一、 弹性分布式数据集 RDD

1. RDD 概述

1.1. 什么是 RDD

RDD（Resilient Distributed Dataset）弹性、可复原的分布式数据集，是 Spark 中最基本的数据抽象，代表一个不可变的、可分区的、可并行计算的集合。

RDD 具有数据流模型的特点：自动容错、位置感知调度和可伸缩性。

RDD 允许用户执行查询时将数据集缓存到内存中（Spark 计算快的重要原因之一），后续的查询重用该数据集，极大提升查询速度。

1.2. RDD 的属性

```
* Internally, each RDD is characterized by five main properties:
*
*  - A list of partitions
*  - A function for computing each split
*  - A list of dependencies on other RDDs
*  - Optionally, a Partitioner for key-value RDDs (e.g. to say that the RDD is hash-partitioned)
*  - Optionally, a list of preferred locations to compute each split on (e.g. block locations for
*    an HDFS file)
```

1）RDD 包含一系列 Partition 分区。分区是 RDD 的基本组成单位，数据都是存储在分区中。RDD 包含的每个分区都会被计算任务处理，且分区数决定并行计算的粒度。可以在创建 RDD 时指定分区数，若没有指定，则默认为程序所分配到的 CPU 核数。（一个分区只属于一台机器，一台机器可包含多个分区）

2）函数是作用到每个分区上。Spark 中 RDD 的计算是以分区为单位，且一个分区对应 hdfs 中的一个数据块（对应 mapreduce 中的一个 mapper）。

3）RDD 之间存在依赖关系。RDD 的容错机制：部分分区数据丢失时，可以通过依赖关系重新计算丢失的分区数据，而不是对所有分区重新计算。

4）分区器 partitioner 作用在 KV 格式的 RDD 上。Spark 实现了两种类型的分区器：基于哈希的 HashPartitioner(默认)、基于范围的 RangePartitioner。只有 KV 类型的 RDD 才有 Partitioner；非 KV 类型 RDD 的 Parititioner 为 None。分区器不但决定 RDD 自身的分区数，而且决定父 RDD Shuffle 输出时的分区数。mapreduce 的 partitioner 决定了数据写到哪个 reducer；spark 的 partitioner 决定数据属于哪个分区。

5）RDD 提供最佳计算位置。若读取 HDFS 数据，数据块的位置即是分区的最佳计算位置。按照“移动计算而不移动数据”的理念，Spark 任务调度会尽可能将计算任务启动到数据所在机器，实现数据本地化。

2. 创建 RDD

1）通过并行化的方式将 Scala 集合或数组转化为 RDD：

```
val rdd1 = sc.parallelize(Array(1,2,3,4,5,6,7,8))
```

2）从外部存储系统（本地文件系统、Hadoop 等）读取数据创建 RDD：

```
val rdd2 = sc.textFile("hdfs://node1.itcast.cn:9000/words.txt")
```

3. RDD 编程 API

RDD 中的算子（方法）包括 Transformation 和 Action。Transformation 延迟加载（lazy）；Action 立即执行，触发 Action 时将任务提交到集群上运行。

Scala 原生集合上的方法操作单机的数组或集合；Spark 程序调用 RDD 上的方法，操作分布式数据集。(两者功能一样，实现不同）

3.1. Transformation

RDD 中所有转换都是延迟加载，即不会直接计算结果，只是记录应用到数据集上的转换动作，只有触发返回结果给 Driver 的动作时，才真正执行这些转换。这种设计让 Spark 运行更加高效。

常用的 Transformation：

转换	含义
map(func)	返回新的 RDD，该 RDD 由每一个输入元素经过 func 函数转换而成（对每个分区中的数据进行某种类型的操作）
filter(func)	返回新的 RDD，该 RDD 由 func 函数计算返回 true 的输入元素组成（对每个分区中的数据进行过滤）
flatMap(func)	类似于 map，但是每一个输入可以映射为 0 个或多个输出（func 返回一个序列，而不是单个元素）
mapPartitions(func)	类似于 map，在 RDD 的每个分区上运行，在类型为 T 的 RDD 上运行时，func 函数的类型必须是 Iterator[T] => Iterator[U]
mapPartitionsWithIndex(func)	类似于 mapPartitions，func 带有一个整型参数的分区索引，在类型为 T 的 RDD 上运行时，func 函数的类型必须是(Int, Interator[T]) => Iterator[U]
sample(withReplacement, fraction, seed)	根据 fraction 指定的比例对数据进行采样，可以选择是否使用随机数进行替换，seed 用于指定随机数生成器种子
union(otherDataset)	对源 RDD 和参数 RDD 求并集，返回新的 RDD

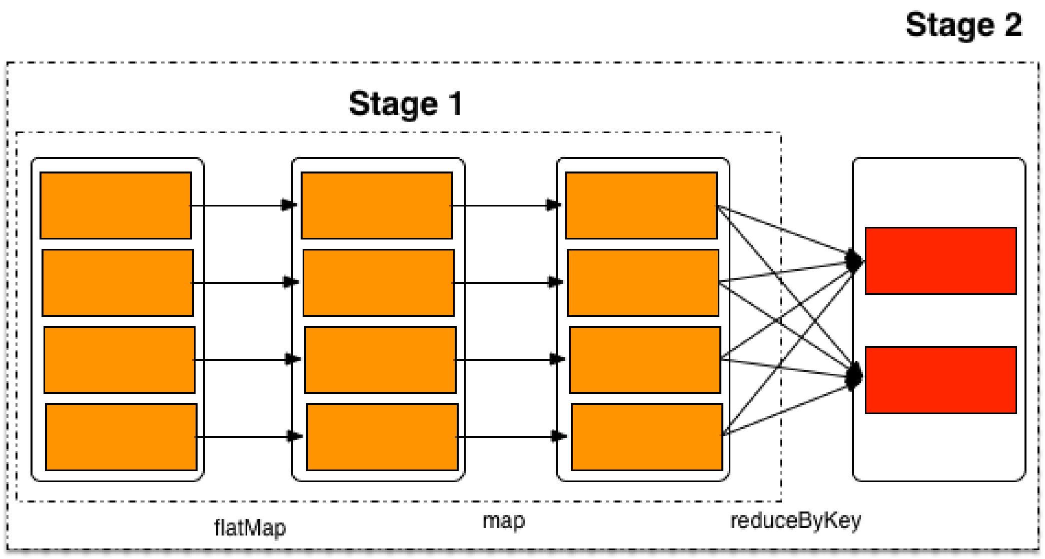
intersection(otherDataset)	对源 RDD 和参数 RDD 求交集，返回新的 RDD
distinct([numTasks]))	对源 RDD 去重，返回新的 RDD
groupByKey([numTasks])	作用在(K,V)的 RDD 上，返回(K, Iterator[V])的 RDD
reduceByKey(func, [numTasks])	作用在(K,V)的 RDD 上，返回(K,V)的 RDD。使用指定的 reduce 函数聚合相同 key 的 value。与 groupByKey 类似，reduce 任务的个数可以通过可选参数来设置
aggregateByKey(zeroValue)(seqOp, combOp, [numTasks])	由自定义函数决定聚合方式。reduceByKey 和 aggregateByKey 底层都调用了 combineByKey
sortByKey([ascending], [numTasks])	作用在(K,V)的 RDD 上，K 必须实现 Ordered 接口，返回按 key 排序的(K,V)的 RDD
sortBy(func,[ascending], [numTasks])	与 sortByKey 类似，更灵活
join(otherDataset, [numTasks])	作用在(K,V)和(K,W)的 RDD 上，返回相同 key 对应所有元素对的(K,(V,W))的 RDD
cogroup(otherDataset, [numTasks])	作用在(K,V)和(K,W)的 RDD 上，返回(K,(Iterable<V>,Iterable<W>))类型的 RDD
cartesian(otherDataset)	笛卡尔积
pipe(command, [envVars])	
coalesce(numPartitions)	重新分区
repartition(numPartitions)	重新分区，底层调用 coalesce，默认 shuffle
repartitionAndSortWithinPartitions(partitioner)	重新分区，并在分区内排序

3.2. Action

动作	含义
reduce(func)	func 函数聚合 RDD 的所有元素
collect()	在驱动程序中以数组的形式返回数据集的所有元素
count()	返回 RDD 的元素个数
first()	返回 RDD 的第一个元素（类似于 take(1)）
take(n)	返回数据集的前 n 个元素组成的数组
takeSample(withReplacement,num, [seed])	返回数据集中随机采样 num 个元素组成数组，可以选择是否使用随机数替换，seed 用于指定随机数生成器种子
top	先排序再取值（降序）
takeOrdered(n, [ordering])	先排序再取前 n 个（升序）
saveAsTextFile(path)	将数据集中的元素以 textfile 的格式保存到 HDFS 文件系统或支持的其他文件系统。对于每个元素，Spark 调用 toString()方法转换为文本。
saveAsSequenceFile(path)	将数据集中的元素以 Hadoop sequencefile 的格式保存到指定目录，可以使用 HDFS 或支持的其他文件系统。
saveAsObjectFile(path)	

countByKey()	作用在(K,V)类型的 RDD 上，返回(K,Int)类型的 map，表示每一个 key 对应的元素个数
foreach(func)	func 函数作用在数据集的每一个元素上。区别于 map 返回一个新的 RDD，foreach 则是取出每一条数据

3.3. WordCount 中的 RDD



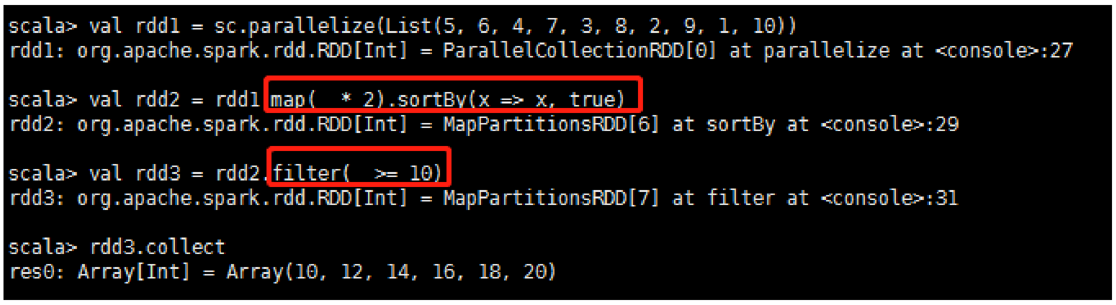
3.4. 练习

启动 spark-shell：

```
/usr/local/spark-1.5.2-bin-hadoop2.6/bin/spark-shell --master spark://node1.itcast.cn:7077
```

练习 1：

```
//通过并行化方式生成 rdd
val rdd1 = sc.parallelize(List(5, 6, 4, 7, 3, 8, 2, 9, 1, 10))
//对 rdd1 的每一个元素乘 2 然后排序
val rdd2 = rdd1.map(_ * 2).sortBy(x => x, true)
//过滤出大于等于 10 的元素
val rdd3 = rdd2.filter(_ >= 10)
//将元素以数组的方式在客户端显示
rdd3.collect
```



```
scala> val rdd1 = sc.parallelize(List(5, 6, 4, 7, 3, 8, 2, 9, 1, 10))
rdd1: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[0] at parallelize at <console>:27

scala> val rdd2 = rdd1.map(_ * 2).sortBy(x => x, true)
rdd2: org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD[6] at sortBy at <console>:29

scala> val rdd3 = rdd2.filter(_ >= 10)
rdd3: org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD[7] at filter at <console>:31

scala> rdd3.collect
res0: Array[Int] = Array(10, 12, 14, 16, 18, 20)
```

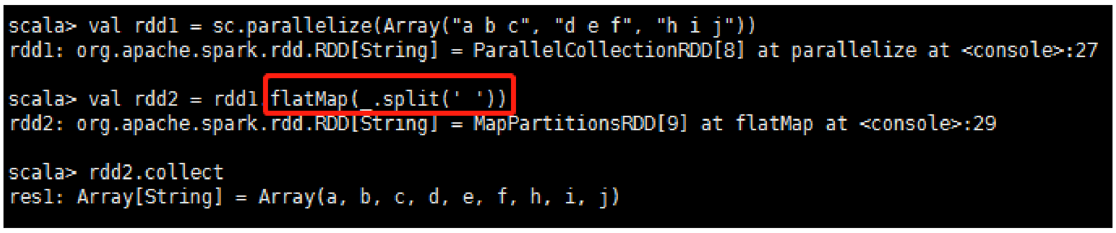
练习 2：

val rdd1 = sc.parallelize(Array("a b c", "d e f", "h i j"))

//将 rdd1 的每一个元素先切分再压平

val rdd2 = rdd1.flatMap(_.split(' '))

rdd2.collect



```
scala> val rdd1 = sc.parallelize(Array("a b c", "d e f", "h i j"))
rdd1: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[8] at parallelize at <console>:27

scala> val rdd2 = rdd1.flatMap(_.split(' '))
rdd2: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[9] at flatMap at <console>:29

scala> rdd2.collect
res1: Array[String] = Array(a, b, c, d, e, f, h, i, j)
```

练习 3：

val rdd1 = sc.parallelize(List(5, 6, 4, 3))

val rdd2 = sc.parallelize(List(1, 2, 3, 4))

//求并集

val rdd3 = rdd1.union(rdd2)

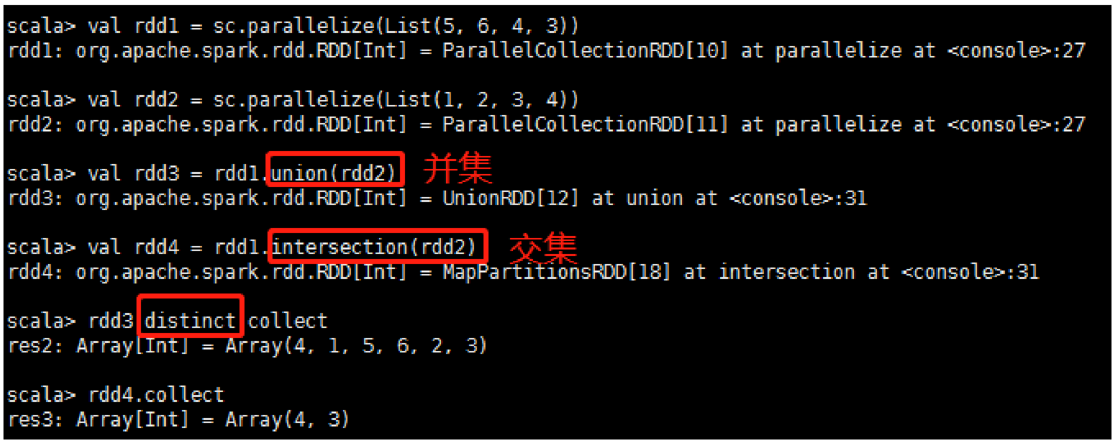
//求交集

val rdd4 = rdd1.intersection(rdd2)

//去重

rdd3.distinct.collect

rdd4.collect



```
scala> val rdd1 = sc.parallelize(List(5, 6, 4, 3))
rdd1: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[10] at parallelize at <console>:27

scala> val rdd2 = sc.parallelize(List(1, 2, 3, 4))
rdd2: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[11] at parallelize at <console>:27

scala> val rdd3 = rdd1.union(rdd2)
rdd3: org.apache.spark.rdd.RDD[Int] = UnionRDD[12] at union at <console>:31

scala> val rdd4 = rdd1.intersection(rdd2)
rdd4: org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD[18] at intersection at <console>:31

scala> rdd3.distinct.collect
res2: Array[Int] = Array(4, 1, 5, 6, 2, 3)

scala> rdd4.collect
res3: Array[Int] = Array(4, 3)
```

练习 4：

val rdd1 = sc.parallelize(List(("tom", 1), ("jerry", 3), ("kitty", 2))) *val rdd2 = sc.parallelize(List(("jerry", 2), ("tom", 1), ("shuke", 2)))*

//求 join

val rdd3 = rdd1.join(rdd2)

rdd3.collect

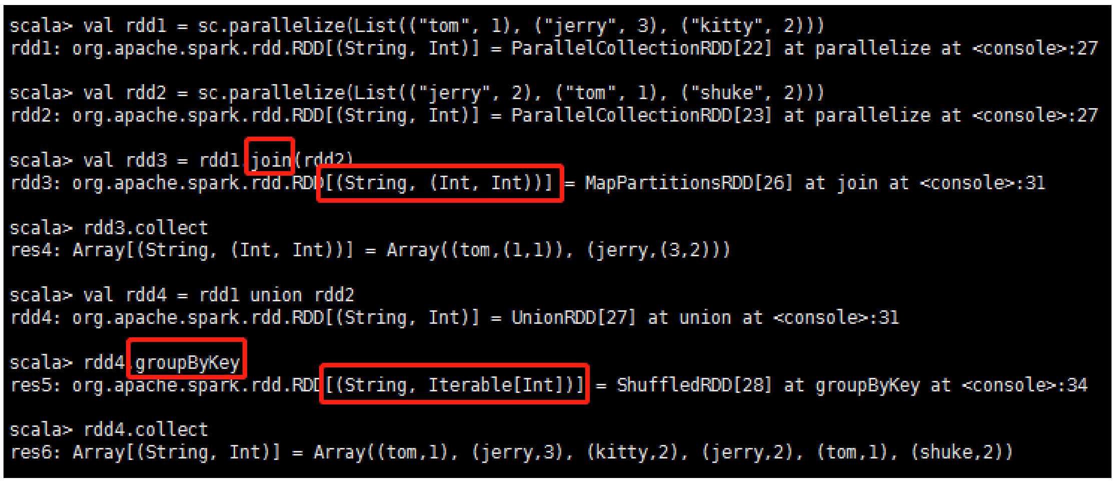
//求并集

val rdd4 = rdd1 union rdd2

//按 key 进行分组

rdd4.groupByKey

rdd4.collect



```
scala> val rdd1 = sc.parallelize(List(("tom", 1), ("jerry", 3), ("kitty", 2)))
rdd1: org.apache.spark.rdd.RDD[(String, Int)] = ParallelCollectionRDD[22] at parallelize at <console>:27

scala> val rdd2 = sc.parallelize(List(("jerry", 2), ("tom", 1), ("shuke", 2)))
rdd2: org.apache.spark.rdd.RDD[(String, Int)] = ParallelCollectionRDD[23] at parallelize at <console>:27

scala> val rdd3 = rdd1.join(rdd2)
rdd3: org.apache.spark.rdd.RDD[(String, (Int, Int))] = MapPartitionsRDD[26] at join at <console>:31

scala> rdd3.collect
res4: Array[(String, (Int, Int))] = Array((tom,(1,1)), (jerry,(3,2)))

scala> val rdd4 = rdd1 union rdd2
rdd4: org.apache.spark.rdd.RDD[(String, Int)] = UnionRDD[27] at union at <console>:31

scala> rdd4.groupByKey
res5: org.apache.spark.rdd.RDD[(String, Iterable[Int])] = ShuffledRDD[28] at groupByKey at <console>:34

scala> rdd4.collect
res6: Array[(String, Int)] = Array((tom,1), (jerry,3), (kitty,2), (jerry,2), (tom,1), (shuke,2))
```

练习 5：

val rdd1 = sc.parallelize(List(("tom", 1), ("tom", 2), ("jerry", 3), ("kitty", 2)))

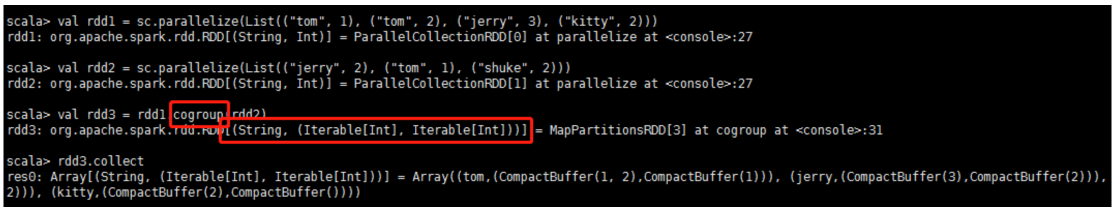
val rdd2 = sc.parallelize(List(("jerry", 2), ("tom", 1), ("shuke", 2)))

//cogroup

val rdd3 = rdd1.cogroup(rdd2)

//注意 cogroup 与 groupByKey 的区别

rdd3.collect



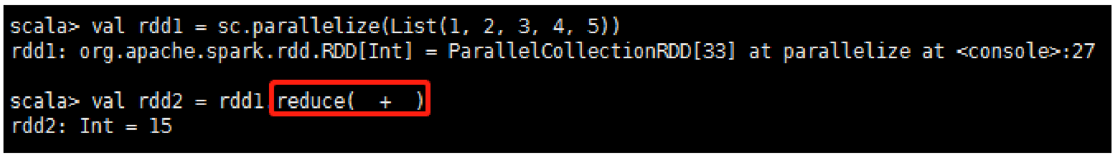
```
scala> val rdd1 = sc.parallelize(List(("tom", 1), ("tom", 2), ("jerry", 3), ("kitty", 2)))
rdd1: org.apache.spark.rdd.RDD[(String, Int)] = ParallelCollectionRDD[0] at parallelize at <console>:27

scala> val rdd2 = sc.parallelize(List(("jerry", 2), ("tom", 1), ("shuke", 2)))
rdd2: org.apache.spark.rdd.RDD[(String, Int)] = ParallelCollectionRDD[1] at parallelize at <console>:27

scala> val rdd3 = rdd1.cogroup(rdd2)
rdd3: org.apache.spark.rdd.RDD[(String, (Iterable[Int], Iterable[Int]))] = MapPartitionsRDD[3] at cogroup at <console>:31

scala> rdd3.collect
res0: Array[(String, (Iterable[Int], Iterable[Int]))] = Array((tom,(CompactBuffer(1, 2),CompactBuffer(1))), (jerry,(CompactBuffer(3),CompactBuffer(2))),
2))), (kitty,(CompactBuffer(2),CompactBuffer())))
```

练习 6：

val rdd1 = sc.parallelize(List(1, 2, 3, 4, 5))

//reduce 聚合

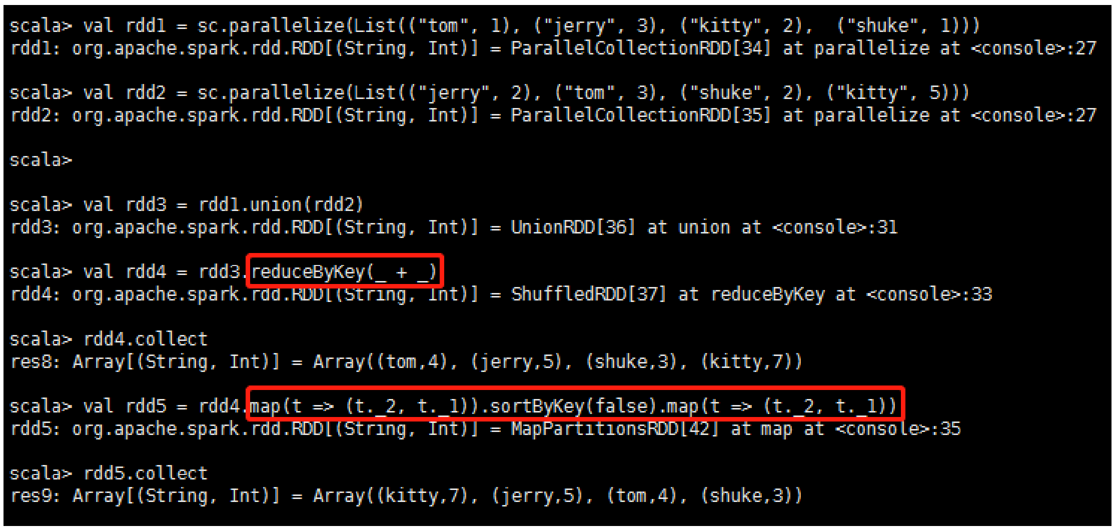
val rdd2 = rdd1.reduce(_ + _)



```
scala> val rdd1 = sc.parallelize(List(1, 2, 3, 4, 5))
rdd1: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[33] at parallelize at <console>:27

scala> val rdd2 = rdd1.reduce(_ + _)
rdd2: Int = 15
```

练习 7：

```
val rdd1 = sc.parallelize(List(("tom", 1), ("jerry", 3), ("kitty", 2),  ("shuke", 1)))
val rdd2 = sc.parallelize(List(("jerry", 2), ("tom", 3), ("shuke", 2), ("kitty", 5)))
val rdd3 = rdd1.union(rdd2)
//按 key 进行聚合
val rdd4 = rdd3.reduceByKey(_ + _)
rdd4.collect
//按 value 降序排序
val rdd5 = rdd4.map(t => (t._2, t._1)).sortByKey(false).map(t => (t._2, t._1))
rdd5.collect
```



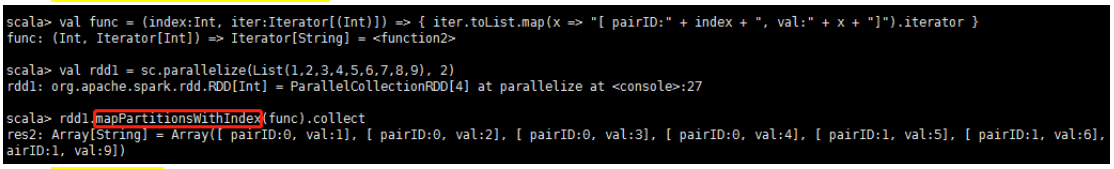
```
scala> val rdd1 = sc.parallelize(List(("tom", 1), ("jerry", 3), ("kitty", 2),  ("shuke", 1)))
rdd1: org.apache.spark.rdd.RDD[(String, Int)] = ParallelCollectionRDD[34] at parallelize at <console>:27

scala> val rdd2 = sc.parallelize(List(("jerry", 2), ("tom", 3), ("shuke", 2), ("kitty", 5)))
rdd2: org.apache.spark.rdd.RDD[(String, Int)] = ParallelCollectionRDD[35] at parallelize at <console>:27

scala>

scala> val rdd3 = rdd1.union(rdd2)
rdd3: org.apache.spark.rdd.RDD[(String, Int)] = UnionRDD[36] at union at <console>:31

scala> val rdd4 = rdd3.reduceByKey(_ + _)
rdd4: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[37] at reduceByKey at <console>:33

scala> rdd4.collect
res8: Array[(String, Int)] = Array((tom,4), (jerry,5), (shuke,3), (kitty,7))

scala> val rdd5 = rdd4.map(t => (t._2, t._1)).sortByKey(false).map(t => (t._2, t._1))
rdd5: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[42] at map at <console>:35

scala> rdd5.collect
res9: Array[(String, Int)] = Array((kitty,7), (jerry,5), (tom,4), (shuke,3))
```

//想了解更多，访问下面的地址

http://homepage.cs.latrobe.edu.au/zhe/ZhenHeSparkRDDAPIExamples.html

补充：复杂算子

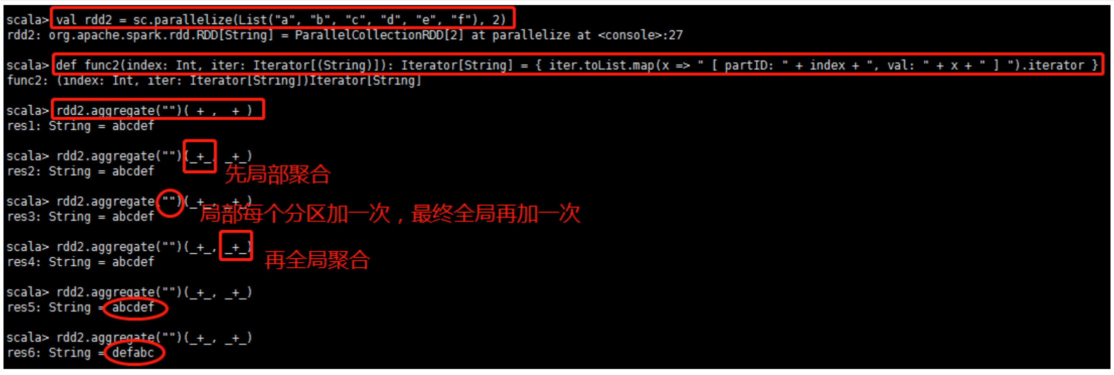
1. mapPartitionsWithIndex



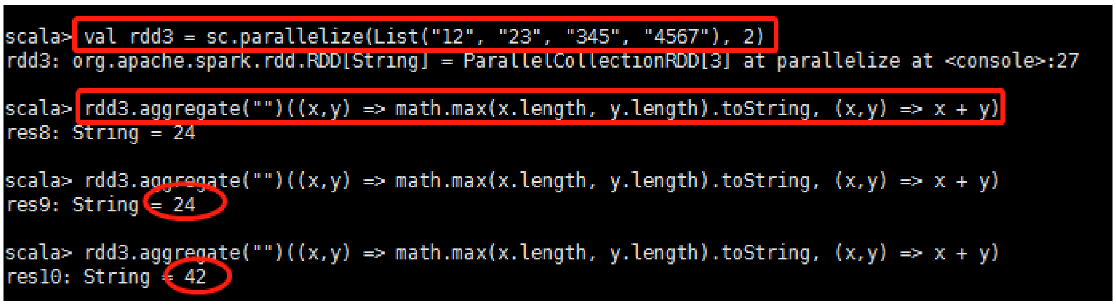
```
scala> val func = (index:Int, iter:Iterator[(Int)]) => { iter.toList.map(x => "[ pairID:" + index + ", val:" + x + "]").iterator }
func: (Int, Iterator[Int]) => Iterator[String] = <function2>

scala> val rdd1 = sc.parallelize(List(1,2,3,4,5,6,7,8,9), 2)
rdd1: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[4] at parallelize at <console>:27

scala> rdd1.mapPartitionsWithIndex(func).collect
res2: Array[String] = Array([ pairID:0, val:1], [ pairID:0, val:2], [ pairID:0, val:3], [ pairID:0, val:4], [ pairID:1, val:5], [ pairID:1, val:6],
airID:1, val:9])
```

2. aggregate

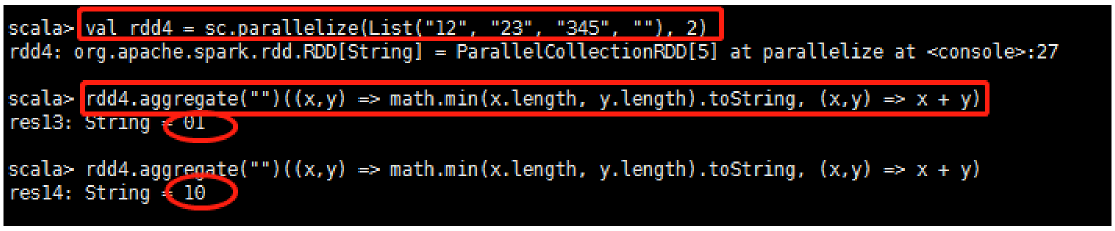
aggregate 是 Action 算子，先局部聚合再全局聚合。



```
scala> val rdd2 = sc.parallelize(List("a", "b", "c", "d", "e", "f"), 2)
rdd2: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[2] at parallelize at <console>:27

scala> def func2(index: Int, iter: Iterator[(String)]): Iterator[String] = { iter.toList.map(x => " [ partID: " + index + ", val: " + x + " ] ").iterator }
func2: (index: Int, iter: Iterator[String])Iterator[String]

scala> rdd2.aggregate("")(_+_, _+_)
res1: String = abcdef

scala> rdd2.aggregate("")(_+_, _+_)
res2: String = abcdef

scala> rdd2.aggregate("")(_+_, _+_)
res3: String = abcdef

scala> rdd2.aggregate("")(_+_, _+_)
res4: String = abcdef

scala> rdd2.aggregate("")(_+_, _+_)
res5: String = abcdef

scala> rdd2.aggregate("")(_+_, _+_)
res6: String = defabc
```



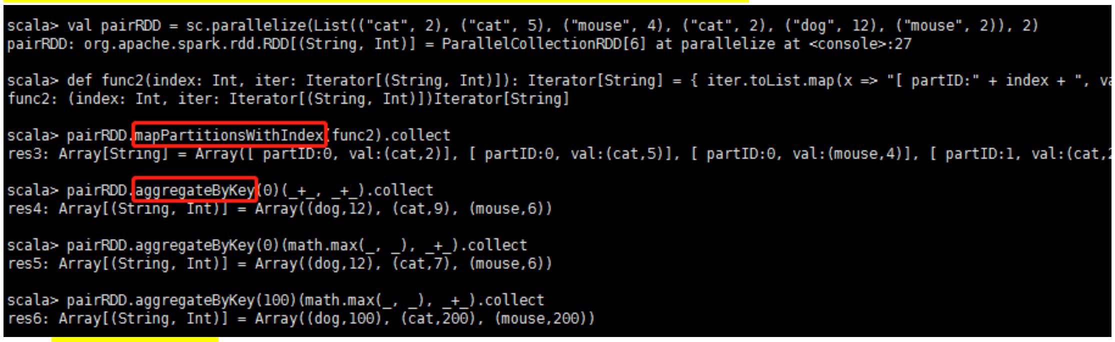
```
scala> val rdd3 = sc.parallelize(List("12", "23", "345", "4567"), 2)
rdd3: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[3] at parallelize at <console>:27

scala> rdd3.aggregate("")((x,y) => math.max(x.length, y.length).toString, (x,y) => x + y)
res8: String = 24

scala> rdd3.aggregate("")((x,y) => math.max(x.length, y.length).toString, (x,y) => x + y)
res9: String = 24

scala> rdd3.aggregate("")((x,y) => math.max(x.length, y.length).toString, (x,y) => x + y)
res10: String = 42
```



```
scala> val rdd4 = sc.parallelize(List("12", "23", "345", ""), 2)
rdd4: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[5] at parallelize at <console>:27

scala> rdd4.aggregate("")((x,y) => math.min(x.length, y.length).toString, (x,y) => x + y)
res13: String = 01

scala> rdd4.aggregate("")((x,y) => math.min(x.length, y.length).toString, (x,y) => x + y)
res14: String = 10
```

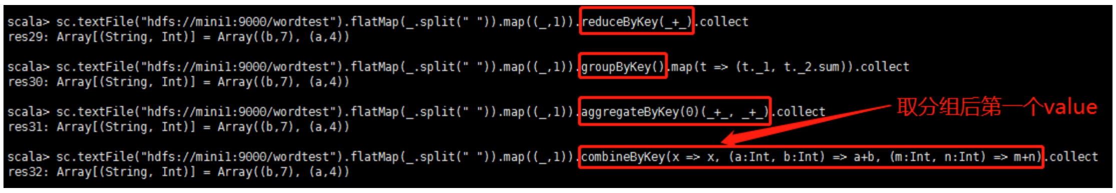
3.　aggregateByKey

reduceByKey 和 aggregateByKey 底层都是调用 combineByKey。

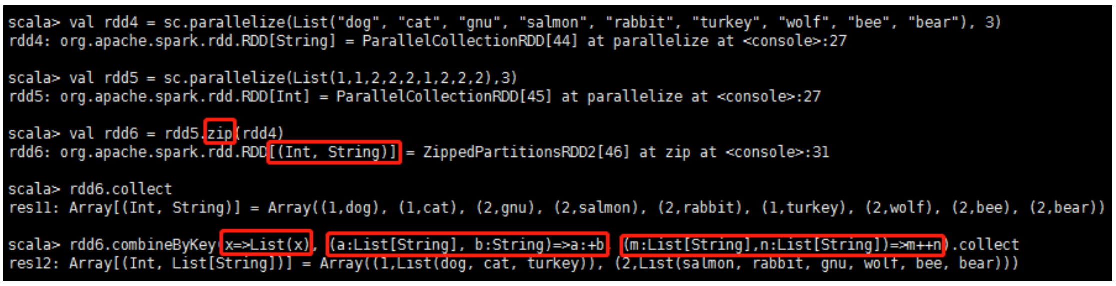


```
scala> val pairRDD = sc.parallelize(List(("cat", 2), ("cat", 5), ("mouse", 4), ("cat", 2), ("dog", 12), ("mouse", 2)), 2)
pairRDD: org.apache.spark.rdd.RDD[(String, Int)] = ParallelCollectionRDD[6] at parallelize at <console>:27

scala> def func2(index: Int, iter: Iterator[(String, Int)]): Iterator[String] = { iter.toList.map(x => "[ partID:" + index + ", va
func2: (index: Int, iter: Iterator[(String, Int)])Iterator[String]

scala> pairRDD.mapPartitionsWithIndex(func2).collect
res3: Array[String] = Array([ partID:0, val:(cat,2)], [ partID:0, val:(cat,5)], [ partID:0, val:(mouse,4)], [ partID:1, val:(cat,2

scala> pairRDD.aggregateByKey(0)(_+_, _+_).collect
res4: Array[(String, Int)] = Array((dog,12), (cat,9), (mouse,6))

scala> pairRDD.aggregateByKey(0)(math.max(_, _), _+_).collect
res5: Array[(String, Int)] = Array((dog,12), (cat,7), (mouse,6))

scala> pairRDD.aggregateByKey(100)(math.max(_, _), _+_).collect
res6: Array[(String, Int)] = Array((dog,100), (cat,200), (mouse,200))
```

4.　combineByKey

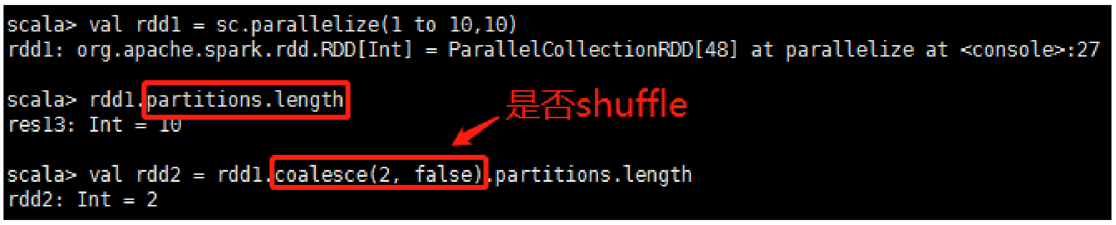


```
scala> sc.textFile("hdfs://mini1:9000/wordtest").flatMap(_.split(" ")).map((_,1)).reduceByKey(_+_).collect
res29: Array[(String, Int)] = Array((b,7), (a,4))

scala> sc.textFile("hdfs://mini1:9000/wordtest").flatMap(_.split(" ")).map((_,1)).groupByKey().map(t => (t._1, t._2.sum)).collect
res30: Array[(String, Int)] = Array((b,7), (a,4))

scala> sc.textFile("hdfs://mini1:9000/wordtest").flatMap(_.split(" ")).map((_,1)).aggregateByKey(0)(_+_, _+_).collect
res31: Array[(String, Int)] = Array((b,7), (a,4))

scala> sc.textFile("hdfs://mini1:9000/wordtest").flatMap(_.split(" ")).map((_,1)).combineByKey(x => x, (a:Int, b:Int) => a+b, (m:Int, n:Int) => m+n).collect
res32: Array[(String, Int)] = Array((b,7), (a,4))
```



```
scala> val rdd4 = sc.parallelize(List("dog", "cat", "gnu", "salmon", "rabbit", "turkey", "wolf", "bee", "bear"), 3)
rdd4: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[44] at parallelize at <console>:27

scala> val rdd5 = sc.parallelize(List(1,1,2,2,2,1,2,2,2),3)
rdd5: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[45] at parallelize at <console>:27

scala> val rdd6 = rdd5.zip(rdd4)
rdd6: org.apache.spark.rdd.RDD[(Int, String)] = ZippedPartitionsRDD2[46] at zip at <console>:31

scala> rdd6.collect
res11: Array[(Int, String)] = Array((1,dog), (1,cat), (2,gnu), (2,salmon), (2,rabbit), (1,turkey), (2,wolf), (2,bee), (2,bear))

scala> rdd6.combineByKey(x=>List(x), (a:List[String], b:String)=>a:+b, (m:List[String],n:List[String])=>m++n).collect
res12: Array[(Int, List[String])] = Array((1,List(dog, cat, turkey)), (2,List(salmon, rabbit, gnu, wolf, bee, bear)))
```

5． coalesce、repartition



```
scala> val rdd1 = sc.parallelize(1 to 10,10)
rdd1: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[48] at parallelize at <console>:27

scala> rdd1.partitions.length
res13: Int = 10

scala> val rdd2 = rdd1.coalesce(2, false).partitions.length
rdd2: Int = 2
```

repartition 底层调用 coalesce，默认 shuffle。