SparkRDD 的 Transformations/Actions 操作实战

在前面Spark编程原理及RDD的特性与基本操作介绍了SparkRDD的操作分为两个部分Transformatio和Action。这两种操作分为多个算子（即操作函数)。Transformation针对已有的RDD创建一个新的RD，主要是对数据进行映射，变换，统计，过滤。。。Action主要是对数据进行最后的执行操作，遍，聚合，保存等操作。下面来看下这些操作的具体实现。

Transformations

Map(func):对RDD中的每个元素通过函数func进行映射。

```
/**
 * map算子：给集合每个元素*2
 */
private static void mapTest() {
    SparkConf conf = new SparkConf().setAppName("mapTest").setMaster("local");
    JavaSparkContext jsc = new JavaSparkContext(conf);

    List<Integer> list = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8);
    JavaRDD<Integer> numRDD = jsc.parallelize(list);
    JavaRDD<Integer> num2RDD = numRDD.map(num -> num * 2);

    num2RDD.foreach(num2 -> System.out.printf(num2.toString() + " "));
    jsc.close();
}
/**
 * map算子：给集合每个元素*2
 */
def mapTest(){
    val conf = new SparkConf().setAppName("mapTest").setMaster("local")
    val sc   = new SparkContext(conf)
    val list = Array(1,2,3,4,5,6,7,8)
    val numRDD = sc.parallelize(list)
    val num2RDD = numRDD.map(num=>num*2)
    num2RDD.foreach(num2=>print(num2+" "))
}
map结果：  2 4 6 8 10 12 14 16
```

filter:过滤或者选择满足条件的数据

```
/**
 * fileter算子：满足过滤条件的保留
 */
private static void filterTest() {
    SparkConf conf = new SparkConf().setAppName("filterTest").setMaster("local");
    JavaSparkContext jsc = new JavaSparkContext(conf);

    List<Integer> list = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
    JavaRDD<Integer> numRDD = jsc.parallelize(list);
    //过滤偶数集合
    JavaRDD<Integer> num2RDD = numRDD.filter(num -> num % 2 == 0);
    num2RDD.foreach(num2 -> System.out.print(num2 + " "));
    jsc.close();
}
/**
 * fileter算子：满足过滤条件的保留
 */
def filterTest(): Unit ={
  val conf = new SparkConf().setAppName("filterTest").setMaster("local")
  val sc    = new SparkContext(conf)
  val list = Array(1,2,3,4,5,6,7,8,9,10)
  val numRDD = sc.parallelize(list)
  //filter过滤偶数
  val evenNumRDD = numRDD.filter(num=> num % 2 == 0)
  evenNumRDD.foreach(evennNum => print(evennNum + " "))
}
filter结果： 2 4 6 8 10
```

flatMap:将一个输入映射成0-多个输出

```
/**
 * flatmap算子：接收RDD中所有元素，并进行运算，然后返回多个元素
 * 这里是对元素分割成单词
 */
private static void flatMapTest() {
    SparkConf conf = new SparkConf().setAppName("flatMapTest").setMaster("local");
    JavaSparkContext jsc = new JavaSparkContext(conf);

    List<String> list = Arrays.asList("hello world", "hello you", "good morning");
    JavaRDD<String> wordsRDD = jsc.parallelize(list);
    //将每条数据通过空格分隔程多个数据
    JavaRDD<String> wordRDD = wordsRDD.flatMap(
            words -> Arrays.asList(words.split(" ")).listIterator()
    );

    wordRDD.foreach(
            word -> System.out.printf(word+" ")
    );
    jsc.close();
}
/**
 * flatmap算子：接收RDD中所有元素，并进行运算，然后返回多个元素
 * 这里是对元素分割成单词
 */
def flatMapTest(): Unit ={
  val conf = new SparkConf().setMaster("local").setAppName("flatMapTest")
  val sc = new SparkContext(conf)
  val list = Array("hello world", "hello you", "good morning")
  val wordsRDD = sc.parallelize(list)
  val wordRDD = wordsRDD.flatMap(words=>words.split(" "))
  wordRDD.foreach(word=>println(word).toString)
}
flatMap结果:
  hello
  world
  hello
  you
  good
  morning
```

groupByKey:按k分组(k,v),(k,v2),(k2,v)(k3,v2) =>(k,(v,v)),(k2,(v)),(k3,(v2)),sortByKey:按key排序

```
/**
 * groupByKey算子：按key分组,
 * sortByKey:按key排序,无参由小到大，参数false,由大到小
 */
private static void groupByKeyTest() {
    SparkConf conf = new SparkConf().setAppName("groupByKeyTest").setMaster("local");
    JavaSparkContext jsc = new JavaSparkContext(conf);

    List<Tuple2<String,Integer>> list = Arrays.asList(
            new Tuple2<>("A",88),
            new Tuple2<>("B",78),
            new Tuple2<>("C",55),
            new Tuple2<>("A",95),
            new Tuple2<>("C",34)
    );
    JavaPairRDD<String,Integer> scoreRDD = jsc.parallelizePairs(list);

    //不加sortBykey()结果乱序
    //JavaPairRDD<String,Iterable<Integer>> scoreGroupRDD = scoreRDD.groupByKey();
    JavaPairRDD<String,Iterable<Integer>> scoreGroupRDD = scoreRDD.groupByKey().sortBy
ey();

    scoreGroupRDD.foreach(
            scoreGroup-> {
                System.out.printf(scoreGroup._1 + ": ");
                scoreGroup._2.forEach(score-> System.out.printf(score+" "));
                System.out.println();
            }
    );
    jsc.close();
}
/**
 * groupByKey算子：按key分组,
 * sortByKey:按key排序,无参由小到大，参数false,由大到小
 */
def groupByKeyTest(): Unit ={
  val conf = new SparkConf().setAppName("groupByKeyTest").setMaster("local")
  val sc = new SparkContext(conf)
  val list = Array(
    Tuple2("A",89),
    Tuple2("c",59),
    Tuple2("B",74),
    Tuple2("c",50),
    Tuple2("A",98))
  val scoreRDD = sc.parallelize(list)
  //不加sortByKey结果乱序
  // val groupRDD = scoreRDD.groupByKey()
  val groupRDD = scoreRDD.groupByKey().sortByKey()
  groupRDD.foreach(
    scoreTP=> {
      print(scoreTP._1 + ": ")
      scoreTP._2.foreach(score=>print(score+" "))
      println()
    })
```

```
groupByKey结果
A: 88 95
B: 78
C: 55 34
```

reduceByKey（func）:key相同V用func函数聚合，这

```
/**
 * reduceByKey算子：按key分组并聚合(聚合函数可以自己定义)这里是相加，即key相同则V相加，（k
,2）,(k1,3),(k2,5) => (K1,5),(k2,5)
 */
private static void reduceByKeyTest() {
    SparkConf conf = new SparkConf().setAppName("reduceByKeyTest").setMaster("local");
    JavaSparkContext jsc = new JavaSparkContext(conf);
    List<Tuple2<String,Integer>> list = Arrays.asList(
            new Tuple2<>("A",88),
            new Tuple2<>("B",78),
            new Tuple2<>("C",55),
            new Tuple2<>("A",95),
            new Tuple2<>("C",34)
    );
    JavaPairRDD<String,Integer> scoreRDD = jsc.parallelizePairs(list);
    //定义的聚合函数是：v相加
    JavaPairRDD<String,Integer> scoreGroupRDD = scoreRDD.reduceByKey((v1,v2)->v1+v2);
    scoreGroupRDD.foreach(
            score->System.out.println(score._1+": "+score._2)
    );
    jsc.close();
}
/**
 * reduceByKey算子：按key分组并聚合(聚合函数可以自己定义)这里是相加，即key相同则V相加，（k
,2）,(k1,3),(k2,5) => (K1,5),(k2,5)
 */
def reduceByKeyTest(): Unit ={
  val conf = new SparkConf().setAppName("reduceByKeyTest").setMaster("local")
  val sc = new SparkContext(conf)
  val list = Array(
    Tuple2("A",89),
    Tuple2("c",59),
    Tuple2("B",74),
    Tuple2("c",50),
    Tuple2("A",98))

  val scoreRDD = sc.parallelize(list)
  val totalRDD = scoreRDD.reduceByKey(_+_).sortByKey()
  totalRDD.foreach(
    total=>println(total._1+": "+total._2)
  )
}
reduceByKey结果:
(B,78)
(A,183)
(C,89)
```

join和cogroup:对RDD进行连接

```
/**
 * join算子：关联兩個RDD，(K,V).join(K,W)=>(K,(V,W)),都存在的K，V保留。其他的丢弃,相当于做集。
 * cogroup算子;相当于集合+，做并集。只要k存在就有一条记录。
 */
private static void joinAndCoGroupTest(){
    SparkConf conf = new SparkConf().setMaster("local").setAppName("joinAndCoGroupTest");
    JavaSparkContext jsc = new JavaSparkContext(conf);

    List<Tuple2<Integer,Integer>> scoreList = Arrays.asList(
            new Tuple2<>(1,88),
            new Tuple2<>(2,98),
            new Tuple2<>(3,75),
            new Tuple2<>(4,84),
            new Tuple2<>(6,77)
    );
    List<Tuple2<Integer,String>> stuList = Arrays.asList(
            new Tuple2<>(1,"A"),
            new Tuple2<>(2,"B"),
            new Tuple2<>(3,"C"),
            new Tuple2<>(4,"D"),
            new Tuple2<>(5,"E")
    );
    JavaPairRDD<Integer,Integer> scoreRDD = jsc.parallelizePairs(scoreList);
    JavaPairRDD<Integer,String> stuRDD = jsc.parallelizePairs(stuList);

    //join链接操作(通过Key做链接，保留都存在的key)
    JavaPairRDD  scoreStuRDD= scoreRDD.join(stuRDD).sortByKey();
    scoreStuRDD.foreach(
            score-> System.out.println(score.toString())
    );

    JavaPairRDD  stuScoreRDD= stuRDD.join(scoreRDD).sortByKey();
    scoreStuRDD.foreach(
            score-> System.out.println(score.toString())
    );

    //cogroup连接操作:保留所有的key
    JavaPairRDD  scoreStuCGRDD= scoreRDD.cogroup(stuRDD).sortByKey();
    scoreStuCGRDD.foreach(
            score-> System.out.println(score.toString())
    );
    JavaPairRDD  stuScoreCGRDD= stuRDD.cogroup(scoreRDD).sortByKey();
    scoreStuCGRDD.foreach(
            score-> System.out.println(score.toString())
    );
  }
  /**
   * join算子：关联兩個RDD，(K,V).join(K,W)=>(K,(V,W)),都存在的K，V保留。其他的丢弃,相当于做
  集。
   * cogroup算子;相当于集合+，做并集。只要k存在就有一条记录。
   */
  def joinAndCoGroupTest(): Unit ={
    val conf = new SparkConf().setAppName("joinAndCoGroupTest").setMaster("local")
    val sc = new SparkContext(conf)

    val scoreList = Array(
      Tuple2(1,89),
      Tuple2(2,59),
      Tuple2(3,74),
      Tuple2(4,50),
      Tuple2(5,98)
    )
    val stuList = Array(
      Tuple2(1,"A"),
      Tuple2(2,"B"),
      Tuple2(3,"C"),
      Tuple2(4,"D"),
      Tuple2(6,"E")
    )
    val scoreRDD = sc.parallelize(scoreList)
    val stuRDD = sc.parallelize(stuList)
    //join联合两个RDD
    val joinRDD = scoreRDD.join(stuRDD).sortByKey()
    joinRDD.foreach(
      join=>println(join.toString())
    )
    //cogroup联合两个RDD
    val cogroupRDD = scoreRDD.cogroup(stuRDD).sortByKey()
    cogroupRDD.foreach(
      co=>println(co.toString())
    )
  }
  join结果:
    (1,(88,A))
    (2,(98,B))
    (3,(75,C))
    (4,(84,D))
  cogroup结果:
    (1,([88],[A]))
    (2,([98],[B]))
    (3,([75],[C]))
```

```
 (4,([84],[D]))
 (5,([],[E]))
 (6,([77],[]))
```

Actions算子

reduce算子：

```
/**
 * reduce算子：1，2聚合结果和3聚合结果和4聚合，递归
 */
private static void reduceTest(){
    SparkConf conf = new SparkConf().setAppName("reduceTest").setMaster("local");
    JavaSparkContext jsc = new JavaSparkContext(conf);

    List<Integer> numList = Arrays.asList(1,2,3,4,5,6,7,8,9);
    JavaRDD<Integer> numRDD = jsc.parallelize(numList);

    Integer sumRDD = numRDD.reduce((x1, x2)->x1+x2);
    System.out.printf(sumRDD.toString());

    jsc.close();
}
/**
 * reduce算子：1，2聚合结果和3聚合结果和4聚合，递归
 */
def reduceTest(): Unit ={
    val conf = new SparkConf().setAppName("reduceTest").setMaster("local")
    val sc = new SparkContext(conf)
    val numList = Array(1,2,3,4,5,6,7,8,9)
    val parRDD = sc.parallelize(numList)
    val sum = parRDD.reduce(_+_)
    println(sum)
}
reduce结果： 45
```

count和collect:count统计RDD元素个数，collect将RDD据拉取到本地

```
/**
 * count:统计RDD元素数量
 * collect算子：将RDD数据拉取到本地,大量数据时，性能比较差，高Io，或者oom内存溢出
 */
private static void collectTest(){
    SparkConf conf = new SparkConf().setAppName("collectTest").setMaster("local");
    JavaSparkContext jsc = new JavaSparkContext(conf);

    List<Integer> numList = Arrays.asList(1,2,3,4,5,6,7,8,9);
    JavaRDD<Integer> numRDD = jsc.parallelize(numList);
    //count:
    long count = numRDD.count();
    System.out.printf("元素个数："+count);

    //map:元素翻倍
    JavaRDD<Integer> doubleNumRDD = numRDD.map(x->x*2);

    //collect:将RDD的数据拉取到本地，变成了java的List,
    List<Integer> listNum= doubleNumRDD.collect();
    listNum.forEach(
            num-> System.out.printf(num.toString()+" ")
    );

    jsc.close();
}
/**
 * count:统计RDD元素数量
 * collect算子：将RDD数据拉取到本地,大量数据时，性能比较差，高Io，或者oom内存溢出
 */
def collectTest(): Unit ={
  val conf = new SparkConf().setAppName("collectTest").setMaster("local")
  val sc = new SparkContext(conf)
  val numList = Array(1,2,3,4,5,6,7,8,9)
  val numRDD = sc.parallelize(numList)
  //count
  val count = numRDD.count()
  println("元素个数："+count);
  val doubleRDD = numRDD.map(x=>x*2)
  //collect
  val listNum =  doubleRDD.collect()
  listNum.foreach(
    num=>println(num+" ")
  )
}

count结果：  元素个数：9
collect结果：  2 4 6 8 10 12 14 16 18
```

tabke(N)算子:取RDD的前N条记录

```
/**
 * take(N),类似collect,把RDD拉取到本地，取前N条数据
 */
private  static void takeTest(){
    SparkConf conf = new SparkConf().setAppName("takeTest").setMaster("local");
    JavaSparkContext jsc = new JavaSparkContext(conf);

    List<Integer> numList = Arrays.asList(1,2,3,4,5,6,7,8,9);
    JavaRDD<Integer> numRDD = jsc.parallelize(numList);

    //take(N):
    List<Integer> list = numRDD.take(8);
    list.forEach(
            num-> System.out.printf(num+" ")
    );
    jsc.close();
}
/**
 * take(N),类似collect,把RDD拉取到本地，取前N条数据
 */
def takeTest(): Unit = {
  val conf = new SparkConf().setAppName("takeTest").setMaster("local")
  val sc = new SparkContext(conf)
  val numList = Array(1, 2, 3, 4, 5, 6, 7, 8, 9)
  val numRDD = sc.parallelize(numList)
  val num5 = numRDD.take(5)
  for (n <- num5) {
    println(n)
  }
 // num5.foreach(n => print(n + " "))
}
take(8)结果：  1 2 3 4 5 6 7 8
```

saveAsTextFile算子:保存RDD到本地

```
/**
 * saveAsTextFile:保存到文件
 */
private static void saveAsTextFileTest(){
    SparkConf conf = new SparkConf().setAppName("saveAsTextFile").setMaster("local");
    JavaSparkContext jsc = new JavaSparkContext(conf);

    List<String > list = Arrays.asList("hello world","how are you","nice to");
    JavaRDD<String> strRDD =  jsc.parallelize(list);
    //saveAsTextFile
    strRDD.saveAsTextFile("input/str");

    jsc.close();
}
/**
 * saveAsTextFile:保存到文件
 */
def saveAsTextFileTest(): Unit = {
  val conf = new SparkConf().setAppName("takeTest").setMaster("local")
  val sc = new SparkContext(conf)
  val numList = Array("TaskSchedulerImpl","DAGScheduler")
  val numRDD = sc.parallelize(numList)
  numRDD.saveAsTextFile("input/scalasaveAsTextFile")
}
```

countByKey:统计key相同的记录（相当于对K分组）

```
/**
 * countBykey:统计每个key的数量
 */
private static void countByKeyTest(){
    SparkConf conf = new SparkConf().setAppName("countByKeyTest").setMaster("local");
    JavaSparkContext jsc = new JavaSparkContext(conf);

    List<Tuple2<Integer,Integer>> numList = Arrays.asList(
            new Tuple2<>(1,100),
            new Tuple2<>(2,50),
            new Tuple2<>(1,100),
            new Tuple2<>(4,80),
            new Tuple2<>(5,50),
            new Tuple2<>(6,80)
    );
    //如果数据是tuple类型的则要用parallelizePairs来并行化，JavaPairRDD来接收类型，而你不是Ja
aRDD,
    JavaPairRDD<Integer,Integer> numRDD  = jsc.parallelizePairs(numList);
    Map<Integer,Long> count = numRDD.countByKey();
    count.forEach((k,v)-> System.out.println(k+"--"+v));
    jsc.close();
}
/**
 * countBykey:统计每个key的数量
 */
def countByKeyTest(): Unit = {
  val conf = new SparkConf().setAppName("takeTest").setMaster("local")
  val sc = new SparkContext(conf)
  val numList = Array("TaskSchedulerImpl","DAGScheduler","TaskSchedulerImpl")
  val numRDD = sc.parallelize(numList)

  numRDD.countByValue().foreach(num=>println(num._1+"--"+num._2));
}
countByKey结果:
5--1
1--2
6--1
2--1
4--1
```