Spark在Yarn上运行Wordcount程序

我们在接触Hadoop的时候，第一个列子一般是运行Wordcount程序，在Spark我们可以用Java代码写一个Wordcount程序并部署在Yarn上运行。我们知道，在Spark源码中就存在一个用Java编写好的JavaWordCount程序，源码如下：

```
package org.apache.spark.examples;

import org.apache.spark.api.java.JavaPairRDD;
import org.apache.spark.api.java.JavaRDD;
import org.apache.spark.api.java.JavaSparkContext;
import org.apache.spark.api.java.function.FlatMapFunction;
import org.apache.spark.api.java.function.Function2;
import org.apache.spark.api.java.function.PairFunction;
import scala.Tuple2;

import java.util.Arrays;
import java.util.List;
import java.util.regex.Pattern;

public final class JavaWordCount {
  private static final Pattern SPACE = Pattern.compile(" ");

  public static void main(String[] args) throws Exception {
    if (args.length < 2) {
      System.err.println("Usage: JavaWordCount <master> <file>");
      System.exit(1);
    }

    JavaSparkContext ctx = new JavaSparkContext(args[0],
        "JavaWordCount",
        System.getenv("SPARK_HOME"),
        JavaSparkContext.jarOfClass(JavaWordCount.class));
    JavaRDD<String> lines = ctx.textFile(args[1], 1);

    JavaRDD<String> words = lines.flatMap(
        new FlatMapFunction<String, String>() {
          @Override
          public Iterable<String> call(String s) {
            return Arrays.asList(SPACE.split(s));
          }
        });
```

```
        JavaPairRDD<String, Integer> ones = words.map(
                new PairFunction<String, String, Integer>() {
                    @Override
                    public Tuple2<String, Integer> call(String s) {
                        return new Tuple2<String, Integer>(s, 1);
                    }
                });

        JavaPairRDD<String, Integer> counts = ones.reduceByKey(
                new Function2<Integer, Integer, Integer>() {
                    @Override
                    public Integer call(Integer i1, Integer i2) {
                        return i1 + i2;
                    }
                });

        List<Tuple2<String, Integer>> output = counts.collect();
        for (Tuple2<?, ?> tuple : output) {
            System.out.println(tuple._1() + ": " + tuple._2());
        }
        System.exit(0);
    }
}
```

这里有必要介绍一下这里用到的几个函数。首先是map函数，它根据现有的数据集返回一个新的分布式数据集，由每个原元素经过func函数转换后组成，这个过程一般叫做转换（transformation）；flatMap函数类似于map函数，但是每一个输入元素，会被映射为0到多个输出元素，因此，func函数的返回值是一个Seq，而不是单一元素，可以从上面的代码中看出；reduceByKey函数在一个（K，V)对的数据集上使用，返回一个（K，V）对的数据集，key相同的值，都被使用指定的reduce函数聚合到一起。

运行上面的代码之前你得先编译好（话说我好几次用Maven编译老是不成功啊，不过大家可以用./sbt/sbt assembly进行编译）。编译好之后可以用下面的命令进行运行：

```
./bin/spark-class                       ₩
org.apache.spark.deploy.yarn.Client          ₩
--jar ./jars/spark-examples-assembly-0.9.1.jar    ₩
--class org.apache.spark.examples.JavaWordCount   ₩
--args yarn-standalone                  ₩
--args /home/wyp/cite75_99.txt               ₩
```

org.apache.spark.examples.JavaWordCount类接收两个参数，第一个参数指定你程序运行的master；第二个参数指定你需要计算Wordcount文件的绝对路径，这个文件需要在HDFS上。程序运行的过程中我们可以在Hadoop的WEB
UI上进行查看，程序运行完的时候，可以在logs里面看到运行的结果，类似下面：

submitting: 1
find: 1
versions: 4
Regression: 1
via: 2
tests: 2
open: 2
./bin/spark-shell: 1
When: 1
All: 1
download: 1
requires: 2
SPARK_YARN=true: 3
Testing: 1
take: 1
project: 4
no: 1
systems.: 1
file: 1
Or,: 1
About: 1
project's: 3
programs: 2
given.: 1
obtained: 1
sbt/sbt: 5
artifact: 1
SBT: 1
local[2]: 1
not: 1
runs.: 1
you: 5
building: 1

当然，程序默认的输出直接输到logs里面去了，我们可以将结果输出到文本里面，修改如下：

counts.saveAsTextFile("/home/wyp/result");

或者：

```
counts.saveAsHadoopFile("/home/wyp/result",
                        Text.class,
                    IntWritable.class,
                TextOutputFormat.class);
```

上面的两行代码都可以将计算的结果存储到HDFS上的/home/wyp/result文件夹里面，但是两者输出来的结果内容格式是有区别的，第一种输出内容格式如下：

```
(5,5)
(1,1)
(15,1)
(7,6)
(11,5)
(14,2)
(3,3)
(8,6)
(6,6)
(12,4)
(4,4)
(10,6)
(13,3)
(2,2)
(9,6)
```

格式是(key, value)的；第二种输出内容格式如下：

```
5 5
1 1
15 1
7 6
11 5
14 2
3 3
8 6
6 6
12 4
4 4
10 6
```

13 3
2 2
9 6

格式是key value。我们可以根据自己的需要定义一个自己的输出格式，而且我们在输出的时候如果文件比较大，还可以指定输出文件的压缩方式。