第3章Spark编程

(源自:https://biglab.site)

(版本:Ver1.2-20230828)

学习目标

掌握如何创建RDD

掌握Spark基本转换操作

掌握Spark基本动作操作

了解如何读取不同格式文件数据

了解如何存储数据为不同格式文件

任务背景

2021年，某公司为了提高员工工作的积极性，将对公司员工进行一次调薪，需要根据员工在2020年的薪资情况及在职表现重新调整薪资，对于爱岗敬业的公司员工，公司拟根据其业绩分析情况予以不同程度涨薪

学习重点

理论学习

（1）从内存中已有数据创建RDD。

（2）从外部存储创建RDD。

（3）RDD转化操作和行动操作。

（4）RDD键值对操作。

（5）文件读取与存储。

实验学习

（1）以学生成绩数据创建RDD

（2）查询学生成绩表中的前5名。

（3）输出单科成绩为100分的学生ID。

（4）输出每位学生所有科目的总成绩。

（5）输出每位学生的平均成绩。

（6）将汇总后的学生成绩存储为文本文件。

（7）统计文本中性别为“男”用户数。

（8）单词计数。

学习视频

理论学习

1. 任务3.1： 虚拟机开启与集群服务开启 ， 任务3.1读取员工薪资数据创建RDD , 任务3.1实操演练
2. 任务3.2: 任务3.2查询上半年实际薪资排名前3的员工信息 , 任务3.2实操演练
3. 任务3.3：任务3.3查询上半年或下半年实际薪资大于20万元的员工姓名 ， 任务3.3实操演练
4. 任务3.4: 任务3.4输出每位员工2020年的总实际薪资

概述

准备测试数据

下载测试数据文件spark_data.tar.gz,该文件包含后续章节的测试数据

```
mkdir /root/spark/
cd /root/spark/
wget http://biglab.site/b5951Ospark/file/spark_data.tar.gz
tar -xvzf ./spark_data.tar.gz
hdfs dfs -mkdir /user/myname
hdfs dfs -put /root/spark/spark_data/Employee_salary_first_half.csv
/user/myname/
hdfs dfs -put /root/spark/spark_data/Employee_salary_second_half.csv
/user/myname/
hdfs dfs -ls /user/myname
```

spark_data.tar.gz备用下载地址：

wget http://home.hddly.cn:90/soft/hadoop/spark_data.tar.gz

自建文件

```
vi /root/pslog.txt
```

在文件中输入任意行的字符串，如:

```
abc
bcd
def
```

将该文件上传到hdfs:

```
hdfs dfs -put /root/pslog.txt /user/myname/
```

任务3.1读取员工薪资数据创建RDD

RDD特点

- Resilient Distributed Datasets,弹性分布式数据集
- 它是提供了许多操作接口的数据集合
- 它被划分为一到多个分区，一个分布在集群各节点的存放元素的集合
- RDD是一个容错的，只读的，可进行并行操作的数据结构

RDD的创建方法

- 第一种是将程序中已存在的Seq集合（如集合、列表、数组）转换成RDD。
- 第二种是对已有RDD进行转换得到新的RDD，这两种方法都是通过内存中已有的集合创建RDD的。
- 第三种是直接读取外部存储系统的数据创建RDD

准备RDD测试运行环境

回顾搭建Spark完全分布式集群

请参考：ch01第一章：搭建Spark完全分布式集群。

进入Hadoop集群，开启spark集群

先使用start-all.sh开启hadoop集群，然后进入spark的sbin开启spark集群

```
start-all.sh
cd /usr/local/spark/sbin/
./start-all.sh
```

进入spark-shell运行模式

```
spark-shell
```

运行如下：

```
[root@master sbin]# spark-shell
Spark context Web UI available at http://master:4040
Spark context available as 'sc' (master = spark://master:7077, app id = app-20230914064838-0001).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 3.1.3
      /_/

Using Scala version 2.12.10 (OpenJDK 64-Bit Server VM, Java 1.8.0_322)
Type in expressions to have them evaluated.
Type :help for more information.

scala>
```

从内存中已有数据创建RDD

parallelize

```
val data = Array(1,2,3,4,5)
val disData=sc.parallelize(data)
disData.partitions.size
val distData1=sc.parallelize(data,3)
distData1.partitions.size
```

运行结果：

```
scala> val data = Array(1,2,3,4,5)
data: Array[Int] = Array(1, 2, 3, 4, 5)

scala> val disData=sc.parallelize(data)
disData: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[0] at
parallelize at <console>:26

scala> disData.partitions.size
res0: Int = 6

scala> val distData1=sc.parallelize(data,3)
distData1: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[1] at
parallelize at <console>:26

scala> distData1.partitions.size
res1: Int = 3
```

makeRDD

```
val seq =Seq((1, Seq("iteblog.com","sparkhost1.com")),(3,
Seq("iteblog.com","sparkhost2.com")),(2,
Seq("iteblog.com","sparkhost3.com")))
val iteblog= sc.makeRDD(seq)
iteblog.collect
iteblog.partitions.size
iteblog.preferredLocations(iteblog.partitions(0))
iteblog.preferredLocations(iteblog.partitions(1))
iteblog.preferredLocations(iteblog.partitions(2))
```

从外部存储创建RDD

从外部存储创建RDD是指直接读取一个存放在文件系统 的数据文件创建RDD,

从HDFS文件创建RDD

准备数据(**已按本文件开头完成数据准备的，可忽略本处**)

```
vi /root/pslog.txt
abc
bcd
def
```

上传/root/pslog.txt到hdfs:

```
hdfs dfs -put /root/pslog.txt /user/myname/
```

读取hdfs上的文件

```
spark-shell
```

```
val test = sc.textFile("/user/myname/pslog.txt")
test.collect
```

运行结果

```
scala> val test = sc.textFile("/user/myname/pslog.txt")
test: org.apache.spark.rdd.RDD[String] = /user/myname/pslog.txt
MapPartitionsRDD[11] at textFile at <console>:24

scala> test.collect
res5: Array[String] = Array(abc, bcd, def, "")

scala>
```

从Linux本地文件创建RDD

读linux文件

```
val test = sc.textFile("file:///usr/local/hadoop-3.3.1/etc/hadoop/core-
site.xml")
test.collect
```

运行结果

```
scala> val test = sc.textFile("file:///usr/local/hadoop-3.3.1/etc/hadoop/core-site.xml")
test: org.apache.spark.rdd.RDD[String] = file:///usr/local/hadoop-3.3.1/etc/hadoop/core-site.xml MapPartitionsRDD[13] at textFile at <console>:24

scala> test.collect
res6: Array[String] = Array(<?xml version="1.0" encoding="UTF-8"?>, <?xml-stylesheet type="text/xsl" href="configuration.xsl"?>, <!--, "  Licensed under the Apache License, Version 2.0 (the "License");", "  you may not use this file except in compliance with the License.", "  You may obtain a copy of the License at", "", "    http://www.apache.org/licenses/LICENSE-2.0", "", "  Unless required by applicable law or agreed to in writing, software", "  distributed under the License is distributed on an "AS IS" BASIS,", "  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.", "  See the License for the specific language governing permissions and", "  limitations under the License. See accompanying LICENSE file.", -->, "", <!--Put site-specific ...

scala>
```

任务实现

准备数据

准备数据(**已按本文件开头完成数据准备的，可忽略本处**)

```
mkdir /root/spark/
cd /root/spark/
wget http://biglab.site/b5951Ospark/file/spark_data.tar.gz
tar -xvzf ./spark_data.tar.gz
hdfs dfs -mkdir /user/myname
hdfs dfs -put /root/spark/spark_data/Employee_salary_first_half.csv /user/myname/
hdfs dfs -put /root/spark/spark_data/Employee_salary_second_half.csv /user/myname/
hdfs dfs -ls /user/myname
```

进入Spark-shell

```
spark-shell --master local[2]
```

RDD实现

```
val first_half = sc.textFile("/user/myname/Employee_salary_first_half.csv")
val second_half = sc.textFile("/user/myname/Employee_salary_second_half.csv")
first_half.collect().foreach(x=>println(x))
second_half.collect().foreach(x=>println(x))
```

任务3.2查询上半年实际薪资排名前3的员工信息

任务描述

使用RDD的基本操作完成对员工上半年实际薪资的排名。找出薪资排名前3的员工信息。

使用map转换数据

```
val distData=sc.parallelize(List(1,2,45,3,76))
val sq_dist=distData.map(x=>x*x)
```

使用sortBy()排序

sortBy()是对标准RDD进行

排序的方法，有3个参数

参数1，f:(T)=>K,左边是要被排序对象中的每个元素，
右边返回的值是元素中要进行排序的值

参数2，ascending,排序，默认true,为升序

参数3，numPartitions,排序后的RDD分区个数

源码

```
val data=sc.parallelize(List((1,3),(45,3),(7,6)))
val sort_data= data.sortBy(x=>x._2,false,1)
sort_data.collect
```

运行结果：

```
scala> val data=sc.parallelize(List((1,3),(45,3),(7,6)))
data: org.apache.spark.rdd.RDD[(Int, Int)] = ParallelCollectionRDD[13] at
parallelize at <console>:24

scala> val sort_data= data.sortBy(x=>x._2,false,1)
sort_data: org.apache.spark.rdd.RDD[(Int, Int)] = MapPartitionsRDD[16] at
sortBy at <console>:25

scala> sort_data.collect
res3: Array[(Int, Int)] = Array((7,6), (1,3), (45,3))
```

使用collect()查询

特点

1. 行动操作，会将RDD所有元素操作转换成数组并返回到Driver端
2. 数据量比较大时可能导致内存溢出
3. collect有两种操作方式

collect():Array[T]

collectU:ClassTag:RDD[U]

源码 :collect():Array[T]

```
sq_dist.collect()
sort_data.collect()
```

源码:collectU:ClassTag:RDD[U]

```
// 定义一个函数one
val one:PartialFunction[Int, String] = {case 1 => "one";case _ => "other"}
// 创建RDD
val data = sc.parallelize(List(2, 3, 1))
// 使用collect()方法，将one函数作为参数
data.collect(one).collect
```

使用flatMap转换数据

```
val test = sc.parallelize(List("How are you","I am fine","What about you"))
test.collect
test.map(x=>x.split(" ")).collect
test.flatMap(x=>x.split(" ")).collect
```

运行结果：

```
scala> val test = sc.parallelize(List("How are you","I am fine","What about you"))
test: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[29] at parallelize at <console>:24

scala> test.collect
res13: Array[String] = Array(How are you, I am fine, What about you)

scala> test.map(x=>x.split(" ")).collect
res14: Array[Array[String]] = Array(Array(How, are, you), Array(I, am, fine), Array(What, about, you))

scala> test.flatMap(x=>x.split(" ")).collect
res15: Array[String] = Array(How, are, you, I, am, fine, What, about, you)

scala>
```

使用take()方式查询某几个值

特点

- 类似collect,是返回数组
- 返回指定个数的数据

源码

```
val data = sc.parallelize(1 to 10)
data.take(5)
```

运行结果：

```
scala> val data = sc.parallelize(1 to 10)
data: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[32] at
parallelize at <console>:24

scala> data.take(5)
res16: Array[Int] = Array(1, 2, 3, 4, 5)

scala>
```

任务实现

方法

查询上半年实际薪资排名前3的员工信息，需要对上半年的实际薪资进行排序，而创建RDD时，textFile()方法是将每一行数据作为一条记录存储的，所以在排序前需要先对数据进行转换，实现步骤如下

1. 读取CSV文件，将第一行字段名称删除。
2. 将数据按分隔符“,”分隔，取出第2列员工姓名和第7列实际薪资数据，并将实际薪资数据转换成Int类型数据。
3. 通过sortBy()方法根据实际薪资进行降序排列。
4. 通过take()方法获取上半年实际薪资排名前3的员工信息 。

源码

```
// 创建RDD
val first_half = sc.textFile("/user/myname/Employee_salary_first_half.csv")
// 去除首行数据
val drop_first = first_half.mapPartitionsWithIndex((ix, it) => {
if (ix == 0) it.drop(1)
it
})
// 分割RDD，并取出第2列员工姓名和第7列实际薪资数据
val split_first = drop_first.map(line => {val data = line.split(",");
(data(1), data(6).toInt)})
// 使用sortBy()方法根据实际薪资降序排序
val sort_first = split_first.sortBy(x => x._2, false)
// 取出上半年实际薪资Top3的员工
sort_first.take(3)
```

```
15
```

运行结果：

```
1
2  scala> // 创建RDD
3
4  scala> val first_half =
   sc.textFile("/user/myname/Employee_salary_first_half.csv")
5  first_half: org.apache.spark.rdd.RDD[String] =
   /user/myname/Employee_salary_first_half.csv MapPartitionsRDD[38] at textFile
   at <console>:24
6
7  scala> // 去除首行数据
8
9  scala> val drop_first = first_half.mapPartitionsWithIndex((ix, it) => {
10      | if (ix == 0) it.drop(1)
11      | it
12      | })
13 drop_first: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[39] at
   mapPartitionsWithIndex at <console>:25
14
15 scala> // 分割RDD，并取出第2列员工姓名和第7列实际薪资数据
16
17 scala> val split_first = drop_first.map(line => {val data = line.split(",");
18      | (data(1), data(6).toInt)})
19 split_first: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[40]
   at map at <console>:27
20
21 scala> // 使用sortBy()方法根据实际薪资降序排序
22
23 scala> val sort_first = split_first.sortBy(x => x._2, false)
24 sort_first: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[45]
   at sortBy at <console>:25
25
26 scala> // 取出上半年实际薪资Top3的员工
27
28 scala> sort_first.take(3)
29 res18: Array[(String, Int)] = Array((Alejandro Cantrell,330405), (Brady
   Calhoun,293806), (Alfred Hickman,257302))
30
31 scala>
```

任务3.3查询上半年或下半年实际薪资大于20万元的员工姓名

任务描述

本节的任务如下:

1. 查询上半年或下半年实际薪资大于20万元的员工姓名。
2. 将最终结果合并为一个RDD

使用 union()合并多个RDD

union转换操作，将两个RDD元素合并，不进行去重操作

源码实例

```
val rdd1 = sc.parallelize(List(('a',1),('b',2),('c',3)))
val rdd2 = sc.parallelize(List(('a',1),('d',4),('e',5)))
rdd1.union(rdd2).collect
```

运行结果：

```
scala> val rdd1 = sc.parallelize(List(('a',1),('b',2),('c',3)))
rdd1: org.apache.spark.rdd.RDD[(Char, Int)] = ParallelCollectionRDD[46] at parallelize at <console>:24

scala> val rdd2 = sc.parallelize(List(('a',1),('d',4),('e',5)))
rdd2: org.apache.spark.rdd.RDD[(Char, Int)] = ParallelCollectionRDD[47] at parallelize at <console>:24

scala> rdd1.union(rdd2).collect
res25: Array[(Char, Int)] = Array((a,1), (b,2), (c,3), (a,1), (d,4), (e,5))

scala> res9: Array[(Char, Int)] = Array((a,1), (b,2), (c,3), (a,1), (d,4), (e,5))
```

使用filter()进行过滤

filter 是一种转换操作，需要一个过滤函数作为参数，返回值为true的元素保留

源码实例

```
val rdd1=sc.parallelize(List(('a',1),('b',2),('c',3)))
rdd1.filter(_._2>1).collect
rdd1.filter(x=>x._2>1).collect
```

运行结果：

```
scala> val rdd1=sc.parallelize(List(('a',1),('b',2),('c',3)))
rdd1: org.apache.spark.rdd.RDD[(Char, Int)] = ParallelCollectionRDD[49] at
parallelize at <console>:24

scala> rdd1.filter(_._2>1).collect
res26: Array[(Char, Int)] = Array((b,2), (c,3))

scala> rdd1.filter(x=>x._2>1).collect
res27: Array[(Char, Int)] = Array((b,2), (c,3))

scala>
```

使用distinct()进行去重

distinct是一种转换操作，去除两个完全相同的元素，没有参数

源码实例

```
val rdd=sc.makeRDD(List(('a',1),('b',1),('a',1),('c',1)))
rdd.distinct().collect
```

运行结果：

```
scala> val rdd=sc.makeRDD(List(('a',1),('b',1),('a',1),('c',1)))
rdd: org.apache.spark.rdd.RDD[(Char, Int)] = ParallelCollectionRDD[52] at
makeRDD at <console>:24

scala> rdd.distinct().collect
res28: Array[(Char, Int)] = Array((b,1), (a,1), (c,1))

scala>
```

简单的集合操作

求交集intersetcion()

求出两个RDD共同的元素

源码实例

```
val c_rdd1=sc.parallelize(List(('a',1),('b',1),('a',1),('c',1)))
val c_rdd2=sc.parallelize(List(('a',1),('b',1),('d',1)))
c_rdd1.intersection(c_rdd2).collect
```

运行结果:

```
scala> val c_rdd1=sc.parallelize(List(('a',1),('b',1),('a',1),('c',1)))
c_rdd1: org.apache.spark.rdd.RDD[(Char, Int)] = ParallelCollectionRDD[56] at parallelize at <console>:24

scala> val c_rdd2=sc.parallelize(List(('a',1),('b',1),('d',1)))
c_rdd2: org.apache.spark.rdd.RDD[(Char, Int)] = ParallelCollectionRDD[57] at parallelize at <console>:24

scala> c_rdd1.intersection(c_rdd2).collect
res29: Array[(Char, Int)] = Array((b,1), (a,1))

scala>
```

求并集union()

将两个RDD的元素合并成一个，不进行去重

源码实例

```
val c_rdd1=sc.parallelize(List(('a',1),('b',1),('a',1),('c',1)))
val c_rdd2=sc.parallelize(List(('a',1),('b',1),('d',1)))
c_rdd1.union(c_rdd2).collect
```

运行结果

```
scala> val c_rdd1=sc.parallelize(List(('a',1),('b',1),('a',1),('c',1)))
c_rdd1: org.apache.spark.rdd.RDD[(Char, Int)] = ParallelCollectionRDD[64] at parallelize at <console>:24

scala> val c_rdd2=sc.parallelize(List(('a',1),('b',1),('d',1)))
c_rdd2: org.apache.spark.rdd.RDD[(Char, Int)] = ParallelCollectionRDD[65] at parallelize at <console>:24

scala> c_rdd1.union(c_rdd2).collect
res30: Array[(Char, Int)] = Array((a,1), (b,1), (a,1), (c,1), (a,1), (b,1), (d,1))

scala>
```

求补集substract()

参数是RDD,将原RDD里和参数RDD里相同的元素去掉

源码实例

```
val rdd1 = sc.parallelize(List(('a',1),('b',1),('c',1)))
val rdd2 = sc.parallelize(List(('d',1),('e',1),('c',1)))
rdd1.subtract(rdd2).collect
rdd2.subtract(rdd1).collect
```

运行结果

```
scala> val rdd1 = sc.parallelize(List(('a',1),('b',1),('c',1)))
rdd1: org.apache.spark.rdd.RDD[(Char, Int)] = ParallelCollectionRDD[67] at
parallelize at <console>:24

scala> val rdd2 = sc.parallelize(List(('d',1),('e',1),('c',1)))
rdd2: org.apache.spark.rdd.RDD[(Char, Int)] = ParallelCollectionRDD[68] at
parallelize at <console>:24

scala> rdd1.subtract(rdd2).collect
res31: Array[(Char, Int)] = Array((b,1), (a,1))

scala> rdd2.subtract(rdd1).collect
res32: Array[(Char, Int)] = Array((d,1), (e,1))

scala>
```

求笛卡儿积cartesian()

将两个集合的元素两两组合成一组

源码

```
val rdd01 = sc.parallelize(List(1,3,5,3))
val rdd02 = sc.parallelize(List(2,4,5,1))
rdd01.cartesian(rdd02).collect
```

运行结果：

```
scala> val rdd01 = sc.parallelize(List(1,3,5,3))
rdd01: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[77] at
parallelize at <console>:24

scala> val rdd02 = sc.parallelize(List(2,4,5,1))
rdd02: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[78] at
parallelize at <console>:24

scala> rdd01.cartesian(rdd02).collect
res33: Array[(Int, Int)] = Array((1,2), (1,4), (3,2), (3,4), (1,5), (1,1),
(3,5), (3,1), (5,2), (5,4), (3,2), (3,4), (5,5), (5,1), (3,5), (3,1))

```

任务实现

实现步骤

输出上半年或下半年实际薪资大于20万元的员工姓名。

1. 首先需要过滤出两个RDD中实际薪资大于20万元的员工姓名。
2. 再将两个RDD得到的员工姓名合并到一个RDD中，对员工姓名进行去重

实现源码

```
// 创建RDD
val first_half = sc.textFile("/user/myname/Employee_salary_first_half.csv")
val second_half =
sc.textFile("/user/myname/Employee_salary_second_half.csv")
// 删除首行字段名称数据
val drop_first = first_half.mapPartitionsWithIndex((ix, it) => {
if (ix == 0) it.drop(1)
it
})
val drop_second = second_half.mapPartitionsWithIndex((ix, it) => {
if (ix == 0) it.drop(1)
it
})
// 分割RDD，并取出第2列员工姓名和第7列实际薪资数据
val split_first = drop_first.map(
line => {val data = line.split(",");(data(1), data(6).toInt)})
val split_second = drop_second.map(
line => {val data = line.split(",");(data(1), data(6).toInt)})
// 筛选出上半年或下半年实际薪资大于20万的员工姓名
val filter_first = split_first.filter(x => x._2 > 200000).map(x => x._1)
val filter_second = split_second.filter(x => x._2 > 200000).map(x => x._1)
// 合并两个RDD并去重后输出结果
val name = filter_first.union(filter_second).distinct()
name.collect

```

运行结果

```
scala> // 创建RDD

scala> val first_half =
sc.textFile("/user/myname/Employee_salary_first_half.csv")
first_half: org.apache.spark.rdd.RDD[String] =
/user/myname/Employee_salary_first_half.csv MapPartitionsRDD[81] at textFile
at <console>:24

scala> val second_half =
sc.textFile("/user/myname/Employee_salary_second_half.csv")
second_half: org.apache.spark.rdd.RDD[String] =
/user/myname/Employee_salary_second_half.csv MapPartitionsRDD[83] at
textFile at <console>:24

scala> // 删除首行字段名称数据

scala> val drop_first = first_half.mapPartitionsWithIndex((ix, it) => {
     | if (ix == 0) it.drop(1)
     | it
     | })
drop_first: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[84] at
mapPartitionsWithIndex at <console>:25

scala> val drop_second = second_half.mapPartitionsWithIndex((ix, it) => {

```

```
     | if (ix == 0) it.drop(1)
     | it
     | })
drop_second: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[85] at mapPartitionsWithIndex at <console>:25

scala> // 分割RDD，并取出第2列员工姓名和第7列实际薪资数据

scala> val split_first = drop_first.map(
     | line => {val data = line.split(",");(data(1), data(6).toInt)})
split_first: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[86] at map at <console>:27

scala> val split_second = drop_second.map(
     | line => {val data = line.split(",");(data(1), data(6).toInt)})
split_second: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[87] at map at <console>:27

scala> // 筛选出上半年或下半年实际薪资大于20万的员工姓名

scala> val filter_first = split_first.filter(x => x._2 > 200000).map(x => x._1)
filter_first: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[89] at map at <console>:25

scala> val filter_second = split_second.filter(x => x._2 > 200000).map(x => x._1)
filter_second: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[91] at map at <console>:25

scala> // 合并两个RDD并去重后输出结果

scala> val name = filter_first.union(filter_second).distinct()
name: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[95] at distinct at <console>:27

scala> name.collect
res34: Array[String] = Array(Tyrell Cross, Paris Lozano, Brady Calhoun, Nestor Parsons, Clyde Franklin, Luther Glenn, Alfred Hickman, Alejandro Cantrell, Alfredo Hodges)

scala>
```

任务3.4输出每位员工2020年的总实际薪资

任务描述

- 计算每位员工2020年的总实际薪资，要求对上、下半年员工的实际薪资进行相加

了解键值对RDD

键值对RDD由一组组的键值对组成，这些RDD称为PairRDD

可以把两个RDD中的键相同的元素组合在一起，合并为一个RDD

创建键值对RDD

使用map函数来回返回键值 对RDD，即PairRDD

源码

```
val rdd = sc.parallelize(List("this is a test","How are you","I am fine","Can you tell me"));
val words=rdd.map(x=>(x.split(" ")(0),x));
words.collect
```

运行结果：

```
scala> val rdd = sc.parallelize(List("this is a test","How are you","I am fine","Can you tell me"));
rdd: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[96] at parallelize at <console>:24

scala> val words=rdd.map(x=>(x.split(" ")(0),x));
words: org.apache.spark.rdd.RDD[(String, String)] = MapPartitionsRDD[97] at map at <console>:25

scala> words.collect
res35: Array[(String, String)] = Array((this,this is a test), (How,How are you), (I,I am fine), (Can,Can you tell me))

scala>
```

转换操作keys与values

通过keys返回一个仅包含键的RDD,values返回一个仅包含值 的RDD

源码：

```
val key=words.keys
val value=words.values
key.collect
value.collect
```

运行结果：

```
scala> val key=words.keys
key: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[98] at keys at <console>:25

scala> val value=words.values
value: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[99] at values at <console>:25

scala> key.collect
res36: Array[String] = Array(this, How, I, Can)

scala> value.collect
res37: Array[String] = Array(this is a test, How are you, I am fine, Can you tell me)

scala>
```

转换操作reduceByKey

特点

reduceByKey的功能是合并具有相同键的值

需要接收一个输入函数，根据函数进行合并并且创建一个新的RDD作为返回结果

源码

```
val rdd_1 =sc.parallelize(List(('a',1),('a',2),('b',1),('c',1),('c',1)))
val re_rdd_1=rdd_1.reduceByKey((a,b)=>a+b)
re_rdd_1.collect
```

运行结果

```
scala> val rdd_1 =sc.parallelize(List(('a',1),('a',2),('b',1),('c',1),('c',1)))
rdd_1: org.apache.spark.rdd.RDD[(Char, Int)] = ParallelCollectionRDD[108] at parallelize at <console>:24

scala> val re_rdd_1=rdd_1.reduceByKey((a,b)=>a+b)
re_rdd_1: org.apache.spark.rdd.RDD[(Char, Int)] = ShuffledRDD[109] at reduceByKey at <console>:25

scala> re_rdd_1.collect
res43: Array[(Char, Int)] = Array((b,1), (a,3), (c,2))

scala>
```

转换操作groupByKey

特点

对具有相同的键的值进行分组

源码

```
// 使用groupByKey()方法对具有相同键的值进行分组
val g_rdd = rdd_1.groupByKey()
// 查看分组结果
g_rdd.collect
// 使用map()方法查看分组后每个value中的元素个数
g_rdd.map(x => (x._1, x._2.size)).collect
```

运行结果：

```

scala> // 使用groupByKey()方法对具有相同键的值进行分组

scala> val g_rdd = rdd_1.groupByKey()
g_rdd: org.apache.spark.rdd.RDD[(Char, Iterable[Int])] = ShuffledRDD[110] at groupByKey at <console>:25

scala> // 查看分组结果

scala> g_rdd.collect
res44: Array[(Char, Iterable[Int])] = Array((b,CompactBuffer(1)), (a,CompactBuffer(1, 2)), (c,CompactBuffer(1, 1)))

scala> // 使用map()方法查看分组后每个value中的元素个数

scala> g_rdd.map(x => (x._1, x._2.size)).collect
res45: Array[(Char, Int)] = Array((b,1), (a,2), (c,2))

scala>
```

任务实现

方法

统计每位员工2020年的总实际薪资，首先需要将数据合并到一个RDD中，通过相同的键对同一个员工的上半年实际薪资和下半年实际薪资进行累加，实现步骤如下

1. 获取上、下半年员工薪资数据并将其转换为RDD，分别为split_first和split_second。
2. 使用union()方法将两个RDD合并成一个新的RDD。
3. 通过reduceByKey()方法统计员工总实际薪资并输出结果

源码

```
// 将前面的两个RDD：split_first和split_second 使用union()方法合并
val all_salary = split_first.union(split_second)
// 使用reduceByKey()方法统计员工总薪资
val salary = all_salary.reduceByKey((a, b) => a+b)
salary.collect

```

运行结果

```

scala> // 将前面的两个RDD：split_first和split_second 使用union()方法合并

scala> val all_salary = split_first.union(split_second)
all_salary: org.apache.spark.rdd.RDD[(String, Int)] = UnionRDD[114] at union at <console>:27

scala> // 使用reduceByKey()方法统计员工总薪资

scala> val salary = all_salary.reduceByKey((a, b) => a+b)
salary: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[115] at reduceByKey at <console>:25

scala> salary.collect
res47: Array[(String, Int)] = Array((Bobby Horton,98683), (Victoria Werner,88513), (Harland Murray,302502), (Patsy Ferguson,113279), (Benito Owen,84841), (Alonso Abbott,119544), (Domenic Ross,78651), (Lottie Johnston,97446), (Alyson Cook,119719), (Maxwell Wilcox,98238), (Lorna Bryant,121566), (Nickolas Gill,112686), (Gabriel Knapp,110468), (Royal Hahn,140892), (Houston Montes,263356), (Monroe Washington,153167), (Eldridge Mclaughlin,98007), (Darrin Guzman,242791), (Earnest Park,161030), (Fritz Rivers,107738), (Sammie Porter,183731), (Andre Malone,226959), (Scott Brewer,121620), (Augusta Zimmerman,131190), (Kasey Hoffman,181162), (Aldo Barajas,98424), (Michale Vincent,122753), (Kristy Orozco,100404), (Lynne Mccann,102086), (Wendell Valencia,125453), (Ashlee Coch...

scala>
```

任务3.5查询每位员工2020年的月均实际薪资

任务描述

本节的任务是输出每位员工2020年的每月平均实际薪资，需要先计算每位员工2020年的总实际薪资，再求出每位员工2020年的月均实际薪资

使用join连接两个RDD

特点

1. 将有键的数据与另一组有键的数据一起使用
2. 连接方式有：右外连接、左外连接、全外连接、内连接
3. 与union的元素合并不同，join的连接操作会将键相同的值进行合并

分类

1. join:对两个RDD进行内连接
2. rightOuterJoin:对两个RDD进行连接，确保第二个RDD的键必须存在
3. leftOuterJoin:对两个RDD进行连接，确保第一个RDD的键必须存在
4. fullOuterJoin:对两个RDD进行全外连接

连接

join

方法

根据两个RDD进行内连接，将两个RDD中键相同的数据的值存放在一个元组中，
最后只返回两个RDD都存在的键的连接结果

例如(K,V)和(K,W)通过join后得到(K,(V,W))

测试源码

```
val rdd1=sc.parallelize(List(('a',1),('b',2),('c',3)))
val rdd2=sc.parallelize(List(('a',1),('d',4),('e',5)))
val j_rdd=rdd1.join(rdd2)
j_rdd.collect
```

运行结果：

```
scala> val rdd1=sc.parallelize(List(('a',1),('b',2),('c',3)))
rdd1: org.apache.spark.rdd.RDD[(Char, Int)] = ParallelCollectionRDD[116] at parallelize at <console>:24

scala> val rdd2=sc.parallelize(List(('a',1),('d',4),('e',5)))
rdd2: org.apache.spark.rdd.RDD[(Char, Int)] = ParallelCollectionRDD[117] at parallelize at <console>:24

scala> val j_rdd=rdd1.join(rdd2)
j_rdd: org.apache.spark.rdd.RDD[(Char, (Int, Int))] = MapPartitionsRDD[120] at join at <console>:27

scala> j_rdd.collect
res48: Array[(Char, (Int, Int))] = Array((a,(1,1)))

scala>
```

rightOuterJoin

结果返回第二个RDD的所有键的连接结果，存在的键对应的值为Some类型，不存在的键对应的值为None类型

测试源码

```
val right_join=rdd1 rightOuterJoin rdd2
right_join.collect
```

运行结果：

```
scala> val right_join=rdd1 rightOuterJoin rdd2
right_join: org.apache.spark.rdd.RDD[(Char, (Option[Int], Int))] =
MapPartitionsRDD[123] at rightOuterJoin at <console>:27

scala> right_join.collect
res49: Array[(Char, (Option[Int], Int))] = Array((d,(None,4)), (e,(None,5)),
(a,(Some(1),1)))

scala>
```

leftOuterJoin

结果返回保留第一个RDD的所有键，在rdd2存在的键对应的值为Some类型，不存在的键对应的值为None类型

测试源码

```
val left_join=rdd1 leftOuterJoin rdd2
left_join.collect
```

运行结果

```
scala> val left_join=rdd1 leftOuterJoin rdd2
left_join: org.apache.spark.rdd.RDD[(Char, (Int, Option[Int]))] =
MapPartitionsRDD[126] at leftOuterJoin at <console>:27

scala> left_join.collect
res50: Array[(Char, (Int, Option[Int]))] = Array((b,(2,None)), (a,
(1,Some(1))), (c,(3,None)))

```

fullOuterJoin

全外连接，会保留两个连接的RDD中所有键的连接结果

测试源码

```
val full_join=rdd1 fullOuterJoin rdd2
full_join.collect
```

运行结果：

```
scala> val full_join=rdd1 fullOuterJoin rdd2
full_join: org.apache.spark.rdd.RDD[(Char, (Option[Int], Option[Int]))] =
MapPartitionsRDD[129] at fullOuterJoin at <console>:27

scala> full_join.collect
res51: Array[(Char, (Option[Int], Option[Int]))] = Array((d,(None,Some(4))),
(b,(Some(2),None)), (e,(None,Some(5))), (a,(Some(1),Some(1))), (c,
(Some(3),None)))

scala>
```

使用zip组合两个RDD

特点

zip函数用于将两个RDD组合成Key/Value形式的RDD,这里要求两个RDD的partition数量及元素数量都相同

源码

```
var rdd1 = sc.makeRDD(1 to 5,2)
var rdd2 = sc.makeRDD(Seq("A","B","C","D","E"),2)
rdd1.zip(rdd2).collect
```

运行结果：

```
scala> var rdd1 = sc.makeRDD(1 to 5,2)
rdd1: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[130] at makeRDD
at <console>:24

scala> var rdd2 = sc.makeRDD(Seq("A","B","C","D","E"),2)
rdd2: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[131] at
makeRDD at <console>:24

scala> rdd1.zip(rdd2).collect
res52: Array[(Int, String)] = Array((1,A), (2,B), (3,C), (4,D), (5,E))

```

使用combineByKey合并相同键的值

特点

是比较核心的高级函数，其他一些高阶键值对函数底层都是用它实现的

定义

combineByKey(createCombiner,mergeValue,mergeCombiners,numPartitions=None)

1. createCombiner:V=>C，V是键值对RDD中的值部分，将该值转换为另一种类型的值C，C会作为每一个键的累加器的初始值。
2. mergeValue:(C,V)=>C，该函数将元素V聚合到之前的元素C(createCombiner)上（这个操作在每个分区内进行）。
3. mergeCombiners:(C,C)=>C，该函数将两个元素C进行合并（这个操作在不同分区间进行）

源码

```
val test = sc.parallelize(List(("panda",1),("panda",8),("pink",4),("pink",8),("pirate",5)))
val cb_test = test.combineByKey(
count => (count,1),
(acc:(Int,Int),count) => (acc._1+count,acc._2+1),
(acc1:(Int,Int),acc2:(Int,Int)) => (acc1._1+acc2._1,acc1._2+acc2._2))
cb_test.map(x=>(x._1,x._2._1.toDouble/x._2._2)).collect
```

运行结果

```
scala> val test = sc.parallelize(List(("panda",1),("panda",8),("pink",4),("pink",8),("pirate",5)))
test: org.apache.spark.rdd.RDD[(String, Int)] = ParallelCollectionRDD[133] at parallelize at <console>:24

scala> val cb_test = test.combineByKey(
     | count => (count,1),
     | (acc:(Int,Int),count) => (acc._1+count,acc._2+1),
     | (acc1:(Int,Int),acc2:(Int,Int)) => (acc1._1+acc2._1,acc1._2+acc2._2))
cb_test: org.apache.spark.rdd.RDD[(String, (Int, Int))] = ShuffledRDD[134] at combineByKey at <console>:28

scala> cb_test.map(x=>(x._1,x._2._1.toDouble/x._2._2)).collect
res56: Array[(String, Double)] = Array((panda,4.5), (pink,6.0), (pirate,5.0))

scala>
```

使用lookup查找指定键的值

lookup(k)作用于key/value类型的 RDD上，返回指定key的所有value值

源码

```
val test = sc.parallelize(List(("panda",1),("panda",8),("pink",4),("pink",8),("pirate",5)))
test.lookup("panda")
```

运行结果

```
scala> val test = sc.parallelize(List(("panda",1),("panda",8),("pink",4),("pink",8),("pirate",5)))
test: org.apache.spark.rdd.RDD[(String, Int)] = ParallelCollectionRDD[138] at parallelize at <console>:24

scala> test.lookup("panda")
res59: Seq[Int] = WrappedArray(1, 8)
```

任务实现

步骤

查询每位员工2020年的月均实际薪资需要先筛选出上、下半年的员工薪资数据中的员工姓名和实际薪资两个字段数据并创建RDD，然后将筛选后的两个RDD合并，再根据员工姓名对实际薪资求和，最后查询出2020年的每位员工的月均实际薪资，具体实现步骤如下：

1. 获取两个RDD，即split_first和split_second，使用union()方法合并两个RDD。
2. 使用combineByKey()方法计算每位员工2020年的月均实际薪资.

源码实现

```
// 合并两个RDD：split_first和split_second
val salary = split_first.union(split_second)
// 求每位员工2020年的每月平均实际薪资
val avg_salary = salary.combineByKey(
count => (count, 0),
(acc:(Int, Int), count) => (acc._1 + count, acc._2 + 0),
(acc1:(Int, Int), acc2:(Int, Int)) => (acc1._1 + acc2._1, acc1._2 + acc2._2)
)
avg_salary.map(x => (x._1, x._2._1.toDouble / 12)).collect
```

运行结果

```
scala> // 合并两个RDD：split_first和split_second

scala> val salary = split_first.union(split_second)
salary: org.apache.spark.rdd.RDD[(String, Int)] = UnionRDD[141] at union at <console>:27

scala> // 求每位员工2020年的每月平均实际薪资

scala> val avg_salary = salary.combineByKey(
     | count => (count, 0),
     | (acc:(Int, Int), count) => (acc._1 + count, acc._2 + 0),
     | (acc1:(Int, Int), acc2:(Int, Int)) => (acc1._1 + acc2._1, acc1._2 + acc2._2)
     | )
```

```
avg_salary: org.apache.spark.rdd.RDD[(String, (Int, Int))] =
ShuffledRDD[142] at combineByKey at <console>:28

scala> avg_salary.map(x => (x._1, x._2._1.toDouble / 12)).collect
res60: Array[(String, Double)] = Array((Bobby Horton,8223.583333333334),
(Victoria Werner,7376.083333333333), (Harland Murray,25208.5), (Patsy
Ferguson,9439.916666666666), (Benito Owen,7070.083333333333), (Alonso
Abbott,9962.0), (Domenic Ross,6554.25), (Lottie Johnston,8120.5), (Alyson
Cook,9976.583333333334), (Maxwell Wilcox,8186.5), (Lorna Bryant,10130.5),
(Nickolas Gill,9390.5), (Gabriel Knapp,9205.666666666666), (Royal
Hahn,11741.0), (Houston Montes,21946.333333333332), (Monroe
Washington,12763.916666666666), (Eldridge Mclaughlin,8167.25), (Darrin
Guzman,20232.583333333332), (Earnest Park,13419.166666666666), (Fritz
Rivers,8978.166666666666), (Sammie Porter,15310.916666666666), (Andre
Malone,18913.25), (Scott Brewer,10135.0), (Augusta Zimmerman,10932.5), (K...

scala>
```

任务3.6存储汇总后的员工薪资为文本文件

在实际生产环境中，需要读取的文本格式不仅包含普通的文本文件，还包含其他格式的文件，如JSON、SequenceFile等。此外，当数据计算或处理结束后，通常需要将结果保存，以便后续环节的分析与应用。本节的任务是学习不同格式文件的数据读取和保存，并对员工薪资统计结果进行汇总并存储为文本文件

读取与存储JSON文件

Json文件读取

准备数据下载(**已按本文件开头完成数据准备的，可忽略本处**)

```
mkdir /root/spark/
cd /root/spark/
wget http://biglab.site/b5951Ospark/file/spark_data.tar.gz
tar -xvzf ./spark_data.tar.gz
hdfs dfs -mkdir /user/myname
```

将测试数据上传到hdfs

```
hdfs dfs -put /root/spark/spark_data/testjson.json /user/myname
hdfs dfs -ls /user/myname
```

测试脚本

```
import org.json4s._
import org.json4s.jackson.JsonMethods._
val input = sc.textFile("/user/myname/testjson.json")
case class Person(name:String,age:Int)
implicit val formats = DefaultFormats
val in_json = input.collect.map{x => parse(x).extract[Person]}
```

运行结果：

```

scala> import org.json4s._
import org.json4s._

scala> import org.json4s.jackson.JsonMethods._
import org.json4s.jackson.JsonMethods._

scala> val input = sc.textFile("/user/myname/testjson.json")
input: org.apache.spark.rdd.RDD[String] = /user/myname/testjson.json
MapPartitionsRDD[1] at textFile at <console>:30

scala> case class Person(name:String,age:Int)
defined class Person

scala> implicit val formats = DefaultFormats
formats: org.json4s.DefaultFormats.type =
org.json4s.DefaultFormats$@7e35c8b5

scala> val in_json = input.collect.map{x => parse(x).extract[Person]}
in_json: Array[Person] = Array(Person(jack,12), Person(lili,22),
Person(cc,11), Person(vv,13), Person(lee,14))

scala>
```

Json文件存储

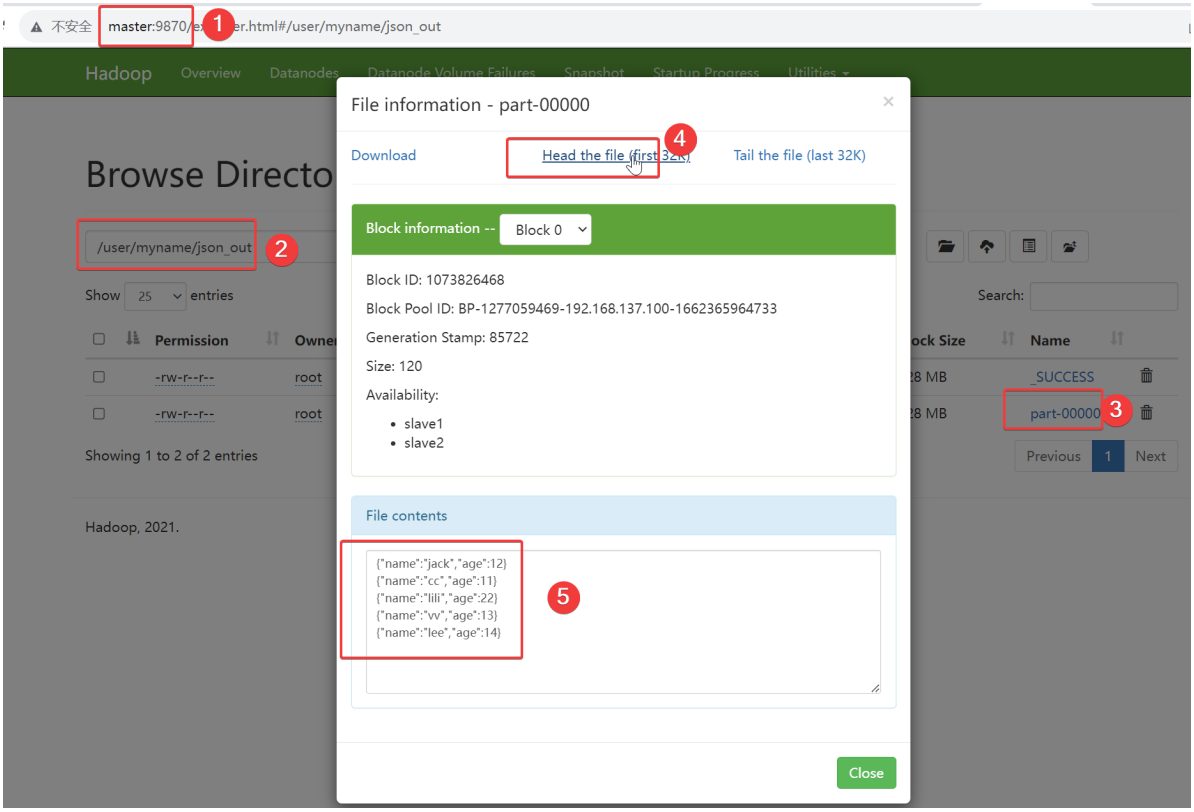
测试脚本

```
import org.json4s.JsonDSL._
val json = in_json.map{x=>("name" -> x.name) ~ ("age" -> x.age)}
val jsons = json.map{x=>compact(render(x))}
sc.parallelize(jsons).repartition(1).saveAsTextFile("/user/myname/json_out")
```

运行结果：

```
scala> import org.json4s.JsonDSL._
import org.json4s.JsonDSL._

scala> val json = in_json.map{x=>("name" -> x.name) ~ ("age" -> x.age)}
json: Array[org.json4s.JsonAST.JObject] =
Array(JObject(List((name,JString(jack)), (age,JInt(12)))),
JObject(List((name,JString(lili)), (age,JInt(22)))),
JObject(List((name,JString(cc)), (age,JInt(11)))),
JObject(List((name,JString(vv)), (age,JInt(13)))),
JObject(List((name,JString(lee)), (age,JInt(14)))))

scala> val jsons = json.map{x=>compact(render(x))}
jsons: Array[String] = Array({"name":"jack","age":12},
{"name":"lili","age":22}, {"name":"cc","age":11}, {"name":"vv","age":13},
{"name":"lee","age":14})

scala>
sc.parallelize(jsons).repartition(1).saveAsTextFile("/user/myname/json_out")

scala>
```

HDFS结果：



读取与存储CSV文件

Csv文件读取

准备数据下载(**已按本文件开头完成数据准备的，可忽略本处**)

```
mkdir /root/spark/
cd /root/spark/
wget http://biglab.site/b59510spark/file/spark_data.tar.gz
tar -xvzf ./spark_data.tar.gz
hdfs dfs -mkdir /user/myname
```

将测试数据上传到hdfs

```
hdfs dfs -put /root/spark/spark_data/testcsv.csv /user/myname
hdfs dfs -ls /user/myname
```

测试脚本_行读

```
import java.io.StringReader
import au.com.bytecode.opencsv.CSVReader
val input = sc.textFile("/user/myname/testcsv.csv")
val result = input.map{line=>val reader = new CSVReader(new StringReader(line));reader.readNext();}
result.collect

```

运行结果_行读

```
scala> import java.io.StringReader
import java.io.StringReader

scala> import au.com.bytecode.opencsv.CSVReader
import au.com.bytecode.opencsv.CSVReader

scala> val input = sc.textFile("/user/myname/testcsv.csv")
input: org.apache.spark.rdd.RDD[String] = /user/myname/testcsv.csv MapPartitionsRDD[15] at textFile at <console>:43

scala> val result = input.map{line=>val reader = new CSVReader(new StringReader(line));reader.readNext();}
result: org.apache.spark.rdd.RDD[Array[String]] = MapPartitionsRDD[16] at map at <console>:44

scala> result.collect
res3: Array[Array[String]] = Array(Array(0, first, first line), Array(1, second, second line))

scala>
```

测试脚本_读全文件

```
import java.io._
import scala.collection.JavaConversions._
import au.com.bytecode.opencsv._
//定义一个CSVData类
case class CSVData(index:String,title:String,content:String)
//读取CSV文件
val csvinput = sc.wholeTextFiles("/user/myname/testcsv.csv")
//解析CSV文件数据
val result = csvinput.flatMap{case(_,txt)=>
    val reader = new CSVReader(new StringReader(txt));
    reader.readAll().map(x=>CSVData(x(0),x(1),x(2)))
}
result.collect
```

运行结果：

```

scala> import java.io._
import java.io._

scala> import scala.collection.JavaConversions._
import scala.collection.JavaConversions._

scala> import au.com.bytecode.opencsv._
import au.com.bytecode.opencsv._

scala> //定义一个CSVData类

scala> case class CSVData(index:String,title:String,content:String)
defined class CSVData

scala> //读取CSV文件

scala> val csvinput = sc.wholeTextFiles("/user/myname/testcsv.csv")
csvinput: org.apache.spark.rdd.RDD[(String, String)] =
/user/myname/testcsv.csv MapPartitionsRDD[1] at wholeTextFiles at
<console>:33

scala> //解析CSV文件数据

scala> val result = csvinput.flatMap{case(_,txt)=>
     |     val reader = new CSVReader(new StringReader(txt));
     |     reader.readAll().map(x=>CSVData(x(0),x(1),x(2)))
     | }
warning: there was one deprecation warning (since 2.12.0); for details,
enable `:setting -deprecation' or `:replay -deprecation'
result: org.apache.spark.rdd.RDD[CSVData] = MapPartitionsRDD[2] at flatMap
at <console>:36

scala> result.collect
res0: Array[CSVData] = Array(CSVData(0,first,first line),
CSVData(1,second,second line))

scala>
```

Csv文件存储

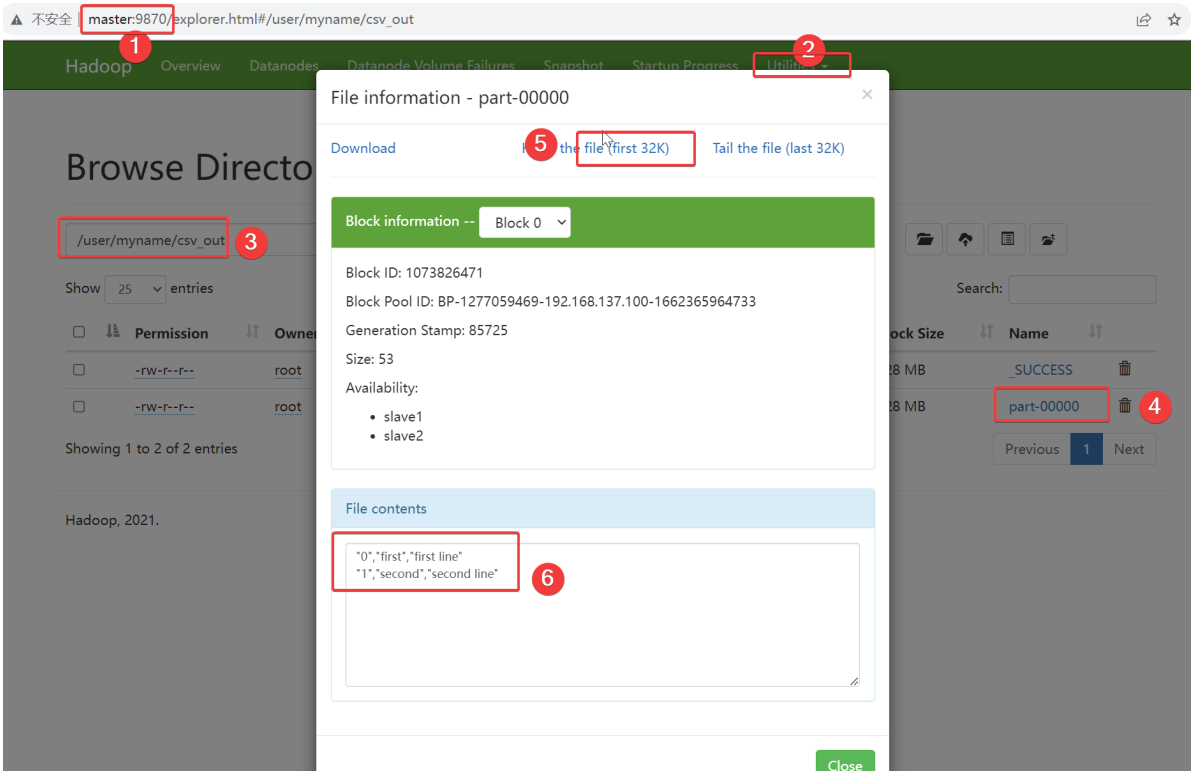
测试脚本

```
import java.io.{StringReader,StringWriter}
import au.com.bytecode.opencsv.{CSVReader,CSVWriter}
result.map(data =>
List(data.index,data.title,data.content).toArray).mapPartitions{data=>
val stringWriter = new StringWriter();
val csvWriter = new CSVWriter(stringWriter);
csvWriter.writeAll(data.toList)
Iterator(stringWriter.toString)}.saveAsTextFile("/user/myname/csv_out")
```

运行结果：

```
scala> import java.io.{StringReader,StringWriter}
import java.io.{StringReader, StringWriter}

scala> import au.com.bytecode.opencsv.{CSVReader,CSVWriter}
import au.com.bytecode.opencsv.{CSVReader, CSVWriter}

scala> result.map(data =>
List(data.index,data.title,data.content).toArray).mapPartitions{data=>
     | val stringWriter = new StringWriter();
     | val csvWriter = new CSVWriter(stringWriter);
     | csvWriter.writeAll(data.toList)
     |
Iterator(stringWriter.toString)}.saveAsTextFile("/user/myname/csv_out")
warning: there was one deprecation warning (since 2.12.0); for details,
enable `:setting -deprecation' or `:replay -deprecation'


scala>
```

HDFS运行结果：



SquenceFile的读取与存储

SquenceFile的存储

首先要保证有一个键值对类型的 RDD,然后直接调用saveSequenceFile(path)保存数据

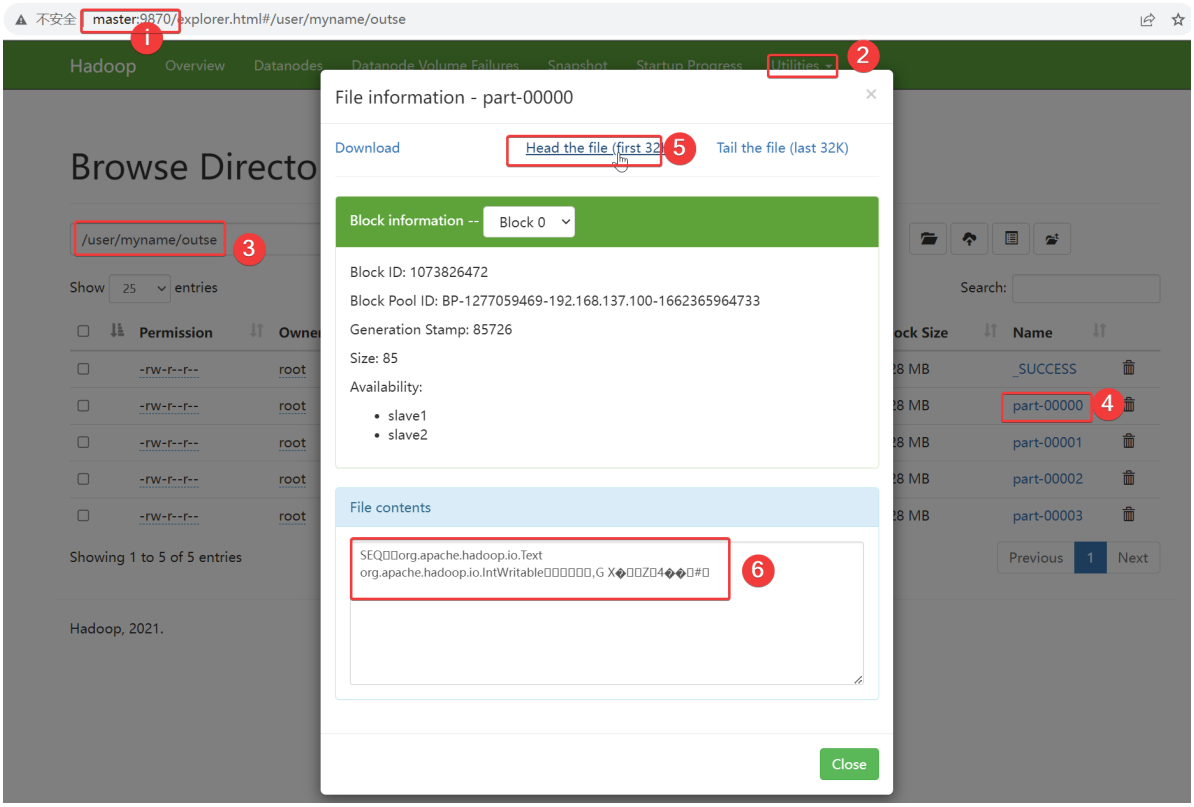
测试脚本

```
import org.apache.hadoop.io.{IntWritable,Text}
val rdd = sc.parallelize(List(("Panda",3),("Kay",6),("Snall",2)))
rdd.saveAsSequenceFile("/user/myname/outse")
```

运行结果：

```
scala> import org.apache.hadoop.io.{IntWritable,Text}
import org.apache.hadoop.io.{IntWritable, Text}

scala> val rdd = sc.parallelize(List(("Panda",3),("Kay",6),("Snall",2)))
rdd: org.apache.spark.rdd.RDD[(String, Int)] = ParallelCollectionRDD[6] at
parallelize at <console>:36

scala> rdd.saveAsSequenceFile("/user/myname/outse")


scala>
```

HDFS上结果文件内容：



SequenceFile文件读取

使用SparkContext的sequenceFile实现

测试脚本

```
val output =
sc.sequenceFile("/user/myname/outse",classOf[Text],classOf[IntWritable]).map{
case (x,y) => (x.toString,y.get())}
output.collect.foreach(println)
```

运行结果：

```
scala> val output =
sc.sequenceFile("/user/myname/outse",classOf[Text],classOf[IntWritable]).map{
case (x,y) => (x.toString,y.get())}
output: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[11] at map
at <console>:36

scala> output.collect.foreach(println)
(Panda,3)
(Kay,6)
(Snall,2)

scala>
```

读取与存储 文本文件

准备数据下载(**已按本文件开头完成数据准备的，可忽略本处**)

```
mkdir /root/spark/
cd /root/spark/
wget http://biglab.site/b59510spark/file/spark_data.tar.gz
tar -xvzf ./spark_data.tar.gz
hdfs dfs -mkdir /user/myname
```

将测试数据上传到hdfs

```
hdfs dfs -put /root/spark/spark_data/bigdata.txt /user/myname
hdfs dfs -ls /user/myname/bigdata.txt
```

使用textFile读文本文件，使用 saveAsTextFile(path)将数据存入文件

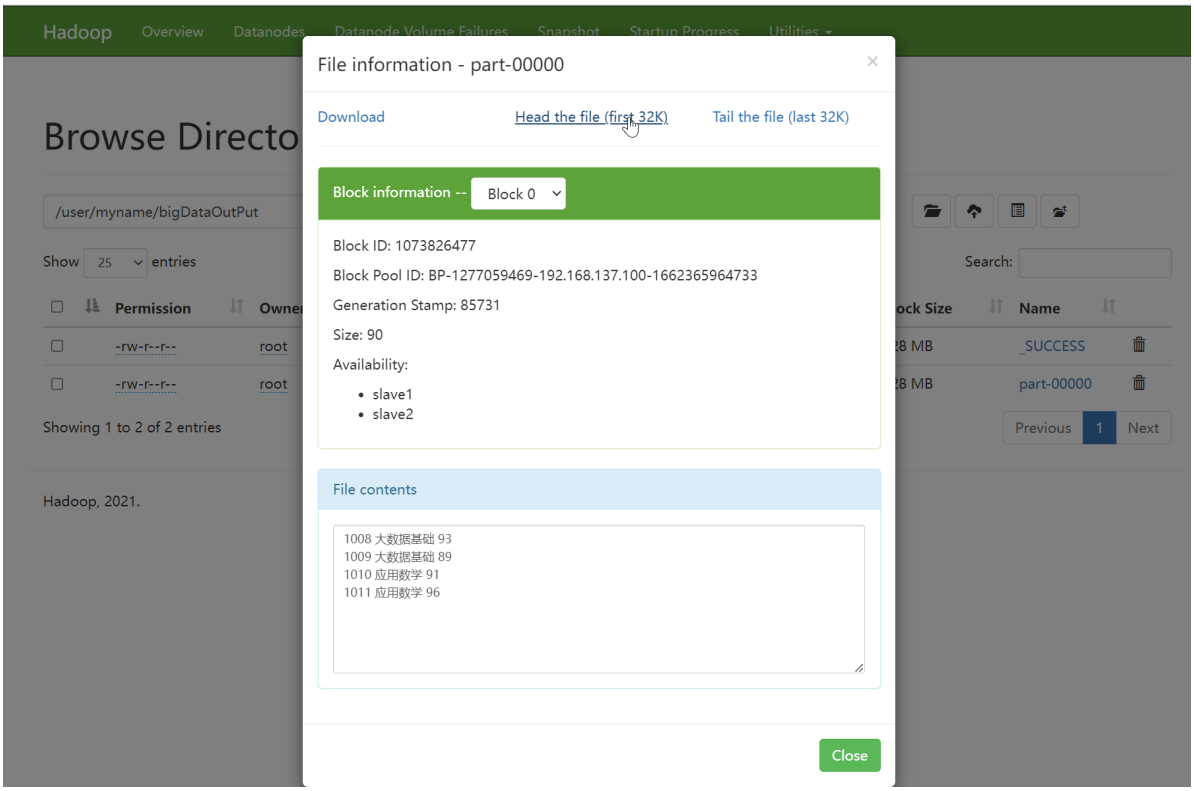
测试脚本

```
// 读取文本文件
val rdd=sc.textFile("/user/myname/bigdata.txt")
// 查看读取的文件内容
rdd.collect
// 文本文件的存储
rdd.repartition(1).saveAsTextFile("/user/myname/bigDataOutPut")
```

运行结果：

```
scala> // 读取文本文件

scala> val rdd=sc.textFile("/user/myname/bigdata.txt")
rdd: org.apache.spark.rdd.RDD[String] = /user/myname/bigdata.txt
MapPartitionsRDD[13] at textFile at <console>:36

scala> // 查看读取的文件内容

scala> rdd.collect
res5: Array[String] = Array(1008 大数据基础 93, 1009 大数据基础 89, 1010 应用数学
91, 1011 应用数学 96)

scala> // 文本文件的存储

scala> rdd.repartition(1).saveAsTextFile("/user/myname/bigDataOutPut")


scala>
```

HDFS上运行输出文件内容：



任务实现

准备数据下载(**已按本文件开头完成数据准备的，可忽略本处**)

```
mkdir /root/spark/
cd /root/spark/
wget http://biglab.site/b59510spark/file/spark_data.tar.gz
tar -xvzf ./spark_data.tar.gz
hdfs dfs -mkdir /user/myname
```

将测试数据上传到hdfs

```
hdfs dfs -put /root/spark/spark_data/Employee_salary_first_half.csv
/user/myname
hdfs dfs -put /root/spark/spark_data/Employee_salary_second_half.csv
/user/myname
hdfs dfs -ls /user/myname/Employee_salary*
```

测试脚本

```
  // 读取上下半年员工薪资数据
  val first_half =
sc.textFile("/user/myname/Employee_salary_first_half.csv")
  val second_half =
sc.textFile("/user/myname/Employee_salary_second_half.csv")
  // 去除首行
  val drop_first = first_half.mapPartitionsWithIndex((ix, it) => {
    if (ix == 0) it.drop(1)
    it
  })
```

```
  val drop_second = second_half.mapPartitionsWithIndex((ix, it) => {
    if (ix == 0) it.drop(1)
    it
  })
  // 获取员工姓名、上半年实际薪资和下半年实际薪资
  val split_first = drop_first.map(
    line => {
      val data = line.split(",");
      (data(1), data(6).toInt)
    })
  val split_second = drop_second.map(
    line => {
      val data = line.split(",");
      (data(1), data(6).toInt)
    })
  // 获取员工2020年总实际薪资
  val all_salary = split_first.union(split_second)
  val salary = all_salary.reduceByKey((a, b) => a + b)
  // 2020年平均薪资
  val avg_salary = all_salary.combineByKey(
    count => (count, 0),
    (acc: (Int, Int), count) => (acc._1 + count, acc._2 + 0),
    (acc1: (Int, Int), acc2: (Int, Int)) => (acc1._1 + acc2._1, acc1._2 + acc2._2)
  )
  val avg = avg_salary.map(x => (x._1, x._2._1.toDouble / 12))
  // 使用join合并所需RDD，并转换数据成（员工姓名，上半年实际薪资，下半年实际薪资，2020年总实际薪资，2020年平均薪资）的形式
  val total = split_first.join(split_second).join(salary).join(avg).map(
    x => Array(x._1, x._2._1._1._1, x._2._1._1._2,
      x._2._1._2, x._2._2).mkString(
      ","))
  // 保存结果成文本文件到HDFS上
  total.repartition(1).saveAsTextFile("/user/myname/total")
```

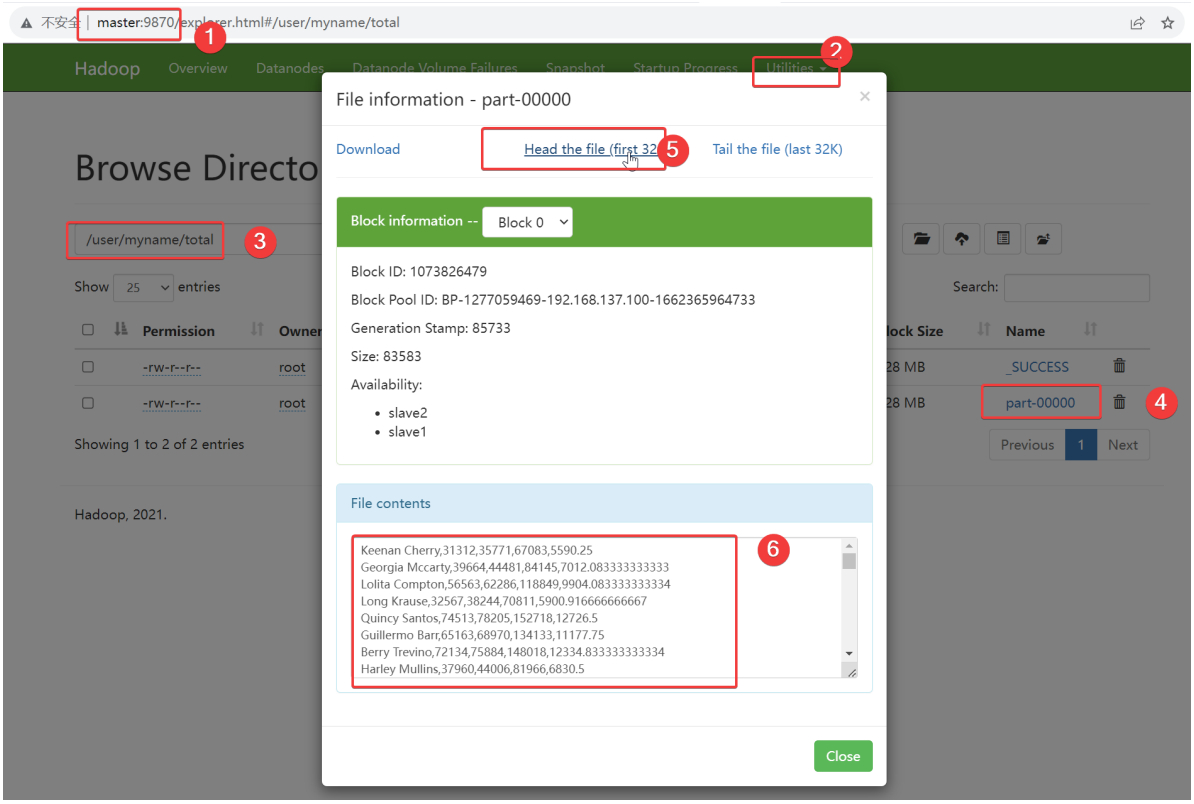
运行结果：

```
scala>   val first_half = sc.textFile("/user/myname/Employee_salary_first_half.csv")
first_half: org.apache.spark.rdd.RDD[String] = /user/myname/Employee_salary_first_half.csv MapPartitionsRDD[36] at textFile at <console>:36

scala>   val second_half = sc.textFile("/user/myname/Employee_salary_second_half.csv")
second_half: org.apache.spark.rdd.RDD[String] = /user/myname/Employee_salary_second_half.csv MapPartitionsRDD[38] at textFile at <console>:36

scala>   // 去除首行

scala>   val drop_first = first_half.mapPartitionsWithIndex((ix, it) => {
     |     if (ix == 0) it.drop(1)
     |     it
     |   })
```

```
drop_first: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[39] at mapPartitionsWithIndex at <console>:37

scala>   val drop_second = second_half.mapPartitionsWithIndex((ix, it) => {
     |     if (ix == 0) it.drop(1)
     |     it
     |   })
drop_second: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[40] at mapPartitionsWithIndex at <console>:37

scala>   // 获取员工姓名、上半年实际薪资和下半年实际薪资

scala>   val split_first = drop_first.map(
     |     line => {
     |       val data = line.split(",");
     |       (data(1), data(6).toInt)
     |     })
split_first: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[41] at map at <console>:37

scala>   val split_second = drop_second.map(
     |     line => {
     |       val data = line.split(",");
     |       (data(1), data(6).toInt)
     |     })
split_second: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[42] at map at <console>:37

scala>   // 获取员工2020年总实际薪资

scala>   val all_salary = split_first.union(split_second)
all_salary: org.apache.spark.rdd.RDD[(String, Int)] = UnionRDD[43] at union at <console>:39

scala>   val salary = all_salary.reduceByKey((a, b) => a + b)
salary: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[44] at reduceByKey at <console>:37

scala>   // 2020年平均薪资

scala>   val avg_salary = all_salary.combineByKey(
     |     count => (count, 0),
     |     (acc: (Int, Int), count) => (acc._1 + count, acc._2 + 0),
     |     (acc1: (Int, Int), acc2: (Int, Int)) => (acc1._1 + acc2._1, acc1._2 + acc2._2)
     |   )
avg_salary: org.apache.spark.rdd.RDD[(String, (Int, Int))] = ShuffledRDD[45] at combineByKey at <console>:40

scala>   val avg = avg_salary.map(x => (x._1, x._2._1.toDouble / 12))
avg: org.apache.spark.rdd.RDD[(String, Double)] = MapPartitionsRDD[46] at map at <console>:37

scala>   // 使用join合并所需RDD，并转换数据成（员工姓名，上半年实际薪资，下半年实际薪资，2020年总实际薪资，2020年平均薪资）的形式
```

```
58
59  scala>   val total =
    split_first.join(split_second).join(salary).join(avg).map(
60       |     x => Array(x._1, x._2._1._1._1, x._2._1._1._2,
61       |       x._2._1._2, x._2._2).mkString(
62       |       ","))
63  total: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[56] at map at
    <console>:43
64
65  scala>   // 保存结果成文本文件到HDFS上
66
67  scala>  total.repartition(1).saveAsTextFile("/user/myname/total")
68
69  scala>
```

HDFS上运行输出文件内容：



实训作业

在spark-shell中实现词频统计

训练要点

1. RDD创建方法。
2. flatMap操作方法。

需求说明

数据文件words.txt如图所示，文件中包含了多行句子，现在要求对文档中的单词计数，并把单词计数超过3的结果存储到HDFS上

words.txt内容如:

WHat is going on there?
I talked to John on email. We talked about some computer stuff that's it.

I went bike riding in the rain, it was not that cold.

We went to the museum in SF yesterday it was $3 to get in and they had
free food. At the same time was a SF Giants game, when we got done we
had to take the train with all the Giants fans, they are 1/2 drunk.

可以在linux下使用wget获取该文件：
wget http://bigdata.hddly.cn/b46488/file/chap3/words.txt

实现思路及步骤

1. 通过textFile的方法读取数据。
2. 通过flatMap将字符串切分成单词。
3. 通过map将单词转化为(单词，1)的形式。
4. 通过reduceByKey将同一一个单词的所有值相加。
5. 通过filter将单词计数大于3的结果过滤出来。
6. 通过saveAsTextFile将结果写人到HDFS

作业要求

1，将测试文件words.txt上传到hdfs上的/user/myname/目录下,并截图(myname要改为自已名字全拼)

2，通过crt进入spark-shell命令窗口，执行词频分析脚本，并截图；
请将词频统计结果文件存于 /user/myname/output_wordcount

3，提交hdfs上生成的结果文件: /user/myname/output_wordcount 内容截图

实现参考

在linux下执行

```
wget http://bigdata.hddly.cn/b46488/file/chap3/words.txt
hdfs dfs -put ./words.txt /user/myname/
```

在spark-shell上执行

```
val worddata=sc.textFile("hdfs://master:9864/user/myname/words.txt").flatMap(x=>x.split(" ")).map(x=>(x,1)).reduceByKey((x,y)=>x+y)

val worddata_3=worddata.filter(x=>x._2>3).map(x=>x._1)

worddata_3.repartition(1).saveAsTextFile("/user/myname/output_wordcount")
```

在hdfs上观察

/user/myname/output_wordcount

版本历史

- Ver1.2-20220927
- Ver1.3-20220928
    - 随堂练习增加：1)以学生成绩创建成RDD2)查询学生成绩表中的前5名3)输出单科成绩为100分的学生ID4)输出每位学生所有科目的总成绩5)输出每位学生的平均成绩6)统计文本中性别为“男”用户数
- Ver1.4-20220930
    - 添加数据准备；修改hdfs上root目录为myname；增加RDD的union操作实例和运行结果
- Ver1.5-20221002
    - 添加视频学习内容

上一章节 下一章节