Spark: sortBy和sortByKey函数详解

在很多应用场景都需要对结果数据进行排序，Spark中有时也不例外。在Spark中存在两种对RDD进行排序的函数，分别是
sortBy和sortByKey函数。sortBy是对标准的RDD进行排序，它是从Spark 0.9.0之后才引入的（可以参见SPARK-1063）。而sortByKey函数是对PairRDD进行排序，也就是有Key和Value的RDD。下面将分别对这两个函数的实现以及使用进行说明。

一、sortBy函数实现以及使用

sortBy函数是在org.apache.spark.rdd.RDD类中实现的，它的实现如下：

```
/**
 * Return this RDD sorted by the given key function.
 */
def sortBy[K](
    f: (T) => K,
    ascending: Boolean = true,
    numPartitions: Int = this.partitions.size)
    (implicit ord: Ordering[K], ctag: ClassTag[K]): RDD[T] =
  this.keyBy[K](f)
      .sortByKey(ascending, numPartitions)
      .values
```

该函数最多可以传三个参数：
第一个参数
是一个函数，该函数的也有一个带T泛型的参数，返回类型和RDD中元素的类型是一致的；
第二个参数
是ascending，从字面的意思大家应该可以猜到，是的，这参数决定排序后RDD中的元素是升序还是降序，默认是true，也就是升序；
第三个参数
是numPartitions，该参数决定排序后的RDD的分区个数，默认排序后的分区个数和排序之前的个数相等，即为this.partitions.size。
从sortBy函数的实现可以看出，第一个参数是必须传入的，而后面的两个参数可以不传入。而且sortBy函数函数的实现依赖于sortByKey函数，关于sortByKey函数后面会进行说明。keyBy函数也是RDD类中进行实现的，它的主要作用就是将将传进来的每个元素作用于f(x)中，并返回tuples类型的元素，也就变成了Key-Value类型的RDD了，它的实现如下：

```
/**
```

```
* Creates tuples of the elements in this RDD by applying `f`.
*/
def keyBy[K](f: T => K): RDD[(K, T)] = {
   map(x => (f(x), x))
}
```

那么，如何使用sortBy函数呢？

```
/**
 * User: 过往记忆
 * Date: 14-12-26
 * Time: 上午10:16
 * bolg:
 * 本文地址：/archives/1240
 * 过往记忆博客，专注于hadoop、hive、spark、shark、flume的技术博客，大量的干货
 * 过往记忆博客微信公共帐号：iteblog_hadoop
 */
scala> val data = List(3,1,90,3,5,12)
data: List[Int] = List(3, 1, 90, 3, 5, 12)

scala> val rdd = sc.parallelize(data)
rdd: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[0] at parallelize at <console>:14

scala> rdd.collect
res0: Array[Int] = Array(3, 1, 90, 3, 5, 12)

scala> rdd.sortBy(x => x).collect
res1: Array[Int] = Array(1, 3, 3, 5, 12, 90)

scala> rdd.sortBy(x => x, false).collect
res3: Array[Int] = Array(90, 12, 5, 3, 3, 1)

scala> val result = rdd.sortBy(x => x, false)
result: org.apache.spark.rdd.RDD[Int] = MappedRDD[23] at sortBy at <console>:16

scala> result.partitions.size
res9: Int = 2

scala> val result = rdd.sortBy(x => x, false, 1)
result: org.apache.spark.rdd.RDD[Int] = MappedRDD[26] at sortBy at <console>:16

scala> result.partitions.size
res10: Int = 1
```

上面的实例对rdd中的元素进行升序排序。并对排序后的RDD的分区个数进行了修改，上面的result就是排序后的RDD，默认的分区个数是2，而我们对它进行了修改，所以最后变成了1。

二、sortByKey函数实现以及使用

sortByKey函数作用于Key-Value形式的RDD，并对Key进行排序。它是在org.apache.spark.rdd.OrderedRDDFunctions中实现的，实现如下

```
 def sortByKey(ascending: Boolean = true, numPartitions: Int = self.partitions.size)
    : RDD[(K, V)] =
 {
  val part = new RangePartitioner(numPartitions, self, ascending)
  new ShuffledRDD[K, V, V](self, part)
    .setKeyOrdering(if (ascending) ordering else ordering.reverse)
 }
```

从函数的实现可以看出，它主要接受两个函数，含义和sortBy一样，这里就不进行解释了。该函数返回的RDD一定是ShuffledRDD类型的，因为对源RDD进行排序，必须进行Shuffle操作，而Shuffle操作的结果RDD就是ShuffledRDD。其实这个函数的实现很优雅，里面用到了RangePartitioner，它可以使得相应的范围Key数据分到同一个partition中，然后内部用到了mapPartitions对每个partition中的数据进行排序，而每个partition中数据的排序用到了标准的sort机制，避免了大量数据的shuffle。下面对sortByKey的使用进行说明：

```
/**
 * User: 过往记忆
 * Date: 14-12-26
 * Time: 上午10:16
 * bolg:
 * 本文地址：/archives/1240
 * 过往记忆博客，专注于hadoop、hive、spark、shark、flume的技术博客，大量的干货
 * 过往记忆博客微信公共帐号：iteblog_hadoop
 */
scala> val a = sc.parallelize(List("wyp", "iteblog", "com", "397090770", "test"), 2)
a: org.apache.spark.rdd.RDD[String] =
ParallelCollectionRDD[30] at parallelize at <console>:12

scala> val b = sc. parallelize (1 to a.count.toInt , 2)
```

```
b: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[31] at parallelize at <console>:14

scala> val c = a.zip(b)
c: org.apache.spark.rdd.RDD[(String, Int)] = ZippedPartitionsRDD2[32] at zip at <console>:16

scala> c.sortByKey().collect
res11: Array[(String, Int)] = Array((397090770,4), (com,3), (iteblog,2), (test,5), (wyp,1))
```

上面对Key进行了排序。细心的读者可能会问，sortBy函数中的第一个参数可以对排序方式进行重写。为什么sortByKey没有呢？难道只能用默认的排序规则。不是，是有的。其实在OrderedRDDFunctions类中有个变量ordering它是隐形的：private val ordering = implicitly[Ordering[K]]。他就是默认的排序规则，我们可以对它进行重写，如下：

```
scala> val b = sc.parallelize(List(3,1,9,12,4))
b: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[38] at parallelize at <console>:12

scala> val c = b.zip(a)
c: org.apache.spark.rdd.RDD[(Int, String)] = ZippedPartitionsRDD2[39] at zip at <console>:16

scala> c.sortByKey().collect
res15: Array[(Int, String)] = Array((1,iteblog), (3,wyp), (4,test), (9,com), (12,397090770))

scala> implicit val sortIntegersByString = new Ordering[Int]{
     | override def compare(a: Int, b: Int) =
     | a.toString.compare(b.toString)}
sortIntegersByString: Ordering[Int] = $iwC$$iwC$$iwC$$iwC$$iwC$$anon$1@5d533f7a

scala>  c.sortByKey().collect
res17: Array[(Int, String)] = Array((1,iteblog), (12,397090770), (3,wyp), (4,test), (9,com))
```

例子中的sortIntegersByString就是修改了默认的排序规则。这样将默认按照Int大小排序改成了对字符串的排序，所以12会排序在3之前。