链滴

Spark 的见解 & 优化 (一)

作者：superYangYang
原文链接：https://ld246.com/article/1548344785815
来源网站：链滴

spark是什么

 spark是一个分布式的内存型的流式计算框架，支持java，python,scala,数据源可以是流式流，可以是文本，数据库，有schema的json或者parquet等

概念&见解(附java示例代码)

rdd

Spark revolves around the concept of a _resilient distributed dataset_ (RDD), which is a fault-tolerant collection of elements that can be
operated on in parallel. There are two ways to create RDDs: _parallelizing_ an existing collection in your driver program, or referencing a
dataset in an external storage system, such as a shared filesystem, HDFS, HBase, or any data source offering a Hadoop InputFormat.

 rdd是分布式的的可容错的数据集合,2种创建方式上面已由上面给出，在此就不做赘述。

Transformations算子(带示例代码)

 Transformation属于延迟计算，当一个RDD转换成另一个RDD时并没有立即进行转换，仅是记住了数据集的逻辑操作

 1）　　**map**(通过函数把rdd变换成为一个新的数据集)

```
SparkConf conf = new SparkConf();
// local[1]表示使用1个线程
conf.setMaster("local[1]");
conf.setAppName("test");
List list = new ArrayList<>();
for(int i=1;i<=10;i++){
   list.add(i);
}
JavaSparkContext jc = new JavaSparkContext(conf);
JavaRDD<Integer> parallelize = jc.parallelize(list);
// 每个值前面加上字符串str:
JavaRDD<String> map = parallelize.map(x -> "str:"+x);
map.foreach(x-> System.out.println(x));

结果：
str:1
str:2
str:3
str:4
str:5
str:6
str:7
str:8
str:9
str:10
```

 2)　　**filter**(返回函数结果为true的数据集)

```
SparkConf conf = new SparkConf();
conf.setMaster("local[1]");
conf.setAppName("test");
List list = new ArrayList<>();
for(int i=1;i<=10;i++){
   list.add(i);
}
JavaSparkContext jc = new JavaSparkContext(conf);
JavaRDD parallelize = jc.parallelize(list);
// 过滤偶数
JavaRDD map = parallelize.filter(x->x%2==0?false:true);
map.foreach(x-> System.out.println(x));
```

结果:
1
3
5
7
9

 3)　　**flatMap**(把一个结果集的每个元素变成为多个元素)

```
SparkConf conf = new SparkConf();
conf.setMaster("local[1]");
conf.setAppName("test");
JavaSparkContext jc = new JavaSparkContext(conf);
// 内容为:1,2,3,4,5,6,7,8,9
// 加载该文件并按逗号分隔
JavaRDD stringJavaRDD = jc.textFile("/Users/yangjunwei/data/spark.txt")
      .flatMap(x -> Arrays.asList(x.split(",")).iterator());
stringJavaRDD.foreach(x-> System.out.println(x));
```

结果:
1
2
3
4
5
6
7
8
9

 4)　　**mapPartitions**(以分区为单位，对每个partition的rdd做map操作)

```
SparkConf conf = new SparkConf();
conf.setMaster("local[1]");
conf.setAppName("test");
JavaSparkContext jc = new JavaSparkContext(conf);
// 内容为:1,2,3,4,5,6,7,8,9
// 加载该文件并按逗号分隔
// 每行开头加str:
JavaRDD stringJavaRDD = jc.textFile("/Users/yangjunwei/data/spark.txt")
```

```
        .flatMap(x -> Arrays.asList(x.split(",")).iterator())
        .mapPartitions(x->{
          List list = new ArrayList<>();
          x.forEachRemaining(x1->{
            list.add("str:"+x1);
          });
          return list.iterator();
 });
 stringJavaRDD.foreach(x-> System.out.println(x));
 结果:
 str:1
 str:2
 str:3
 str:4
 str:5
 str:6
 str:7
 str:8
 str:9
```

 5) **union**(返回2个数据集的并集)

```
SparkConf conf = new SparkConf();
conf.setMaster("local[1]");
conf.setAppName("test");
JavaSparkContext jc = new JavaSparkContext(conf);
List list1 = new ArrayList<>();
for(int i=1;i<=5;i++){
  list1.add(i);
}
List list2 = new ArrayList<>();
for(int i=5;i<=10;i++){
  list2.add(i);
}
JavaRDD rdd1 = jc.parallelize(list1);
JavaRDD rdd2 = jc.parallelize(list2);
JavaRDD union = rdd1.union(rdd2);
System.out.println("rdd1:");
rdd1.foreach(x-> System.out.println(x));
System.out.println("rdd2:");
rdd2.foreach(x-> System.out.println(x));
System.out.println("union:");
union.foreach(x-> System.out.println(x));

结果:
rdd1:
1
2
3
4
5
rdd2:
5
6
```

7
8
9
10
union:
1
2
3
4
5
5
6
7
8
9
10

 6) **distinct**(数据集去重)

 7) **sortBy**(对数据集处理后的值做二次排序)

接5)的代码

```
// 按原值分1个partition进行升序
union.distinct().sortBy(x->x,true,1).foreach(x-> System.out.println(x));
```

结果:
1
2
3
4
5
6
7
8
9
10

 9) **mapToPair**(将数据集转化为<K,V>数据集)

 10) **sortByKey**(对<K,V>数据集(pairs)基于key进行排序)

```
SparkConf conf = new SparkConf();
conf.setMaster("local[1]");
conf.setAppName("test");
JavaSparkContext jc = new JavaSparkContext(conf);
List list1 = new ArrayList<>();
for(int i=5;i>=1;i--){
   list1.add(i);
}
JavaPairRDD, String> pairRDD = jc.parallelize(list1).mapToPair(x -> new Tuple2<>(x, ""));
System.out.println("排序前：");
pairRDD.foreach(x-> System.out.println(x));
JavaPairRDD, String> sortPair = pairRDD.sortByKey();
System.out.println("排序后：");
sortPair.foreach(x-> System.out.println(x));
```

```
结果:
排序前:
(5,)
(4,)
(3,)
(2,)
(1,)
排序后:
(1,)
(2,)
(3,)
(4,)
(5,)
```

 11) **groupByKey**(对<K,V>数据集(pairs)基于key进行分组)

```
SparkConf conf = new SparkConf();
conf.setMaster("local[1]");
conf.setAppName("test");
JavaSparkContext jc = new JavaSparkContext(conf);
List<Map<Integer,Integer>> list = new ArrayList<>();
Map<Integer,Integer> item = new HashMap<>();
item.put(1,1);
list.add(item);
item = new HashMap<>();
item.put(1,2);
list.add(item);
item = new HashMap<>();
item.put(1,3);
list.add(item);
item = new HashMap<>();
item.put(2,1);
list.add(item);
item = new HashMap<>();
item.put(2,2);
list.add(item);
JavaPairRDD<Integer,Integer> javaPairRDD = jc.parallelize(list).flatMap(x -> x.entrySet().itera
or()).mapToPair(x -> new Tuple2<>(x.getKey(),x.getValue()));
System.out.println("分组前:");
javaPairRDD.foreach(x-> System.out.println(x));
System.out.println("分组后:");
javaPairRDD.groupByKey().foreach(x-> System.out.println(x));
```

```
结果:
分组前:
(1,1)
(1,2)
(1,3)
(2,1)
(2,2)
分组后:
(1,[1, 2, 3])
(2,[1, 2])
```

 12) **reduceByKey**(对<K,V>数据集(pairs)基于key进行reduce操作)

```
SparkConf conf = new SparkConf();
conf.setMaster("local[1]");
conf.setAppName("test");
JavaSparkContext jc = new JavaSparkContext(conf);
List list1 = new ArrayList<>();
for(int i=1;i<=5;i++){
   list1.add(i);
}
List list2 = new ArrayList<>();
for(int i=3;i<=5;i++){
   list2.add(i);
}
JavaPairRDD, String> pairRdd1 = jc.parallelize(list1).mapToPair(x -> new Tuple2<>(x, ""));
JavaPairRDD, String> pairRdd2 = jc.parallelize(list2).mapToPair(x -> new Tuple2<>(x, ""));
JavaPairRDD, String> union = pairRdd1.union(pairRdd2);
System.out.println("key去重前:");
union.sortByKey().map(x->x._1).foreach(x-> System.out.println(x));
System.out.println("key去重后:");
union.reduceByKey((var1, var2) -> var1).sortByKey().map(x->x._1).foreach(x-> System.out.prin
ln(x));
```

action算子(带示例代码)

 下篇