1.回顾
    Spark通用计算引擎，速度快、通用性、运行方式多，易用性。

    core
    sql
    streaming
    mllib
    graphx

    集群部署模式
    ----
        local
        standalone
        yarn
        mesos

RDD
---
    resilient distributed dataset,弹性分布式数据集。
    java集合(List)。

    [scala]
    val conf = new SparkConf()
    conf.setAppName("wc")
    conf.setMaster("spark://s101:7077")
    val sc = new SparkContext(conf)

    val rdd1 = sc.textFile("hdfs://mycluster/user/centos/1.txt")
    val rdd2 = rdd1.flatMap(_.split(" "))
    val rdd3 = rdd2.map((_,1))
    val rdd4 = rdd3.reduceByKey(_+_)
    val res = rdd4.collect();

    [java]
    JavaSparkContext
    JavaRDD
    JavaPairRDD


```
List<Tuple2<String, Tuple2<String, Integer>>> list = new ArrayList<Tuple2<String, Tuple2<String, Integer>>>() ;
String busid = t._1._1 ;
String tag = t._1._2 ;
Integer count = t._2() ;

list.add(new Tuple2<String, Tuple2<String, Integer>>(busid ,new Tuple2<String, Integer>(tag, count)));
return null ;
```

Spark实现标签生成

    1.scala版
        import com.oldboy.spark.Util.TagUtil
        import org.apache.spark.{SparkConf, SparkContext}

        import scala.collection.mutable.ArrayBuffer

        /**
          * Created by Administrator on 2018/5/10.
          */
        object TaggenScala {
            def main(args: Array[String]): Unit = {
                //1.创建SparkContex
                val conf = new SparkConf
                conf.setAppName("taggen")
                conf.setMaster("local")
                val sc = new SparkContext(conf)

                //2.加载文件
                val rdd1 = sc.textFile("d:/mr/temptags.txt")

```
71
72                  //3.压扁成RDD[(busid,tag)]
73                  val rdd2 = rdd1.flatMap(line=>{
74                      val buf = ArrayBuffer[Tuple2[Tuple2[String, String], Int]]()
75                      val arr = line.split("\t")
76                      val busid = arr(0)
77                      val json = arr(1)
78                      //
79                      import scala.collection.JavaConversions._
80                      val tags = TagUtil.extractTags(json)
81                      for(tag <- tags){
82                          buf.+=(((busid , tag),1))
83                      }
84                      buf
85                  })
86
87                  //4.聚合
88                  val rdd3 = rdd2.reduceByKey(_+_)
89
90                  //5.变换组合成(busid, List((tag,count)))
91                  val rdd4 = rdd3.map(t=>{
92                      (t._1._1, (t._1._2 , t._2)::Nil)
93                  })
94
95                  //6.聚合
96                  val rdd5 = rdd4.reduceByKey((a,b)=> a ::: b)
97
98                  //7.商家内评论倒排序并取前三
99                  val rdd6 = rdd5.mapValues(list=>{
100                     list.sortBy(- _._2).take(3)
101                 })
102
103                 val rdd7 = rdd6.sortBy(t=> -t._2(0)._2)
104
105                 rdd7.foreach(println)
106             }
107         }
108
109     2.java版
110         package com.oldboy.spark.java;
111
112         import com.oldboy.spark.Util.TagUtil;
113         import org.apache.spark.SparkConf;
114         import org.apache.spark.api.java.JavaPairRDD;
115         import org.apache.spark.api.java.JavaRDD;
116         import org.apache.spark.api.java.JavaSparkContext;
117         import org.apache.spark.api.java.function.*;
118         import scala.Tuple2;
119         import scala.Tuple4;
120
121         import java.util.ArrayList;
122         import java.util.Comparator;
123         import java.util.Iterator;
124         import java.util.List;
125
126         /**
127          * 统计气温数据
128          */
129         public class TaggenJava {
130             public static void main(String[] args) {
131                 SparkConf conf = new SparkConf();
132                 conf.setAppName("TaggenJava");
133                 conf.setMaster("local") ;
134
135                 JavaSparkContext sc = new JavaSparkContext(conf);
136
137                 //1.加载文件
138                 JavaRDD<String> rdd1 = sc.textFile("d:/mr/temptags.txt");
139
140                 //2.压扁
141                 JavaPairRDD<Tuple2<String, String>, Integer> rdd2 =
                    rdd1.flatMapToPair(new PairFlatMapFunction<String,
                    Tuple2<String,String>, Integer>() {
```

```
142                     public Iterator<Tuple2<Tuple2<String, String>, Integer>>
                        call(String s) throws Exception {
143                         List<Tuple2<Tuple2<String, String>, Integer>> list = new
                            ArrayList<Tuple2<Tuple2<String, String>, Integer>>() ;
144                         String[] arr = s.split("\t") ;
145                         String busid = arr[0] ;
146                         String json = arr[1] ;
147                         List<String> tags = TagUtil.extractTags(json) ;
148                         for(String tag : tags){
149                             list.add(new Tuple2<Tuple2<String, String>, Integer>(new
                                Tuple2<String, String>(busid,tag) , 1)) ;
150                         }
151                         return list.iterator();
152                     }
153                 }) ;
154
155                 //3.聚合((busid,tag),count)
156                 JavaPairRDD<Tuple2<String, String>, Integer> rdd3
                    =rdd2.reduceByKey(new Function2<Integer, Integer, Integer>() {
157                     public Integer call(Integer v1, Integer v2) throws Exception {
158                         return v1 + v2;
159                     }
160                 }) ;
161
162                 //4.交换元组形成新元组(busid , List((tag,count)))
163                 JavaPairRDD<String, List<Tuple2<String, Integer>>> rdd4 =
                    rdd3.mapToPair(new
                    PairFunction<Tuple2<Tuple2<String,String>,Integer>, String,
                    List<Tuple2<String,Integer>>>() {
164                     public Tuple2<String, List<Tuple2<String, Integer>>>
                        call(Tuple2<Tuple2<String, String>, Integer> t) throws Exception {
165                         List<Tuple2<String,Integer>> list = new
                            ArrayList<Tuple2<String, Integer>>() ;
166                         list.add(new Tuple2<String, Integer>(t._1._2() , t._2())) ;
167                         return new Tuple2<String, List<Tuple2<String,
                            Integer>>>(t._1._1 , list);
168                     }
169                 });
170
171                 //5.聚合,(busid,List((tag,count),(tag1,count1)))
172                 JavaPairRDD<String, List<Tuple2<String, Integer>>> rdd5
                    =rdd4.reduceByKey(
173                         new Function2<List<Tuple2<String, Integer>>,
                            List<Tuple2<String, Integer>>, List<Tuple2<String,
                            Integer>>>() {
174                             public List<Tuple2<String, Integer>>
                                call(List<Tuple2<String, Integer>> v1,
                                List<Tuple2<String, Integer>> v2) throws Exception {
175                                 v1.addAll(v2);
176                                 return v1 ;
177                             }
178                         }) ;
179
180                 //6.排序
181                 JavaPairRDD<String, List<Tuple2<String, Integer>>> rdd6 =
                    rdd5.mapValues(new Function<List<Tuple2<String,Integer>>,
                    List<Tuple2<String, Integer>>>() {
182                     public List<Tuple2<String, Integer>> call(List<Tuple2<String,
                        Integer>> v1) throws Exception {
183                         v1.sort(new Comparator<Tuple2<String, Integer>>() {
184                             public int compare(Tuple2<String, Integer> o1,
                                Tuple2<String, Integer> o2) {
185                                 return -(o1._2() - o2._2());
186                             }
187                         });
188                         return new ArrayList<Tuple2<String, Integer>>(v1.size() > 4
                            ? v1.subList(0,5): v1);
189                     }
190                 }) ;
191                 //7.全排序
192                 JavaRDD<Tuple2<String, List<Tuple2<String, Integer>>>>
                    rdd7=rdd6.map(new
                    Function<Tuple2<String,List<Tuple2<String,Integer>>>, Tuple2<String,
```

```
                List<Tuple2<String, Integer>>>>() {
193                 public Tuple2<String, List<Tuple2<String, Integer>>>
                    call(Tuple2<String, List<Tuple2<String, Integer>>> v1) throws
                    Exception {
194                     return v1;
195                 }
196             }) ;
197
198             //7.全排序
199             JavaRDD<Tuple2<String, List<Tuple2<String, Integer>>>> rdd8
                =rdd7.sortBy(new
                Function<Tuple2<String,List<Tuple2<String,Integer>>>, Integer>() {
200                 public Integer call(Tuple2<String, List<Tuple2<String,
                    Integer>>> v1) throws Exception {
201                     return v1._2().get(0)._2();
202                 }
203             } , false , 1) ;
204
205             List<Tuple2<String, List<Tuple2<String, Integer>>>> res =
                rdd8.collect();
206             for(Object o : res){
207                 System.out.println(o);
208             }
209         }
210     }
211
212
213 Spark创建RDD方式
214 ----------------
215     1.textFile
216         文本加载
217     2.通过RDD进行特定变换
218         rdd.map()
219         rdd.flatMap()
220         rdd.filter()
221         ...
222     3.parallelize
223         创建静态rdd.
224     4.
225
226 RDD的算子
227 ----------------
228     RDD算子包括变换和动作两种类型，只要返回值是rdd就是变换，反之大部分action.
229     最终通过是否调用SparkContext的runJob方法来界定。
230     变换不会触发job的执行，只有action方法调用时才会触发job的执行.
231
232
233     [transformation]
234     map                     //变换
235     flatMap                 //压扁
236     filter                  //过滤
237     reduceByKey()           //聚合
238     groupBy                 //分组
239     mapPartitions           //*****对分区进行变换.*****
240     mapPartitionsWithIndex  //*****对分区进行变换.*****
241     union                   //同型操作,等价于sql union,不需要
242     intersect               //交集,需要shuffle ,
243     distinct                //交集,需要shuffle
244     groupByKey()            //变换
245     aggregateByKey()        //按照key进行聚合，可以改变类型。
246
247     join                    //shuffle，按照key连接，等价于sql的连接操作
248     leftOuterJoin
249     rightOuterJoin
250     fullOuterJoin
251     cogroup                 //协分组，对两个rdd的value进行聚合成二元组
252     cartesian               //笛卡尔积，交叉连接。不需要shuffle
253
254     repartition
        //再分区,不论是增加还是减少分区，都需要跨网络进行数据shuffle
255                             //处理，负载平衡。一旦涉及shuffle，性能会有影响。
256     coalesce                //*****
257
```

```
                                //改变分区，需要携带shuffle=true参数来控制是否需要shuffle。
258
                                //默认false，不进行shuffle，只能用于减少分区，repartition调用
259                             //调用的就是该方法，但是shuffle设置为true。
260
261
262 解决数据倾斜
263 ------------------------
264     1.重新设计key
265         import org.apache.spark.{SparkConf, SparkContext}
266
267         import scala.util.Random
268
269         /**
270           * Created by Administrator on 2018/5/8.
271           */
272         object DataLeanScala {
273             def main(args: Array[String]): Unit = {
274                 //1.创建spark配置对象
275                 val conf = new SparkConf()
276                 conf.setAppName("wcApp")
277                 conf.setMaster("local")
278
279                 val sc = new SparkContext(conf)
280
281                 val rdd1 = sc.textFile("d:/mr/1.txt")
282                 val rdd2 = rdd1.flatMap(line=>{
283                     var arr = line.split(" ")
284                     arr
285                 })
286                 val rdd3= rdd2.map((_,1))
287
288                 val rdd4 = rdd3.map(t=>{
289                     val n = Random.nextInt(3)
290                     (t._1 + "_" + n , t._2)
291                 })
292
293                 val rdd5 = rdd4.reduceByKey(_+_)
294                 val rdd6 = rdd5.map(t=>{
295                     var arr = t._1.split("_")
296                     (arr(0),t._2)
297                 })
298                 val rdd7 = rdd6.reduceByKey(_+_)
299                 rdd7.collect().foreach(println)
300
301
302                 while(true){
303                     Thread.sleep(1000)
304                 }
305             }
306
307         }
308
309
310     2.重新设计分区类。
311         import org.apache.spark.{HashPartitioner, Partitioner, SparkConf,
            SparkContext}
312
313         import scala.util.Random
314
315         /**
316           * Created by Administrator on 2018/5/8.
317           */
318         object DataLean2Scala {
319             def main(args: Array[String]): Unit = {
320                 //1.创建spark配置对象
321                 val conf = new SparkConf()
322                 conf.setAppName("wcApp")
323                 conf.setMaster("local")
324
325                 val sc = new SparkContext(conf)
```

```
                val rdd1 = sc.textFile("d:/mr/1.txt")
                val rdd2 = rdd1.flatMap(line=>{
                    var arr = line.split(" ")
                    arr
                })
                val rdd3= rdd2.map((_,1))

                class MyPartioner(n:Int) extends Partitioner{
                    override def numPartitions: Int = n

                    override def getPartition(key: Any): Int = {
                        Random.nextInt(n)
                    }
                }

                val rdd4 = rdd3.reduceByKey(new MyPartioner(4) , (a,b)=>{a + b})
                //需要重新指定分区类对象，否则使用之前的分区对象
                val rdd5 = rdd4.reduceByKey(new HashPartitioner(3) , (a,b)=>a + b)

                rdd5.foreach(println)

                while(true){
                    Thread.sleep(1000)
                }
            }

        }


Spark aggregateByKey()
-------------------------
    aggregateByKey(zeroValue: U)(seqOp: (U, V) => U, combOp: (U, U) => U): RDD[(K, U)]
    1.(zeroValue: U)
        初始值
    2.(seqOp: (U, V) => U, combOp: (U, U) => U)

    3.实现每个年度下，各个分区内的气温最大值列表
        import java.sql.DriverManager

        import org.apache.spark.{SparkConf, SparkContext}

        /**
          * Created by Administrator on 2018/5/10.
          */
        object RDDTestScala {

            //返回当前线程
            def tname() = {
                Thread.currentThread().getName
            }

            def main(args: Array[String]): Unit = {
                //1.创建SparkContex
                val conf = new SparkConf
                conf.setAppName("taggen")
                conf.setMaster("local[4]")
                val sc = new SparkContext(conf)

        //        val rdd1 = sc.parallelize(Array(1,2,2,3) , 3)
        //        val rdd2 = sc.parallelize(Array(2, 2, 3, 4), 3)
        //        val rdd3 = rdd1.intersection(rdd2 , 2)
        //        //        val rdd3 = rdd1.union(rdd2)
        ////        println("3 ): " + rdd3.partitions.length)
        ////
        ////        val rdd4 = rdd3.distinct(3)
        ////        println("4 ): " + rdd4.partitions.length)
        ////        println(rdd4.partitions.length)
        //        rdd3.collect().foreach(println)

        //        val rdd1 =
        sc.parallelize(Array("1.1"->"tom","1.1"->"tomas","1.2"->"bob","1.2"->"alice"))
```

```
398         //        val rdd2 = rdd1.aggregateByKey()
399         //        rdd2.collect()
400
401                 val rdd1 = sc.textFile("d:/mr/temp15.txt", 3)
402                 val rdd2 = rdd1.map(line=>{
403                     val arr = line.split(" ")
404                     (arr(0).toInt , arr(1).toInt)
405                 })
406
407                 //初始值,在每个分区内按key聚合,
408                 val zero:List[Int] = Nil
409                 //分区内按key聚合的聚合函数 (U,V) => U
410                 def seqOp(a:List[Int] , b:Int) = {
411                     if(a.isEmpty){
412                         b :: a
413                     }
414                     else{
415                         import scala.math._
416                         max(a(0),b) :: Nil
417                     }
418                 }
419                 //shuffle之后，对key下的shuffle过来的U进行聚合， (U,U)=>U
420                 def combOp(a:List[Int],b:List[Int]) ={
421                     a:::b
422                 }
423                 val rdd3 = rdd2.mapPartitionsWithIndex((index, it)=>{
424                     for(t <- it){
425                         println(index + " : " + t )
426                     }
427                     it
428                 })
429                 rdd3.collect().foreach(println)
430                 println("===============")
431                 val rdd4 = rdd2.aggregateByKey(zero)(seqOp _, combOp _)
432                 //练习： 提取每个年度下，每个分区内的最高气温和最低温形成的元组列表。
433                 //加强题:0-(20,-19),1-(,)
434                 rdd4.collect().foreach(println)
435
436                 while(true){
437                     Thread.sleep(1000)
438                 }
439             }
440         }
441
442
443 作业
444 -----------------
445     1.aggregateByKey改成java版实现气温统计。
446     2.统计实现每年内，在每个分区下最高最低气温形成列表
447     3.统计实现每年内，在每个分区下最高最低气温形成列表并携带分区索引信息。
448     4.java版实现上述2,3两题。
449
450     [action]
451     collect                 //Array
452     first                   //take(1)
453     take                    //
454     foreach()               //循环
455     reduce()                //T
456     save...                 //保存.
457
458 spark分区
459 -----------------
460     hadoop中的切片 == spaark.rdd的分区
461
462 启动spark-shell
463 ----------------
464     spark-shell --master  local[4]  //启动4个线程模拟worker
465     spark-shell --master  local[*]  //*匹配CPU内核数
466
467
468 Spark常用变换
469 -----------------
470     map
```

```
    flatMap
    mapValues()
    filter()                //


spark运行
---------------
    spark-submit --class ... --master ... xxx.jar


List<Integer> = 12345
list2
for(Integer n : list){
    list.add(n * 2);
}

list.map(_ * 2)

spark相关概念
---------------
    1.RDD
        resilient distributed dataset,弹性(容错)分布式数据集,spark的基本抽象。
        代表不可变、分区化Partition的元素集合，可以进行并行计算。
        包含可用于所有RDD的基本操作，例如map、filter、persist等。
        PairRDDFunctions包含针对kv对类型的RDD的操作。
        轻量级。
        [内部表现为5个主要属性]
        a.分区列表

        b.作用于每个切片的计算函数

        c.到其他RDD的依赖列表
            linage , 血统

        e.(可选)针对kv的RDD的分区类

        f.(可选)计算每个切片的首选位置列表(hdfs)。

        g.HadoopRDD
            该RDD为用于读取hdfs文件提供了核心功能。

        h.hadoop.textFile()
            textFile() -> HadoopRDD(inputFormat,k,v,minParts)
    2.SparkContext
        spark程序的主入口点，代表到cluster的连接，可以创建RDD、累加器、广播变量、
        每个JVM只有一个active的SparkContext。在创建新的上下文之前必须调用stop停止
        当前的上下文。
        Spark调度机制是三级调度,DagScheduler -> TaskScheduler -> BackendSheduler

        创建SparkContext -> createTaskScheduler -> {new TaskSchedulerImpl +
        match(local) => SchedulerBackend}

        LocalSchedulerBackend -> local[n]
        //local模式下
        sc.textFile(path , minPartitions = defaultMinPartitons=>
        min((get("spark.default.parallelism" , totleCores)) ,2))




    3.DAGScheduler
        Direct acycline graph,有向无环图调度器。
        面向stage的高级调出层。计算每个job的stage DAG，跟踪RDD和stage，寻找最短路径。

        提交taskset给下层的调度器(task调度器),taskset包含的是完全独立的任务(可以在clus
        ter直接运行).
        stage划分以shuffle边界，具有窄依赖的RDD操作(map |
        filter)进入同一taskset的管线中。
        Shuffle依赖的操作需要多个Stage，上一个的输出做下一个的输入。
        每个Stage只有一次依赖于其他stage的shuffle操作。
        DAG调度器决定task的首选运行位置，
```

DAG调度器处理因为shuffle输出文件丢失导致的故障。不是由shuffle输出文件丢失引发的故障由task调度器处理。
取消整个stage之前重试少量次数。

job :ActiveJob表示，最顶层的工作单元。action发生时，提交的activeJob
stage
:task集合，计算中间结果。同一RDD的每个分区都执行相同的计算过程。stage以shuffle边界作为划分标准。
task :工作单元，每个task发送一个machine.

4.ActiveJob
DAG调度器中运行的job，job类型有两种。
一种是result job，执行action，对ResultStage进行计算。
另一个是MapStage job,计算shuffleMapStage的map输出。
使用finalStage字段来区分是哪种类型.

5.Stage

并行执行的task集合，所有task运行同一函数。所有task都有相同的shuffle依赖，DAG调度器以拓扑顺序执行stage。
stage类型有两种，
一种是ShuffleMapStage,的输出是下一个stage的输入。
另一种是ResultStage，通过执行一个rdd的函数运行action.
每个stage都有一个firstJobid，标识第一个提交的stage，每个阶段都对应一个rdd.
Stage类型有两种：
5.1)ShuffleMapStage
在RDD的某些分区上执行函数计算结果

5.2)ResultStage
在RDD的某些分区上执行函数计算结果

6.Task
Spark执行单元，两种类型:ShuffleMapTask + ResultTask.

Job由多个Stage构成，每个job的最后stage由多个ResultTask构成,之前的stage由多个ShuffleMapTask构成。
ResultTask执行task并将结果回传到Driver(执行入口点)。
ShuffleMapTask执行task并按照分区类将task的输出划分到多个bucket(分区)中。

7.TaskScheduler
底层任务调度器接口，当前只有TaskSchedulerImpl实现。
该接口可插拔，可以使用不同的实现类。每个任务调度器为一个SparkContext调度任务.
该调度器从上层调度器(DagScheduler)得到提交过来的task set，并发送任务给集群。
运行在driver端,

通过后台调度器为多种集群类型调度任务。是过渡阶段。是以taskset为单位进行调度。

8.Executor
spark执行程序，通过线程池运行task.

9.依赖
Dependency.
RDD的分区和上级RDD分区之间的对应关系。

[窄依赖]
子RDD的每个分区依赖于父RDD的少量分区。
OneToOneDependency
RangeDependency
PruneDependency

[宽依赖]
需要shuffle,
ShuffleDependency

10.driver
驱动。
client : 入口程序。

spark job提交流程[local]

并发度 :

```
-------------
    最大并发度，等价于切片数(分区数)
    sc.textFile("",10) ;


并发能力
-------------
    并行线程数决定。
    conf.setMaster("local[4]")


spark的textFile()默认并发度计算
------------------------------
    1.sc.textFile(,,)
        def defaultMinPartitions: Int = math.min(defaultParallelism, 2)

    2.sc.parallelize(1 to 10,3)
        defaultParallelism = scheduler.conf.getInt("spark.default.parallelism",
        totalCores)


后台调度器的创建时间
----------------------
    sc.createTaskScheduler()
        -->new LocalSchedulerBackend()

再分区
-----------------------
    shuffle动作可以重新指定分区。
    reduceByKey(f, n) ;
    groupByKey(n) ;

map变换的操作
---------------------
    1.mapPartitions
        //映射分区
        val rdd1 = sc.parallelize(1 to 10 , 2)
        val rdd2 = rdd1.mapPartitions(it => {
            println("start")
            var list:List[Int] = Nil
            for(e <- it){
                list = (e * 2)::list
            }
            println("end")
            list.iterator
        } )

        //数据入库问题,可以使用forEachPartition
        val  rdd2 = rdd1.mapPartitions(it=>{
            Class.forName("com.mysql.jdbc.Driver")
            val url = "jdbc:mysql://localhost:3306/big9"
            val user = "root"
            val pass = "root"
            val conn = DriverManager.getConnection(url,user,pass)
            conn.setAutoCommit(false);
            val sql ="insert into tt(id) values(?)" ;
            val ppst = conn.prepareStatement(sql)
            for(e <- it){
                ppst.setInt(1, e)
                ppst.executeUpdate()
            }
            conn.commit()
            it
        })

    2.union
        联合的分区数是父RDD的分区数总和.
        rdd1.union(rdd2) ;

    3.intersection
        交集,需要shuffle。采用协分组。
        val rdd3 = rdd1.intersection(rdd2,4)
        println(rdd3.getNumPartitions)
```

```
        val rdd4 = rdd3.mapPartitionsWithIndex((n,it)=>{
            for(e <- it){
                println(n + " : " + e)
            }
            it
        })

    4.distinct
        去重，内部通过reduceByKey实现。
        map(x => (x, null)).reduceByKey((x, y) => x, numPartitions).map(_._1)

    5.groupByKey
        需要shuffle。
        val rdd3 = rdd1.map(e=>{
            (if(e%2==0) "even" else "odd" , e)
        })
        val rdd4 = rdd3.groupByKey(4);

    6.reduceByKey()
        可以指定分区数。


    7.aggregateByKey
        按key聚合，
        zeroU是初始值。
        该函数现在分区内进行预聚合，使用函数f1.
        分区间聚合使用f2函数。

        val zeroU = "X"
        def f1(a:String,b:Int) :String = a + "(f1)" + b
        def f2(a:String,b:String) = a + "(f2)" + b
        val rdd4 = rdd3.aggregateByKey(zeroU)(f1,f2)

    8.join
        leftOuterJoin
        rightOuterJoin
        fullOuterJoin

    9.cogroup
        协分组.
        对当前的rdd和传递的rdd参数进行联合分组，将相同的key的v组合成元组.
        rdd1.cogroup(rdd2)
        (K, (List[V1],List[V2]))

    10.groupByKey
        对当前的rdd进行按key分组。

    11.cartesian
        不需要shuffle.
        for (x <- rdd1.iterator(currSplit.s1, context);
             y <- rdd2.iterator(currSplit.s2, context)) yield (x, y)

    12.coalesce
        再分区。
        推荐不进行shuffle来降低分区数，产生新的rdd。
        如果增加分区数，必须shuffle=true，否则无法实现增加目的。

    13.repartition
        恒使用shuffle操作，对于减少分区的动作，推荐使用coalesce.
        coalesce(numPartitions, shuffle = true)

    14.repartitionAndSortWithinPartitions
        分区带排序
        rdd1.repartitionAndSortWithinPartitions(new HashPartitioner(3))

groupByKey和reduceByKey
-----------------------
    groupByKey不会进行map端combine操作。
    reduceByKey()默认进行map端combine操作，减少网络负载。


练习
```

```
--------------------
    1.使用aggregateByKey函数实现groupByKey功能，将所有v放入到集合List中.
        val zeroU:List[Int] = Nil
        def f1(a:List[Int] , b:Int) = b :: a
        def f2(a:List[Int],b:List[Int]) = a ++ b

    2.准备cust.txt和orders.txt,
        使用spark实现如下计算
        2.1)查询每个客户的订单编号集合
            集合形式:(1,tomas,List(No001,No002))
        2.2)查询每个客户订单价格的总和
            集合形式:(1,tomas,300.4)
        2.3)查询没有订单的客户


        import org.apache.spark.{SparkConf, SparkContext}

        /**
          * Created by Administrator on 2018/3/2.
          */
        object SimpleApp3 {
            case class Customer(id:Int,name:String,age:Int)
            case class Order(id:Int,orderno:String,price:Float,cid:Int)
            def main(args: Array[String]): Unit = {
                val conf = new SparkConf()
                conf.setAppName("app")
                conf.setMaster("local[3]")

                val sc = new SparkContext(conf)

                val crdd1 = sc.textFile("file:///d:/java/custs.txt")
                val ordd1 = sc.textFile("file:///d:/java/orders.txt")

                val crdd2 = crdd1.map(e=>{
                    val arr = e.split(",")
                    (arr(0).toInt , Customer(arr(0).toInt ,arr(1),arr(2).toInt))
                })

                val ordd2 = ordd1.map(e=>{
                    val arr = e.split(",")
                    (arr(3).toInt, Order(arr(0).toInt, arr(1), arr(2).toFloat ,
                    arr(3).toInt))
                })

                //连接
                val ardd1 = crdd2.join(ordd2)
                    .map(t=>(t._2._1,t._2._2.orderno))
                    .groupByKey().map(t=>(t._1.id,t._1.name,t._2.toList))

                //ordd2.foreach(println)
                ardd1.foreach(println)
                while(true){
                    Thread.sleep(1000)
                }
            }
        }
```