5.4 综合案例（Comprehensive Cases)

第1页PPT：目录介绍

好，同学们，现在我们要把之前学过的RDD(Resilient Distributed Dataset，弹性分布式数据集)理论知识通过实际的综合案例来应用一下。

今天我们要讲的三个综合案例是：

- 第一个，求TOP值的问题
- 第二个，文件排序(File Sorting)问题
- 第三个，二次排序(Secondary Sorting)问题

这三个案例呢，都是在实际工作中经常会遇到的典型大数据处理模式。

5.4.1 案例1：求TOP值（Case 1: Finding TOP Values)

第2页PPT：问题定义

咱们先看第一个案例。

这里大家看到有订单数据，每条记录包含四个字段：订单ID(orderid)、用户ID(userid)、支付金额(payment)、商品ID(productid)。

File00.txt和file01.txt两个文件里都有这种格式的数据，我们的目标就是从所有数据中按照支付金额(payment)找出前N个最大的值。

第3页PPT：代码实现

好，现在我们来看具体的PySpark代码实现。

导入必要的PySpark模块

from pyspark import SparkConf, SparkContext

```
# 主程序入口
if __name__ == '__main__':
    # 1. 创建Spark配置和上下文
    conf = SparkConf().setAppName("find TOP value")  # 设置应用名称
    sc = SparkContext(conf=conf)  # 创建Spark上下文对象

    # 2. 读取输入文件
```

```
# 从本地文件系统读取两个文本文件创建RDD
lines = sc.textFile("file:///usr/local/spark/mycode/rdd/file00.txt,"
                    "file:///usr/local/spark/mycode/rdd/file01.txt")

# 3. 数据清洗和转换
# 过滤掉空行和不符合格式的行(列数不等于4的行)
result1 = lines.filter(lambda line: (len(line.strip()) > 0) and (len(line.split(",")) == 4))

# 提取每行第三列的数据(索引为2)
result2 = result1.map(lambda x: x.split(",")[2])

# 将字符串转换为整数，并转换为键值对格式(值设为空字符串)
result3 = result2.map(lambda x: (int(x), ""))

# 4. 数据重分区和排序
# 将所有数据重新分区到1个分区(为了全局排序)
result4 = result3.repartition(1)

# 按键(整数)降序排序
result5 = result4.sortByKey(False)

# 5. 结果提取和输出
# 只保留键(去掉空字符串的值)
result6 = result5.map(lambda x: x[0])

# 取前5个最大的数字
result7 = result6.take(5)

# 打印结果
for a in result7:
    print(a)

# 6. 停止Spark上下文
sc.stop()
```

首先要创建SparkContext(Spark上下文)。这里注意啊，setMaster("local")表示在本地模式下运行。

第4-5页PPT：数据加载和过滤

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

result1 = lines.filter(lambda line:(len(line.strip()) > 0) and (len(line.split(","))== 4))

用textFile()方法读取文件创建RDD(弹性分布式数据集)。然后用filter()转换操作(Transformation)过滤掉空行和字段不是4个的错误数据。

大家看这里，每个文件的内容会以字符串形式存储在RDD中，比如："1,1768,50,155"、"2,1218,600,211"这样。

第6页PPT：数据提取和转换

result2 = result1.map(lambda x:x.split(",")[2])

result3 = result2.map(lambda x:(int(x),""))

现在用map()转换操作把每行按逗号分割，取第三个索引位置，也就是支付金额部分。

然后再用一个map()操作把字符串转换成整数，并构成键值对(Key-Value Pair)形式。这里键是支付金额，值是空字符串。

第7页PPT：排序和结果提取

result4 = result3.repartition(1)

result5 = result4.sortByKey(False)

result6 = result5.map(lambda x:x[0])

result7 = result6.take(5)

用repartition(1)把所有数据合并到一个分区(Partition)中，然后用sortByKey(False)按降序排列。

最后提取键值，用take(5)行动操作(Action)取前5个结果。

第8页PPT：实例2介绍

现在我们看一个稍微复杂点的例子。

有个商品销售数据集sales.txt，每条记录包含5个字段：时间戳、省份、城市、用户、购买商品。

比如"1602031567000,21,1005,95001,20001"这条数据中，21代表福建省，1005代表厦门市，95001代表客户王明，20001代表华为Mate60手机。

我们的目标是求出每个省份销量前3名的商品。

第9-10页PPT：解题思路过程

解题过程我们分步骤来看：

1. 从sales.txt加载数据生成rdd1
2. 过滤出需要的字段，即省份和商品，得到rdd2
3. 把rdd2中每个元素从(省份,商品)转换成(省份,商品,1)，得到rdd3
4. 把每个元素中的省份和商品名称组合在一起，得到rdd4
5. 根据(省份,商品)进行分组，得到rdd5
6. 对每个分组进行汇总，得到rdd6
7. 把每个元素从((省份,商品),销量)转换成(省份,(商品,销量))，得到rdd7
8. 对元素按照省份进行分组，得到rdd8
9. 最后对元素进行排序，求出每个省份销量前3名的商品

第11-13页PPT：详细代码实现

```
# coding:utf8
from pyspark import SparkConf, SparkContext
if __name__ == '__main__':
    conf = SparkConf().setMaster("local[*]").setAppName("Page 123")
    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:///usr/local/spark/mycode/rdd/sales.txt")
    dataRDD = sourceRDD.filter(lambda line : len(line.split(",")) == 5)
    # 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)
    # 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()
    # 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), key=lambda tup: tup[1],reverse=True)[:3])
    # 7.把数据打印出到控制台
    resultRDD.foreach(print)
    sc.stop()
```

这里的关键是用reduceByKey()转换操作对相同键的值进行求和，这就是MapReduce模式的核心思想。

用mapValues()对每个省份分组后的商品按销量降序排列，然后取前3个。

5.4.2 案例2：文件排序 (Case 2: File Sorting)

第14-15页PPT：问题定义和解决方案

第二个案例是文件排序。

有多个输入文件，每个文件的每一行都是一个整数。要求读取所有文件中的整数，进行排序后输出到新文件中。

输出格式是每行两个整数，第一个是排序位次，第二个是原来的整数值。

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

可以用通配符(*)一次读取多个文件。

第16-19页PPT：分步实现

```
# 导入必要的库
from pyspark import SparkConf, SparkContext  # Spark相关库
import shutil  # 文件操作库
import os  # 操作系统接口库

# 全局索引变量
index = 0


# 获取并递增索引的函数
def getindex():
    global index  # 声明使用全局变量
    index += 1  # 索引值加1
    return index  # 返回新索引值


# 主函数
def main():
    # 1. 配置Spark环境
    conf = SparkConf().setMaster("local[1]").setAppName("FileSort")  # 本地单线程模式，应用名"FileSort"
    sc = SparkContext(conf=conf)  # 创建Spark上下文

    # 2. 清理输出目录
    output_path = "/usr/local/spark/mycode/rdd/filesort/sortresult"  # 输出目录路径
    if os.path.exists(output_path):  # 检查目录是否存在
        shutil.rmtree(output_path)  # 递归删除目录及其内容
```

```
    # 3. 读取并处理数据
    lines = sc.textFile("file:///usr/local/spark/mycode/rdd/filesort/file*.txt")  # 读取多个文本文件
    index = 0  # 重置索引

    # 4. 数据转换流程
    result1 = lines.filter(lambda line: (len(line.strip()) > 0))  # 过滤空行
    result2 = result1.map(lambda x: (int(x.strip()), ""))  # 转换为(数字,"")键值对
    result3 = result2.repartition(1)  # 重新分区为1个(便于全局排序)
    result4 = result3.sortByKey(True)  # 按键(数字)升序排序
    result5 = result4.map(lambda x: x[0])  # 只保留数字(去掉空字符串)
    result6 = result5.map(lambda x: (getindex(), x))  # 添加行号索引

    # 5. 输出结果
    result6.foreach(print)  # 打印每个元素
    result6.saveAsTextFile("file://" + output_path)  # 保存结果到文件


# 程序入口
if __name__ == '__main__':
    main()
```

这里要注意的是用全局变量来生成排序号的方法。不过这种方法在分布式环境中可能会有问题，需要小心使用。

5.4.3 案例3：二次排序（Case 3: Secondary Sorting)

第20-21页PPT：二次排序的必要性

第三个也是最复杂的案例——二次排序。

给定文件中的数据，要求首先根据第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。

普通的sortByKey()只能按一个键排序，所以我们需要创建复合排序键(Composite Key)。

第22-23页PPT：解决策略

实现二次排序的四步方法：

1. 定义SecondarySortKey类——实现自定义的用于排序的key
2. 将要进行二次排序的文件加载进来生成(key,value)类型的RDD
3. 使用sortByKey()基于自定义的key进行二次排序
4. 去除掉排序的key，只保留排序的结果

第24-25页PPT：自定义键类

```
#!/usr/bin/env python3
# 导入必要的模块
from operator import gt  # 用于比较操作
from pyspark import SparkContext, SparkConf  # Spark相关模块

# 自定义二级排序键类
class SecondarySortKey():
    def __init__(self, k):
        """
        初始化方法
        参数:
            k: 包含两个元素的元组，作为排序键
        """
        self.column1 = k[0]  # 主排序列
        self.column2 = k[1]  # 次排序列

    def __gt__(self, other):
        """
        重载大于比较运算符
```

```
        规则:
            1. 先比较主排序列(column1)
            2. 如果主排序列相等，再比较次排序列(column2)
        """
        if other.column1 == self.column1:
            return gt(self.column2, other.column2)  # 比较次排序列
        else:
            return gt(self.column1, other.column1)  # 比较主排序列
# 主函数
def main():
    # 1. 配置Spark环境
    conf = SparkConf().setAppName('spark_sort').setMaster('local[1]')  # 本地单线程模式
    sc = SparkContext(conf=conf)  # 创建Spark上下文

    # 2. 读取输入文件
    file = "file:///usr/local/spark/mycode/rdd/secondarysort/file4.txt"
    rdd1 = sc.textFile(file)  # 读取文本文件创建RDD

    # 3. 数据清洗和转换
    rdd2 = rdd1.filter(lambda x: (len(x.strip()) > 0))  # 过滤空行
    # 将每行数据转换为((col1,col2), 原始行)的格式
    rdd3 = rdd2.map(lambda x: ((int(x.split(" ")[0]), int(x.split(" ")[1])), x))

    # 4. 使用自定义键进行排序
    # 将键转换为SecondarySortKey对象
    rdd4 = rdd3.map(lambda x: (SecondarySortKey(x[0]), x[1]))
    # 按自定义键降序排序(False表示降序)
    rdd5 = rdd4.sortByKey(False)

    # 5. 结果处理
    rdd6 = rdd5.map(lambda x: x[1])  # 只保留原始行数据(去掉排序键)
    rdd6.foreach(print)  # 打印排序后的结果

# 程序入口
if __name__ == '__main__':
    main()
```

关键是实现__gt__()方法来定义两个对象如何比较。如果第一列相等就比较第二列，否则比较第一列。

第26-28页PPT：执行过程

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)

解析字符串生成元组，然后用SecondarySortKey对象包装使其可排序。

5.5 本章小结 (Chapter Summary)

第29页PPT：章节总结

好，现在我们来总结一下这一章的内容。

这一章我们学习了RDD编程的基础知识。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。

核心要点总结如下：

1. **RDD创建**：可以通过本地文件、HDFS或者parallelize()方法从集合中创建RDD
2. **转换操作和行动操作**：map、filter、reduceByKey等转换操作，以及collect、take等行动操作
3. **持久化(Persistence)**：把RDD保存在内存或者磁盘中，避免多次重复计算
4. **分区(Partitioning)**：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
5. **键值对RDD**：Spark编程中经常被使用的常见RDD类型
6. **文件数据读写和数据库读写**：介绍了与各种数据源的连接方法

特别是今天学的三个综合实例——求TOP值、文件排序、二次排序，这些都是实务中会经常遇到的模式，大家一定要记住。

下次课我们会讲更高级的RDD操作和实际生产环境中的优化技巧。

大家有什么问题吗？