5.4 综合案例

5.4.1 案例1：求TOP值

5.4.2 案例2：文件排序

5.4.3 案例3：二次排序

5.4.1 案例1：求TOP值



1. 实例1

任务描述：

orderid,userid,payment,productid

file1.txt

```
1,1768,50,155
2,1218, 600,211
3,2239,788,242
4,3101,28,599
5,4899,290,129
6,3110,54,1201
7,4436,259,877
8,2369,7890,27
```

file2.txt

```
100,4287,226,233
101,6562,489,124
102,1124,33,17
103,3267,159,179
104,4569,57,125
105,1438,37,116
```

求Top N个payment值

5.4.1 案例1：求TOP值



TopN.py

```
#!/usr/bin/env python3

from pyspark import SparkConf, SparkContext

conf = SparkConf().setMaster("local").setAppName("ReadHBase")
sc = SparkContext(conf = conf)
lines = sc.textFile("file:///usr/local/spark/mycode/rdd/file")
result1 = lines.filter(lambda line:(len(line.strip()) > 0) and (len(line.split(","))== 4))
result2 = result1.map(lambda x:x.split(",")[2])
result3 = result2.map(lambda x:(int(x),""))
result4 = result3.repartition(1)
result5 = result4.sortByKey(False)
result6 = result5.map(lambda x:x[0])
result7 = result6.take(5)
for a in result7:
    print(a)
```

Ch05RDD 〉 p121_Top.py

p121_Top

Project

Ch05RDD ~/PycharmProjects/Ch05

- line_for_each.py
- main.py
- p93_parallel.py
- p109_example.py
- p112_differece_groupBy_reduce
- p113_sortBy.py
- p116_key_value_pair.py
- p117_read_data.py
- p119_MySQL.py
- p120_SparkWriteMySQL.py
- p121_Top.py
- psedo_dd.py

External Libraries

Scratches and Consoles

p112_differece_groupBy_reduceBy.py | p109_example.py | p119_MySQL.py | p120_SparkWriteMySQL.py | p121_Top.py

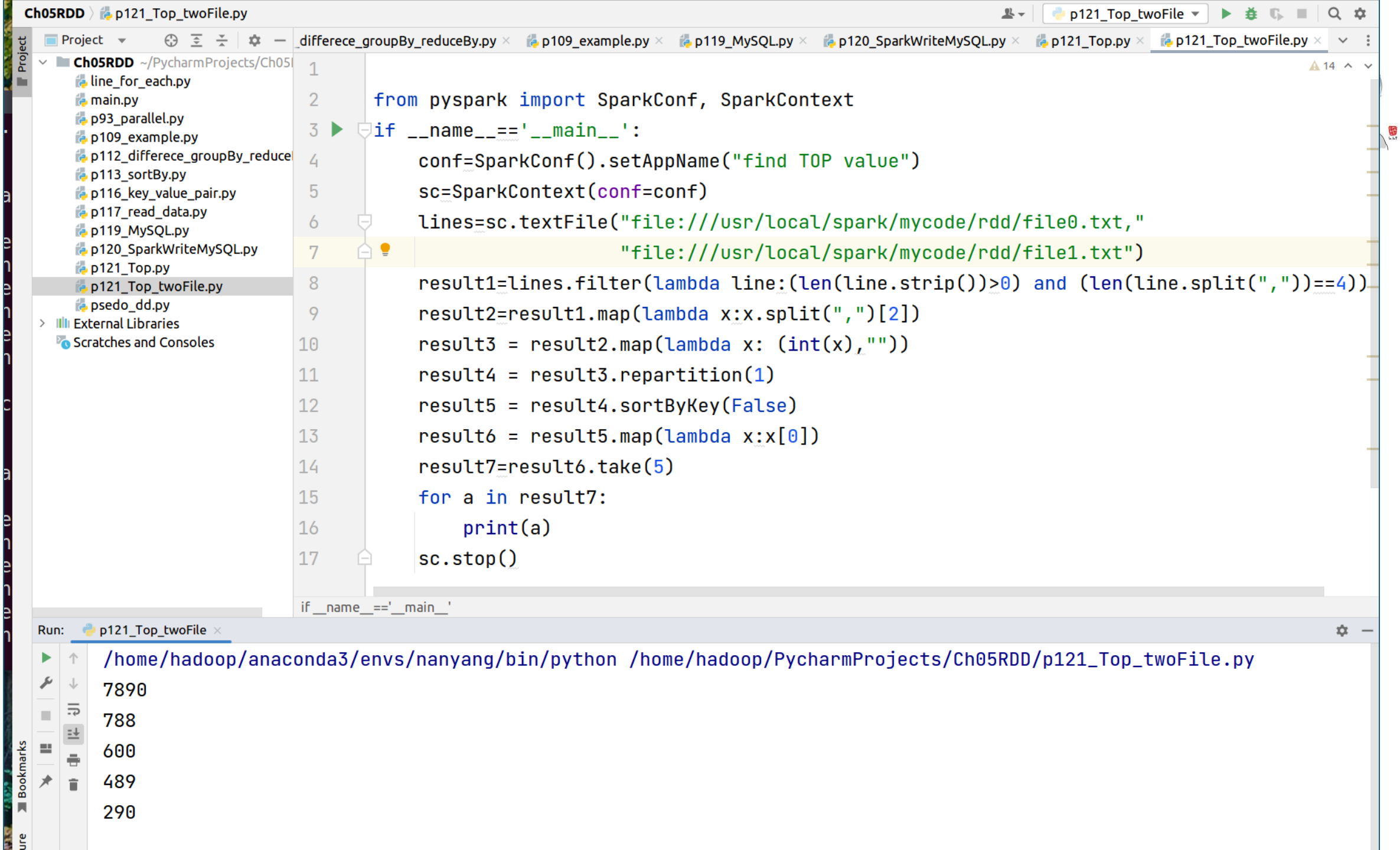
```
1  from pyspark import SparkConf, SparkContext
2  if __name__=='__main__':
3      conf=SparkConf().setAppName("find TOP value")
4      sc=SparkContext(conf=conf)
5      lines=sc.textFile("file:///usr/local/spark/mycode/rdd/file0.txt")
6      result1=lines.filter(lambda line:(len(line.strip())>0) and (len(line.split(","))==4))
7      result2=result1.map(lambda x:x.split(",")[2])
8      result3 = result2.map(lambda x: (int(x),""))
9      result4 = result3.repartition((1))
10     result5 = result4.sortByKey(False)
11     result6 = result5.map(lambda x:x[0])
12     result7=result6.take(5)
13     for a in result7:
14         print(a)
15     sc.stop()
```

Run: p121_Top

```
/home/hadoop/anaconda3/envs/nanyang/bin/python /home/hadoop/PycharmProjects/Ch05RDD/p121_Top.py
7890
788
600
290
259

Process finished with exit code 0
```

Bookmarks

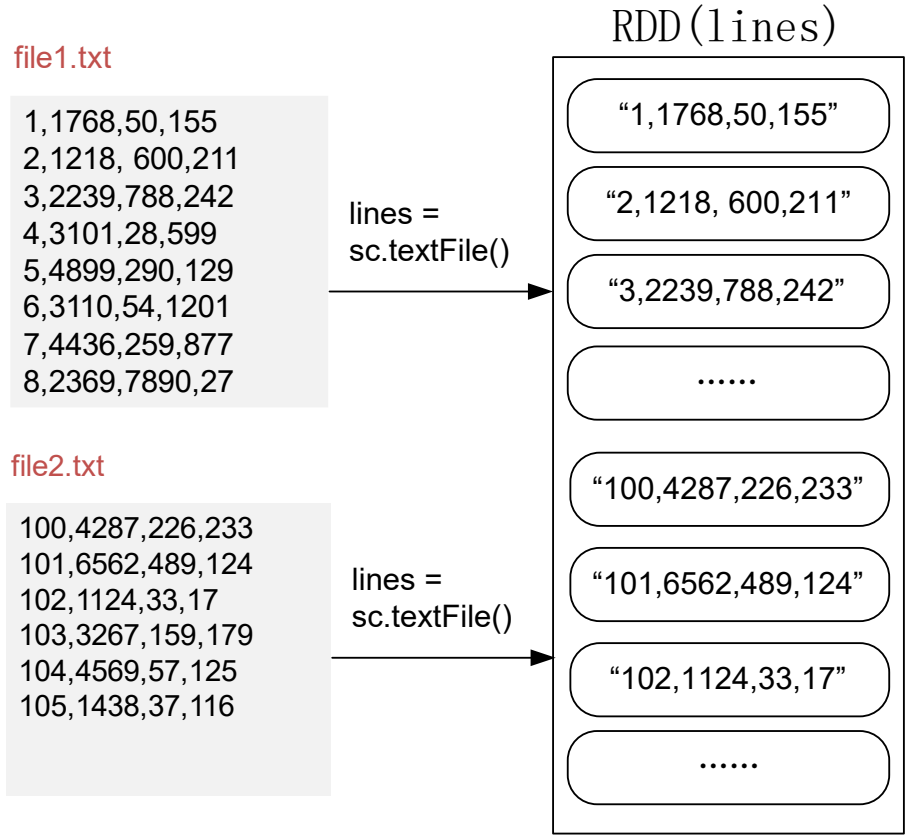

Ch05RDD
p121_Top_twoFile.py
Project
Ch05RDD ~/PycharmProjects/Ch05
line_for_each.py
main.py
p93_parallel.py
p109_example.py
p112_differece_groupBy_reduce
p113_sortBy.py
p116_key_value_pair.py
p117_read_data.py
p119_MySQL.py
p120_SparkWriteMySQL.py
p121_Top.py
p121_Top_twoFile.py
psedo_dd.py
External Libraries
Scratches and Consoles
differece_groupBy_reduceBy.py
p109_example.py
p119_MySQL.py
p120_SparkWriteMySQL.py
p121_Top.py
p121_Top_twoFile.py
14
from pyspark import SparkConf, SparkContext
if __name__=='__main__':
conf=SparkConf().setAppName("find TOP value")
sc=SparkContext(conf=conf)
lines=sc.textFile("file:///usr/local/spark/mycode/rdd/file0.txt,"
"file:///usr/local/spark/mycode/rdd/file1.txt")
result1=lines.filter(lambda line:(len(line.strip())>0) and (len(line.split(","))==4))
result2=result1.map(lambda x:x.split(",")[2])
result3 = result2.map(lambda x: (int(x),""))
result4 = result3.repartition(1)
result5 = result4.sortByKey(False)
result6 = result5.map(lambda x:x[0])
result7=result6.take(5)
for a in result7:
print(a)
sc.stop()
if __name__=='__main__'
Run:
p121_Top_twoFile
/home/hadoop/anaconda3/envs/nanyang/bin/python /home/hadoop/PycharmProjects/Ch05RDD/p121_Top_twoFile.py
7890
788
600
489
290
Bookmarks

5.4.1 案例1：求TOP值



```
lines = sc.textFile("file:///usr/local/spark/mycode/rdd/file")
```

该语句从文件中读取数据生成RDD（名称为lines），执行后的效果如下：

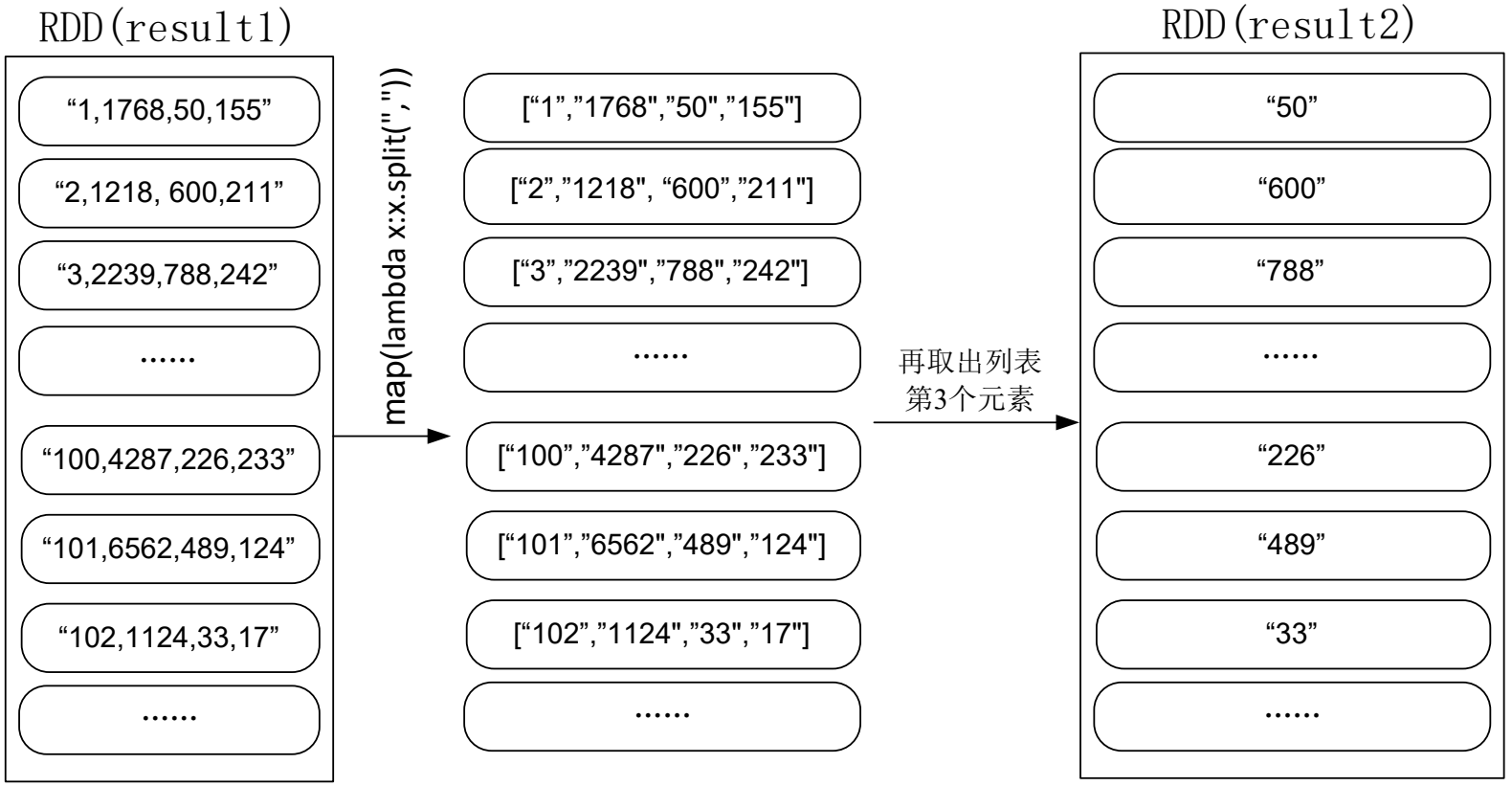


5.4.1 案例1：求TOP值



```
result1 = lines.filter(lambda line:(len(line.strip()) > 0) and (len(line.split(","))== 4))
result2 = result1.map(lambda x:x.split(",")[2])
```

该语句执行效果如下：

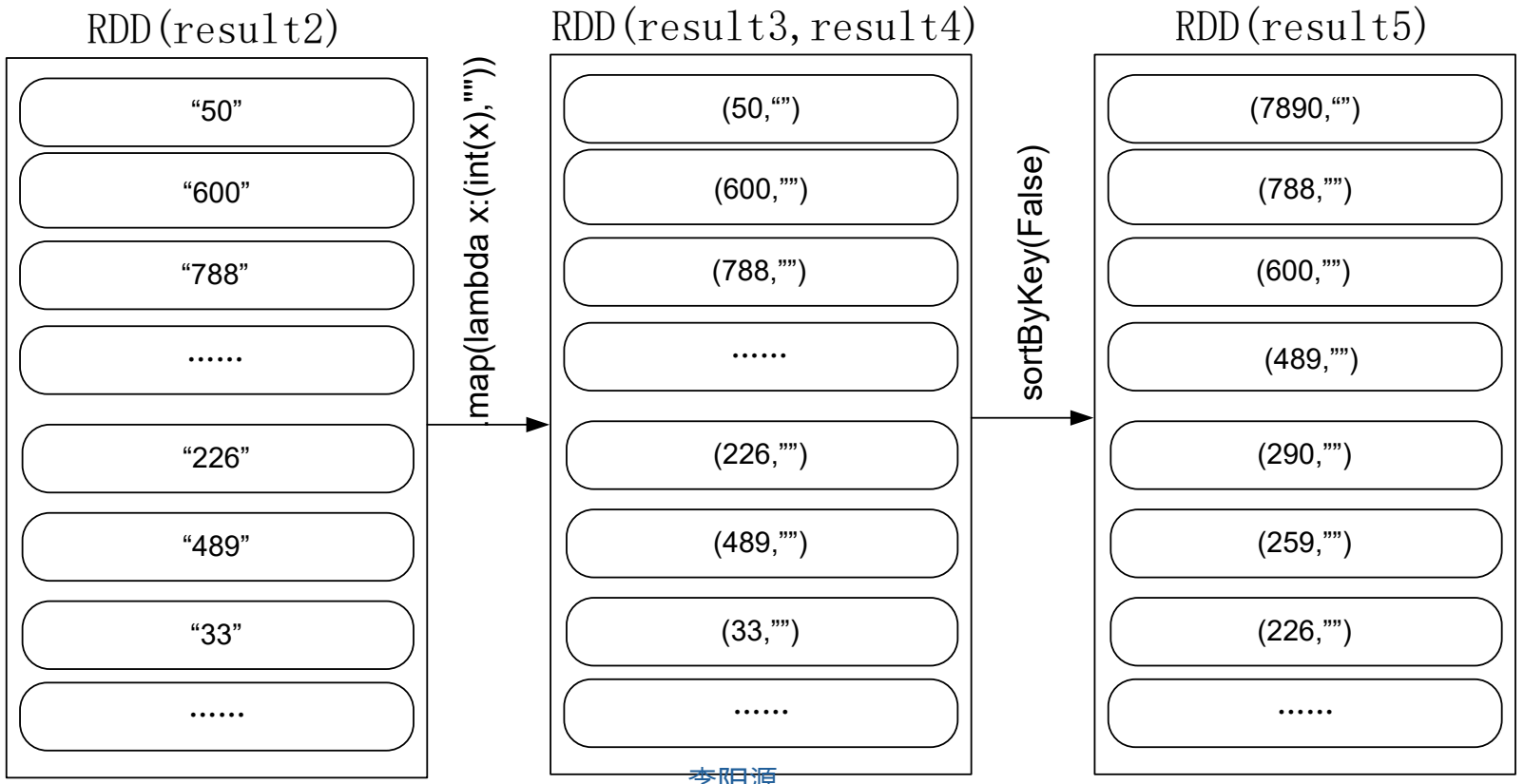


5.4.1 案例1：求TOP值



```
result3 = result2.map(lambda x:(int(x),""))
result4 = result3.repartition(1)
result5 = result4.sortByKey(False)
```

该语句执行效果如下：

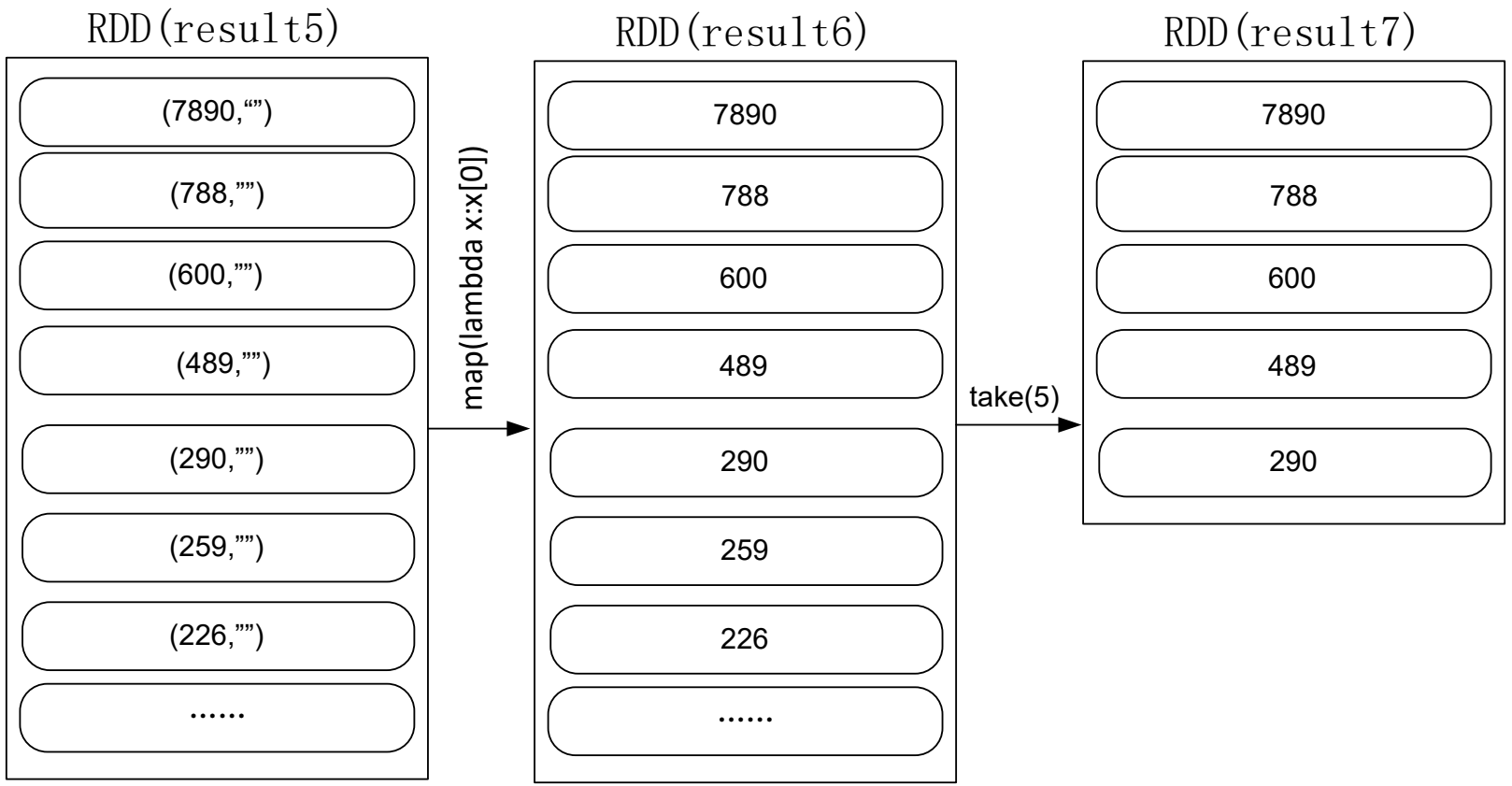


5.4.1 案例1：求TOP值



```
result6 = result5.map(lambda x:x[0])
result7 = result6.take(5)
```

该语句执行效果如下：



5.4.1 案例1：求TOP值



2. 实例2

有一个商品销售数据集sales.txt，里面的每行记录包含5个字段：时间戳、省份、城市、用户、购买的商品，字段之间用英文逗号隔开，下面是某条数据的样例：

```
1602031567000,21,1005,95001,20001
```

在这条数据中，
“1602031567000”表示时间戳，
“21”表示福建省，
“1005”表示厦门市，
“95001”表示顾客王明，
“20001”表示华为Mate60手机。

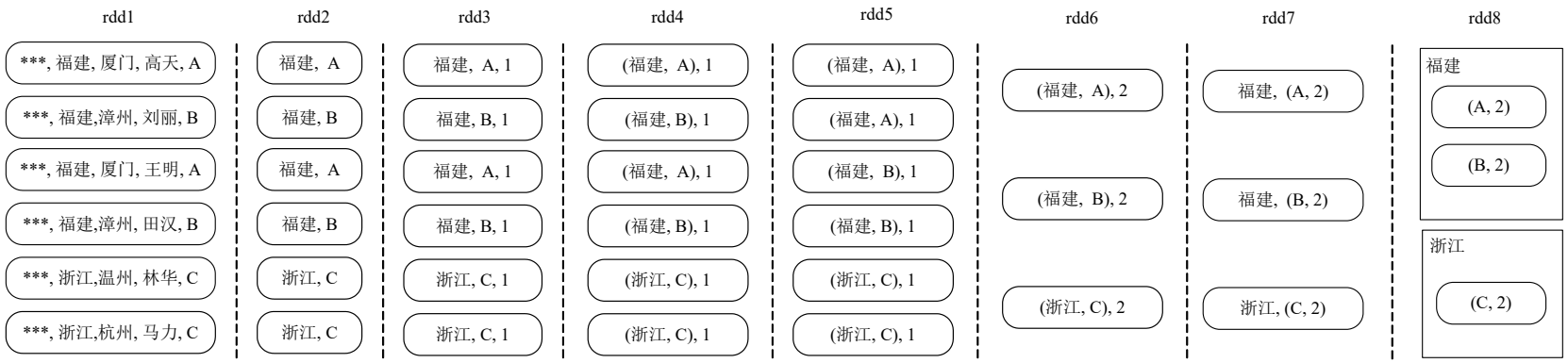
要求求出每个省份销量前3名的商品。

5.4.1 案例1：求TOP值



- 图5-25给出了结题思路，为了方便理解，这里直接使用了省份名字、城市名字、用户名字和商品名称，而不是使用编号，图中的“***”表示时间戳。
- 首先从sales.txt中加载数据生成rdd1；然后，过滤出需要的字段，即省份和商品，得到rdd2；
- 再把rdd2中的每个元素从(省份,商品)转换成(省份,商品,1)，得到rdd3；再把每个元素中的省份和商品名称组合在一起，得到rdd4；
- 再根据(省份,商品)进行分组，得到rdd5；
- 再对每个分组进行汇总，得到rdd6；再把每个元素从((省份,商品),销量))转换成(省份,(商品,销量))，得到rdd7；
- 再对元素按照省份进行分组，得到rdd8；最后，对元素进行排序，求出每个省份销量前3名的商品。

实际上，在编写程序时，可以更加精炼，也就是，可以直接从rdd1转换得到rdd4。



5.4.1 案例1：求TOP值



```
# coding:utf8
from pyspark import SparkConf, SparkContext
if __name__ == '__main__':
    conf = SparkConf().setMaster("local[*]").setAppName("My App")
    sc = SparkContext(conf = conf)
    # 1.获取原始数据：时间戳，省份，城市，用户，商品
    sourceRDD = sc.textFile("file:///home/hadoop/sales.txt")
    dataRDD = sourceRDD.filter(lambda line : len(line.split(",")) == 5)
```

5.4.1 案例1：求TOP值



```
# 2.将原始数据进行结构转换
    # 把数据从(时间戳, 省份, 城市, 用户, 商品)
    # 转变成((省份, 商品), 1)
    def myMapFunction1(line):
        datas = line.split(",")
        return ((datas[1], datas[4]), 1)
    mapRDD = dataRDD.map(myMapFunction1)
    # 3.将数据进行分组聚合
    # 得到((省份, 商品), sum)形式的结果，sum是汇总结果
    reduceRDD = mapRDD.reduceByKey(lambda a,b:a+b)
```

5.4.1 案例1：求TOP值



```
# 4.将聚合的结果进行数据结构转换
 # 从((省份, 商品), sum)转变成(省份, (商品, sum))
 def myMapFunction2(tuple):
     x = tuple[0] # 值是(省份, 商品)
     y = tuple[1] # 值是sum
     return (x[0], (x[1],y))
 newMapRDD = reduceRDD.map(myMapFunction2)
 # 5.根据省份进行分组
 # 得到的数据形式是(省份, ((商品1, sum1), (商品2, sum2), ...))
 groupRDD = newMapRDD.groupByKey()
```

5.4.1 案例1：求TOP值



```
    # 6.将分组后的数据进行组内排序（降序），取前3名
    resultRDD = groupRDD.mapValues(lambda iter: sorted(list(iter), ke
y=lambda tup: tup[1],reverse=True)[:3])
    # 7.把数据打印出到控制台
    resultRDD.foreach(print)
    sc.stop()
```

程序的运行结果如下：

```
('21', [('20002', 3), ('20001', 2), ('20003', 1)])
('20', [('20002', 4), ('20001', 2), ('20003', 2)])
```

5.4.2 案例2：文件排序



任务描述：
有多个输入文件，每个文件中的每一行内容均为一个整数。要求读取所有文件中的整数，进行排序后，输出到一个新的文件中，输出的内容个数为每行两个整数，第一个整数为第二个整数的排序位次，第二个整数为原待排序的整数

输入文件

file1.txt	file2.txt	file3.txt
33	4	1
37	16	45
12	39	25
40	5	

输出文件

1	1
2	4
3	5
4	12
5	16
6	25
7	33
8	37
9	39
10	40
11	45

5.4.2 案例2：文件排序



FileSort.py

```
#!/usr/bin/env python3

from pyspark import SparkConf, SparkContext

index = 0

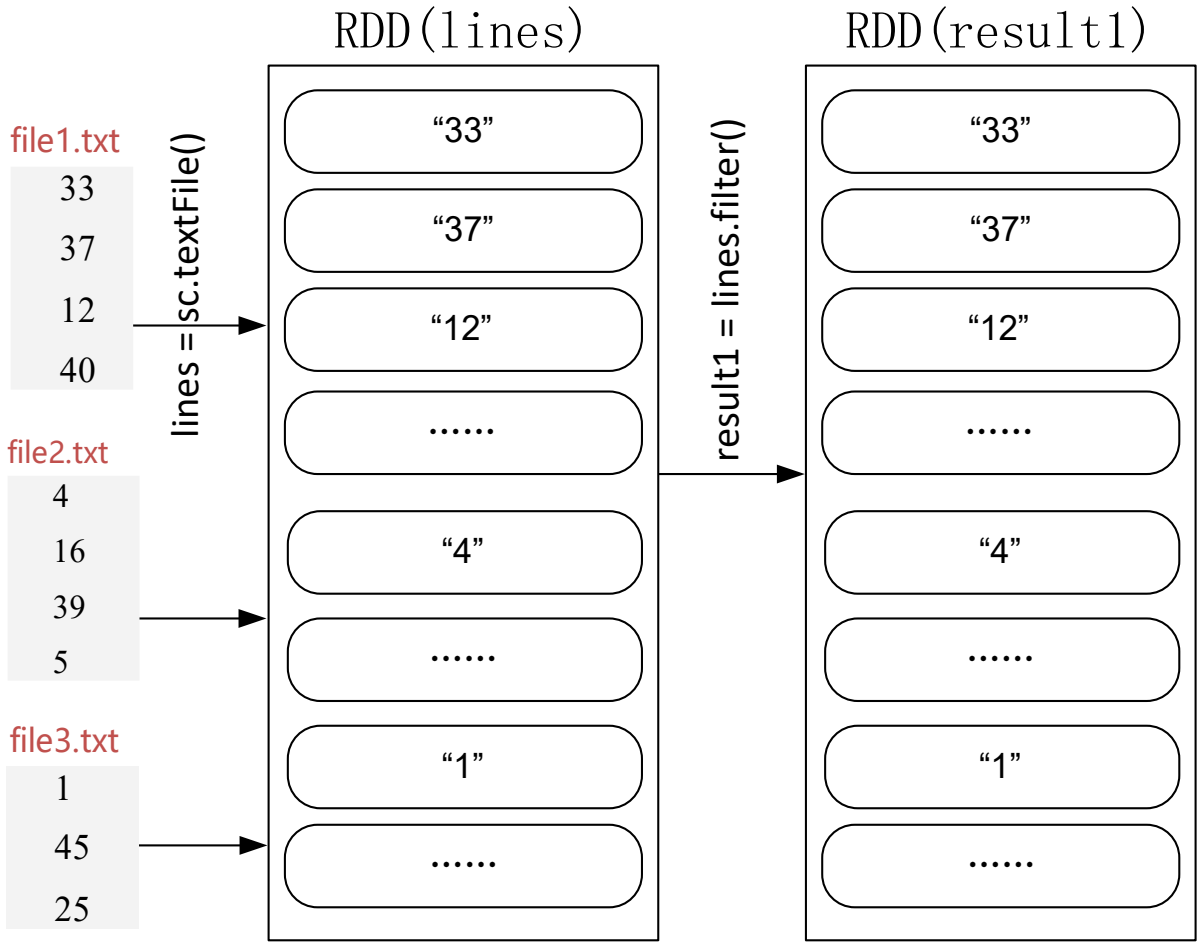
def getindex():
    global index
    index+=1
    return index

def main():
    conf = SparkConf().setMaster("local[1]").setAppName("FileSort")
    sc = SparkContext(conf = conf)
    lines = sc.textFile("file:///usr/local/spark/mycode/rdd/filesort/file*.txt")
    index = 0
    result1 = lines.filter(lambda line:(len(line.strip()) > 0))
    result2 = result1.map(lambda x:(int(x.strip()),""))
    result3 = result2.repartition(1)
    result4 = result3.sortByKey(True)
    result5 = result4.map(lambda x:x[0])
    result6 = result5.map(lambda x:(getindex(),x))
    result6.foreach(print)
    result6.saveAsTextFile("file:///usr/local/spark/mycode/rdd/filesort/sortresult")
if __name__ == '__main__':
    main()
```

5.4.2 案例2：文件排序



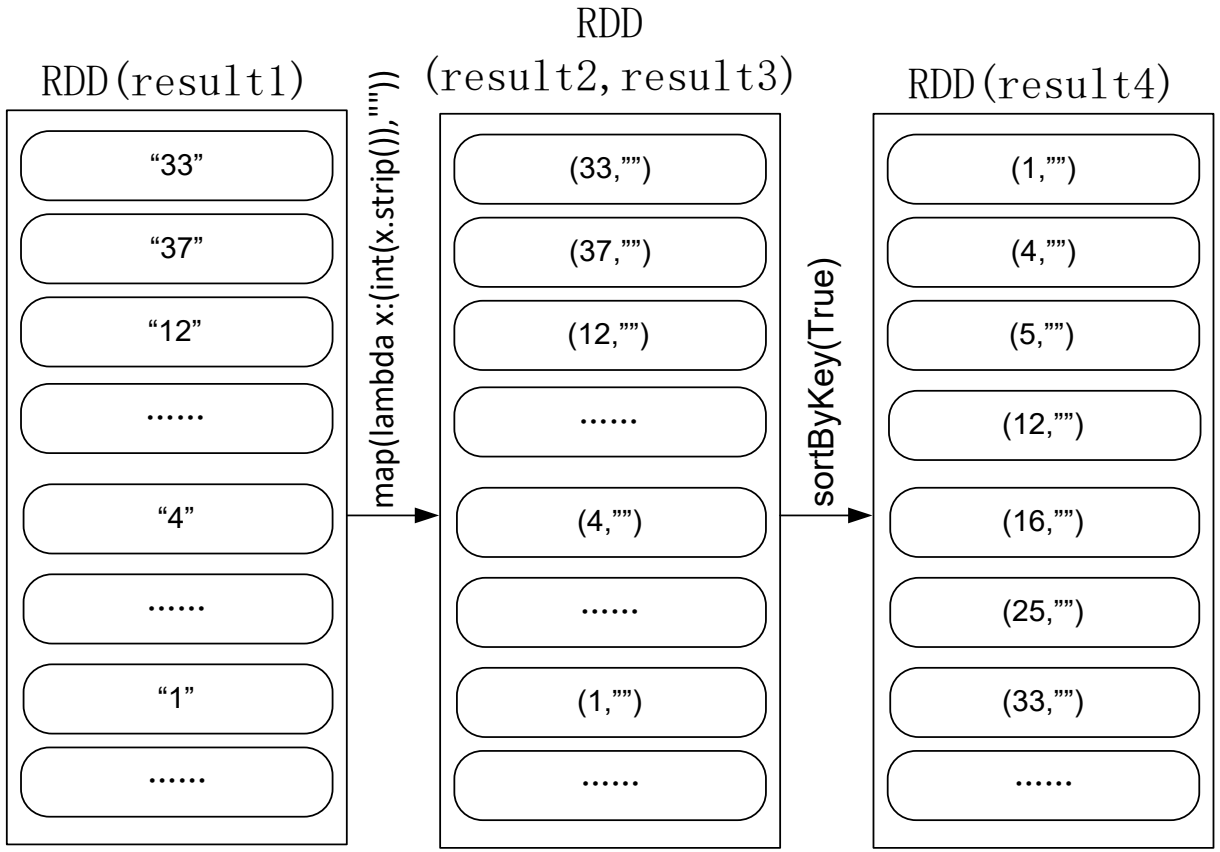
```
lines = sc.textFile("file:///usr/local/spark/mycode/rdd/filesort/file*.txt")
result1 = lines.filter(lambda line:(len(line.strip()) > 0))
```



5.4.2 案例2：文件排序



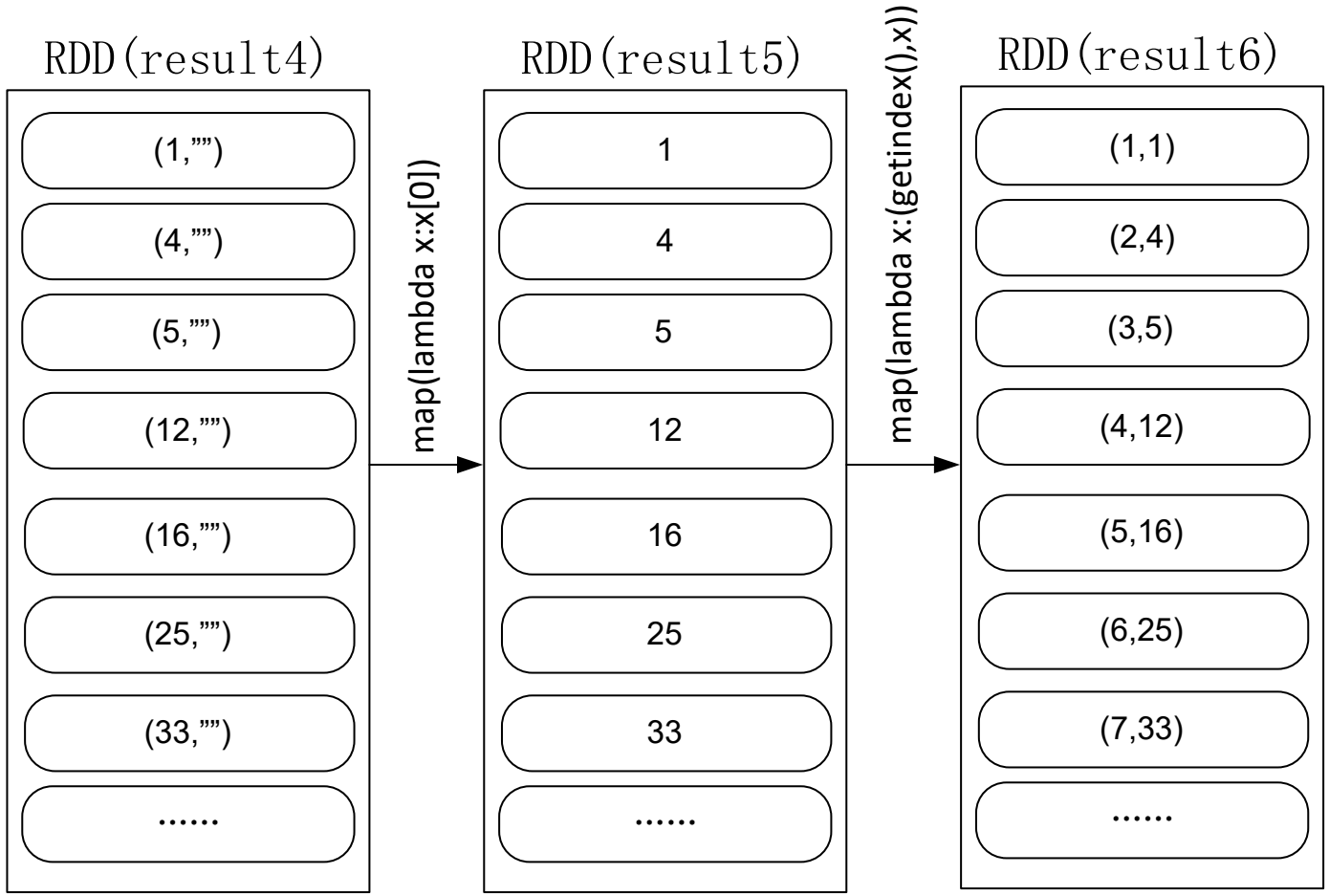
```
result2 = result1.map(lambda x:(int(x.strip()),""))
result3 = result2.repartition(1)
result4 = result3.sortByKey(True)
```



5.4.2 案例2：文件排序



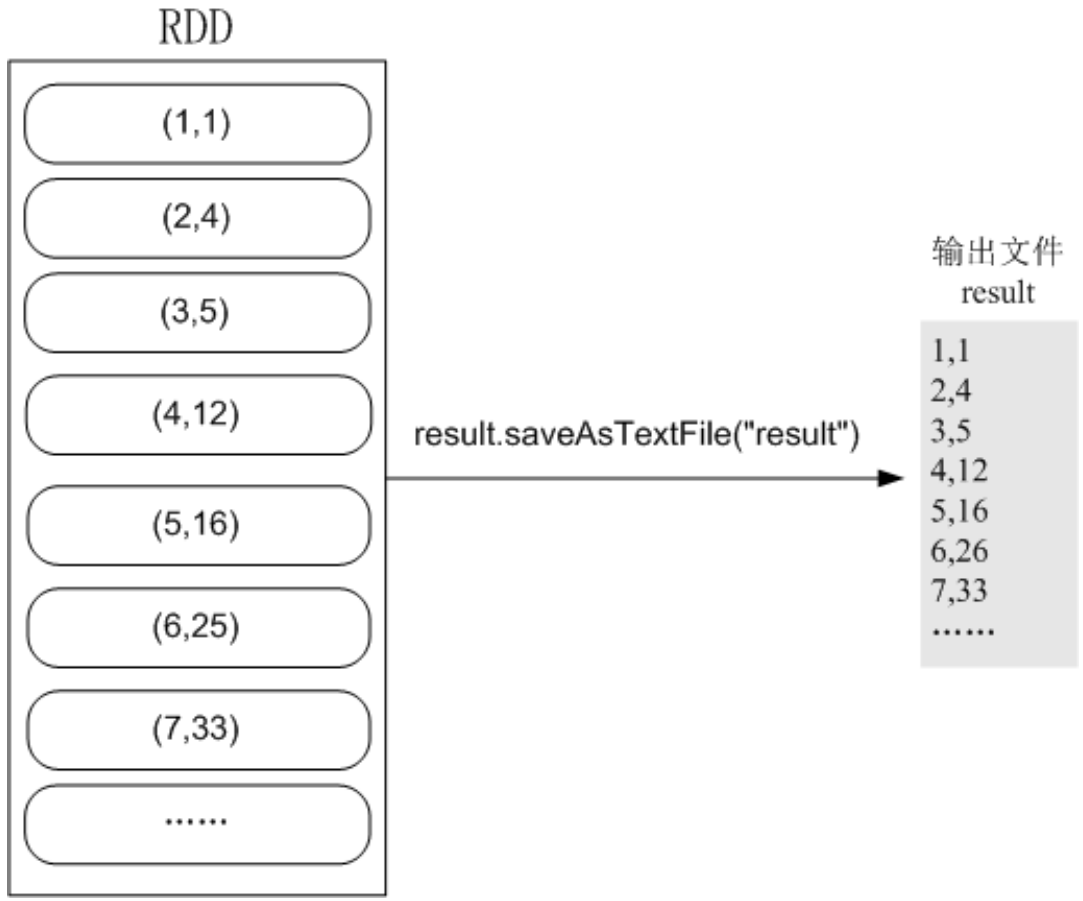
```
result5 = result4.map(lambda x:x[0])
result6 = result5.map(lambda x:(getindex(),x))
```



5.4.2 案例2：文件排序



```
result6.saveAsTextFile("file:///usr/local/spark/mycode/rdd/filesort/sortresult")
```



5.4.3 案例3：二次排序



任务要求：
对于一个给定的文件（数据如file1.txt所示），请对数据进行排序，首先根据第1列数据降序排序，如果第1列数据相等，则根据第2列数据降序排序

输入文件file1.txt

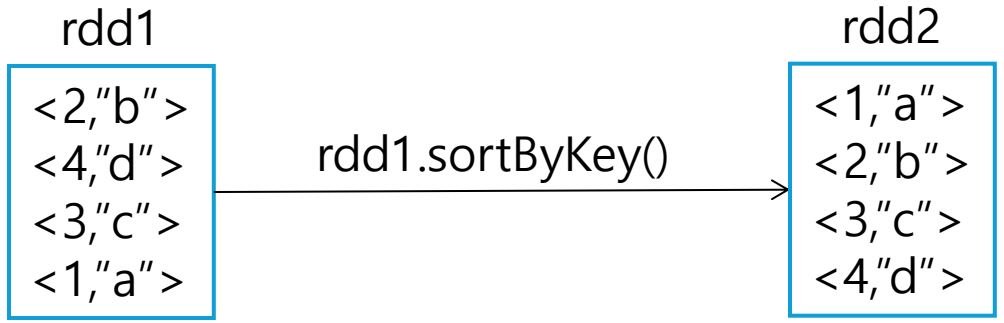
5	3
1	6
4	9
8	3
4	7
5	6
3	2

输出结果

8	3
5	6
5	3
4	9
4	7
3	2
1	6

5.4.3 案例3：二次排序





用于排序的key必须是可比较的对象

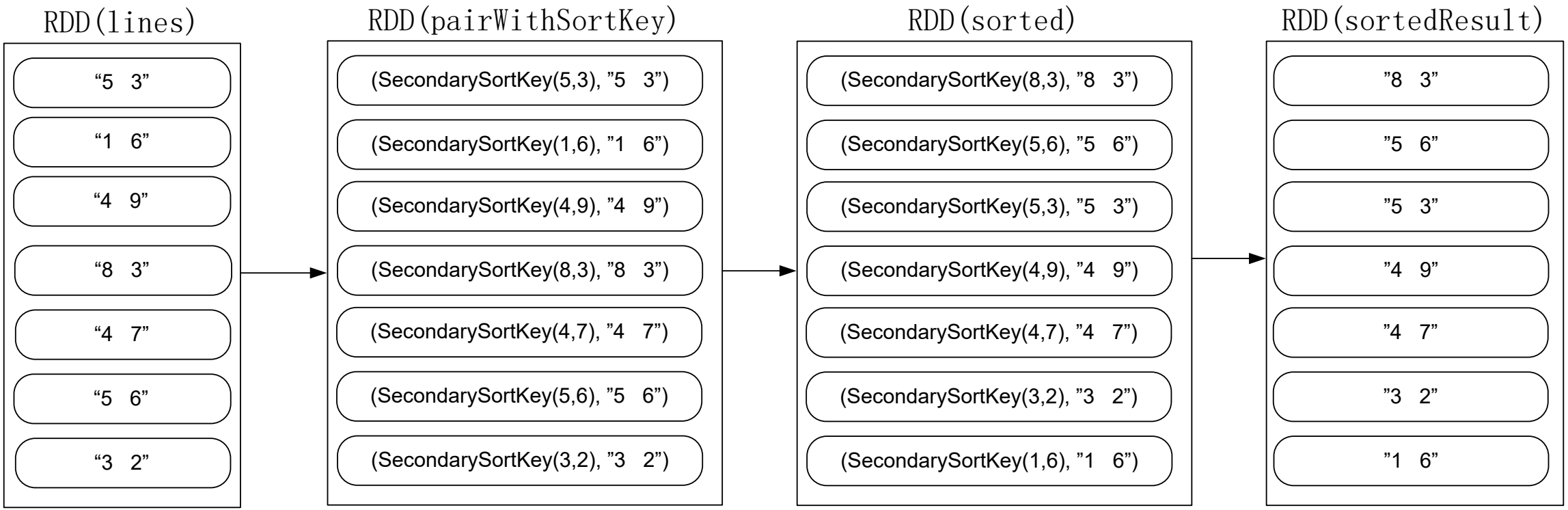
“5 3”
“1 6”
“4 9”
“8 3”
“4 7”
“5 6”
“3 2”

如何比较？

5.4.3 案例3：二次排序



解题思路：



5.4.3 案例3：二次排序



二次排序，具体的实现步骤如下

01 第一步：定义一个类SecondarySortKey，实现自定义的用于排序的key

02 第二步：将要进行二次排序的文件加载进来生成(key,value)类型的RDD

03 第三步：使用sortByKey()基于自定义的key进行二次排序

04 第四步：去除掉排序的key，只保留排序的结果

5.4.3 案例3：二次排序



SecondarySortKey.py代码如下：

```
#!/usr/bin/env python3

from operator import gt
from pyspark import SparkContext, SparkConf

class SecondarySortKey():
    def __init__(self, k):
        self.column1 = k[0]
        self.column2 = k[1]

    def __gt__(self, other):
        if other.column1 == self.column1:
            return gt(self.column2,other.column2)
        else:
            return gt(self.column1, other.column1)
```

剩余代码见下一页

5.4.3 案例3：二次排序



SecondarySortApp.py代码如下：

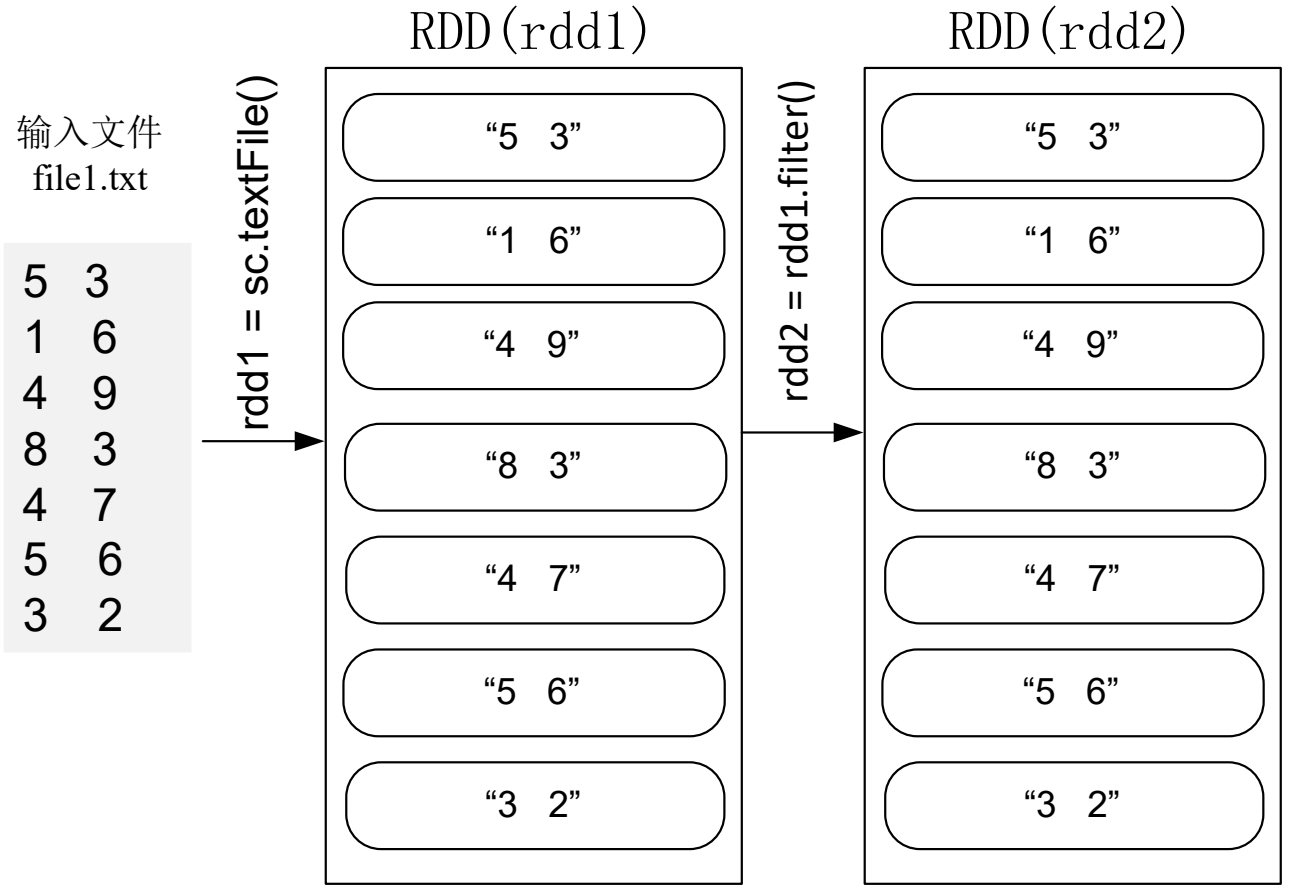
```
def main():
    conf = SparkConf().setAppName('spark_sort').setMaster('local[1]')
    sc = SparkContext(conf=conf)
    file="file:///usr/local/spark/mycode/rdd/secondarysort/file4.txt"
    rdd1 = sc.textFile(file)
    rdd2 = rdd1.filter(lambda x:(len(x.strip()) > 0))
    rdd3 = rdd2.map(lambda x:((int(x.split(" ")[0]),int(x.split(" ")[1])),x))
    rdd4 = rdd3.map(lambda x: (SecondarySortKey(x[0]),x[1]))
    rdd5 = rdd4.sortByKey(False)
    rdd6 = rdd5.map(lambda x:x[1])
    rdd6.foreach(print)

if __name__ == '__main__':
    main()
```

5.4.3 案例3：二次排序



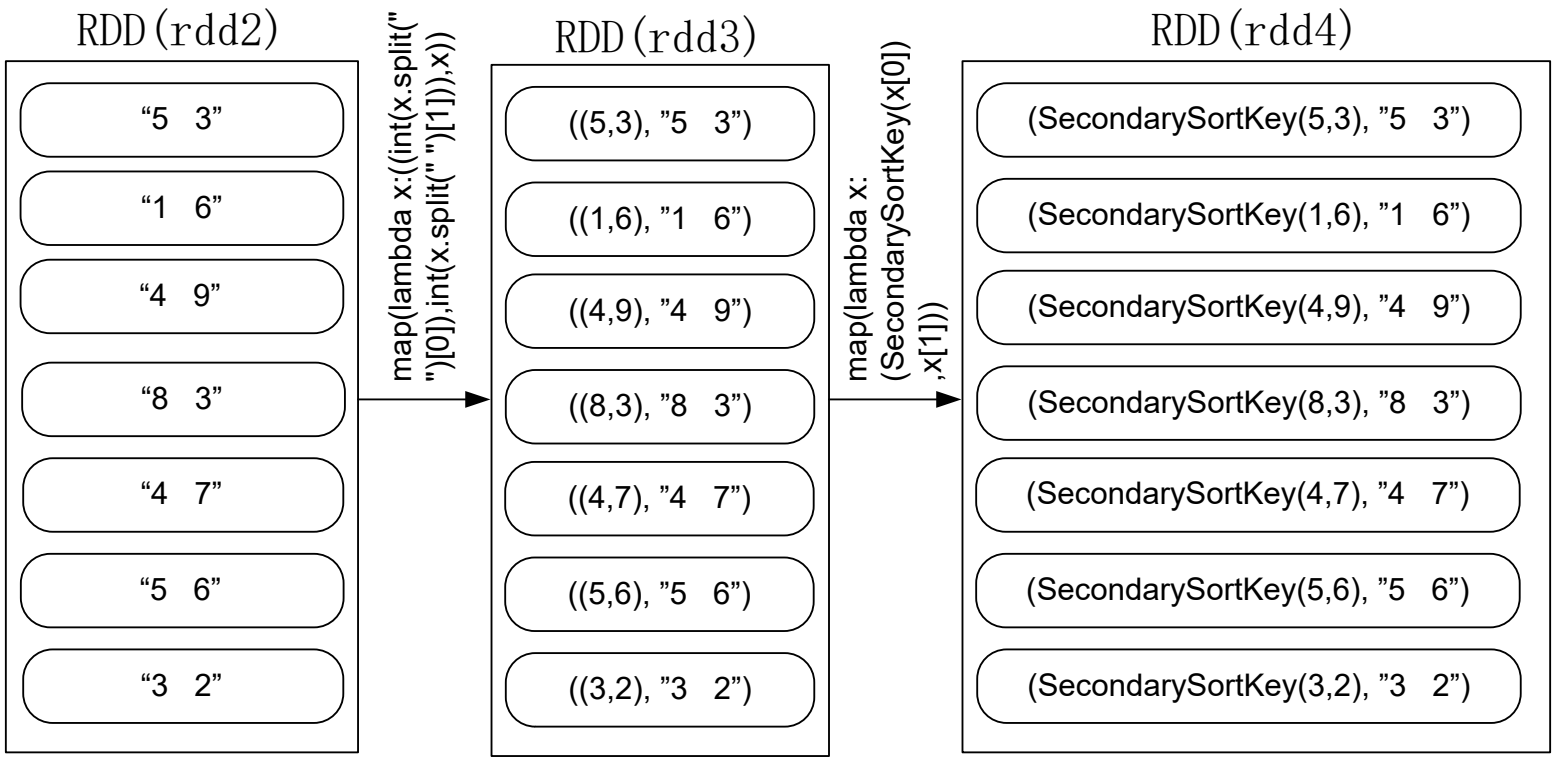
```
rdd1 = sc.textFile(file)
rdd2 = rdd1.filter(lambda x:(len(x.strip()) > 0))
```



5.4.3 案例3：二次排序



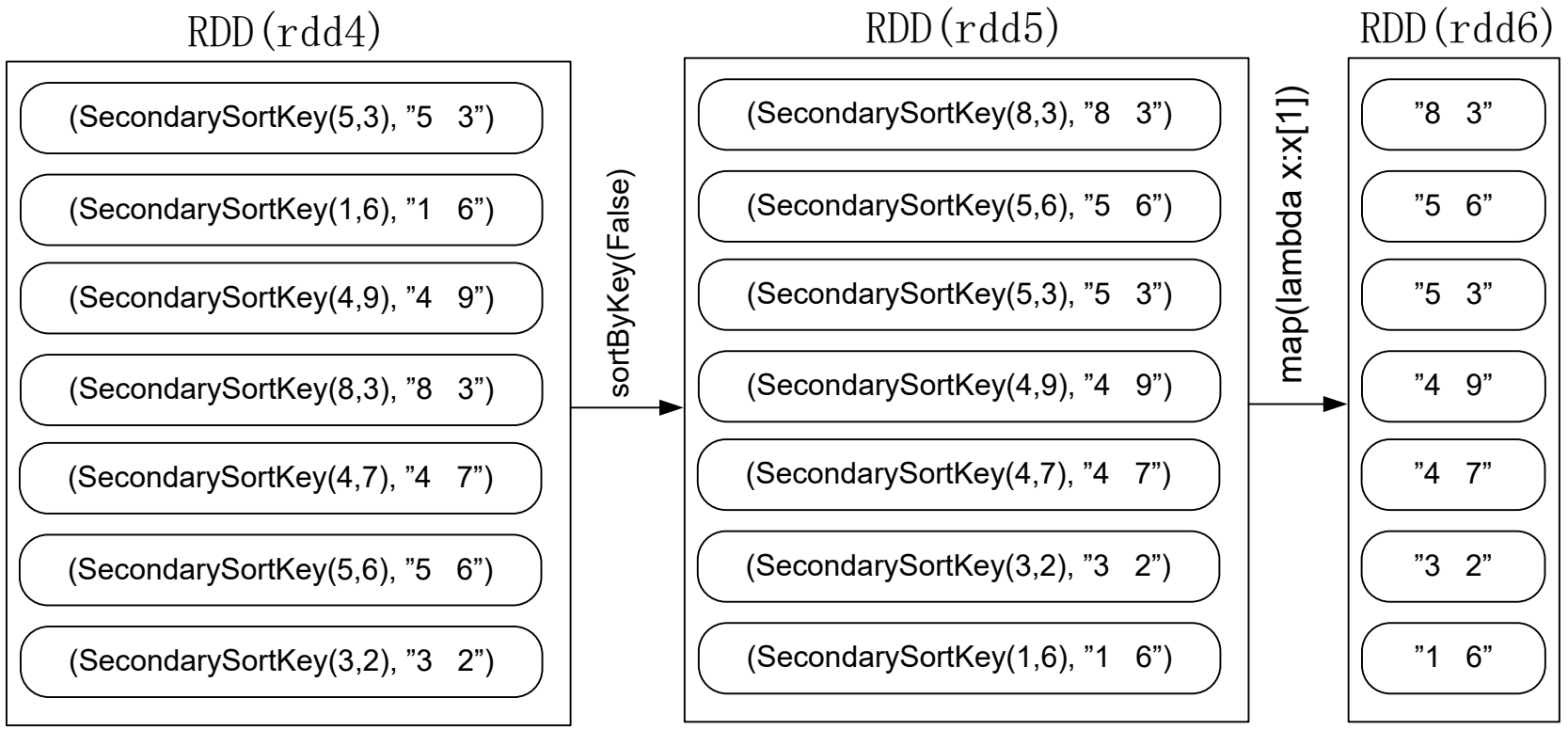
```
rdd3 = rdd2.map(lambda x:((int(x.split(" ")[0]),int(x.split(" ")[1])),x))
rdd4 = rdd3.map(lambda x: (SecondarySortKey(x[0]),x[1]))
```



5.4.3 案例3：二次排序



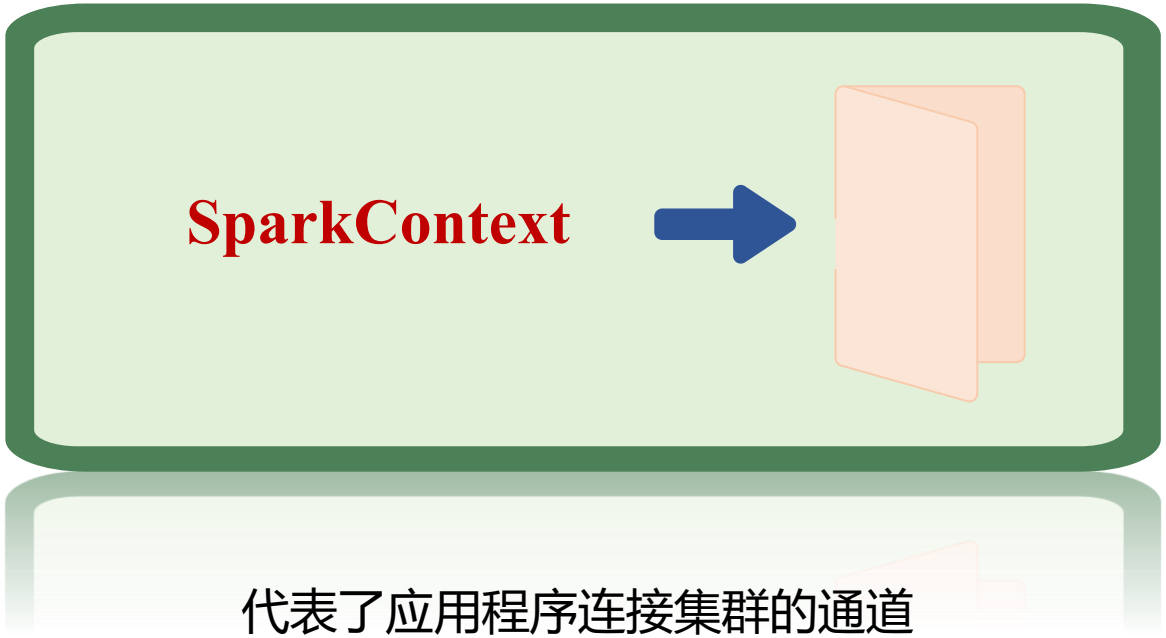
```
rdd5 = rdd4.sortByKey(False)
rdd6 = rdd5.map(lambda x:x[1])
```



5.5 本章小结

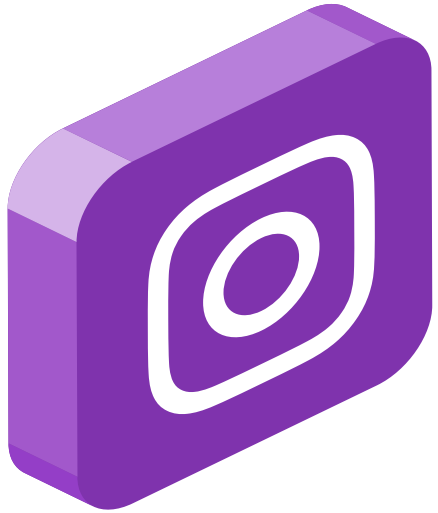
- 本章介绍了RDD编程基础知识，主要是RDD各种操作API的使用，无论多复杂的Spark应用程序，最终都是借助于这些RDD操作来实现的。
- RDD编程都是从创建RDD开始的，我们可以通过多种方式创建得到RDD，比如，从本地文件或者分布式文件系统HDFS中读取数据创建RDD，或者使用parallelize()方法从一个集合中创建得到RDD。
- 创建得到RDD以后，就可以对RDD执行各种操作，包括转换操作和行动操作，本章通过多个实例详细介绍了每个操作API的使用方法。另外，通过持久化，可以把RDD保存在内存或者磁盘中，避免多次重复计算。通过对RDD进行分区，不仅可以增加程序并行度，而且在一些应用场景中可以降低网络通信开销。
- 键值对RDD是一种常见的RDD类型，在Spark编程中经常被使用，本章介绍了键值对RDD的各种操作，并给出了一个综合实例。
- 此外，本章还介绍了文件数据读写和MySQL数据库读写的方法，最后，给出了3个综合实例。

2.3.2 Spark运行基本流程



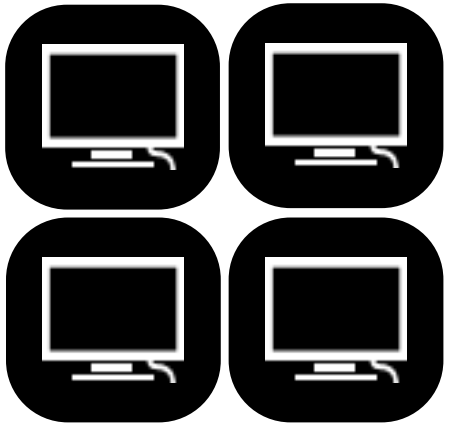
代表了应用程序连接集群的通道

2.3.2 Spark运行基本流程



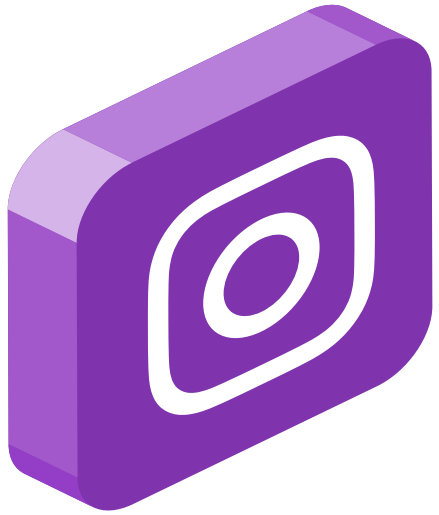
应用



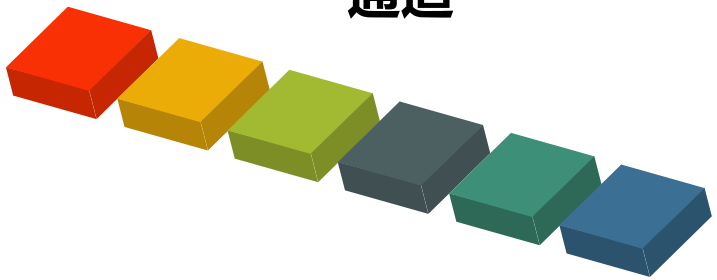


集群

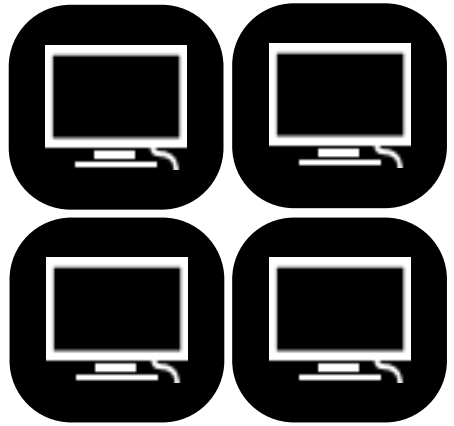
2.3.2 Spark运行基本流程



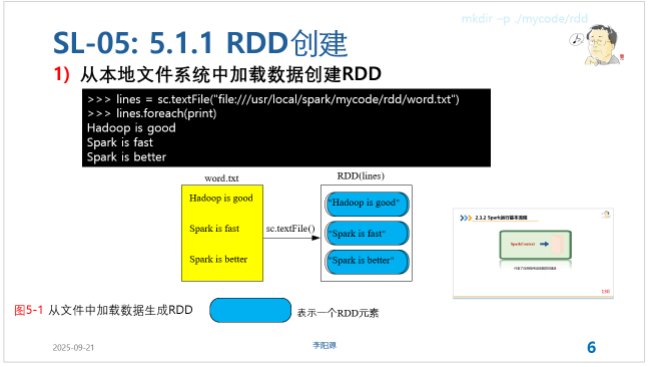
应用



通道



集群



2025-11-11

실험시 에러 발생하면

1. SSH 호스트 키 재생성

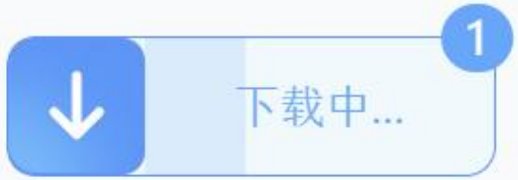
bash

Copy Download

```bash
# 기존 SSH 호스트 키 백업 및 삭제
sudo mv /etc/ssh/ssh_host_* /tmp/

# 새로운 SSH 호스트 키 생성
sudo dpkg-reconfigure openssh-server

# 또는 수동으로 생성
sudo ssh-keygen -A
```



실험시 에러 발생하면

1. 이벤트 로그 디렉토리 생성 (가장 빠른 해결책)

bash Copy Download

```
sudo mkdir -p /tmp/spark-events
sudo chmod 777 /tmp/spark-events  # 모든 사용자에게 권한 부여
```

2. Spark 설정 변경 (영구적 해결)

bash Copy Download

```
# Spark 기본 설정 파일 수정
sudo nano /usr/local/spark/conf/spark-defaults.conf
```

다음 내용 추가:

properties Copy Download

```
spark.eventLog.enabled false
spark.history.fs.logDirectory /tmp/spark-events  # (옵션) 다른 경로 지정 시
```

什么是lambda函数?



Lambda函数是没有名字的简单函数

- 可以把它看作“一次性小型函数”
- 不使用`def`关键字定义函数，而是简单表达
- Lambda这个名称来源于数学中的概念

普通函数 vs Lambda函数对比



定义普通函数

```
def add(x, y):
    return x + y

result = add(3, 5)
print(result)  # 输出: 8
```

普通函数 vs Lambda函数对比



使用Lambda函数定义

```
add = lambda x, y: x + y
result = add(3, 5)
print(result)  # 输出: 8
```

Lambda函数基本结构



lambda 参数1, 参数2, ... ：表达式

Lambda函数的特点

1.没有名称（匿名函数）

2.可以一行编写

3.即使没有return语句也会返回结果

4.只用于**简单功能**

Lambda函数的真正威力：与其他函数配合使用



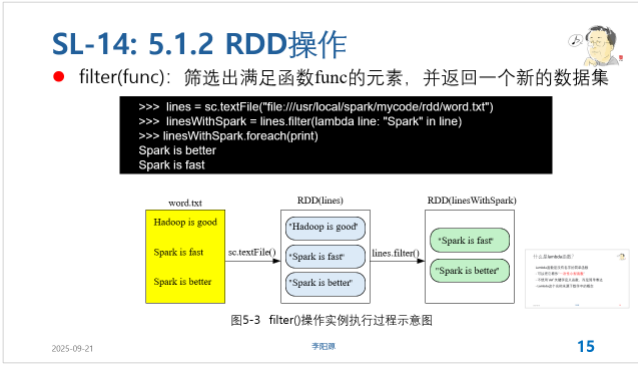
Lambda函数与map()、filter()、sorted()等函数一起使用时能发挥真正价值

与map()函数一起使用

```
numbers = [1, 2, 3, 4, 5]

squared = list(map(lambda x: x**2, numbers))

print(squared)  # 输出: [1, 4, 9, 16, 25]
```





data

[1,2,3,4]

sc.parallelize(data,2)

RDD(rdd1)

[1,2]

[3,4]

rdd1.mapPartitions(myFunction)

分句1: [1, 2] → [2, 4]
分句2: [3, 4] → [6, 8]

RDD(rdd2)

[2, 4]

[2, 4]

```
def myFunction(iter):
        result = list()
        print("==============")
        for it in iter:
            result.append(it*2)
        return result
```



Spark reduceByKey() 工作原理



输入数据结构

```
words = sc.parallelize([
    ("Hadoop", 1), ("is", 1),
("good", 1),
    ("Spark", 1), ("is", 1), ("fast",
1),
    ("Spark", 1), ("is", 1),
("better", 1)
])
```

0 初始数据 (RDD)

('Hadoop', 1) ('is', 1) ('good', 1)
('Spark', 1) ('is', 1) ('fast', 1)
('Spark', 1) ('is', 1)
('better', 1)

1 分区 (按键分组)

将相同键的数据分组到同一分区

分区 1
('Hadoop', 1) ('good', 1)
('fast', 1)

分区 2
('is', 1) ('is', 1) ('is', 1)

分区 3
('Spark', 1) ('Spark', 1)
('better', 1)

2 本地聚合 (分区内聚合)

在每个分区内先应用lambda函数

聚合函数: lambda a, b: a + b

分区 1
('Hadoop', 1) ('good', 1)
('fast', 1)
↓ 本地聚合
('Hadoop', 1), ('good', 1), ('fast', 1)

分区 2
('is', 1) + ('is', 1) +
('is', 1)
↓ 本地聚合
('is', 3)

分区 3
('Spark', 1) + ('Spark', 1)
('better', 1)
↓ 本地聚合
('Spark', 2), ('better', 1)



3 最终聚合 (全局合并)

将所有分区的结果合并计算最终值

各分区本地聚合结果

分区1: ('Hadoop',1), ('good',1), ('fast',1)

分区2: ('is', 3)

分区3: ('Spark',2), ('better',1)

⬇️

最终合并结果

('Hadoop', 1) ('is', 3)

('good', 1) ('Spark', 2)

('fast', 1) ('better', 1)

lambda函数示例说明

示例1: ('is', 1) + ('is', 1) + ('is', 1) → ('is', 3)
示例2: ('Spark', 1) + ('Spark', 1) → ('Spark', 2)

工作原理： lambda函数 `lambda a, b: a + b` 将两个值相加来计算累积和。这个过程对所有相同键的数据重复执行，最终产生聚合结果。

性能优势：

- 网络传输优化：先进行本地聚合，减少数据传输量
- 并行处理：各分区同时进行聚合操作
- 内存高效：最小化中间结果存储



PySpark sortBy() 实际示例分析



初始数据 / Initial Data

myList=[3,1,2,4] →
rdd1=sc.parallelize(myList,1)

3 1 2 4

分区数 / Partitions: 1 (所有数据在同一分区 / All data in one partition)

Lambda函数分析 / Lambda Function Analysis

lambda x: x

中文: 恒等函数，返回元素本身作为排序键
English: Identity function, returns the element itself as sort key

键提取过程 / Key Extraction Process

对每个元素应用 lambda x: x / Apply lambda x: x to each element

3 → 3　1 → 1　2 → 2　4 → 4

两种排序方式比较 / Comparison of Two Sorting Methods

rdd2 = rdd1.sortBy(lambda x:x)
升序排序 / Ascending Sort
(默认 ascending=True / Default ascending=True)

原始数据 / Original Data

3 1 2 4

↓ 升序排序 / Ascending Sort

排序后 / After Sorting

1 2 3 4

```
输出结果 / Output:
1
2
3
4
```

rdd3 = rdd1.sortBy(lambda x:x, False)
降序排序 / Descending Sort
(明确指定 ascending=False / Explicitly set ascending=False)

原始数据 / Original Data

3 1 2 4

↓ 降序排序 / Descending Sort

排序后 / After Sorting

4 3 2 1

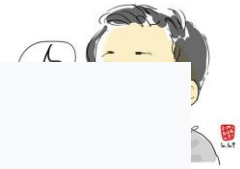
```
输出结果 / Output:
4
3
2
1
```



执行流程分析 / Execution Flow Analysis

代码执行顺序 / Code Execution Order:

1. 创建RDD: rdd1 = sc.parallelize([3,1,2,4], 1)
   → 在单个分区中创建包含4个元素的RDD
   → Create RDD with 4 elements in a single partition
2. 升序排序: rdd2 = rdd1.sortBy(lambda x:x)
   → 使用默认升序排序 / Use default ascending order
3. 输出升序结果: rdd2.foreach(print)
   → 打印: 1, 2, 3, 4
4. 降序排序: rdd3 = rdd1.sortBy(lambda x:x, False)
   → 明确指定降序排序 / Explicitly specify descending order
5. 输出降序结果: rdd3.foreach(print)
   → 打印: 4, 3, 2, 1



关键要点总结 / Key Points Summary

🎯 sortBy() 参数说明 / sortBy() Parameters:

- **keyfunc**: lambda x: x (提取排序键的函数 / Function to extract sort key)
- **ascending**: True(默认升序) / False(降序) / True(default ascending) / False(descending)

☑ 代码特点 / Code Characteristics:

- 使用单分区处理，避免了跨分区洗牌的复杂性
- Single partition processing, avoiding cross-partition shuffle complexity
- lambda x: x 是最简单的恒等函数
- lambda x: x is the simplest identity function
- foreach(print) 直接输出每个元素
- foreach(print) directly outputs each element

🔧 实际应用场景 / Practical Use Cases:

- 数据预处理中的排序需求 / Sorting requirements in data preprocessing
- 生成有序的输出结果 / Generate ordered output results
- 为后续分析准备有序数据 / Prepare ordered data for subsequent analysis

