MOBIN

离开舒适区，坚持不懈，持续学习！！！

Spark函数详解系列之RDD基本转换

摘要：

RDD：弹性分布式数据集，是一种特殊集合，支持多种来源，有容错机制，可以被缓存，支持并行操作，一个RDD代表一个分区里的数据集

RDD有两种操作算子：

Transformation（转换）：Transformation属于延迟计算，当一个RDD转换成另一个RDD时并没有立即进行转换，仅仅是记住了数据集的逻辑操作

Ation（执行）：触发Spark作业的运行，真正触发转换算子的计算

本系列主要讲解Spark中常用的函数操作：

1.RDD基本转换

2.键-值RDD转换

3.Action操作篇

本节所讲函数

1.map(func)

2.flatMap(func)
3.mapPartitions(func)
4.mapPartitionsWithIndex(func)
5.simple(withReplacement,fraction,seed)
6.union(ortherDataset)
7.intersection(otherDataset)
8.distinct([numTasks])
9.cartesian(otherDataset)
10.coalesce(numPartitions，shuffle)
11.repartition(numPartition)
12.glom()
13.randomSplit(weight:Array[Double],seed)

基础转换操作：

1.map(func)：数据集中的每个元素经过用户自定义的函数转换形成一个新的RDD，新的RDD叫MappedRDD

（例1）

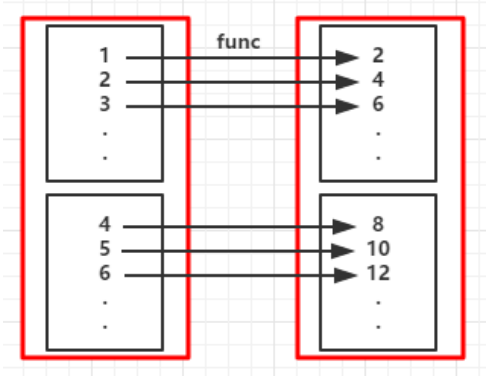
```
object Map {
  def main(args: Array[String]) {
    val conf = new SparkConf().setMaster("local").setAppName("map")
    val sc = new SparkContext(conf)
    val rdd = sc.parallelize(1 to 10)  //创建RDD
    val map = rdd.map(_*2)             //对RDD中的每个元素都乘于2
    map.foreach(x => print(x+" "))
    sc.stop()
  }
}
```

输出：

```
2 4 6 8 10 12 14 16 18 20
```

(RDD依赖图：红色块表示一个RDD区，黑色块表示该分区集合，下同)

14 0



2.flatMap(func):与map类似，但每个元素输入项都可以被映射到0个或多个的输出项，最终将结果"扁平化"后输出
（例2）

```
//...省略sc
   val rdd = sc.parallelize(1 to 5)
   val fm = rdd.flatMap(x => (1 to x)).collect()
   fm.foreach( x => print(x + " "))
```

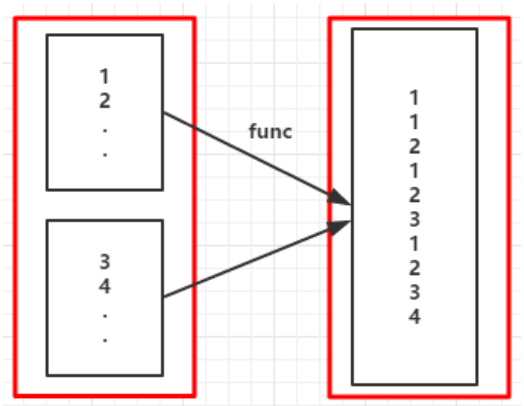
输出：

```
1 1 2 1 2 3 1 2 3 4 1 2 3 4 5
```

如果是map函数其输出如下：

```
Range(1) Range(1, 2) Range(1, 2, 3) Range(1, 2, 3, 4) Range(1, 2, 3, 4, 5)
```

(RDD依赖图)



3.mapPartitions(func):类似与map，map作用于每个分区的每个元素，但mapPartitions作用于每个分区工
func的类型：Iterator[T] => Iterator[U]
假设有N个元素，有M个分区，那么map的函数的将被调用N次,而mapPartitions被调用M次,当在映射的过程中不断的创建对象时就可以使用mapPartitions比map的效率要高很多，比如当向数据库写入数据时，如果使用map就需要为每个元素创建connection对象，但使用mapPartitions的话就需要为每个分区创建connetcion对象
(例3)：输出有女性的名字：

```
object MapPartitions {
//定义函数
  def partitionsFun(/*index : Int,*/iter : Iterator[(String,String)]) : Iterator[String] = {
    var woman = List[String]()
    while (iter.hasNext){
      val next = iter.next()
      next match {
        case (_,"female") => woman = /*"["+index+"]"+*/next._1 :: woman
        case _ =>
      }
    }
    return  woman.iterator
  }

  def main(args: Array[String]) {
```

14 0

随笔档案

文章分类

积分与排名

积分 - 87178
排名 - 3955

最新评论

1. Re:Actor模型原理
二楼的例子更简单
--nuc093
2. Re:Spark常用函数讲解之键值RDD转换
讲的很好，受教了。
--Super<John
3. Re:Hive2.0函数大全(中文版)
收藏了多谢
--哈士奇说喵
4. Re:Hive2.0函数大全(中文版)
非常方便，感谢
--柏原森森
5. Re:图解堆排序
for(int i = len/2 - 1; i >=0; i --){ //堆构造
这里的应该用arr.length / 2，而不应该再拿arr.length减1了吧
--cumtli

阅读排行榜

评论排行榜

```
    val conf = new SparkConf().setMaster("local").setAppName("mappartitions")
    val sc = new SparkContext(conf)
    val l = List(("kpop","female"),("zorro","male"),("mobin","male"),("lucy","female"))
    val rdd = sc.parallelize(l,2)
    val mp = rdd.mapPartitions(partitionsFun)
    /*val mp = rdd.mapPartitionsWithIndex(partitionsFun)*/
    mp.collect.foreach(x => (print(x +" ")))    //将分区中的元素转换成Aarray再输出
  }
}
```

输出：

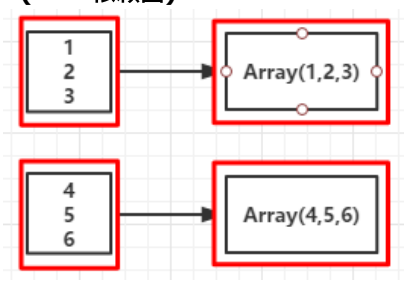
```
kpop lucy
```

其实这个效果可以用一条语句完成

```
val mp = rdd.mapPartitions(x => x.filter(_._2 == "female")).map(x => x._1)
```

之所以不那么做是为了演示函数的定义

(RDD依赖图)



4.mapPartitionsWithIndex(func):与mapPartitions类似，不同的时函数多了个分区索引的参数
func类型：(Int, Iterator[T]) => Iterator[U]
（例4）：将例3橙色的注释部分去掉即是
输出：（带了分区索引）

```
[0]kpop [1]lucy
```

5.sample(withReplacement,fraction,seed):以指定的随机种子随机抽样出数量为fraction的数据，withReplacement表示是抽出的数据是否放回，true为有放回的抽样，false为无放回的抽样
(例5)：从RDD中随机且有放回的抽出50%的数据，随机种子值为3（即可能以1 2 3的其中一个起始值）

```
//省略
    val rdd = sc.parallelize(1 to 10)
    val sample1 = rdd.sample(true,0.5,3)
    sample1.collect.foreach(x => print(x + " "))
    sc.stop
```

6.union(ortherDataset):将两个RDD中的数据集进行合并，最终返回两个RDD的并集，若RDD中存在相同的元素也不会去重

```
//省略sc
   val rdd1 = sc.parallelize(1 to 3)
   val rdd2 = sc.parallelize(3 to 5)
   val unionRDD = rdd1.union(rdd2)
   unionRDD.collect.foreach(x => print(x + " "))
   sc.stop
```

输出：

```
1 2 3 3 4 5
```

7.intersection(otherDataset):返回两个RDD的交集

```
//省略sc
val rdd1 = sc.parallelize(1 to 3)
val rdd2 = sc.parallelize(3 to 5)
val unionRDD = rdd1.intersection(rdd2)
```

14 0

推荐排行榜

1. 图解快速排序(36)
2. 深度剖析JDK动态代理机制(30)
3. java并发编程--Executor框架(25)
4. Hive2.0函数大全(中文版)(20)
5. Spark函数详解系列之RDD基本转换(14)

```
unionRDD.collect.foreach(x => print(x + " "))
sc.stop
```

输出：

```
3 4
```

8.distinct([numTasks]):对RDD中的元素进行去重

```
//省略sc
val list = List(1,1,2,5,2,9,6,1)
val distinctRDD = sc.parallelize(list)
val unionRDD = distinctRDD.distinct()
unionRDD.collect.foreach(x => print(x + " "))
```

输出：

```
1 6 9 5 2
```

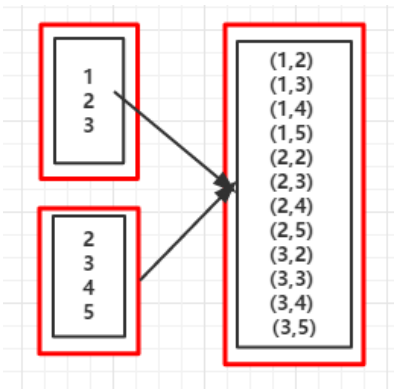
9.cartesian(otherDataset):对两个RDD中的所有元素进行笛卡尔积操作

```
//省略
val rdd1 = sc.parallelize(1 to 3)
val rdd2 = sc.parallelize(2 to 5)
val cartesianRDD = rdd1.cartesian(rdd2)
cartesianRDD.foreach(x => println(x + " "))
```

输出：

```
(1,2)
(1,3)
(1,4)
(1,5)
(2,2)
(2,3)
(2,4)
(2,5)
(3,2)
(3,3)
(3,4)
(3,5)
```

（RDD依赖图）



10.coalesce(numPartitions，shuffle):对RDD的分区进行重新分区，shuffle默认值为false,当shuffle=false时，不能增加分区数目,但不会报错，只是分区个数还是原来的

(例9:）**shuffle=false**

```
//省略
val rdd = sc.parallelize(1 to 16,4)
val coalesceRDD = rdd.coalesce(3) //当suffle的值为false时，不能增加分区数(即
```

14 0

```
4 println("重新分区后的分区个数:"+coalesceRDD.partitions.size)
```

输出:

```
重新分区后的分区个数:3
//分区后的数据集
List(1, 2, 3, 4)
List(5, 6, 7, 8)
List(9, 10, 11, 12, 13, 14, 15, 16)
```

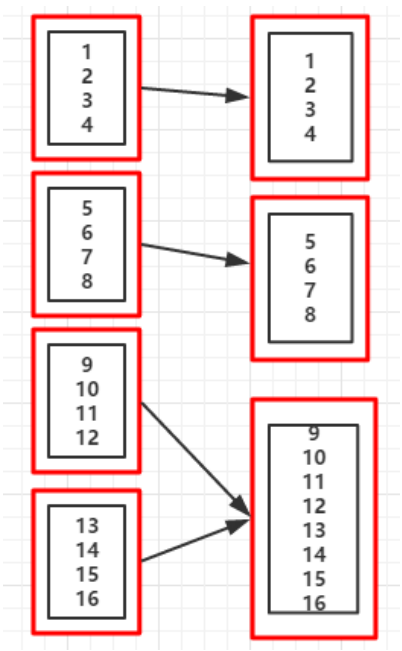
(例9.1:）**shuffle=true**

```
1 //...省略
2 val rdd = sc.parallelize(1 to 16,4)
3 val coalesceRDD = rdd.coalesce(7,true)
4 println("重新分区后的分区个数:"+coalesceRDD.partitions.size)
5 println("RDD依赖关系:"+coalesceRDD.toDebugString)
```

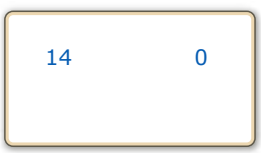
输出:

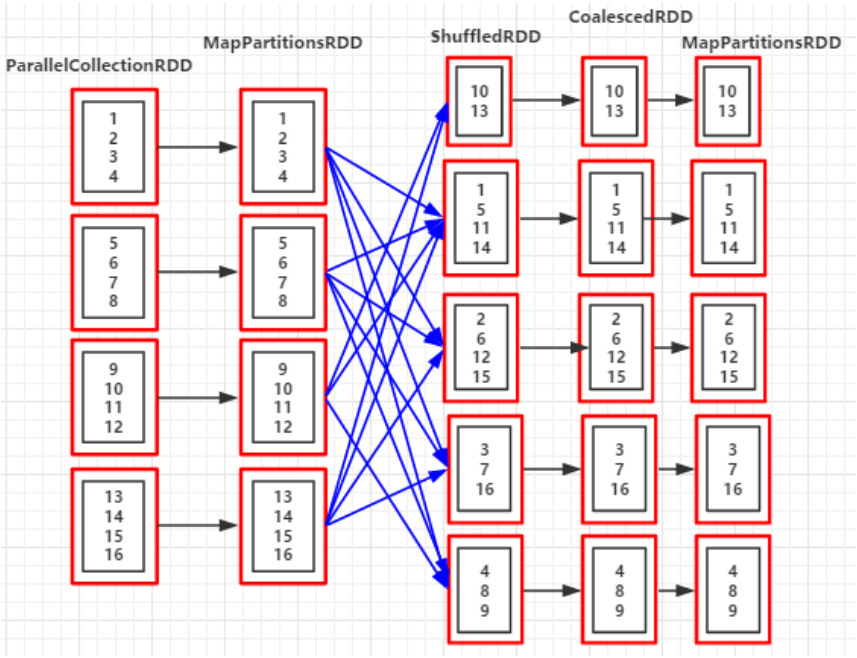
```
重新分区后的分区个数:5
RDD依赖关系:(5) MapPartitionsRDD[4] at coalesce at Coalesce.scala:14 []
| CoalescedRDD[3] at coalesce at Coalesce.scala:14 []
| ShuffledRDD[2] at coalesce at Coalesce.scala:14 []
+-(4) MapPartitionsRDD[1] at coalesce at Coalesce.scala:14 []
| ParallelCollectionRDD[0] at parallelize at Coalesce.scala:13 []
//分区后的数据集
List(10, 13)
List(1, 5, 11, 14)
List(2, 6, 12, 15)
List(3, 7, 16)
List(4, 8, 9)
```

(RDD依赖图:coalesce(3,flase))



(RDD依赖图:coalesce(3,true))





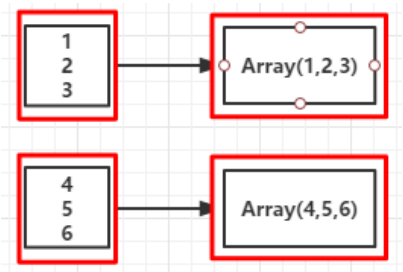
11.repartition(numPartition):是函数coalesce(numPartition,true)的实现，效果和例9.1的coalesce(numPartition,true)的一样

12.glom():将RDD的每个分区中的类型为T的元素转换换数组Array[T]

```
//省略
val rdd = sc.parallelize(1 to 16,4)
val glomRDD = rdd.glom() //RDD[Array[T]]
glomRDD.foreach(rdd => println(rdd.getClass.getSimpleName))
sc.stop
```

输出：

```
int[] //说明RDD中的元素被转换成数组Array[Int]
```



13.randomSplit(weight:Array[Double],seed):根据weight权重值将一个RDD划分成多个RDD,权重越高划分得到的元素较多的几率就越大

```
//省略sc
val rdd = sc.parallelize(1 to 10)
val randomSplitRDD = rdd.randomSplit(Array(1.0,2.0,7.0))
randomSplitRDD(0).foreach(x => print(x +" "))
randomSplitRDD(1).foreach(x => print(x +" "))
randomSplitRDD(2).foreach(x => print(x +" "))
sc.stop
```

输出：

```
2 4
3 8 9
1 5 6 7 10
```

以上例子源码地址：https://github.com/Mobin-F/SparkExample/tree/master/src/main/scala/com/mobin/SparkRDDFun/TransFormation/KVRDD