第17课：RDD案例(常用算子解释)

1. 创建RDD

代码：

**def** sparkContext(name:String)=  
{  
 **val** conf = **new** SparkConf().setAppName(name).setMaster(**"local"**)  
 **val** sc = **new** SparkContext(conf)  
 sc  
}

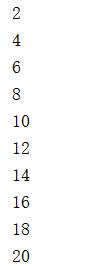
1. Map

作用：适用于任何集合，且对其作用的集合中的每一个元素循环遍历，并调用其作为参数的函数对每一个遍历的元素进行具体化处理。

代码：

**def** mapTransformation(sc:SparkContext): Unit ={  
 **val** nums = sc.parallelize(1 to 10)*//根据集合创建RDD* **val** mapped = nums.map(item=> 2 \* item)  
 mapped.collect.foreach(*println*)  
}

结果：

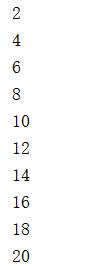


1. Filter

作用：遍历集合中的所有元素，将每个元素作为参数放入函数中进行判断，将判断结果为真的元素筛选出来。

代码：

**def** filterTransformation(sc:SparkContext): Unit ={  
 **val** nums = sc.parallelize(1 to 20)*//根据集合创建RDD* **val** filtered = nums.filter(item => item % 2 == 0)  
 filtered.collect.foreach(*println*)  
}

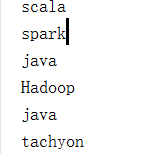


1. Flatmap

作用：通过传入的作为参数的函数来作用与RDD的每个字符串进行单词切分，然后把切分后的结果合并成一个大的集合

代码：

**def** flatmapTransformation(sc:SparkContext): Unit ={  
 **val** bigData = *Array*(**"scala"**,**"spark"**,**"java Hadoop"**,**"java tachyon"**)  
 **val** bigDataString =sc.parallelize(bigData)  
 **val** words= bigDataString.flatMap(line=>line.split(**" "**))  
 words.collect.foreach(*println*)  
}

结果：  


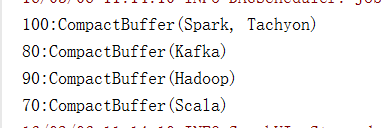
1. groupByKey

作用:将传入的tuple数组生成为RDD，通过groupByKey方法将RDD通过key进行分组汇总，并生成一个新的RDD

代码：

**def** groupByKeyTransformation(sc:SparkContext): Unit ={  
 **val** data = *Array*(*Tuple2*(100,**"Spark"**),*Tuple2*(100,**"Tachyon"**),*Tuple2*(90,**"Hadoop"**),*Tuple2*(80,**"Kafka"**),*Tuple2*(70,**"Scala"**))  
 **val** dataRDD = sc.parallelize(data)  
 **val** group = dataRDD.groupByKey()  
 group.collect.foreach(pair=>*println*(pair.\_1+**":"**+pair.\_2))  
}

结果：



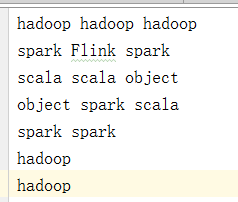
1. reduceByKey

作用：对key相同的元素进行value值得相加。

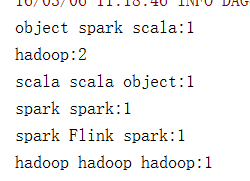
代码：

**def** reduceByKeyTransformation(sc:SparkContext): Unit ={  
 **val** lines =sc.textFile(**"C://Users//feng//IdeaProjects//WordCount//src//SparkText.txt"**,1)  
 **val** reduce= lines.map(line=>(line,1)).reduceByKey(\_+\_)  
 reduce.collect.foreach(pair=>*println*(pair.\_1+**":"**+pair.\_2))  
}

文件内容：



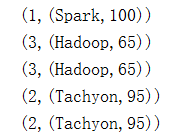
结果：



1. Join

作用：根据相同key,把不同的RDD合并为一个RDD  
代码：

**def** joinTransformation(sc:SparkContext): Unit ={  
 *//大数据中最重要的算子* **val** studentNames=*Array*(  
 *Tuple2*(1,**"Spark"**),  
 *Tuple2*(2,**"Tachyon"**),  
 *Tuple2*(3,**"Hadoop"**)  
 )  
 **val** studentScore=*Array*(  
 *Tuple2*(1,100),  
 *Tuple2*(2,95),  
 *Tuple2*(3,65),  
 *Tuple2*(2,95),  
 *Tuple2*(3,65)  
 )  
 **val** names = sc.parallelize(studentNames)  
 **val** scores = sc.parallelize(studentScore)  
 **val** studentNameAndScore=names.join(scores)  
 studentNameAndScore.collect.foreach(*println*)  
}  
结果：



1. cogroup

作用：协同分组，首先将两个RDD的内容进行join,在此基础上，以ID为key的情况下将改ID内容的所有分数聚合到一起。

代码：

**def** cogroupTransformation(sc:SparkContext): Unit ={  
 **val** nameList = *Array*(  
 *Tuple2*(1,**"Spark"**),  
 *Tuple2*(2,**"Scala"**),  
 *Tuple2*(3,**"Hadoop"**)  
 )  
 **val** scoreList = *Array*(  
 *Tuple2*(1,100),  
 *Tuple2*(2,90),  
 *Tuple2*(3,87),  
 *Tuple2*(1,80),  
 *Tuple2*(2,90),  
 *Tuple2*(2,60)  
 )  
 **val** names = sc.parallelize(nameList)  
 **val** scores =sc.parallelize(scoreList)  
 **val** nameScores= names.cogroup(scores)  
 nameScores.collect.foreach(*println*)  
}

结果：

