Day06_MapReduce进阶和倒排索引

大数据-张军锋 Day06 倒排索引

Day06_MapReduce进阶和倒排索引
Mr文件读取的几种方式 的区别
书写MapReducer为什么要使用静态内部类？
书写map时，为什么将变量定义在成员变量位置？
设置分隔符的两种方式
程序优化原理分析—combinner
如何在map端进行聚合

MR倒排索引
TopN问题

Mr文件读取的几种方式 的区别

TextInputFormat(LongWritable,Text)：文件偏移量 ：整行数据(默认状态)
KeyValueTextInputFormat(Text,Text)：第一个” “前的数据 ： 后面的整行数据
SequenceFileInputFormat：因为这是二进制文件，所以Key-Value都是由用户指定

KeyValueTextInputFormat默认情况下是以“\t”作为分隔符号，将每行记录按照“\t”分隔成key，value两个部分

书写MapReducer为什么要使用静态内部类？

添加static的话属于静态内部类，它和普通的类型没有什么区别，在使用上直接new类型来实例化对象

```
IpLoginNewTweetMap map = new IpLoginNewTweetMap();
```

如果不加static的话,内部类的实例化之前要先实例化外部类

```
IpLoginNewTweet ilnt = new IpLoginNewTweet();
IpLoginNewTweetMap map = new ilnt.IpLoginNewTweetMap();
```

书写map时，为什么将变量定义在成员变量位置？

Mr运行过程中，每个map节点，只实例化一个map对象,但是每行记录都会调用一次map方法，定义在成员变量位置,减少对象在堆中创建资源，减少了垃圾回收机制回收资源,提高了代码的质量

设置分隔符的两种方式

程序优化原理分析—combinner

Shuffle过程是从map读取kv结束后到传递给reducerwork调用reducer方法的过程
Shuffle过程是mr中最耗时间的过程，减少shuffle的数据量，能提高mr的运行速度
如果在map上将数据进行聚合，聚合后的结果在发送到reduce上再次聚合，这样

1. 减少shuffle的数据量
2. 把reducer上的计算的压力在每一个reducer上进行分担(分担了reduce的压力)

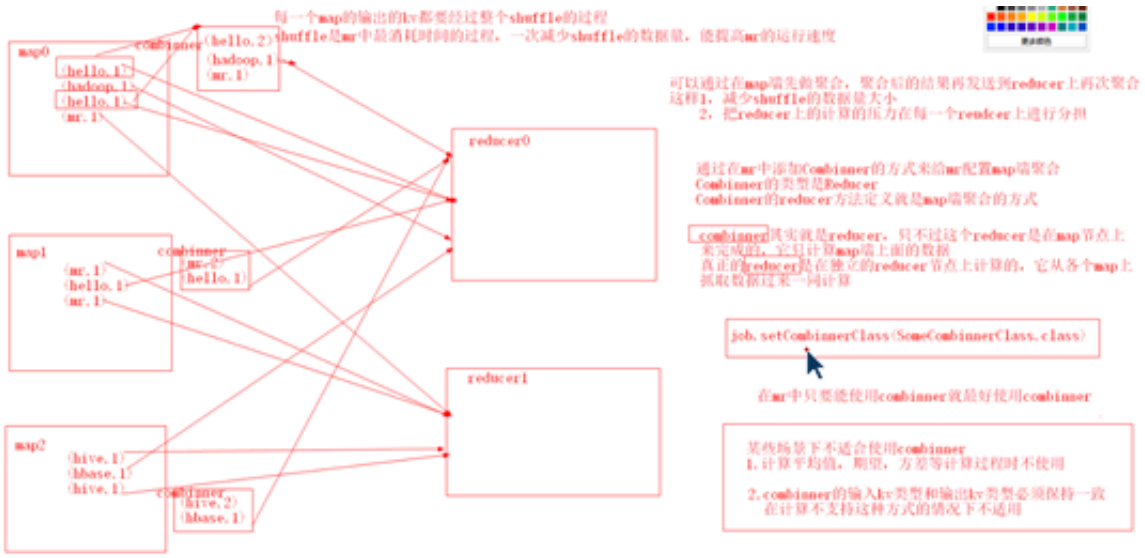
如何在map端进行聚合

通过在mr中添加Combinner的方式给mr配置map端聚合
Combinner的类型就是Reducer
Combinner的reducer方法定义就是map端聚合的方式
Combinner其实就是reducer，只不过这个reducer是在map节点上完成的，它只是计算map端上的数据
真正的reducer在独立的reducer节点上计算的，它从各个map上找去数据过来一同计算

```
Job.setCombinnerClass(SomeCombinnerClass.class);
```

在mr中只要能使用Combinner就最好使用Combinner,因为它减少了Map端向reducer端传送的数据量
不能使用combinner的场景

1. 计算平均值，期望，方差等计算过程不能使用
2. Combinner的输入kv类型必须和输出类型保持一致，否则不适用



> 注意 ：如果combinner与reducer一样并且满足combinner的使用场景，可以将combinner和reducer进行合并

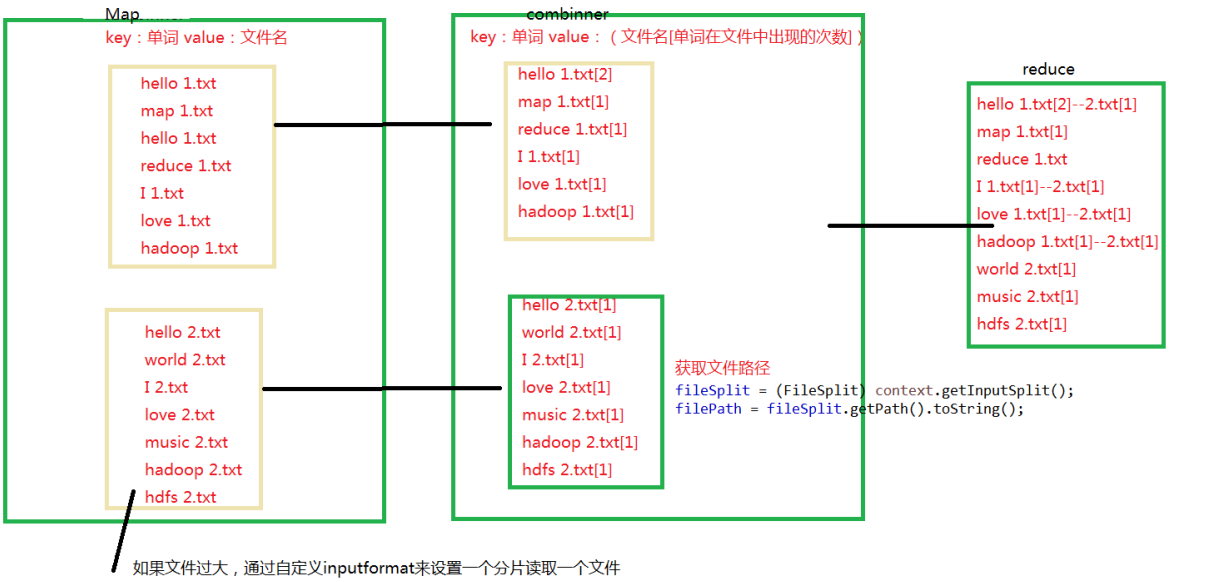
MR倒排索引

> 如果一个文件被两个或两个以上的mapwork进行分隔，就会出现同一个文件被分配到不同的mapwork上，这样就会出现错误，我们可以通过两种方式进行修改，避免这种问题的发生

1. 在reduce上将解析的文件名和次数相同的进行累加，所有文件名和次数解析之后再输出

2.在map上自定义InputFormat来将同一个文件分到一个split中，这样就解决了上述问题

自定义inputFormat上重写isSplitable方法即可



```
import java.io.IOException;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.JobContext;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.FileSplit;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class ReversedIndex {
    /**
    * 项目名称：mapreeduce
    * 类名称：ReversedIndexInputFormat
    * 类描述：为了解决一个文件可能有两个分片，自定义一个inputformat，来设置一
个文件只能有一个分片，前提是文件不宜过大，否则另想办法
    * @version
    */
    public static class ReversedIndexInputFormat extends TextInputF
ormat{
        @Override
        protected boolean isSplitable(JobContext context, Path fil
e) {
            return false;
        }
    }

    /**
    * 项目名称：mapreeduce
    * 类名称：ReversedIndexMap
    * 类描述：加载源文件，解析单词，把单词作为key，文件名作为value输出 ,输
出：key(单词),value(文件名)
    * @version
    */
    public static class ReversedIndexMap extends Mapper<LongWritabl
e, Text, Text, Text> {

        private String[] infos;
        private String filePath;
        private Text outputKey = new Text();
        private Text outputValue = new Text();
        private FileSplit fileSplit;
```

```
        // 执行map任务时执行一次，和测试类中的before类似
        @Override
        protected void setup(Mapper<LongWritable, Text, Text, Tex
t>.Context context)
                throws IOException, InterruptedException {
            // 设置文件路径，文件是指该map正在执行处理的数据文件
            fileSplit = (FileSplit) context.getInputSplit();
            filePath = fileSplit.getPath().toString();
        }

        // map方法每行调用一次
        @Override
        protected void map(LongWritable key, Text value, Mapper<Lon
gWritable, Text, Text, Text>.Context context)
                throws IOException, InterruptedException {
            infos = value.toString().split("[\\s\"\\
(\\)\\:\\;\\.\\,\\*\\/\\#\\!\\[\\]\\{\\}\\-\\<\\>]");
            if (infos != null || infos.length > 0) {
                for (String word : infos) {
                    outputKey.set(word.toLowerCase());
                    outputValue.set(filePath);
                    context.write(outputKey, outputValue);
                }
            }
        }
    }

    /**
    * 项目名称：mapreeduce
    * 类名称：ReversedIndexCombinner
    * 类描述：接受map的输出，然后统计出key(单词)出现的次数，输出key(单词)，va
lue(文件名称[单词在文件中出现的次数])
    * @version
    */
    public static class ReversedIndexCombinner extends Reducer<Tex
t, Text, Text, Text> {
        private int wordCount;
        private Text outputValue = new Text();
        private String filePath;
        private boolean isGetFilePath;
        @Override
        protected void reduce(Text key, Iterable<Text> values, Redu
cer<Text, Text, Text, Text>.Context context)
                throws IOException, InterruptedException {
            isGetFilePath = false;
            wordCount = 0;
            for (Text path : values) {
                wordCount += 1;
                if(!isGetFilePath){
```

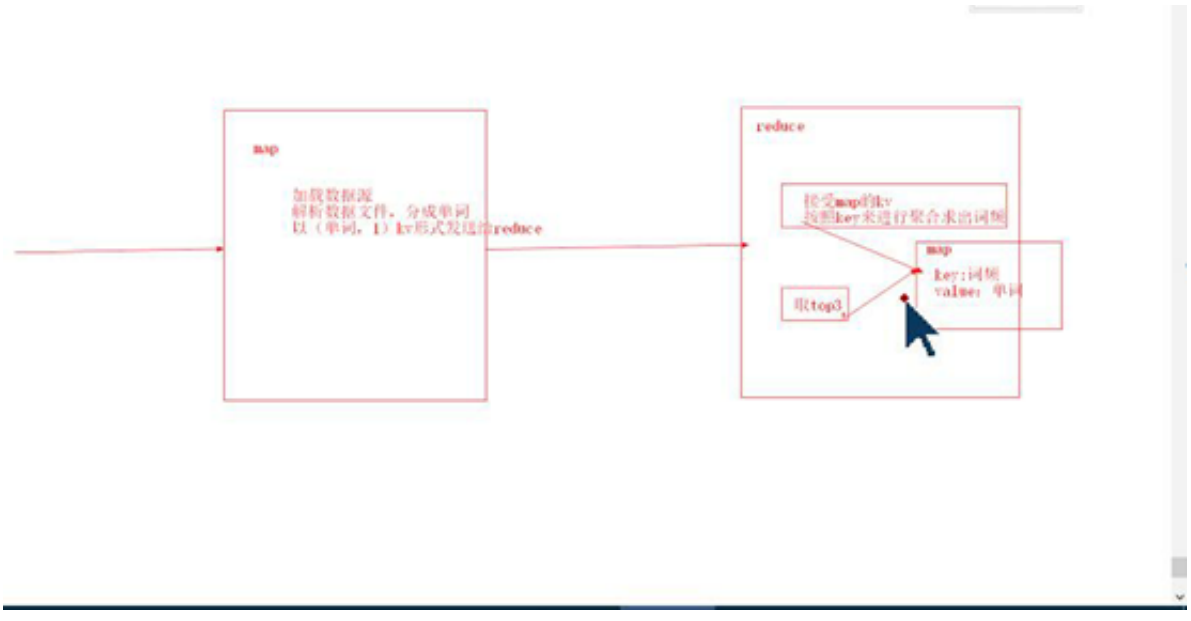
```
                    filePath = path.toString();
                    isGetFilePath = true;
                }
            }
            outputValue.set(filePath + "[" + wordCount + "]");
            context.write(key, outputValue);
        }

    }

    /**
    * 项目名称：mapreeduce
    * 类名称：ReversedIndexReducer
    * 类描述：接受Combinner的输入，并把相同key(单词)下不同的values(文件名称[单词在文件中出现的次数])，
    * 输出key(单词)，value(文件名[词频]----文件名[词频]---...)
    * @version
    */
    public static class ReversedIndexReducer extends Reducer<Text, Text, Text, Text> {
        private StringBuffer valueStr;
        private Text outputValue = new Text();
        private boolean isInitlized;
        @Override
        protected void reduce(Text key, Iterable<Text> values, Reducer<Text, Text, Text, Text>.Context context)
                throws IOException, InterruptedException {
            valueStr = new StringBuffer();
            isInitlized = false;
            for (Text value : values) {
                if(isInitlized){
                    valueStr.append("---" + value.toString());
                }else {
                    // 第一次往valueStr里面放内容
                    valueStr.append(value.toString());
                    isInitlized = true;
                }
            }
            outputValue.set(valueStr.toString());
            context.write(key, outputValue);
        }

    }

    public static void main(String[] args) throws Exception {
        Configuration configuration = new Configuration();
        Job job =  Job.getInstance(configuration);
        job.setJarByClass(ReversedIndex.class);
        job.setJobName("倒序索引");
```

```
        job.setMapperClass(ReversedIndexMap.class);
        job.setCombinerClass(ReversedIndexCombinner.class);
        job.setReducerClass(ReversedIndexReducer.class);

        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(Text.class);

        Path inputPath = new Path("/reversetext");
        Path outputPath = new Path("/bd14/reverseindex");
        outputPath.getFileSystem(configuration).delete(outputPath,t
rue);
        FileInputFormat.addInputPath(job, inputPath);
        FileOutputFormat.setOutputPath(job, outputPath);

        job.setNumReduceTasks(2);
        job.setInputFormatClass(ReversedIndexInputFormat.class);
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

TopN问题

> 对于topN问题，我们以WordCount来展开叙述，对于这个问题，有两种可行的方案
> **方案一**

编写两个mr程序，第一个mr程序进行计数，第二个mr程序获取topN的值，但是这种方式需要书写两个mr程序，过程比较繁琐，不推荐使用



方案二

当然，方案二必须是要书写一个mr了，要不多浪费感情啊。MapReduce分为两个阶段，Map阶段和Reduce阶段

Map

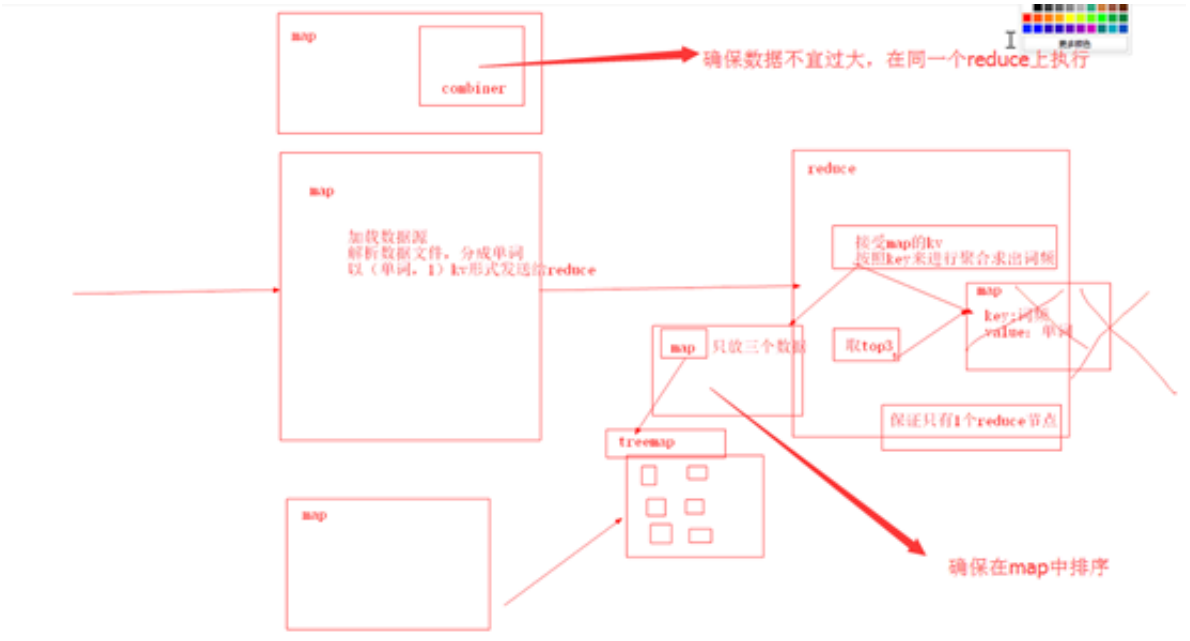
map阶段主要做的事情是加载数据，解析数据，将数据分隔成单词，然后以kv(单词,1)的形式将数据发送给reduce

Reduce

reduce阶段，接收数据，按照key进行统计，计算出词频，然后获取topN，输出结果

计算出词频之后，怎么将结果输出呢？

我们采用的思路是将计算的词频数据放入到map中，但是当数据量过大时，在内存中的数据会写入到磁盘，但是我们不能将数据写入到磁盘，因为这些数据并不是我们想要的，因此我们需要控制map的大小，当数据量小时，这种现象根本不会出现，但是我们是做大数据的，自然要解决这种问题，我们可以控制放入map的数据量，我们只将N对数据放入到map中，这样就解决了我们的问题。但是感觉还是不完美，如果放进map中的数据自己能够排序，那就更方便了。上天总是眷顾那些有思想的人，我们可以使用treeMap，treeMap和map是一样的，但放入treemap中的数据可以帮我们自动排序。现实总是很残酷的，如果有很多的mapwork，我们该怎么办呢，我们怎么能够保证数据在同一个reduce上执行呢，其实也很简单，我们可以使用combinner进行整合



```
import java.io.IOException;
import java.util.Set;
import java.util.TreeMap;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WordCountTopN {

    public static class WordCountTopNMap extends Mapper<LongWritabl
e, Text, Text, IntWritable>{

        private final IntWritable ONE = new IntWritable(1);
        private Text outKey = new Text();
        private String[] infos;
        @Override
        protected void map(LongWritable key, Text value, Mapper<Lon
gWritable, Text, Text, IntWritable>.Context context)
                throws IOException, InterruptedException {
            infos = value.toString().split("[\\s\\(\\)\\.,]");
            for (String word : infos) {
                outKey.set(word);
                context.write(outKey, ONE);
            }
        }
    }

    public static class WordCountTopNReduce extends Reducer<Text, I
ntWritable,Text, IntWritable>{
        private int sum;
        private Text outKey = new Text();
        private IntWritable outValue = new IntWritable();
        // 开辟内存空间保存topN,TreeMap是一个排序的map
        private TreeMap<Integer, String> topN = new TreeMap<>();
        @Override
        protected void reduce(Text key, Iterable<IntWritable> value
s,
                Reducer<Text, IntWritable, Text, IntWritable>.Conte
xt context) throws IOException, InterruptedException {
            sum = 0;
            for (IntWritable value : values) {
```

```
                sum += value.get();
            }
            // 把计算结果放入到topN
            // 先看看topN中有没有相同的key，如果有把topN中相同key对应的val
ue和单词串在一起，如果没有，直接放进去
            // 放进去treeMap会自动排序，这个时候把最后一个再删除，保证topN中
只要N个kv对

            if(topN.size() < 3){
                if(topN.get(sum) != null){
                    // 如果存在，在后面进行追加
                    topN.put(sum, topN.get(sum) + "--" + key.toStri
ng());
                }else{
                    topN.put(sum, key.toString());
                }
            }else {
                // 大于等于N的话放进去一个删除一个，始终保证topN中有N个元素
                if(topN.get(sum) != null){
                    topN.put(sum, topN.get(sum) + "--" + key.toStri
ng());
                }else {
                    topN.put(sum, key.toString());
//                  topN.remove(topN.lastKey()); // 求topN最小
                    topN.remove(topN.firstKey()); // 求最大topN
                }
            }
        }

        @Override
        protected void cleanup(Reducer<Text, IntWritable, Text, Int
Writable>.Context context)
                throws IOException, InterruptedException {
            if(topN != null && !topN.isEmpty()){
                Set<Integer> keys = topN.descendingKeySet(); // //
求最大topN
//              Set<Integer> keys = topN.keySet(); // // 求topN最小
                for (Integer key : keys) {
                    outKey.set(topN.get(key));
                    outValue.set(key);
                    context.write(outKey, outValue);
                }
            }
        }
    }

    public static void main(String[] args) throws Exception {
        Configuration configuration = new Configuration();
        Job job = Job.getInstance(configuration);
```

```
        job.setJarByClass(WordCountTopN.class);
        job.setJobName("词频topN");

        job.setMapperClass(WordCountTopNMap.class);
        job.setReducerClass(WordCountTopNReduce.class);
        job.setCombinerClass(WordCountTopNReduce.class);

        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);

        Path inputPath = new Path("/reversetext/reverse1.txt");
        Path outputPath = new Path("/bd14/topN");
        outputPath.getFileSystem(configuration).delete(outputPath,t
rue);

        FileInputFormat.addInputPath(job, inputPath);
        FileOutputFormat.setOutputPath(job, outputPath);

        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```