Spark算子第二部分

主要理解难度比较大的算子

map是对每个元素操作, mapPartitions是对其中的每个partition操作

-------------------------------------------------------------------------------------------

-------------------------------------------------------------------------------------------

mapPartitionsWithIndex : 把每个partition中的分区号和对应的值拿出来, 看源码

val func = (index: Int, iter: Iterator[(Int)]) => {

iter.toList.map(x => "[partID:" + index + ", val: " + x + "]").iterator

}

val rdd1 = sc.parallelize(List(1,2,3,4,5,6,7,8,9), 2)

rdd1.mapPartitionsWithIndex(func).collect

-------------------------------------------------------------------------------------------

-------------------------------------------------------------------------------------------

aggregate

def func1(index: Int, iter: Iterator[(Int)]) : Iterator[String] = {

iter.toList.map(x => "[partID:" + index + ", val: " + x + "]").iterator

}

val rdd1 = sc.parallelize(List(1,2,3,4,5,6,7,8,9), 2)

rdd1.mapPartitionsWithIndex(func1).collect

###是action操作, 第一个参数是初始值, 二:是2个函数[每个函数都是2个参数(第一个参数:先对个个分区进行合并, 第二个:对个个分区合并后的结果再进行合并), 输出一个参数]

###0 + (0+1+2+3+4 + 0+5+6+7+8+9)

rdd1.aggregate(0)(\_+\_, \_+\_)

rdd1.aggregate(0)(math.max(\_, \_), \_ + \_)

###5和1比, 得5再和234比得5 --> 5和6789比,得9 --> 5 + (5+9)

rdd1.aggregate(5)(math.max(\_, \_), \_ + \_)

val rdd2 = sc.parallelize(List("a","b","c","d","e","f"),2)

def func2(index: Int, iter: Iterator[(String)]) : Iterator[String] = {

iter.toList.map(x => "[partID:" + index + ", val: " + x + "]").iterator

}

rdd2.aggregate("")(\_ + \_, \_ + \_)

rdd2.aggregate("=")(\_ + \_, \_ + \_)

val rdd3 = sc.parallelize(List("12","23","345","4567"),2)

rdd3.aggregate("")((x,y) => math.max(x.length, y.length).toString, (x,y) => x + y)

val func\_str = (index: Int, iter: Iterator[(String)]) => {

iter.toList.map(x => "[partID:" + index + ", val: " + x + "]").iterator

}

val rdd4 = sc.parallelize(List("12","23","345",""),2)

rdd4.aggregate("")((x,y) => math.min(x.length, y.length).toString, (x,y) => x + y)

val rdd5 = sc.parallelize(List("12","23","","345"),2)

rdd5.aggregate("")((x,y) => math.min(x.length, y.length).toString, (x,y) => x + y)

-------------------------------------------------------------------------------------------

-------------------------------------------------------------------------------------------

aggregateByKey

val pairRDD = sc.parallelize(List( ("cat",2), ("cat", 5), ("mouse", 4),("cat", 12), ("dog", 12), ("mouse", 2)), 2)

def func2(index: Int, iter: Iterator[(String, Int)]) : Iterator[String] = {

iter.toList.map(x => "[partID:" + index + ", val: " + x + "]").iterator

}

pairRDD.mapPartitionsWithIndex(func2).collect

pairRDD.aggregateByKey(0)(math.max(\_, \_), \_ + \_).collect

pairRDD.aggregateByKey(100)(math.max(\_, \_), \_ + \_).collect

-------------------------------------------------------------------------------------------

-------------------------------------------------------------------------------------------

checkpoint

sc.setCheckpointDir("hdfs://bigdata01:9000/ck")

val rdd = sc.textFile("hdfs://bigdata01:9000/wc").flatMap(\_.split(" ")).map((\_, 1)).reduceByKey(\_+\_)

rdd.checkpoint

rdd.isCheckpointed

rdd.count

rdd.isCheckpointed

rdd.getCheckpointFile

-------------------------------------------------------------------------------------------

-------------------------------------------------------------------------------------------

coalesce, repartition

val rdd1 = sc.parallelize(1 to 10, 10)

val rdd2 = rdd1.coalesce(2, false)

rdd2.partitions.length

-------------------------------------------------------------------------------------------

-------------------------------------------------------------------------------------------

collectAsMap : Map(b -> 2, a -> 1)

val rdd = sc.parallelize(List(("a", 1), ("b", 2)))

rdd.collectAsMap

-------------------------------------------------------------------------------------------

-------------------------------------------------------------------------------------------

combineByKey : 和reduceByKey是相同的效果

###第一个参数x:原封不动取出来, 第二个参数:是函数, 局部运算, 第三个:是函数, 对局部运算后的结果再做运算

###每个分区中每个key中value中的第一个值, (hello,1)(hello,1)(good,1)-->(hello(1,1),good(1))-->x就相当于hello的第一个1, good中的1

val rdd1 = sc.textFile("hdfs://bigdata01:9000/spark/").flatMap(\_.split(" ")).map((\_, 1))

val rdd2 = rdd1.combineByKey(x => x, (a: Int, b: Int) => a + b, (m: Int, n: Int) => m + n)

rdd1.collect

rdd2.collect

###当input下有3个文件时(有3个block块, 不是有3个文件就有3个block, ), 每个会多加3个10

val rdd3 = rdd1.combineByKey(x => x + 10, (a: Int, b: Int) => a + b, (m: Int, n: Int) => m + n)

rdd3.collect

val rdd4 = sc.parallelize(List("dog","cat","gnu","salmon","rabbit","turkey","wolf","bear","bee"), 3)

val rdd5 = sc.parallelize(List(1,1,2,2,2,1,2,2,2), 3)

val rdd6 = rdd5.zip(rdd4)

val rdd7 = rdd6.combineByKey(List(\_), (x: List[String], y: String) => x :+ y, (m: List[String], n: List[String]) => m ++ n)

-------------------------------------------------------------------------------------------

-------------------------------------------------------------------------------------------

countByKey

val rdd1 = sc.parallelize(List(("a", 1), ("b", 2), ("b", 2), ("c", 2), ("c", 1)))

rdd1.countByKey

rdd1.countByValue

-------------------------------------------------------------------------------------------

-------------------------------------------------------------------------------------------

filterByRange

val rdd1 = sc.parallelize(List(("e", 5), ("c", 3), ("d", 4), ("c", 2), ("a", 1)))

val rdd2 = rdd1.filterByRange("b", "d")

rdd2.collect

-------------------------------------------------------------------------------------------

-------------------------------------------------------------------------------------------

flatMapValues : Array((a,1), (a,2), (b,3), (b,4))

val rdd3 = sc.parallelize(List(("a", "1 2"), ("b", "3 4")))

val rdd4 = rdd3.flatMapValues(\_.split(" "))

rdd4.collect

-------------------------------------------------------------------------------------------

-------------------------------------------------------------------------------------------

foldByKey

val rdd1 = sc.parallelize(List("dog", "wolf", "cat", "bear"), 2)

val rdd2 = rdd1.map(x => (x.length, x))

val rdd3 = rdd2.foldByKey("")(\_+\_)

-------------------------------------------------------------------------------------------

-------------------------------------------------------------------------------------------

foreachPartition

val rdd1 = sc.parallelize(List(1, 2, 3, 4, 5, 6, 7, 8, 9), 3)

rdd1.foreachPartition(x => println(x.reduce(\_ + \_)))

-------------------------------------------------------------------------------------------

-------------------------------------------------------------------------------------------

keyBy : 以传入的参数做key

val rdd1 = sc.parallelize(List("dog", "salmon", "salmon", "rat", "elephant"), 3)

val rdd2 = rdd1.keyBy(\_.length)

rdd2.collect

-------------------------------------------------------------------------------------------

-------------------------------------------------------------------------------------------

keys values

val rdd1 = sc.parallelize(List("dog", "tiger", "lion", "cat", "panther", "eagle"), 2)

val rdd2 = rdd1.map(x => (x.length, x))

rdd2.keys.collect

rdd2.values.collect

-------------------------------------------------------------------------------------------

-------------------------------------------------------------------------------------------

mapPartitions

spark 算子第二部分作业

1. 将上面的算子从头到尾敲二遍，边敲边理解
2. 通过折叠的方式计算wordcount里面的单词数，并在本地跑动
3. 统计每个电话号码停留时间最长基站的总停留时间，以及所在经纬度