单：

1. 我们先创建一个Python列表(list)
2. 然后用sc.parallelize()方法将列表转换成RDD
3. 最后用map()函数将每个元素转换成键值对形式

结果我们得到：(Hadoop,1), (Spark,1), (Hive,1), (Spark,1)

这种方式在测试和小规模数据处理时特别有用。

```
from pyspark import SparkConf, SparkContext
if __name__=='__main__':
    conf=SparkConf().setMaster("local").setAppName("5.2.1 pairRDD")
    sc=SparkContext(conf=conf)
    sc.setLogLevel("WARN")
    myList=['Hadoop','Spark','Hive','Spark']
    rdd=sc.parallelize(myList)
    pairRDD=rdd.map(lambda word:(word,1))
    pairRDD.foreach(print)
    sc.stop()
```

슬라이드 4: 常用的键值对RDD转换操作概览

好，现在我们进入今天的重点内容：常用的键值对RDD转换操作(transformation operations)。

我们今天要学习这些操作：

- reduceByKey(func) - 按键聚合
- groupByKey() - 按键分组
- keys - 提取所有键
- values - 提取所有值
- sortByKey() - 按键排序
- mapValues(func) - 对值进行映射
- join - 连接操作
- combineByKey - 复杂的按键组合操作

这些都是我们在实际工作中经常用到的操作。让我们一个一个来详细学习。

슬라이드 5: reduceByKey操作

首先我们来看reduceByKey(func)操作。这个操作的功能是使用func函数合并具有相同键的值

看这个例子：

pairRDD = sc.parallelize([("Hadoop",1),("Spark",1),("Hive",1),("Spark",1)])

pairRDD.reduceByKey(lambda a,b:a+b).foreach(print)

原始数据是：(Hadoop,1), (Spark,1), (Hive,1), (Spark,1)

经过reduceByKey操作后，我们得到：

- ('Spark', 2) - 因为有两个Spark，1+1=2
- ('Hive', 1) - 只有一个Hive
- ('Hadoop', 1) - 只有一个Hadoop

这个操作在词频统计中特别有用！

```
# 导入Spark相关库
from pyspark import SparkConf, SparkContext
if __name__ == '__main__':
    # 创建Spark配置（本地模式），设置应用名称
    conf = SparkConf().setMaster("local").setAppName("综合5.2.3")
    # 创建Spark上下文
    sc = SparkContext(conf=conf)
    # 设置日志级别为WARN，减少不必要的日志输出
    sc.setLogLevel('WARN')

    # 原始数据列表（键值对形式）
    MyList = [('Hadoop', 1), ('Hadoop',2),('Hadoop',5),('Spark', 1), ('Hive', 3), ('Spark', 2)]
    # 将列表转换为RDD
    rdd = sc.parallelize(MyList)
    # 按key进行reduce操作，对值进行累加
    # 例如 ('Spark', 1))和('Spark', 1) -> ('Spark', 2))
    rddrBK = rdd.reduceByKey(lambda x, y: (x+y))
    # 打印最终的平均值结果
    rddrBK.foreach(print)
    # 停止Spark上下文
    sc.stop()
```

슬라이드 6: groupByKey操作

现在我们来看groupByKey()操作。groupByKey()的功能是对具有相同键的值进行分组。

让我们看这个例子：

list = [("spark",1),("spark",2),("hadoop",3),("hadoop",5)]

pairRDD = sc.parallelize(list)

pairRDD.groupByKey().foreach(print)

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))

但是大家注意，当我们打印结果时，我们看到的是ResultIterable对象，这是因为Spark为了节省内存，使用了惰性求值(lazy evaluation)。

```
# 导入Spark相关库
from pyspark import SparkConf, SparkContext
if __name__ == '__main__':
    # 创建Spark配置（本地模式），设置应用名称
    conf = SparkConf().setMaster("local").setAppName("综合5.2.3")
    # 创建Spark上下文
    sc = SparkContext(conf=conf)
    # 设置日志级别为WARN，减少不必要的日志输出
    sc.setLogLevel('WARN')
    # 原始数据列表（键值对形式）
    MyList = [('Hadoop', 1), ('Hadoop',2),('Hadoop',5),('Spark', 1), ('Hive', 3), ('Spark', 2)]

    # 将列表转换为RDD
    rdd = sc.parallelize(MyList)
    rddgBK = rdd.groupByKey()

    # 打印最终的平均值结果
    rddgBK.foreach(print)

    rddgBK.foreach(lambda x: print(f"({x[0]}, {list(x[1])})"))

    # 停止Spark上下文
    sc.stop()
```

슬라이드 7: reduceByKey和groupByKey的区别

这里我要强调一个非常重要的概念：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。

[슬라이드 8 표시]

让我们看一个对比例子：

words = ["one", "two", "two", "three", "three", "three"]

wordPairsRDD = sc.parallelize(words).map(lambda word:(word, 1))

使用reduceByKey

wordCountsWithReduce = wordPairsRDD.reduceByKey(lambda a,b:a+b)

使用groupByKey

wordCountsWithGroup = wordPairsRDD.groupByKey().map(lambda t:(t[0],sum(t[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。从性能角度来说，reduceByKey通常更高效。

```
#coding:utf8
from pyspark import SparkConf, SparkContext
if __name__=='__main__':
    conf=SparkConf().setMaster("local").setAppName("My App")
    sc=SparkContext(conf=conf)
    words = ["one", "two", "two", "three", "three", "three"]
    wordPairRDD=sc.parallelize(words).map(lambda word:(word,1))
    wordCountWithReduce=wordPairRDD.reduceByKey(lambda a,b:a+b)
    wordCountWithReduce.foreach(print)
    wordCountsWithGroup=wordPairRDD.groupByKey().map(lambda t:(t[0],sum(t[1])))
    wordCountsWithGroup.foreach(print)
     sc.stop()
```

슬라이드 9: keys操作

现在我们来看keys操作。keys只会把Pair RDD中的key返回形成一个新的RDD。

list = [("Hadoop",1),("Spark",1),("Hive",1),("Spark",1)]

pairRDD = sc.parallelize(list)

pairRDD.keys().foreach(print)

结果我们得到： Hadoop, Spark, Hive, Spark

就是把所有的键提取出来，值被丢弃了。这个操作在我们只需要键信息时很有用。

```
# 导入Spark相关库
from pyspark import SparkConf, SparkContext

if __name__ == '__main__':
    # 创建Spark配置（本地模式），设置应用名称
    conf = SparkConf().setMaster("local").setAppName("综合5.2.3")
    # 创建Spark上下文
    sc = SparkContext(conf=conf)
    # 设置日志级别为WARN，减少不必要的日志输出
    sc.setLogLevel('WARN')

    # 原始数据列表（键值对形式）
    MyList = [('Hadoop', 1), ('Hadoop',2),('Hadoop',5),('Spark', 1), ('Hive', 3), ('Spark', 2)]

    # 将列表转换为RDD
    rdd = sc.parallelize(MyList)
    rddKeys = rdd.keys()

    # 打印最终的平均值结果
    rddKeys.foreach(print)



    sc.stop()
```

슬라이드 10: values操作

与keys操作相对应的是values操作。values只会把Pair RDD中的value返回形成一个新的RDD。

list = [("Hadoop",1),("Spark",1),("Hive",1),("Spark",1)]

pairRDD = sc.parallelize(list)

pairRDD.values().foreach(print)

结果我们得到：1, 1, 1, 1

就是把所有的值提取出来，键被丢弃了。

```
# 导入Spark相关库
from pyspark import SparkConf, SparkContext

if __name__ == '__main__':
    # 创建Spark配置（本地模式），设置应用名称
    conf = SparkConf().setMaster("local").setAppName("综合5.2.3")
    # 创建Spark上下文
    sc = SparkContext(conf=conf)
    # 设置日志级别为WARN，减少不必要的日志输出
    sc.setLogLevel('WARN')

    # 原始数据列表（键值对形式）
    MyList = [('Hadoop', 1), ('Hadoop',2),('Hadoop',5),('Spark', 1), ('Hive', 3), ('Spark', 2)]

    # 将列表转换为RDD
    rdd = sc.parallelize(MyList)
    rddValues = rdd.values()

    # 打印最终的平均值结果
    rddValues.foreach(print)



    sc.stop()
```

슬라이드 11: sortByKey操作

sortByKey()的功能是返回一个根据键排序的RDD。

list = [("Hadoop",1),("Spark",1),("Hive",1),("Spark",1)]

pairRDD = sc.parallelize(list)

pairRDD.sortByKey().foreach(print)

原始顺序：(Hadoop,1), (Spark,1), (Hive,1), (Spark,1) 排序后：(Hadoop,1), (Hive,1), (Spark,1), (Spark,1)

可以看到按照键的字母顺序进行了排序。

```
# 导入Spark相关库
from pyspark import SparkConf, SparkContext

if __name__ == '__main__':
    # 创建Spark配置（本地模式），设置应用名称
    conf = SparkConf().setMaster("local").setAppName("综合5.2.3")
    # 创建Spark上下文
    sc = SparkContext(conf=conf)
    # 设置日志级别为WARN，减少不必要的日志输出
    sc.setLogLevel('WARN')

    # 原始数据列表（键值对形式）
    MyList = [('Hadoop', 1), ('Hadoop',2),('Hadoop',5),('Spark', 1), ('Hive', 3), ('Spark', 2)]

    # 将列表转换为RDD
    rdd = sc.parallelize(MyList)
    rddsBK = rdd.sortByKey()

    # 打印最终的平均值结果
    rddsBK.foreach(print)



    sc.stop()
```

[슬라이드 12 표시]

我们还可以比较sortByKey()和sortBy()的区别：

- sortByKey(False) - 按键降序排列
- sortBy(lambda x:x[0], False) - 按第一个元素（键）降序排列
- sortBy(lambda x:x[1], False) - 按第二个元素（值）降序排列

sortBy更加灵活，可以指定按什么来排序。

```
# 导入Spark相关库
from pyspark import SparkConf, SparkContext

if __name__ == '__main__':
    # 创建Spark配置（本地模式），设置应用名称
    conf = SparkConf().setMaster("local").setAppName("综合5.2.3")
    # 创建Spark上下文
    sc = SparkContext(conf=conf)
    # 设置日志级别为WARN，减少不必要的日志输出
    sc.setLogLevel('WARN')

    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())
    print(d1.reduceByKey(lambda a, b: a + b).sortBy(lambda x: x, False).collect())
    print(d1.reduceByKey(lambda a, b: a + b).sortBy(lambda x: x[0], False).collect())
    print(d1.reduceByKey(lambda a, b: a + b).sortBy(lambda x: x[1], False).collect())

    sc.stop()
```

슬라이드 13: mapValues操作

mapValues(func)操作对键值对RDD中的每个value都应用一个函数，但是key不会发生变化。

list = [("Hadoop",1),("Spark",1),("Hive",1),("Spark",1)]

pairRDD = sc.parallelize(list)

pairRDD1 = pairRDD.mapValues(lambda x:x+1)

pairRDD1.foreach(print)

原始：(Hadoop,1), (Spark,1), (Hive,1), (Spark,1) 结果：(Hadoop,2), (Spark,2), (Hive,2), (Spark,2)

可以看到，键保持不变，但每个值都加了1。这个操作在我们需要对值进行统一变换时很有用。

```
# 导入Spark相关库
from pyspark import SparkConf, SparkContext

if __name__ == '__main__':
    # 创建Spark配置（本地模式），设置应用名称
    conf = SparkConf().setMaster("local").setAppName("综合5.2.3")
    # 创建Spark上下文
    sc = SparkContext(conf=conf)
    # 设置日志级别为WARN，减少不必要的日志输出
    sc.setLogLevel('WARN')

    # 原始数据列表（键值对形式）
    MyList = [('Hadoop', 1),('Spark', 1), ('Hive', 3), ('Spark', 2)]

    # 将列表转换为RDD
    rdd = sc.parallelize(MyList)
    pairRDD = rdd.mapValues(lambda x:x+1)

    # 打印最终的平均值结果
    pairRDD.foreach(print)

    sc.stop()
```

슬라이드 14: join操作

join操作表示内连接(inner join)。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)

结果： ('spark', (1, 'fast')) ('spark', (2, 'fast'))

注意，只有spark这个键在两个RDD中都存在，所以只有spark相关的记录被连接了。hadoop因为在pairRDD2中不存在，所以被过滤掉了。

```
# 导入Spark相关库
from pyspark import SparkConf, SparkContext

if __name__ == '__main__':
    # 创建Spark配置（本地模式），设置应用名称
    conf = SparkConf().setMaster("local").setAppName("综合5.2.3")
    # 创建Spark上下文
    sc = SparkContext(conf=conf)
    # 设置日志级别为WARN，减少不必要的日志输出
    sc.setLogLevel('WARN')

    # 原始数据列表（键值对形式）
    MyList = [('Hadoop', 1),('Spark', 1), ('Hive', 3), ('Spark', 2)]
    SecondList = [('Spark', 'fast') ]

    # 将列表转换为RDD
    rdd = sc.parallelize(MyList)
    rdd2=sc.parallelize(SecondList)
    pairRDD = rdd.join(rdd2)

    # 打印最终的平均值结果
    pairRDD.foreach(print)
    sc.stop()
```

슬라이드 15: combineByKey操作介绍

现在我们来学习最复杂但也最强大的操作：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

[슬라이드 16 표시]

让我们通过一个实际例子来理解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。

```
# 导入Spark相关库
from pyspark import SparkConf, SparkContext
import shutil
import os

if __name__ == '__main__':
    # 创建Spark配置（本地模式）
    conf = SparkConf().setMaster("local").setAppName("My App")
    sc = SparkContext(conf=conf)

    # 原始数据集：公司收入信息（公司名，收入）
    companyInfo = [('Samsung', 100), ('Samsung', 200), ('Samsung', 300),
                   ('Huawei', 80), ('Huawei', 90), ('Huawei', 100),
                   ('Shaomi', 50), ('Shaomi', 60), ('Shaomi', 70)]

    # 创建RDD（3个分区）
    rdd = sc.parallelize(companyInfo, 3)


    # 定义分区打印函数（调试用）
    def print_partition(iterator):
        print(list(iterator))


    print("Original RDD partitions:")
    rdd.foreachPartition(print_partition)

    # 使用combineByKey转换操作：
    # 1. 创建组合器（收入值，计数器）
    # 2. 分区内合并
    # 3. 分区间合并
    res = rdd.combineByKey(
        lambda income: (income, 1),  # 初始值格式（值,1）
```

```
        lambda acc, income: (acc[0] + income, acc[1] + 1),  # 分区内聚合
        lambda acc1, acc2: (acc1[0] + acc2[0], acc1[1] + acc2[1])  # 分区间合并
)

print("₩ncombineByKey results:")
print(res.collect())

# 计算平均收入（公司名，总收入，平均收入）
avg_res = res.map(lambda x: (x[0], x[1][0], x[1][0] / x[1][1]))

print("₩nAverage results:")
print(avg_res.collect())

# 首次输出结果路径
output_path = 'file:///usr/local/spark/mycode/rdd/combinerresult'

# 确保输出目录不存在（本地模式）
if os.path.exists('/usr/local/spark/mycode/rdd/combinerresult'):
    shutil.rmtree('/usr/local/spark/mycode/rdd/combinerresult')

avg_res.saveAsTextFile(output_path)

# 重新分区为1个并保存最终结果
print("₩nRepartiton result:")
final_res = avg_res.repartition(1)

# 最终输出路径（与首次输出不同）
if os.path.exists('/usr/local/spark/mycode/rdd/combinerresult'):
    shutil.rmtree('/usr/local/spark/mycode/rdd/combinerresult')

final_res.saveAsTextFile(output_path)
final_res.foreach(print)

# 关闭Spark上下文
sc.stop()
```

슬라이드 17: combineByKey执行过程详解

让我详细解释一下combineByKey的执行过程：

1. **第一步**：当系统遇到第一个("company-1", 88)时，key是"company-1"，这是第一次遇到这个key，所以Spark会调用createCombiner函数，把88转换成(88,1)
2. **第二步**：当遇到第二个("company-1", 96)时，这是第二次遇到相同的key，所以使用mergeValue函数，将之前的(88,1)和新的96合并成(184,2)
3. **第三步**：如果RDD有多个分区，mergeCombiners函数会将不同分区的结果合并
4. **最后一步**：map函数计算每个公司的总收入和平均收入

执行结果：

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

其中第二个数字是总收入，第三个数字是平均收入。

슬라이드 23: 综合实例

最后，让我们通过一个综合实例来巩固今天学到的知识。

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()
```

结果：[('hadoop', 5.0), ('spark', 4.0)]

슬라이드 24: 综合实例执行过程

让我们详细分析这个执行过程：

1. **mapValues操作**：将("Spark",2)转换成("Spark",(2,1))，其他元素同理
2. **reduceByKey操作**：
    - 对于"Hadoop"：(6,1)和(4,1)合并成(10,2)
    - 对于"Spark"：(2,1)和(6,1)合并成(8,2)
3. **mapValues操作**：计算平均值
    - "Hadoop": 10/2 = 5.0
    - "Spark": 8/2 = 4.0

这个例子很好地展示了如何组合使用多个键值对RDD操作来解决实际问题。

```
# 导入Spark相关库
from pyspark import SparkConf, SparkContext

if __name__ == '__main__':
    # 创建Spark配置（本地模式），设置应用名称
    conf = SparkConf().setMaster("local").setAppName("综合5.2.3")
    # 创建Spark上下文
    sc = SparkContext(conf=conf)
    # 设置日志级别为WARN，减少不必要的日志输出
    sc.setLogLevel('WARN')

    # 原始数据列表（键值对形式）
    MyList = [('Spark', 2), ('Hadoop', 6), ('Hadoop', 4), ('Spark', 6)]

    # 将列表转换为RDD
    rdd = sc.parallelize(MyList)

    # 对值进行映射转换，将每个值转换为元组（原值，1）
    # 例如 ('Spark',2) -> ('Spark', (2,1))
    rddmV = rdd.mapValues(lambda x: (x, 1))
```

```
# 打印转换后的结果
rddmV.foreach(print)

# 按key进行reduce操作，对值进行累加
# 例如 ('Spark', (2,1))和('Spark', (6,1)) -> ('Spark', (8,2))
rddrBK = rddmV.reduceByKey(lambda x, y: (x[0] + y[0], x[1] + y[1]))

# 计算每个key的平均值（总和/计数）
# 例如 ('Spark', (8,2)) -> ('Spark', 4.0)
rddmV2 = rddrBK.mapValues(lambda x: x[0] / x[1]).collect()

# 打印最终的平均值结果
print(rddmV2)

# 停止Spark上下文
sc.stop()
```

总结

今天我们学习了Spark中键值对RDD的创建和常用操作：

1. **创建方式**：从文件加载和并行集合创建
2. **转换操作**：reduceByKey, groupByKey, keys, values, sortByKey, mapValues, join, combineByKey
3. **实际应用**：通过综合实例学会了如何组合使用这些操作

键值对RDD是Spark中最重要的数据结构之一，掌握这些操作对于进行大数据处理非常重要。希望大家课后多练习，熟练掌握这些操作的使用方法。

有什么问题吗？