### 第16课:RDD实战

由于RDD的不可修改的特性，导致RDD的操作与正常面向对象的操作不同，RDD的操作基本分为3大类：transformation，action，contoller

### Transformation

Transformation是通过转化针对已有的RDD创建出新的RDD

map(func):对调用map的RDD数据集中的每个element都使用func，然后返回一个新的RDD,这个返回的数据集是分布式的数据集

filter(func): 对调用filter的RDD数据集中的每个元素都使用func，然后返回一个包含使func为true的元素构成的RDD

flatMap(func):和map差不多，但是flatMap生成的是多个结果

mapPartitions(func):和map很像，但是map是每个element，而mapPartitions是每个partition

mapPartitionsWithSplit(func):和mapPartitions很像，但是func作用的是其中一个split上，所以func中应该有index

sample(withReplacement,faction,seed):抽样

union(otherDataset)：返回一个新的dataset，包含源dataset和给定dataset的元素的集合

distinct([numTasks]):返回一个新的dataset，这个dataset含有的是源dataset中的distinct的element

groupByKey(numTasks):返回(K,Seq[V])，也就是hadoop中reduce函数接受的key-valuelist

reduceByKey(func,[numTasks]):就是用一个给定的reducefunc再作用在groupByKey产生的(K,Seq[V]),比如求和，求平均数

sortByKey([ascending],[numTasks]):按照key来进行排序，是升序还是降序，ascending是boolean类型

join(otherDataset,[numTasks]):当有两个KV的dataset(K,V)和(K,W)，返回的是(K,(V,W))的dataset,numTasks为并发的任务数

cogroup(otherDataset,[numTasks]):当有两个KV的dataset(K,V)和(K,W)，返回的是(K,Seq[V],Seq[W])的dataset,numTasks为并发的任务数

Transformation特性:

lazy优化：由于Tranformation的lazy特性，也就是创建不马上运行，对于框架来说，我有足够的时间查看到尽可能多的步骤，看到的步骤越多，优化的空间就越大。最简单的优化方式就是步骤合并，例如本来的做法是a=b\*3;b=c\*3;c=d\*3;d=3,步骤合并后就是a=3\*3\*3\*3。

### Action

Action操作的目的是得到一个值，或者一个结果

reduce(func)：说白了就是聚集，但是传入的函数是两个参数输入返回一个值，这个函数必须是满足交换律和结合律的

collect()：一般在filter或者足够小的结果的时候，再用collect封装返回一个数组

count():返回的是dataset中的element的个数

first():返回的是dataset中的第一个元素

take(n):返回前n个elements，这个士driverprogram返回的

takeSample(withReplacement，num，seed)：抽样返回一个dataset中的num个元素，随机种子seed

saveAsTextFile（path）：把dataset写到一个textfile中，或者hdfs，或者hdfs支持的文件系统中，spark把每条记录都转换为一行记录，然后写到file中

saveAsSequenceFile(path):只能用在key-value对上，然后生成SequenceFile写到本地或者hadoop文件系统

countByKey()：返回的是key对应的个数的一个map，作用于一个RDD

foreach(func):对dataset中的每个元素都使用func

### Contoller

Contoller动作主要为持久化RDD，例如cache()，persist(),checkpoint();

具体内容在后续刊物中会讲解。

### [Spark WordCount动手实践](#_Toc441349300)

本小节通过IDEA具体逐步调试一个WordCount案例，让学员知道各步骤中RDD的具体类型，并为下一节逐步解析做铺垫

1. 使用的wordCount代码如下：
2. object WordCount {
3. def main (args: Array[String]) {
4. val conf = new SparkConf()//create SparkConf
5. conf.setAppName("Wow,My First Spark App")//set app name
6. conf.setMaster("local")//run local
7. val sc =new SparkContext(conf)
8. val lines =sc.textFile("C://Users//feng//IdeaProjects//WordCount//src//SparkText.txt")
9. val words = lines.flatMap{ lines => lines.split(" ") }
10. val pairs =words.map ( word => (word,1) )
11. val reduce = pairs.reduceByKey(\_+\_)
12. val sort\_1 = reduce.map(pair=>(pair.\_2,pair.\_1))
13. val sort\_2 = sort\_1.sortByKey(false)
14. val sort\_3=sort\_2.map(pair=>(pair.\_2,pair.\_1))
15. val filter=sort\_3.filter(pair=>pair.\_2>2)
16. filter.collect.foreach(wordNumberPair => println(wordNumberPair.\_1+" : "+wordNumberPair.\_2))
17. sc.stop()
18. }
19. }
    1. 程序使用的SparkText.txt文件内容如下

hadoop hadoop hadoop

spark Flink spark

scala scala object

object spark scala

spark spark

Hadoop hadoop

* 1. 程序WordCount调试结果:

通过IDEA的逐步调试，会在调试窗口显示每一行代码具体操作什么类型的RDD，此RDD通过什么依赖关系依赖于父RDD等重要信息(如图2-14所示)，程序运行结果如图2-15所示。

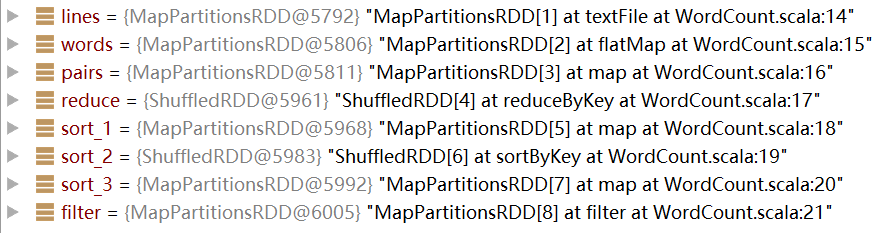


图2-14调试过程图

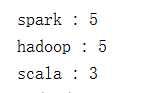


图2-15 wordCount结果

### 2.8.2 [解析RDD生成的内部机制](#_Toc441349301)

本小节基于上小节程序的调试结果，逐条查看调试信息内容，并基于信息内容进行讲解，并在讲解中回顾并复习本章所有内容。

1. line = sc.textFile()

本语句的作用在于从外部数据中读取数据，并生成MapPartitionsRDD。此处需要注意：

如图2-16所示，可以看出次MapPartitionsRDD的deps(dependency,依赖)为HadoopRDD,从这里可以发现其实textFile()过程包含两个步骤，第一步骤将文件内容转化为HadoopRDD(key-value形式，key为行号)，第二步骤将HadoopRDD转化为MapPartitionsRDD(value形式，将key-value类型的key删去)

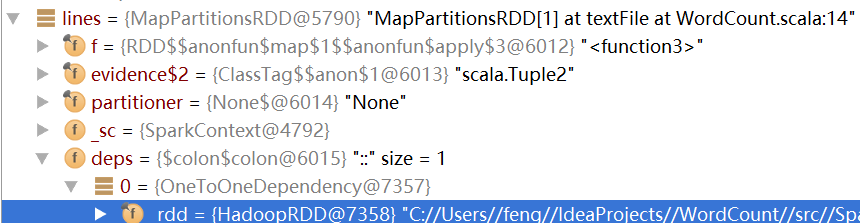


图2-16通过HadoopRDD获取数据

1. words=line.flatMap()

此命令对于RDD采取transformation(转换)操作，作用在于将MapPartitionsRDD中的每一个记录进行以空格为标记的切分，并把每一个RDD的切分的结果放在一个MapPartitionRDD中

1. pairs=words.map(word=>(word,1))

此命令对于RDD采取transformation(转换)操作，作用在于将MapPartitionsRDD中的每一个记录(例：spark(value类型))转换为key-value类型(例: (spark,1)),便于下一步reduceByKey操作

1. reduce = pairs.reduceByKey(\_+\_)

此命令对于RDD采取action(动作)操作，作用在于通过shuffle将pairs中所有的记录按照key相同value相加的规则进行处理，并把结果放到一个shuffleRDD中。例((spark,1),(spark,1))变成((spark,2))。

同时需要注意一下两点：首先本步骤实质上分为两个步骤，第一步骤为local级别的reduce,对当前计算机所拥有的数据先进行reduce操作，生成MapPartitionsRDD;第二步骤为shuffle级别的reduce,基于第一步骤的结果，对结果进行shuffle-reduce操作，生成最终的shuffleRDD。其次 Action操作进行时，对此操作之前的所有转换操作进行执行，所以调试过程中会出现此前的除textFile操作的执行时间均非常短，说明RDD转换操作不直接进行运算。

1. sort\_1 = reduce.map(pair=>(pair.\_2,pair.\_1))

此命令对于RDD采取transformation(转换)操作，作用在于将shuffleRDD中的每一个记录的key和value互换,生成一个新的MapPartitionsRDD。例: (spark,2)变为(2,spark)

1. sort\_2 = sort\_1.sortByKey(false)

此命令对于RDD采取action(动作)操作，作用在于将MapPartitionsRDD根据key进行排序，并生成shuffleRDD

1. sort\_3=sort\_2.map(pair=>(pair.\_2,pair.\_1))

此命令对于RDD采取transformation(转换)操作，作用在于将shuffleRDD中的每一个记录的key和value互换,生成一个新的MapPartitionsRDD。例: (2,spark)变为(spark,2)

1. filter=sort\_3.filter(pair=>pair.\_2>2)

此命令对于RDD采取transformation(转换)操作，作用在于根据value值筛选MapPartitionsRDD中的数据，输出value大于2的记录

1. 最后通过collect()方法将结果收集后，使用foreach()方法遍历数据并通过println()方法打印出所有数据。