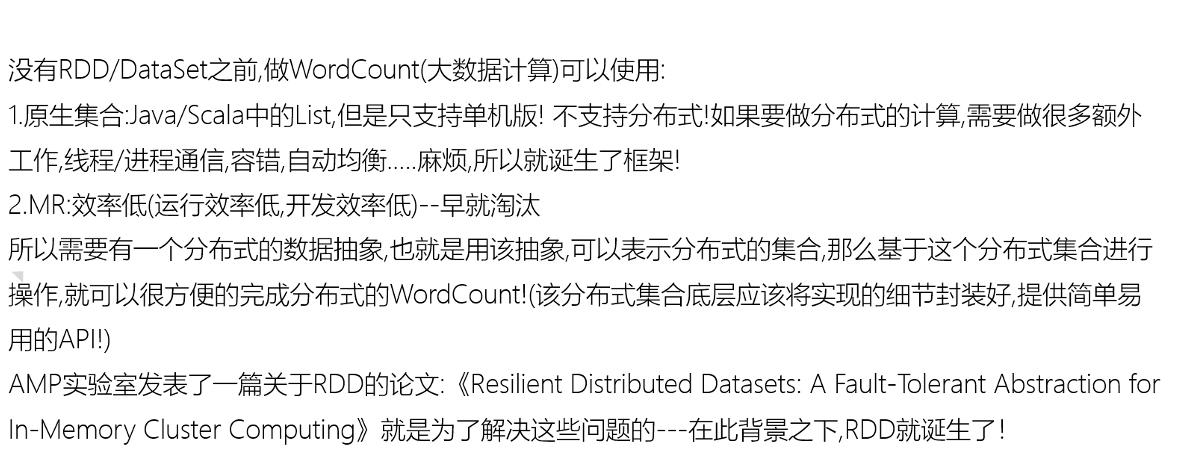
# RDD详解

## why?



## what?

A Resilient Distributed Dataset (RDD), the basic abstraction in Spark.

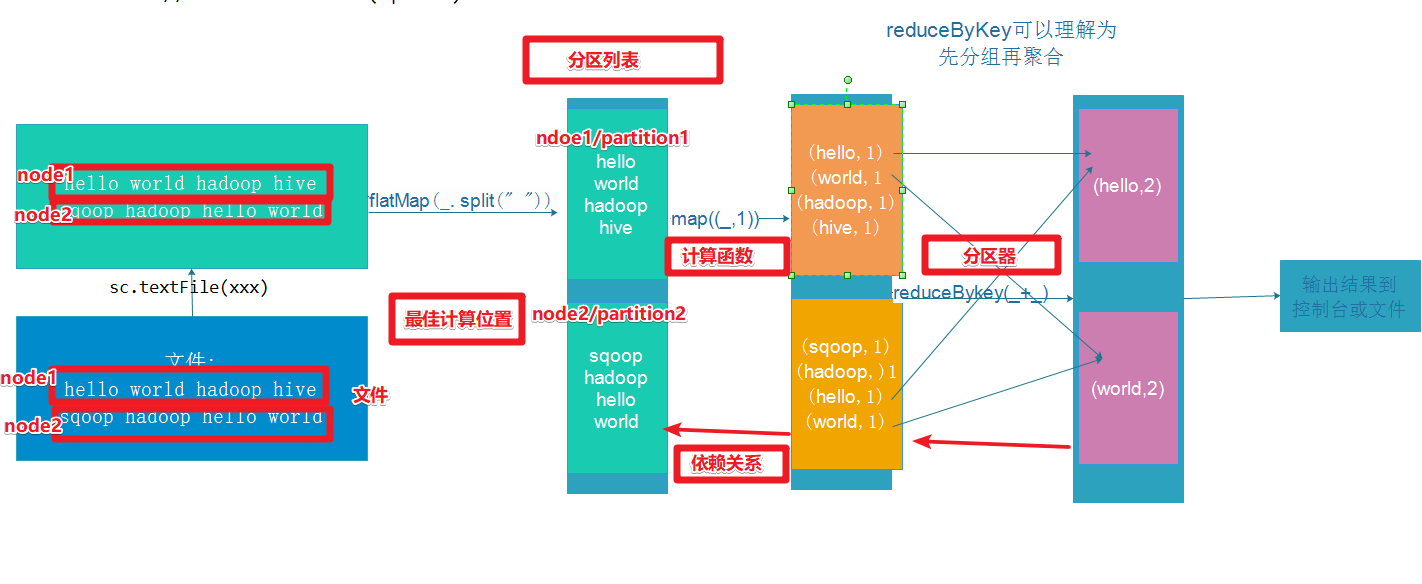
RDD:弹性分布式数据集,是Spark中最基本的数据抽象,用来表示分布式集合,支持分布式操作!

## 五大属性

Internally, each RDD is characterized by five main properties:

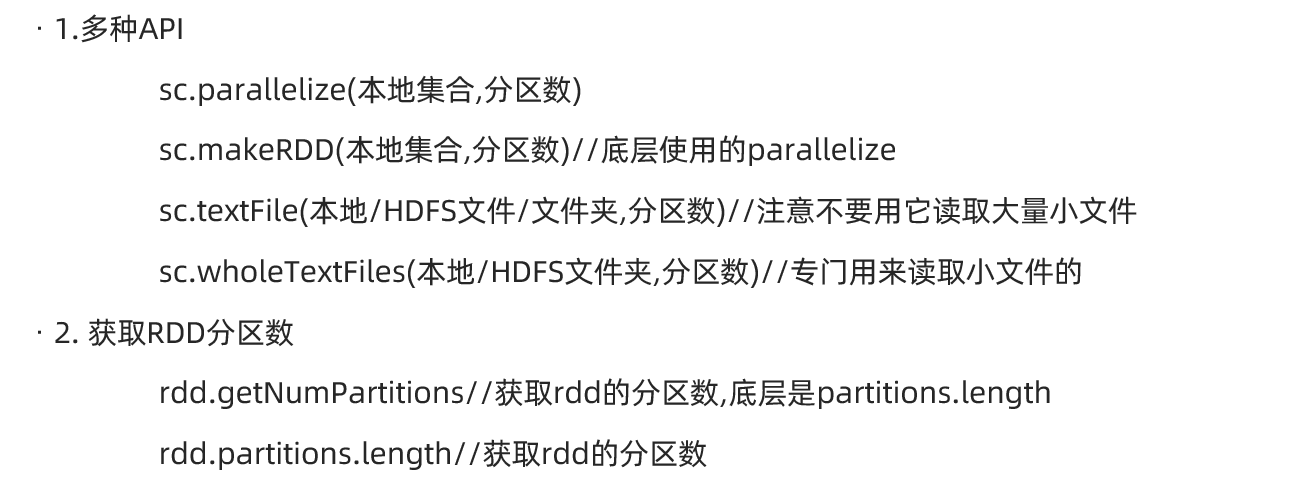
* 分区列表: A list of partitions
* 计算函数: A function for computing each split
* 依赖关系: A list of dependencies on other RDDs
* 分区器: Optionally, a Partitioner for key-value RDDs (e.g. to say that the RDD is hash-partitioned)
* 计算位置：Optionally, a list of preferred locations to compute each split on (e.g. block locations for  
  an HDFS file)

## WordCount中的RDD的五大属性



# RDD的创建

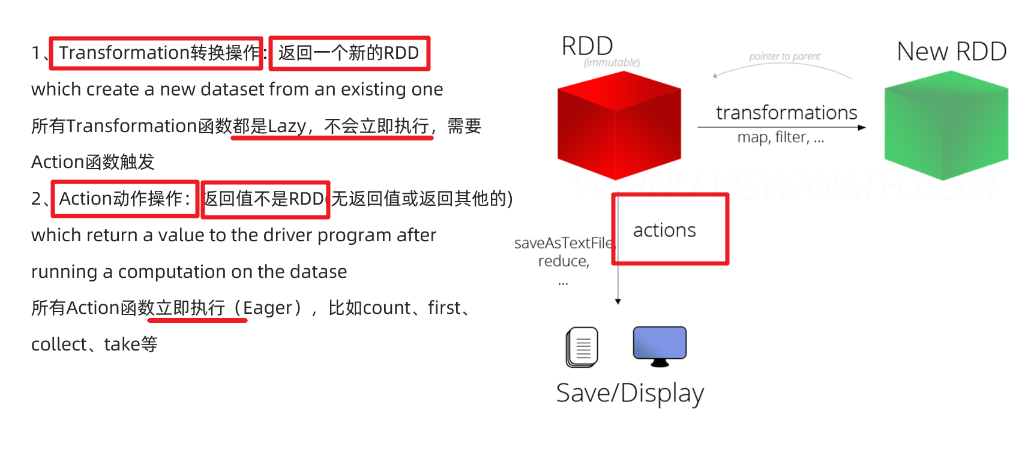
RDD中的数据可以来源于2个地方：本地集合或外部数据源



package com.as.core  
  
import org.apache.spark.rdd.RDD  
import org.apache.spark.{SparkConf, SparkContext}  
  
/\*\*  
 \* Desc 演示RDD的创建  
 \*/  
object RDDDemo01\_Create {  
 def main(args: Array[String]): Unit = {  
 //TODO 0.env/创建环境  
 val conf: SparkConf = new SparkConf().setAppName("spark").setMaster("local[\*]")  
 val sc: SparkContext = new SparkContext(conf)  
 sc.setLogLevel("WARN")  
  
 //TODO 1.source/加载数据/创建RDD  
 val rdd1: RDD[Int] = sc.parallelize(1 to 10) //8  
 val rdd2: RDD[Int] = sc.parallelize(1 to 10,3) //3  
  
 val rdd3: RDD[Int] = sc.makeRDD(1 to 10)//底层是parallelize //8  
 val rdd4: RDD[Int] = sc.makeRDD(1 to 10,4) //4  
  
 //RDD[一行行的数据]  
 val rdd5: RDD[String] = sc.textFile("data/input/words.txt")//2  
 val rdd6: RDD[String] = sc.textFile("data/input/words.txt",3)//3  
 //RDD[一行行的数据]  
 val rdd7: RDD[String] = sc.textFile("data/input/ratings10")//10  
 val rdd8: RDD[String] = sc.textFile("data/input/ratings10",3)//10  
 //RDD[(文件名, 一行行的数据),(文件名, 一行行的数据)....]  
 val rdd9: RDD[(String, String)] = sc.wholeTextFiles("data/input/ratings10")//2  
 val rdd10: RDD[(String, String)] = sc.wholeTextFiles("data/input/ratings10",3)//3  
  
 println(rdd1.getNumPartitions)//8 //底层partitions.length  
 println(rdd2.partitions.length)//3  
 println(rdd3.getNumPartitions)//8  
 println(rdd4.getNumPartitions)//4  
 println(rdd5.getNumPartitions)//2  
 println(rdd6.getNumPartitions)//3  
 println(rdd7.getNumPartitions)//10  
 println(rdd8.getNumPartitions)//10  
 println(rdd9.getNumPartitions)//2  
 println(rdd10.getNumPartitions)//3  
  
 //TODO 2.transformation  
 //TODO 3.sink/输出  
 }  
}

# RDD操作

## 分类



## 基本算子/操作/方法/API

map

faltMap

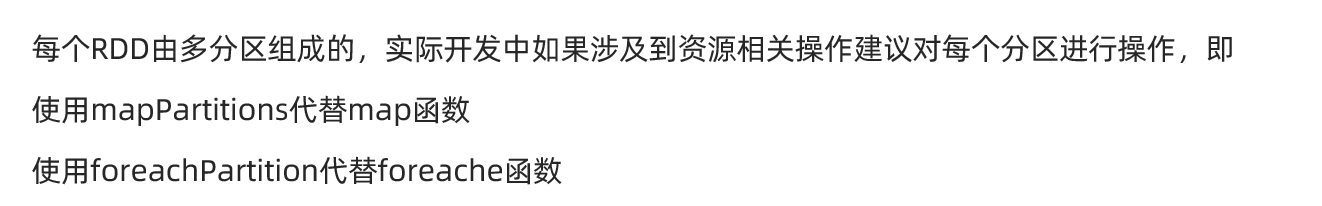
filter

foreach

saveAsTextFile

package com.as.core  
  
import org.apache.commons.lang3.StringUtils  
import org.apache.spark.rdd.RDD  
import org.apache.spark.{SparkConf, SparkContext}  
  
/\*\*  
 \* Desc 演示RDD的基本操作  
 \*/  
object RDDDemo02\_Basic {  
 def main(args: Array[String]): Unit = {  
 //TODO 0.env/创建环境  
 val conf: SparkConf = new SparkConf().setAppName("spark").setMaster("local[\*]")  
 val sc: SparkContext = new SparkContext(conf)  
 sc.setLogLevel("WARN")  
  
 //TODO 1.source/加载数据/创建RDD  
 val lines: RDD[String] = sc.textFile("data/input/words.txt") //2  
  
 //TODO 2.transformation  
 val result: RDD[(String, Int)] = lines.filter(StringUtils.isNoneBlank(\_))  
 .flatMap(\_.split(" "))  
 .map((\_, 1))  
 .reduceByKey(\_ + \_)  
  
 //TODO 3.sink/输出/action  
 result.foreach(println)  
 result.saveAsTextFile("data/output/result4")  
 }  
}

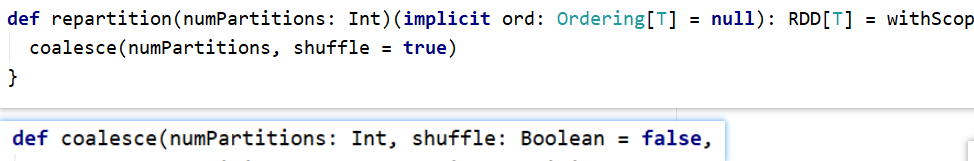
## 分区操作



package com.as.core  
  
import org.apache.commons.lang3.StringUtils  
import org.apache.spark.rdd.RDD  
import org.apache.spark.{SparkConf, SparkContext}  
  
/\*\*  
 \* Desc 演示RDD的分区操作  
 \*/  
object RDDDemo03\_PartitionOperation {  
 def main(args: Array[String]): Unit = {  
 //TODO 0.env/创建环境  
 val conf: SparkConf = new SparkConf().setAppName("spark").setMaster("local[\*]")  
 val sc: SparkContext = new SparkContext(conf)  
 sc.setLogLevel("WARN")  
  
 //TODO 1.source/加载数据/创建RDD  
 val lines: RDD[String] = sc.textFile("data/input/words.txt")  
  
 //TODO 2.transformation  
 val result: RDD[(String, Int)] = lines.filter(StringUtils.isNoneBlank(\_))  
 .flatMap(\_.split(" "))  
 //.map((\_, 1)) //注意:map是针对分区中的每一条数据进行操作  
 /\*.map(word=>{  
 //开启连接--有几条数据就执行几次  
 (word,1)  
 //关闭连接  
 })\*/  
 // f: Iterator[T] => Iterator[U]  
 .mapPartitions(iter=>{//注意:mapPartitions是针对每个分区进行操作  
 //开启连接--有几个分区就执行几次  
 iter.map((\_, 1))//注意:这里是作用在该分区的每一条数据上  
 //关闭连接  
 })  
 .reduceByKey(\_ + \_)  
  
 //TODO 3.sink/输出/action  
 //Applies a function f to all elements of this RDD.  
 /\*result.foreach(i=>{  
 //开启连接--有几条数据就执行几次  
 println(i)  
 //关闭连接  
 })\*/  
 //Applies a function f to each partition of this RDD.  
 result.foreachPartition(iter=>{  
 //开启连接--有几个分区就执行几次  
 iter.foreach(println)  
 //关闭连接  
 })  
  
  
 //result.saveAsTextFile("data/output/result4")  
 }  
}

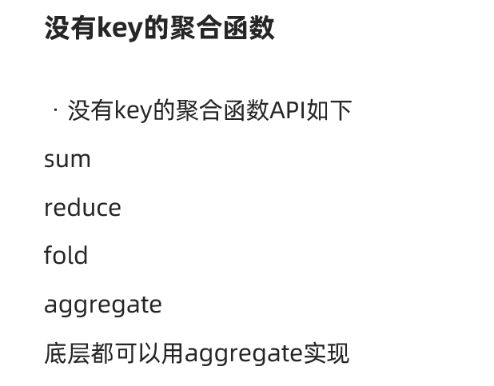
## 重分区操作



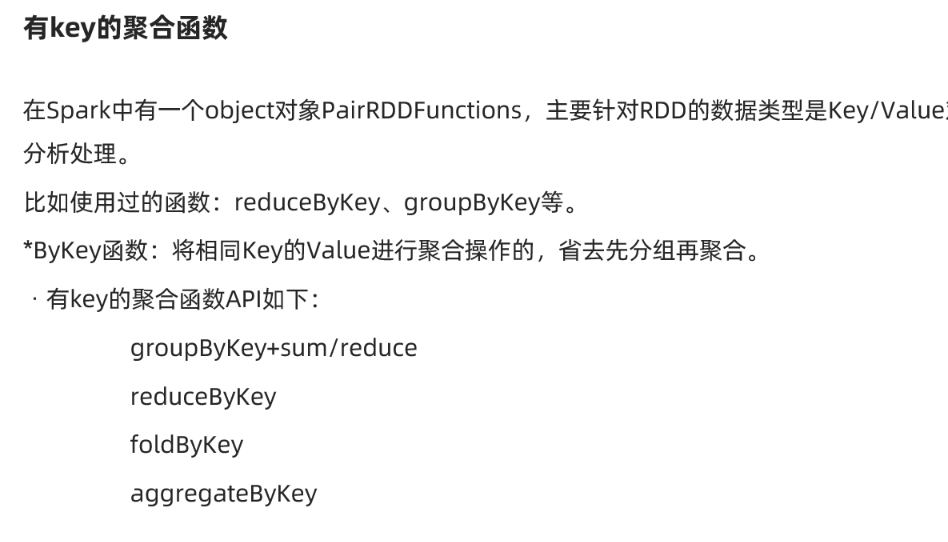


package com.as.core  
  
import org.apache.spark.rdd.RDD  
import org.apache.spark.{SparkConf, SparkContext}  
  
/\*\*  
 \* Desc 演示RDD的重分区  
 \*/  
object RDDDemo04\_RePartition{  
 def main(args: Array[String]): Unit = {  
 //TODO 0.env/创建环境  
 val conf: SparkConf = new SparkConf().setAppName("spark").setMaster("local[\*]")  
 val sc: SparkContext = new SparkContext(conf)  
 sc.setLogLevel("WARN")  
  
 //TODO 1.source/加载数据/创建RDD  
 val rdd1: RDD[Int] = sc.parallelize(1 to 10) //8  
 //repartition可以增加或减少分区,注意:原来的不变  
 val rdd2: RDD[Int] = rdd1.repartition(9)//底层是coalesce(分区数,shuffle=true)  
 val rdd3: RDD[Int] = rdd1.repartition(7)  
 println(rdd1.getNumPartitions)//8  
 println(rdd2.getNumPartitions)//9  
 println(rdd3.getNumPartitions)//7  
  
 //coalesce默认只能减少分区,除非把shuffle指定为true,注意:原来的不变  
 val rdd4: RDD[Int] = rdd1.coalesce(9)//底层是coalesce(分区数,shuffle=false)  
 val rdd5: RDD[Int] = rdd1.coalesce(7)  
 val rdd6: RDD[Int] = rdd1.coalesce(9,true)  
 println(rdd4.getNumPartitions)//8  
 println(rdd5.getNumPartitions)//7  
 println(rdd6.getNumPartitions)//9  
  
 //TODO 2.transformation  
 //TODO 3.sink/输出  
 }  
}

## 聚合操作

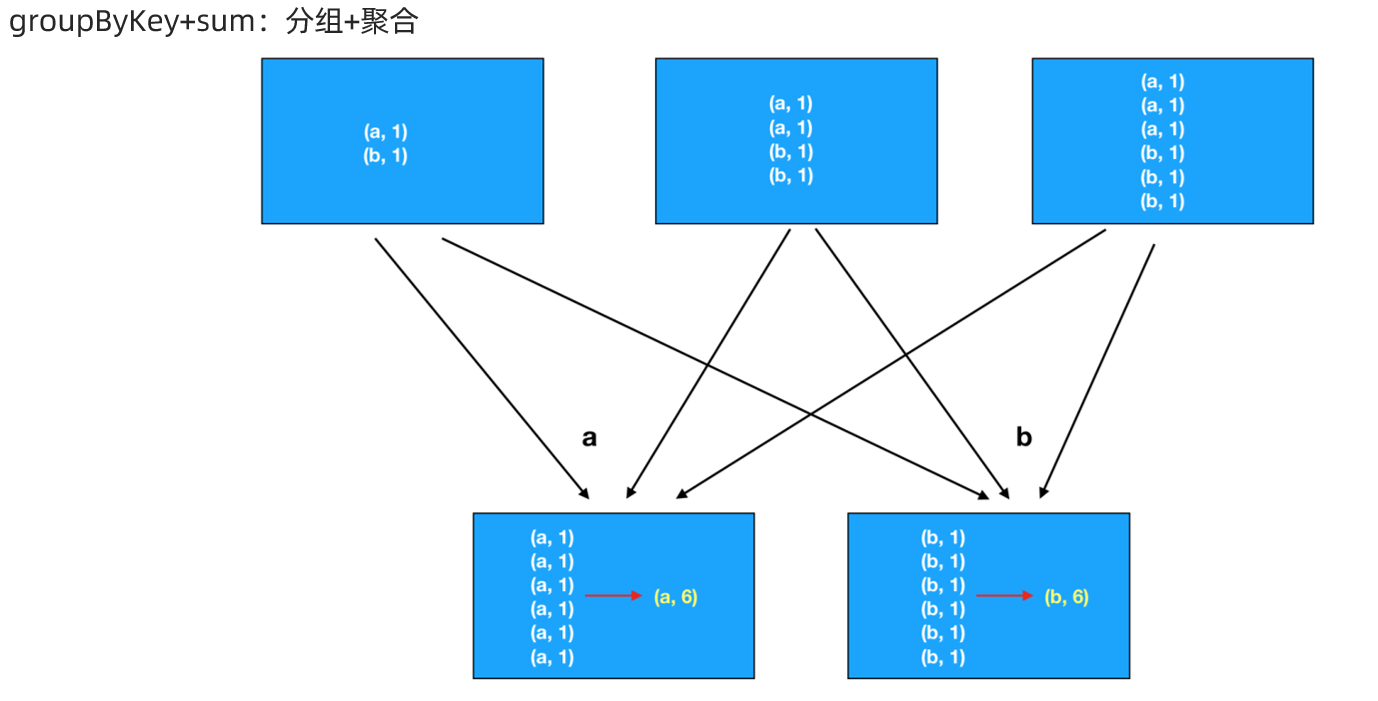


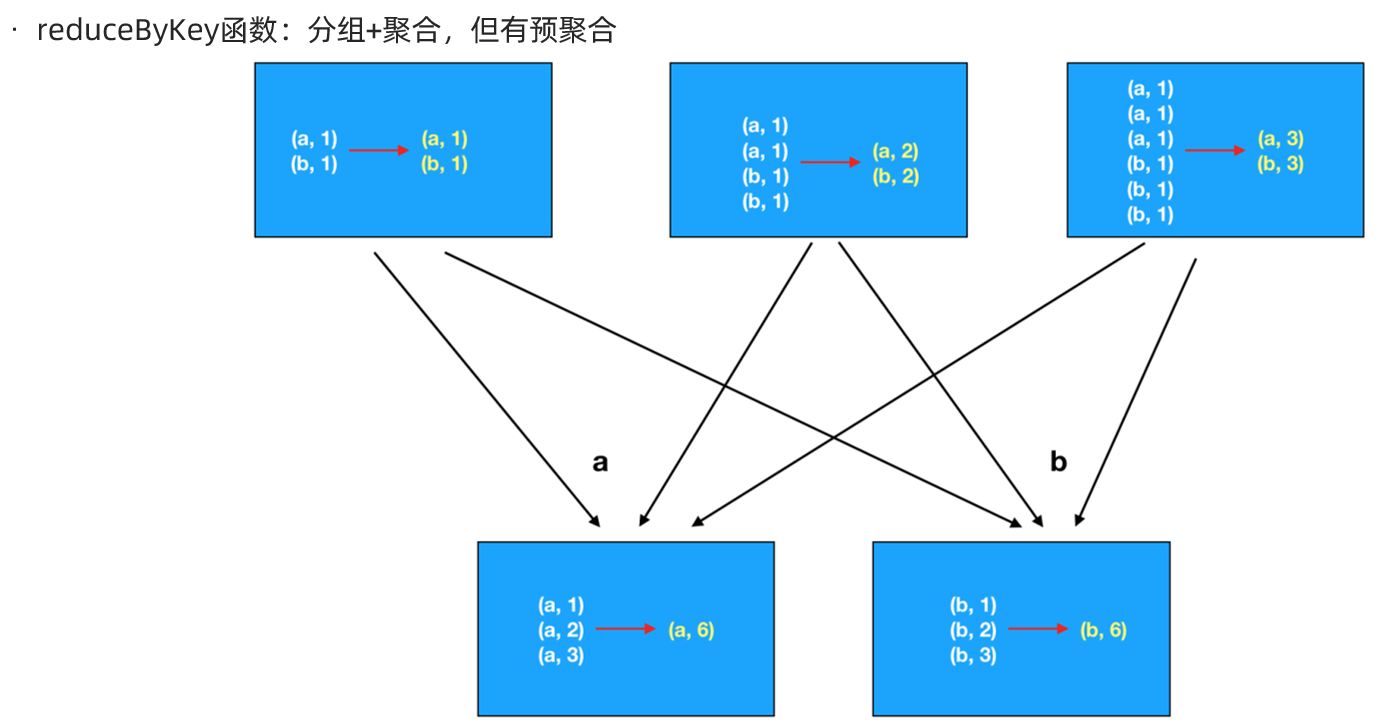
package com.as.core  
  
import org.apache.spark.rdd.RDD  
import org.apache.spark.{SparkConf, SparkContext}  
  
/\*\*  
 \* Desc 演示RDD的聚合-没有key  
 \*/  
object RDDDemo05\_Aggregate\_NoKey{  
 def main(args: Array[String]): Unit = {  
 //TODO 0.env/创建环境  
 val conf: SparkConf = new SparkConf().setAppName("spark").setMaster("local[\*]")  
 val sc: SparkContext = new SparkContext(conf)  
 sc.setLogLevel("WARN")  
  
 //TODO 1.source/加载数据/创建RDD  
 val rdd1: RDD[Int] = sc.parallelize(1 to 10) //8  
 //TODO 2.transformation  
  
 //TODO 3.sink/输出/Action  
 //需求求rdd1中各个元素的和  
 println(rdd1.sum())  
 println(rdd1.reduce(\_ + \_))  
 println(rdd1.fold(0)(\_ + \_))  
 //aggregate(初始值)(局部聚合, 全局聚合)  
 println(rdd1.aggregate(0)(\_ + \_, \_ + \_))  
 //55.0  
 //55  
 //55  
 //55  
 }  
}



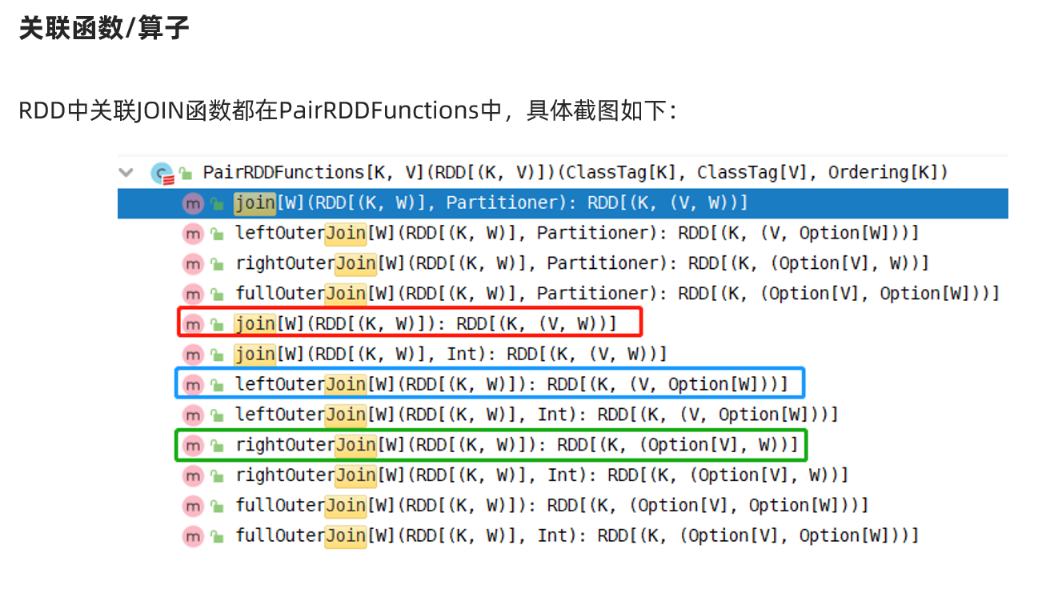
package com.as.core  
  
import org.apache.commons.lang3.StringUtils  
import org.apache.spark.rdd.RDD  
import org.apache.spark.{SparkConf, SparkContext}  
  
/\*\*  
 \* Desc 演示RDD的聚合-有key  
 \*/  
object RDDDemo06\_Aggregate\_Key {  
 def main(args: Array[String]): Unit = {  
 //TODO 0.env/创建环境  
 val conf: SparkConf = new SparkConf().setAppName("spark").setMaster("local[\*]")  
 val sc: SparkContext = new SparkContext(conf)  
 sc.setLogLevel("WARN")  
  
 //TODO 1.source/加载数据/创建RDD  
 //RDD[一行行的数据]  
 val lines: RDD[String] = sc.textFile("data/input/words.txt")  
  
 //TODO 2.transformation  
 //RDD[(单词, 1)]  
 val wordAndOneRDD: RDD[(String, Int)] = lines.filter(StringUtils.isNoneBlank(\_))  
 .flatMap(\_.split(" "))  
 .map((\_, 1))  
 //分组+聚合  
 //groupBy/groupByKey + sum/reduce  
 //wordAndOneRDD.groupBy(\_.\_1)  
 val grouped: RDD[(String, Iterable[Int])] = wordAndOneRDD.groupByKey()  
 //grouped.mapValues(\_.reduce(\_+\_))  
 val result: RDD[(String, Int)] = grouped.mapValues(\_.sum)  
 //reduceByKey  
 val result2: RDD[(String, Int)] = wordAndOneRDD.reduceByKey(\_+\_)  
 //foldByKey  
 val result3: RDD[(String, Int)] = wordAndOneRDD.foldByKey(0)(\_+\_)  
 //aggregateByKeye(初始值)(局部聚合, 全局聚合)  
 val result4: RDD[(String, Int)] = wordAndOneRDD.aggregateByKey(0)(\_ + \_, \_ + \_)  
  
 //TODO 3.sink/输出  
 result.foreach(println)  
 result2.foreach(println)  
 result3.foreach(println)  
 result4.foreach(println)  
 }  
}

## reduceByKey和groupByKey的区别





## 关联操作



package com.as.core  
  
import org.apache.commons.lang3.StringUtils  
import org.apache.spark.rdd.RDD  
import org.apache.spark.{SparkConf, SparkContext}  
  
/\*\*  
 \* Desc 演示RDD的join  
 \*/  
object RDDDemo07\_Join {  
 def main(args: Array[String]): Unit = {  
 //TODO 0.env/创建环境  
 val conf: SparkConf = new SparkConf().setAppName("spark").setMaster("local[\*]")  
 val sc: SparkContext = new SparkContext(conf)  
 sc.setLogLevel("WARN")  
  
 //TODO 1.source/加载数据/创建RDD  
 //员工集合:RDD[(部门编号, 员工姓名)]  
 val empRDD: RDD[(Int, String)] = sc.parallelize(  
 Seq((1001, "zhangsan"), (1002, "lisi"), (1003, "wangwu"))  
 )  
 //部门集合:RDD[(部门编号, 部门名称)]  
 val deptRDD: RDD[(Int, String)] = sc.parallelize(  
 Seq((1001, "销售部"), (1002, "技术部"), (1004, "客服部"))  
 )  
  
 //TODO 2.transformation  
 //需求:求员工对应的部门名称  
 val result1: RDD[(Int, (String, String))] = empRDD.join(deptRDD)  
 val result2: RDD[(Int, (String, Option[String]))] = empRDD.leftOuterJoin(deptRDD)  
 val result3: RDD[(Int, (Option[String], String))] = empRDD.rightOuterJoin(deptRDD)  
  
  
 //TODO 3.sink/输出  
 result1.foreach(println)  
 result2.foreach(println)  
 result3.foreach(println)  
 //(1002,(lisi,技术部))  
 //(1001,(zhangsan,销售部))  
  
 //(1002,(lisi,Some(技术部)))  
 //(1001,(zhangsan,Some(销售部)))  
 //(1003,(wangwu,None))  
  
 //(1004,(None,客服部))  
 //(1001,(Some(zhangsan),销售部))  
 //(1002,(Some(lisi),技术部))  
  
 }  
}

## 排序操作

https://www.runoob.com/w3cnote/ten-sorting-algorithm.html

sortBy

sortByKey

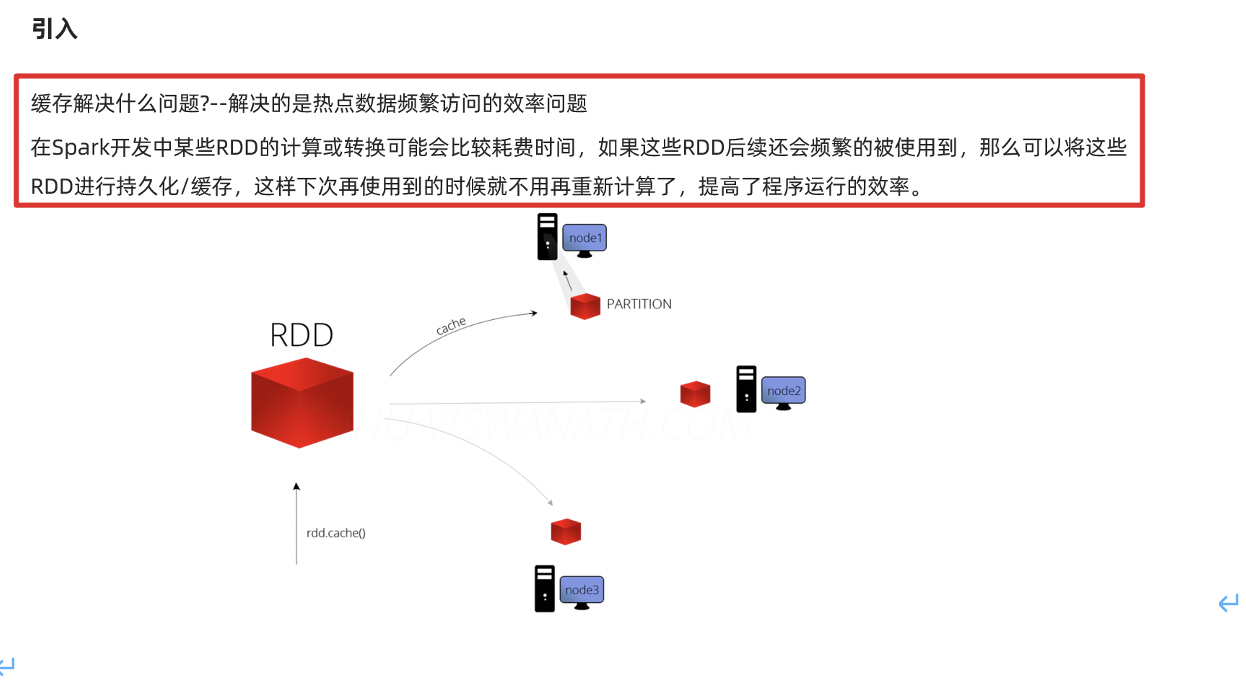
top

//需求:求WordCount结果的top3

package com.as.core  
  
import org.apache.commons.lang3.StringUtils  
import org.apache.spark.rdd.RDD  
import org.apache.spark.{SparkConf, SparkContext}  
import spire.std.tuples  
  
/\*\*  
 \* Desc 演示RDD的排序  
 \*/  
object RDDDemo08\_Sort {  
 def main(args: Array[String]): Unit = {  
 //TODO 0.env/创建环境  
 val conf: SparkConf = new SparkConf().setAppName("spark").setMaster("local[\*]")  
 val sc: SparkContext = new SparkContext(conf)  
 sc.setLogLevel("WARN")  
  
 //TODO 1.source/加载数据/创建RDD  
 //RDD[一行行的数据]  
 val lines: RDD[String] = sc.textFile("data/input/words.txt")  
  
 //TODO 2.transformation  
 //RDD[(单词, 数量)]  
 val result: RDD[(String, Int)] = lines.filter(StringUtils.isNoneBlank(\_))  
 .flatMap(\_.split(" "))  
 .map((\_, 1))  
 .reduceByKey(\_ + \_)  
  
 //需求:对WordCount的结果进行排序,取出top3  
 val sortResult1: Array[(String, Int)] = result  
 .sortBy(\_.\_2, false) //按照数量降序排列  
 .take(3)//取出前3个  
  
 //result.map(t=>(t.\_2,t.\_1))  
 val sortResult2: Array[(Int, String)] = result.map(\_.swap)  
 .sortByKey(false)//按照数量降序排列  
 .take(3)//取出前3个  
  
 val sortResult3: Array[(String, Int)] = result.top(3)(Ordering.by(\_.\_2)) //topN默认就是降序  
  
 //TODO 3.sink/输出  
 result.foreach(println)  
 //(hello,4)  
 //(you,2)  
 //(me,1)  
 //(her,3)  
 sortResult1.foreach(println)  
 //(hello,4)  
 //(her,3)  
 //(you,2)  
 sortResult2.foreach(println)  
 //(4,hello)  
 //(3,her)  
 //(2,you)  
 sortResult3.foreach(println)  
 //(hello,4)  
 //(her,3)  
 //(you,2)  
  
 }  
}

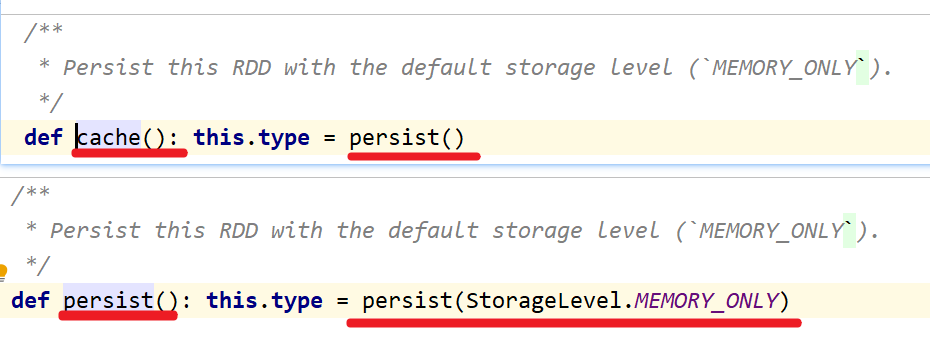
# RDD的缓存/持久化

## API

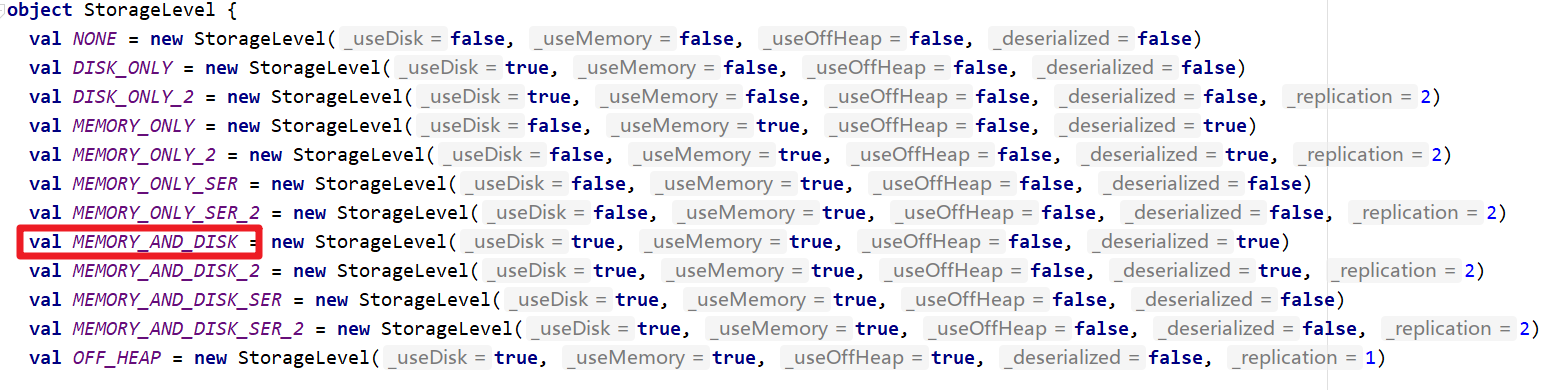




* 源码



* 缓存级别

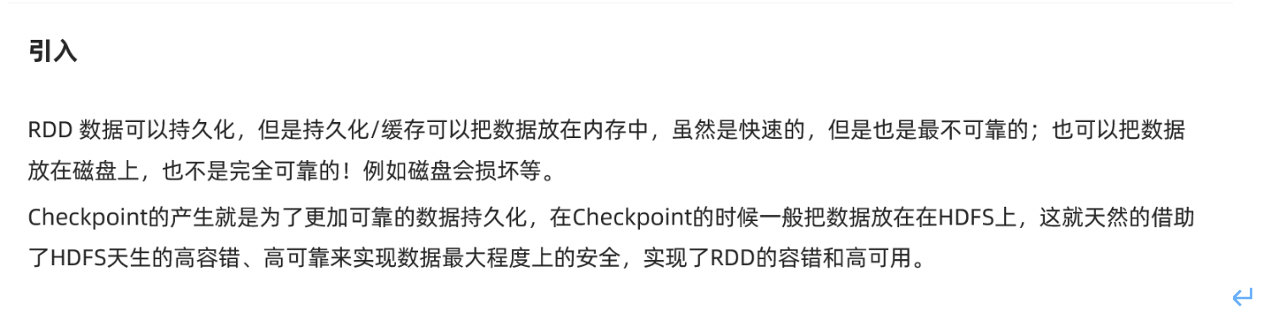


## 代码演示

package com.as.core  
  
import org.apache.commons.lang3.StringUtils  
import org.apache.spark.rdd.RDD  
import org.apache.spark.storage.StorageLevel  
import org.apache.spark.{SparkConf, SparkContext}  
  
/\*\*  
 \* Desc 演示RDD的缓存/持久化  
 \*/  
object RDDDemo09\_Cache\_Persist{  
 def main(args: Array[String]): Unit = {  
 //TODO 0.env/创建环境  
 val conf: SparkConf = new SparkConf().setAppName("spark").setMaster("local[\*]")  
 val sc: SparkContext = new SparkContext(conf)  
 sc.setLogLevel("WARN")  
  
 //TODO 1.source/加载数据/创建RDD  
 //RDD[一行行的数据]  
 val lines: RDD[String] = sc.textFile("data/input/words.txt")  
  
 //TODO 2.transformation  
 //RDD[(单词, 数量)]  
 val result: RDD[(String, Int)] = lines.filter(StringUtils.isNoneBlank(\_))  
 .flatMap(\_.split(" "))  
 .map((\_, 1))  
 .reduceByKey(\_ + \_)  
  
 //TODO =====注意:resultRDD在后续会被频繁使用到,且该RDD的计算过程比较复杂,所以为了提高后续访问该RDD的效率,应该将该RDD放到缓存中  
 //result.cache()//底层persist()  
 //result.persist()//底层persist(StorageLevel.MEMORY\_ONLY)  
 result.persist(StorageLevel.MEMORY\_AND\_DISK)//底层persist(StorageLevel.MEMORY\_ONLY)  
  
  
 //需求:对WordCount的结果进行排序,取出top3  
 val sortResult1: Array[(String, Int)] = result  
 .sortBy(\_.\_2, false) //按照数量降序排列  
 .take(3)//取出前3个  
  
 //result.map(t=>(t.\_2,t.\_1))  
 val sortResult2: Array[(Int, String)] = result.map(\_.swap)  
 .sortByKey(false)//按照数量降序排列  
 .take(3)//取出前3个  
  
 val sortResult3: Array[(String, Int)] = result.top(3)(Ordering.by(\_.\_2)) //topN默认就是降序  
  
 result.unpersist()  
   
 //TODO 3.sink/输出  
 result.foreach(println)  
 //(hello,4)  
 //(you,2)  
 //(me,1)  
 //(her,3)  
 sortResult1.foreach(println)  
 //(hello,4)  
 //(her,3)  
 //(you,2)  
 sortResult2.foreach(println)  
 //(4,hello)  
 //(3,her)  
 //(2,you)  
 sortResult3.foreach(println)  
 //(hello,4)  
 //(her,3)  
 //(you,2)  
  
 }  
}

# RDD的checkpoint

## API





## 代码实现

package com.as.core  
  
import org.apache.commons.lang3.StringUtils  
import org.apache.spark.rdd.RDD  
import org.apache.spark.storage.StorageLevel  
import org.apache.spark.{SparkConf, SparkContext}  
  
/\*\*  
 \* Desc 演示RDD的Checkpoint/检查点设置  
 \*/  
object RDDDemo10\_Checkpoint{  
 def main(args: Array[String]): Unit = {  
 //TODO 0.env/创建环境  
 val conf: SparkConf = new SparkConf().setAppName("spark").setMaster("local[\*]")  
 val sc: SparkContext = new SparkContext(conf)  
 sc.setLogLevel("WARN")  
  
 //TODO 1.source/加载数据/创建RDD  
 //RDD[一行行的数据]  
 val lines: RDD[String] = sc.textFile("data/input/words.txt")  
  
 //TODO 2.transformation  
 //RDD[(单词, 数量)]  
 val result: RDD[(String, Int)] = lines.filter(StringUtils.isNoneBlank(\_))  
 .flatMap(\_.split(" "))  
 .map((\_, 1))  
 .reduceByKey(\_ + \_)  
  
 //TODO =====注意:resultRDD在后续会被频繁使用到,且该RDD的计算过程比较复杂,所以为了提高后续访问该RDD的效率,应该将该RDD放到缓存中  
 //result.cache()//底层persist()  
 //result.persist()//底层persist(StorageLevel.MEMORY\_ONLY)  
 result.persist(StorageLevel.MEMORY\_AND\_DISK)//底层persist(StorageLevel.MEMORY\_ONLY)  
 //TODO =====注意:上面的缓存持久化并不能保证RDD数据的绝对安全,所以应使用Checkpoint把数据发在HDFS上  
 sc.setCheckpointDir("./ckp")//实际中写HDFS目录  
 result.checkpoint()  
  
  
 //需求:对WordCount的结果进行排序,取出top3  
 val sortResult1: Array[(String, Int)] = result  
 .sortBy(\_.\_2, false) //按照数量降序排列  
 .take(3)//取出前3个  
  
 //result.map(t=>(t.\_2,t.\_1))  
 val sortResult2: Array[(Int, String)] = result.map(\_.swap)  
 .sortByKey(false)//按照数量降序排列  
 .take(3)//取出前3个  
  
 val sortResult3: Array[(String, Int)] = result.top(3)(Ordering.by(\_.\_2)) //topN默认就是降序  
  
 result.unpersist()//清空缓存  
  
 //TODO 3.sink/输出  
 result.foreach(println)  
 //(hello,4)  
 //(you,2)  
 //(me,1)  
 //(her,3)  
 sortResult1.foreach(println)  
 //(hello,4)  
 //(her,3)  
 //(you,2)  
 sortResult2.foreach(println)  
 //(4,hello)  
 //(3,her)  
 //(2,you)  
 sortResult3.foreach(println)  
 //(hello,4)  
 //(her,3)  
 //(you,2)  
  
 }  
}

## 注意:缓存/持久化和Checkpoint检查点的区别

1.存储位置

缓存/持久化数据存默认存在内存, 一般设置为内存+磁盘(普通磁盘)

Checkpoint检查点:一般存储在HDFS

2.功能

缓存/持久化:保证数据后续使用的效率高

Checkpoint检查点:保证数据安全/也能一定程度上提高效率

3.对于依赖关系:

缓存/持久化:保留了RDD间的依赖关系

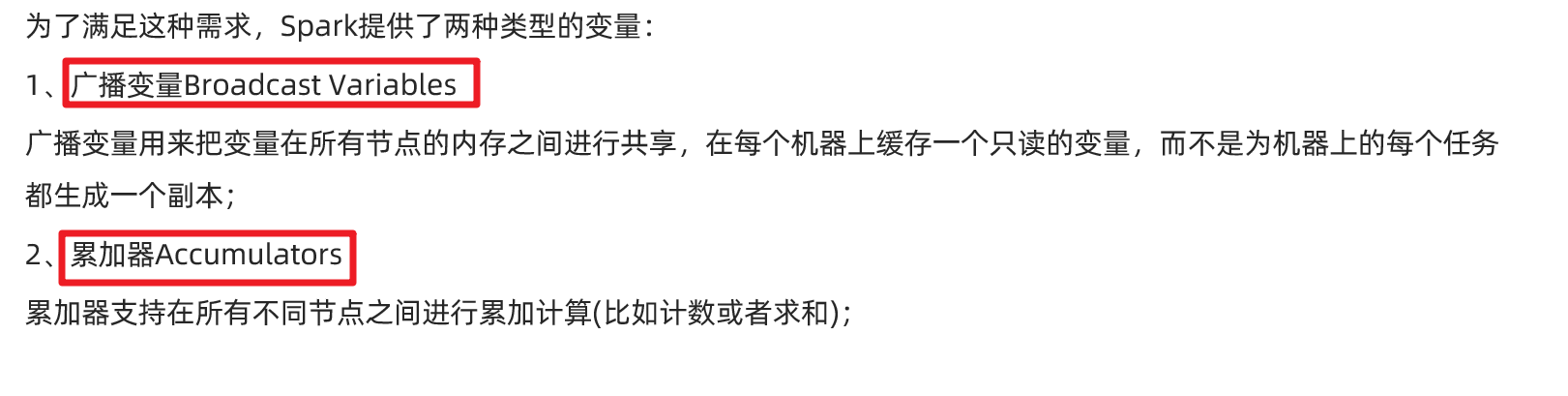
Checkpoint检查点:不保留RDD间的依赖关系

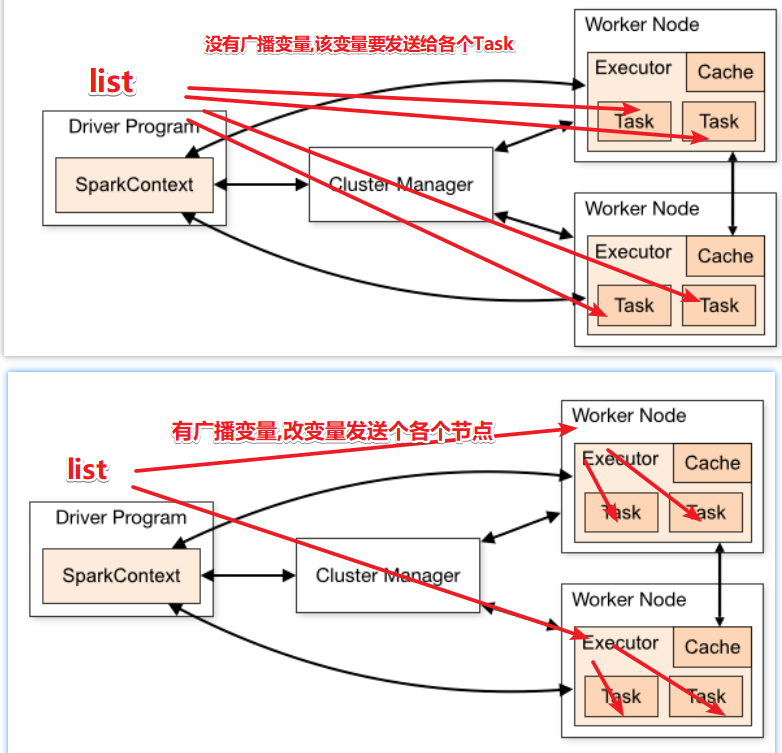
4.开发中如何使用?

对于计算复杂且后续会被频繁使用的RDD先进行缓存/持久化,再进行Checkpoint

sc.setCheckpointDir("./ckp")//实际中写HDFS目录  
rdd.persist(StorageLevel.MEMORY\_AND\_DISK)  
rdd.checkpoint()  
//频繁操作rdd  
result.unpersist()//清空缓存

# 共享变量

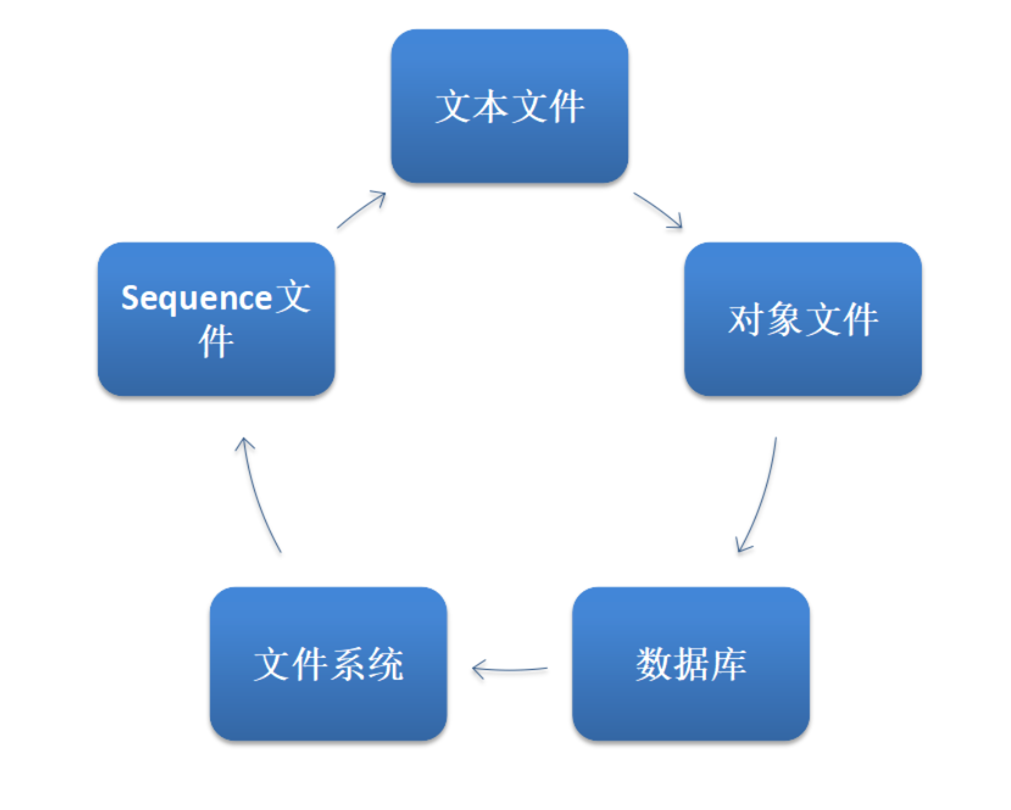




package com.as.core  
  
import java.lang  
  
import org.apache.commons.lang3.StringUtils  
import org.apache.spark.broadcast.Broadcast  
import org.apache.spark.rdd.RDD  
import org.apache.spark.storage.StorageLevel  
import org.apache.spark.util.LongAccumulator  
import org.apache.spark.{SparkConf, SparkContext}  
  
/\*\*  
 \* Desc 演示RDD的共享变量  
 \*/  
object RDDDemo11\_ShareVariable{  
 def main(args: Array[String]): Unit = {  
 //TODO 0.env/创建环境  
 val conf: SparkConf = new SparkConf().setAppName("spark").setMaster("local[\*]")  
 val sc: SparkContext = new SparkContext(conf)  
 sc.setLogLevel("WARN")  
  
 //需求:  
 // 以词频统计WordCount程序为例，处理的数据word2.txt所示，包括非单词符号，  
 // 做WordCount的同时统计出特殊字符的数量  
 //创建一个计数器/累加器  
 val mycounter: LongAccumulator = sc.longAccumulator("mycounter")  
 //定义一个特殊字符集合  
 val ruleList: List[String] = List(",", ".", "!", "#", "$", "%")  
 //将集合作为广播变量广播到各个节点  
 val broadcast: Broadcast[List[String]] = sc.broadcast(ruleList)  
  
 //TODO 1.source/加载数据/创建RDD  
 val lines: RDD[String] = sc.textFile("data/input/words2.txt")  
  
 //TODO 2.transformation  
 val wordcountResult: RDD[(String, Int)] = lines.filter(StringUtils.isNoneBlank(\_))  
 .flatMap(\_.split("\\s+"))  
 .filter(ch => {  
 //获取广播数据  
 val list: List[String] = broadcast.value  
 if (list.contains(ch)) { //如果是特殊字符  
 mycounter.add(1)  
 false  
 } else { //是单词  
 true  
 }  
 }).map((\_, 1))  
 .reduceByKey(\_ + \_)  
  
 //TODO 3.sink/输出  
 wordcountResult.foreach(println)  
 val chResult: lang.Long = mycounter.value  
 println("特殊字符的数量:"+chResult)  
  
 }  
}

# 外部数据源-了解

## 支持的多种格式



package com.as.core  
  
import java.lang  
  
import org.apache.commons.lang3.StringUtils  
import org.apache.spark  
import org.apache.spark.broadcast.Broadcast  
import org.apache.spark.rdd.RDD  
import org.apache.spark.util.LongAccumulator  
import org.apache.spark.{SparkConf, SparkContext, broadcast}  
  
/\*\*  
 \* Desc 演示RDD的外部数据源  
 \*/  
object RDDDemo12\_DataSource{  
 def main(args: Array[String]): Unit = {  
 //TODO 0.env/创建环境  
 val conf: SparkConf = new SparkConf().setAppName("spark").setMaster("local[\*]")  
 val sc: SparkContext = new SparkContext(conf)  
 sc.setLogLevel("WARN")  
  
 //TODO 1.source/加载数据/创建RDD  
 val lines: RDD[String] = sc.textFile("data/input/words.txt")  
  
 //TODO 2.transformation  
 val result: RDD[(String, Int)] = lines.filter(StringUtils.isNoneBlank(\_))  
 .flatMap(\_.split(" "))  
 .map((\_, 1))  
 .reduceByKey(\_ + \_)  
  
 //TODO 3.sink/输出  
 result.repartition(1).saveAsTextFile("data/output/result1")  
 result.repartition(1).saveAsObjectFile("data/output/result2")  
 result.repartition(1).saveAsSequenceFile("data/output/result3")  
  
 }  
}

## 支持的数据源-JDBC

需求:将数据写入到MySQL,再从MySQL读出来

package com.as.core  
  
import java.sql.{Connection, DriverManager, PreparedStatement, ResultSet}  
  
import org.apache.spark.rdd.{JdbcRDD, RDD}  
import org.apache.spark.{SparkConf, SparkContext}  
  
/\*\*  
 \* Desc 演示RDD的外部数据源  
 \*/  
object RDDDemo13\_DataSource2{  
 def main(args: Array[String]): Unit = {  
 //TODO 0.env/创建环境  
 val conf: SparkConf = new SparkConf().setAppName("spark").setMaster("local[\*]")  
 val sc: SparkContext = new SparkContext(conf)  
 sc.setLogLevel("WARN")  
  
 //TODO 1.source/加载数据/创建RDD  
 //RDD[(姓名, 年龄)]  
 val dataRDD: RDD[(String, Int)] = sc.makeRDD(List(("jack", 18), ("tom", 19), ("rose", 20)))  
  
 //TODO 2.transformation  
 //TODO 3.sink/输出  
 //需求:将数据写入到MySQL,再从MySQL读出来  
 /\*  
CREATE TABLE `t\_student` (  
 `id` int(11) NOT NULL AUTO\_INCREMENT,  
 `name` varchar(255) DEFAULT NULL,  
 `age` int(11) DEFAULT NULL,  
 PRIMARY KEY (`id`)  
) ENGINE=InnoDB DEFAULT CHARSET=utf8;  
 \*/  
  
 //写到MySQL  
 //dataRDD.foreach()  
 dataRDD.foreachPartition(iter=>{  
 //开启连接--有几个分区就开启几次  
 //加载驱动  
 //Class.forName("com.mysql.jdbc.Driver")  
 val conn: Connection = DriverManager.getConnection("jdbc:mysql://localhost:3306/bigdata?characterEncoding=UTF-8","root","root")  
 val sql:String = "INSERT INTO `t\_student` (`id`, `name`, `age`) VALUES (NULL, ?, ?);"  
 val ps: PreparedStatement = conn.prepareStatement(sql)  
 iter.foreach(t=>{//t就表示每一条数据  
 val name: String = t.\_1  
 val age: Int = t.\_2  
 ps.setString(1,name)  
 ps.setInt(2,age)  
 ps.addBatch()  
 //ps.executeUpdate()  
 })  
 ps.executeBatch()  
 //关闭连接  
 if (conn != null) conn.close()  
 if (ps != null) ps.close()  
 })  
  
 //从MySQL读取  
 /\*  
 sc: SparkContext,  
 getConnection: () => Connection, //获取连接对象的函数  
 sql: String,//要执行的sql语句  
 lowerBound: Long,//sql语句中的下界  
 upperBound: Long,//sql语句中的上界  
 numPartitions: Int,//分区数  
 mapRow: (ResultSet) => T = JdbcRDD.resultSetToObjectArray \_) //结果集处理函数  
 \*/  
 val getConnection = () => DriverManager.getConnection("jdbc:mysql://localhost:3306/bigdata?characterEncoding=UTF-8","root","root")  
 val sql:String = "select id,name,age from t\_student where id >= ? and id <= ?"  
 val mapRow: ResultSet => (Int, String, Int) = (r:ResultSet) =>{  
 val id: Int = r.getInt("id")  
 val name: String = r.getString("name")  
 val age: Int = r.getInt("age")  
 (id,name,age)  
 }  
 val studentTupleRDD: JdbcRDD[(Int, String, Int)] = new JdbcRDD[(Int,String,Int)](  
 sc,  
 getConnection,  
 sql,  
 4,  
 6,  
 1,  
 mapRow  
 )  
 studentTupleRDD.foreach(println)  
 }  
}

# Shuffle本质

==shuffle本质是洗牌==

