第八节、MapReduce

一、概念阐释

- MapReduce是一种编程模型，用于大规模数据集的并行运算。能自动完成计算任务的并行化处理，自动划分计算数据和计算任务，在集群节点上自动分配和执行任务以及收集计算结果。
- 在函数式语言里，map表示对一个列表中的每个元素做计算，reduce表示对一个列表中的每个元素做迭代计算。
- 在MapReduce里，Map处理的是原始数据，可能是杂乱无章的，Reduce处理的是分组后的数据。

二、流程概述

1. MapReduce结构

- MRAppMaster：负责整个程序的过程调度及状态协调
- MapTask：负责Map阶段的整个数据处理流程
- ReduceTask：负责Reduce阶段的整个数据处理流程

2. 运行流程

- 最先启动的是MRAppMaster，MRAppMaster启动后根据本次job的描述信息，计算出需要的MapTask实例数量，然后向集群申请机器启动相应数量的MapTask进程
- MapTask进程启动之后，根据给定的数据切片范围进行数据处理
- MRAppMaster监控到所有MapTask进程任务完成之后，会根据客户指定的参数启动相应数量的ReduceTask进程，并告知ReduceTask进程要处理的数据范围（数据分区）

- ReduceTask进程启动之后，根据MRAppMaster告知的待处理数据所在位置，从若干台MapTask运行所在机器上获取到若干个MapTask输出结果文件，并在本地进行重新归并排序，然后按照相同key的KV为一个组，调用用户定义的reduce()方法进行逻辑运算，并收集运算输出的结果KV，然后调用用户指定的outputformat将结果数据输出到外部存储

三、程序案例

1. 环境准备

- 将hadoop.dll放在HADOOP_HOME/bin以及System32/SysWOW64目录下
- 找到Hadoop-common包源码包
- 复制org.apache.hadoop.io.nativeio路径下的NativeIO.java
- 在工程下新建相应的包
- 修改609行，将返回值改为true
- 将Hadoop配置文件中的log4j.properties复制到工程src目录下即可

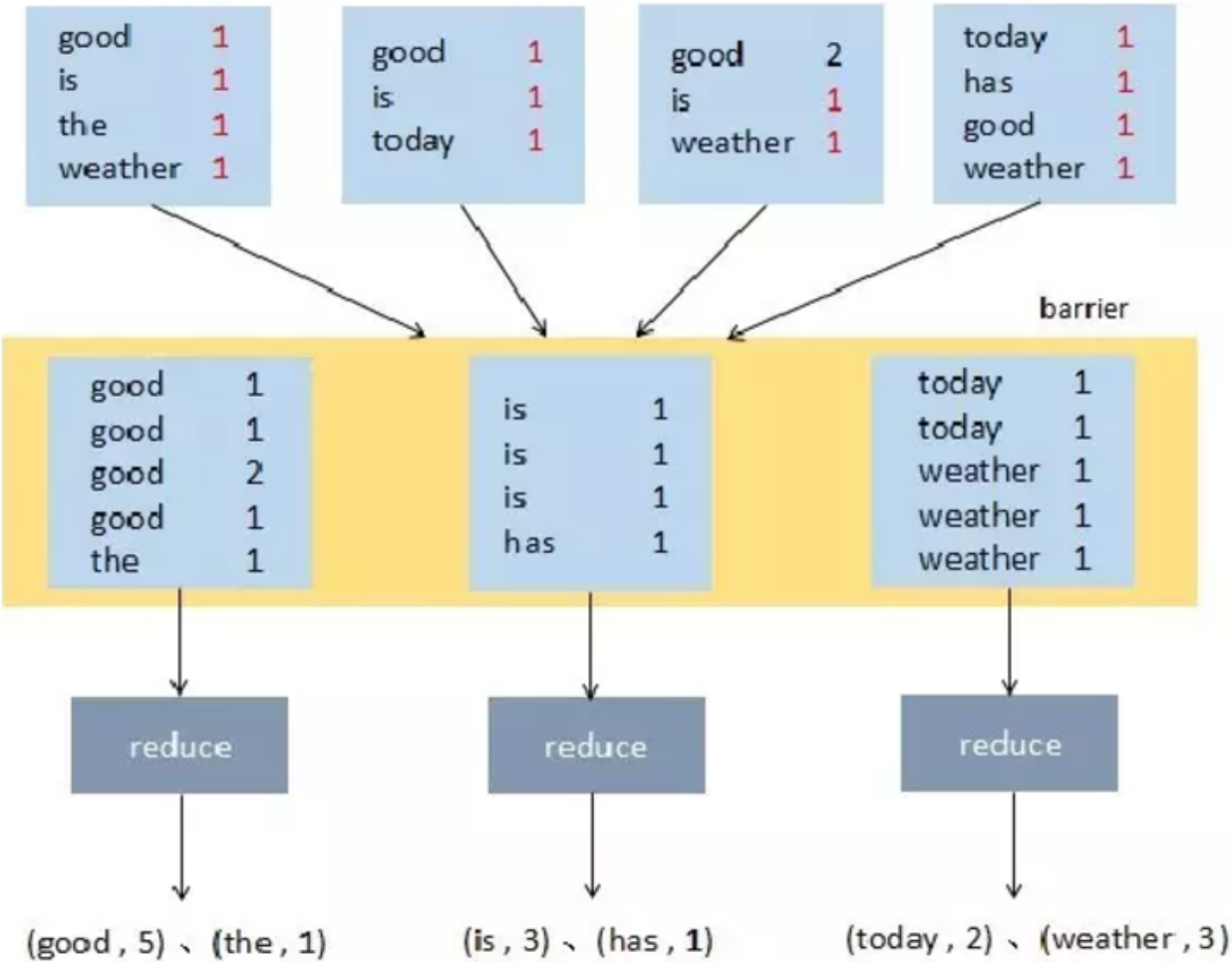
2. WordCount

- 待处理数据

```
the weather is good
today is good
good weather is good
today has good weather
```

- 处理思路

> 将句子进行拆分，变为单个单词进行计数，然后将结果进行合并



- 参数解析
  Mapper<KEYIN,VALUEIN,KEYOUT,VALUEOUT>
  KEYIN：输入数据的KEY，一般为起始偏移量
  VALUEIN：输入数据的VALUE，一般为一行数据
  KEYOUT：输出数据的KEY的数据类型
  VALUEOUT：输出数据的VALUE的数据类型
  Reducer<KEYIN,VALUEIN,KEYOUT,VALUEOUT>
  Reducer的输入为Mapper的输出

3. 程序代码

- Mapper

```java
public class WordCountMapper extends Mapper<LongWritable, Text, Text, IntWritable> {

    @Override
    protected void map(LongWritable key, Text value, Mapper<LongWritable, Text, Text, IntWritable>.Context context)
            throws IOException, InterruptedException {
        //获取读取到的每一行数据
        String line = value.toString();
        //通过空格进行分割
        String[] words = line.split(" ");
        //将初步处理的结果进行输出
        for (String word : words) {
            context.write(new Text(word), new IntWritable(1));
        }
    }

}
```

- Reducer

```java
public class WordCountReducer extends Reducer<Text, IntWritable, Text, IntWritable> {

    @Override
    protected void reduce(Text key, Iterable<IntWritable> values,
            Context context) throws IOException, InterruptedException {
        Integer count = 0;
        for (IntWritable value : values) {
            //将每一个分组中的数据进行累加计数
            count += value.get();
        }
        //将最终的结果进行输出
        context.write(key, new IntWritable(count));
    }

}
```

- Master

```java
public class WordMaster {

    public static void main(String[] args) throws Exception {
        //初始化配置
        Configuration conf = new Configuration();
        //初始化job参数，指定job名称
        Job job = Job.getInstance(conf, "wordCount");
        //设置运行job的类
        job.setJarByClass(WordMaster.class);
        //设置Mapper类
        job.setMapperClass(WordCountMapper.class);
        //设置Reducer类
        job.setReducerClass(WordCountReducer.class);
        //设置Map的输出数据类型
        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(IntWritable.class);
        //设置Reducer的输出数据类型
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        //设置输入的路径
        FileInputFormat.setInputPaths(job, new Path("hdfs://192.168.116.111:8020/input"));
        //设置输出的路径
        FileOutputFormat.setOutputPath(job, new Path("hdfs://192.168.116.111:8020/output/wordCount"));
        //提交job
        boolean result = job.waitForCompletion(true);
        //执行成功后进行后续操作
        if (result) {
            System.out.println("Congratulations!");
        }
    }

}
```