结合Spark源码分析, combineByKey, aggregateByKey, foldByKey, reduceByKey

置顶 2016年06月16日 09:42:03 阅读数：5243

转载请标明出处：小帆的帆的专栏

combineByKey

```
def combineByKey[C](
  createCombiner: V => C,
  mergeValue: (C, V) => C,
  mergeCombiners: (C, C) => C): RDD[(K, C)] = self.withScope {
combineByKeyWithClassTag(createCombiner, mergeValue, mergeCombiners)(null)
}
```

- createCombiner：当combineByKey第一次遇到值为k的Key时，调用createCombiner函数，将v转换为c
- mergeValue：combineByKey不是第一次遇到值为k的Key时，调用mergeValue函数，将v累加到c中
- mergeCombiners：将两个c，合并成一个

```
// 实例
SparkConf conf = new SparkConf().setAppName("test").setMaster("local");
JavaSparkContext sc = new JavaSparkContext(conf);

List<Tuple2<Integer, String>> list = new ArrayList<>();

list.add(new Tuple2<>(1, "www"));
list.add(new Tuple2<>(1, "iteblog"));
list.add(new Tuple2<>(1, "com"));
list.add(new Tuple2<>(2, "bbs"));
list.add(new Tuple2<>(2, "iteblog"));
list.add(new Tuple2<>(2, "com"));
list.add(new Tuple2<>(3, "good"));

JavaPairRDD<Integer, String> data = sc.parallelizePairs(list);

JavaPairRDD<Integer, List<String>> result = data.combineByKey(v -> {
    ArrayList<String> strings = new ArrayList<>();
    strings.add(v);
    return strings;
```

```
}, (c, v) -> {
    c.add(v);
    return c;
}, (c1, c2) -> {
    c1.addAll(c2);
    return c1;
});

result.collect().forEach(System.out::println);
```

aggregateByKey

```
def aggregateByKey[U: ClassTag](zeroValue: U, partitioner: Partitioner)(seqOp: (U,
  combOp: (U, U) => U): RDD[(K, U)] = self.withScope {

// 中间代码省略，主要看最后一个，调用combineByKey

val cleanedSeqOp = self.context.clean(seqOp)
// seqOp，同时是，createCombiner，mergeValue。而combOp是mergeCombiners
combineByKeyWithClassTag[U]((v: V) => cleanedSeqOp(createZero(), v),
  cleanedSeqOp, combOp, partitioner)
}
```

- createCombiner：cleanedSeqOp(createZero(), v)是createCombiner, 也就是传入的seqOp函数, 只不过其中一个值是传入的zeroValue
- mergeValue：seqOp函数同样是mergeValue, createCombiner和mergeValue函数相同是aggregateByKey函数的关键
- mergeCombiners：combOp函数

因此, 当createCombiner和mergeValue函数的操作相同, aggregateByKey更为合适

```
// 例子与combineByKey相同，只是改用aggregateByKey实现
SparkConf conf = new SparkConf().setAppName("test").setMaster("local");
JavaSparkContext sc = new JavaSparkContext(conf);

List<Tuple2<Integer, String>> list = new ArrayList<>();

list.add(new Tuple2<>(1, "www"));
list.add(new Tuple2<>(1, "iteblog"));
list.add(new Tuple2<>(1, "com"));
list.add(new Tuple2<>(2, "bbs"));
list.add(new Tuple2<>(2, "iteblog"));
list.add(new Tuple2<>(2, "com"));
```

```
list.add(new Tuple2<>(3, "good"));

JavaPairRDD<Integer, String> data = sc.parallelizePairs(list);

JavaPairRDD<Integer, List<String>> result = data.aggregateByKey(new ArrayList<Stri
    c.add(v);
    return c;
}, (Function2<List<String>, List<String>, List<String>>) (c1, c2) -> {
    c1.addAll(c2);
    return c1;
});

result.collect().forEach(System.out::println);
```

foldByKey

```
def foldByKey(
  zeroValue: V,
  partitioner: Partitioner)(func: (V, V) => V): RDD[(K, V)] = self.withScope {

// 中间代码省略，主要看最后一个，调用combineByKey

val cleanedFunc = self.context.clean(func)
// 传入的func函数，同时是，createCombiner，mergeValue，mergeCombiners
// createCombiner函数传入了零值，首次遇到一个key时，根据零值进行初始化
combineByKeyWithClassTag[V]((v: V) => cleanedFunc(createZero(), v),
  cleanedFunc, cleanedFunc, partitioner)
}
```

- createCombiner：cleanedFunc(createZero(), v)是createCombiner, 也就是func函数, 只不过其中一个值是传入的zeroValue
- mergeValue, mergeCombiners：func函数也是mergeValue和 mergeCombiners

当createCombiner，mergeValue和mergeCombiners函数操作都相同, 唯独需要一个zeroValue时, 适用

```
// 根据Key把Value相加，但是不从0开始，设置初始值为100
val conf = new SparkConf().setAppName("test").setMaster("local")
val sc = new SparkContext(conf)
val sqlContext = new SQLContext(sc)

var rdd = sc.makeRDD(Array(("A",0),("A",2),("B",1),("B",2),("C",1)))

```

```
8
rdd.foldByKey(100)(_+_).collect.foreach(println)
```

reduceByKey

```
1 def reduceByKey(partitioner: Partitioner, func: (V, V) => V): RDD[(K, V)] = self.w
2   combineByKeyWithClassTag[V]((v: V) => v, func, func, partitioner)
3 }
```

- createCombiner：与foldByKey相比, reduceByKey没有初始值, createCombiner也没有调用func函数, 而是直接将参数作为返回值返回了,
- mergeValue, mergeCombiners：func函数同时是mergeValue和 mergeCombiners

当不需要createCombiner，且mergeValue和mergeCombiners函数操作都相同时, 适用

```
1 val conf = new SparkConf().setAppName("test").setMaster("local")
2 val sc = new SparkContext(conf)
3 val sqlContext = new SQLContext(sc)
4
5 var rdd = sc.makeRDD(Array(("A", 0), ("A", 2), ("B", 1), ("B", 2), ("C", 1)))
6
7 rdd.reduceByKey(_ + _).collect.foreach(println)
```

总结

这几个算子, 核心就要弄明白combineByKey, 其他三个都是调用它. 上文主要也是从combingByKey传入的三个函数的角度在分析.

而在实际运用中, 最先要考虑的应该是类型. combingByKey和aggregateByKey输入跟输出的类型可以不一致, 而foldByKey和reduceByKey不行. 类型确定后再根据自己的业务选择最简洁的算子.