MR 典型编程场景 2

目录

1、自定义 OutputFormat--数据分类输出

1.1、需求

现有一些原始日志需要做增强解析处理，流程：

1、 从原始日志文件中读取数据
2、 根据业务获取业务数据库的数据
3、 根据某个连接条件获取相应的连接结果

典型业务场景如：爬虫 URL 管理，移动号码管理

1.2、分析

程序的关键点是要在一个 MapReduce 程序中根据数据的不同输出两类结果到不同目录，这类灵活的输出需求可以通过自定义 OutputFormat 来实现

1.3、实现

实现要点：

1、 在 MapReduce 中访问外部资源
2、 自定义 OutputFormat，改写其中的 RecordWriter，改写具体输出数据的方法 write()

以 Score.txt 的 32 条学生考试记录为例，现要求把参考次数>=7 的输出到一个文件/output/out1，然后剩下的不合格的参考输出到另外一个文件/output/out2

实现：

第一步，实现自己的 OutputFormat

```java
package com.ghgj.mr.format.output;

import java.io.IOException;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FSDataOutputStream;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IOUtils;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.RecordWriter;
import org.apache.hadoop.mapreduce.TaskAttemptContext;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class LongOutputFormat extends FileOutputFormat<Text, NullWritable>{

    @Override
    public RecordWriter<Text, NullWritable> getRecordWriter(
            TaskAttemptContext job) throws IOException, InterruptedException {

        Configuration configuration = job.getConfiguration();
        FileSystem fs = FileSystem.get(configuration);

        Path p1 = new Path("d:/outputformat/out1");
        Path p2 = new Path("d:/outputformat/out2");

        FSDataOutputStream out1 = fs.create(p1);
        FSDataOutputStream out2 = fs.create(p2);

        return new MyRecordWriter(out1, out2);
    }

    static class MyRecordWriter extends RecordWriter<Text, NullWritable>{

        FSDataOutputStream fsdout = null;
        FSDataOutputStream fsdout1 = null;

        public MyRecordWriter(FSDataOutputStream fsdout,
                FSDataOutputStream fsdout1) {
            super();
            this.fsdout = fsdout;
				this.fsdout1 = fsdout1;
			}

			@Override
			public void write(Text key, NullWritable value) throws IOException,
					InterruptedException {
				String[] strs = key.toString().split("::");
				if(strs[0].equals("1")){
					fsdout.write((strs[1]+"\n").getBytes());
				}else{
					fsdout1.write((strs[1]+"\n").getBytes());
				}
			}

			@Override
			public void close(TaskAttemptContext context) throws IOException,
					InterruptedException {
				IOUtils.closeStream(fsdout);
				IOUtils.closeStream(fsdout1);
			}
		}
}
```

第二步，实现 MapReduce 程序

```
	package com.ghgj.mr.format.output;

	import java.io.IOException;

	import org.apache.hadoop.conf.Configuration;
	import org.apache.hadoop.fs.FileSystem;
	import org.apache.hadoop.fs.Path;
	import org.apache.hadoop.io.LongWritable;
	import org.apache.hadoop.io.NullWritable;
	import org.apache.hadoop.io.Text;
	import org.apache.hadoop.mapreduce.Job;
	import org.apache.hadoop.mapreduce.Mapper;
	import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
	import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

	public class MultipleOutputMR {

		public static void main(String[] args) throws Exception {

			Configuration conf = new Configuration();
		Job job = Job.getInstance(conf);

		job.setJarByClass(MultipleOutputMR.class);

		job.setMapperClass(MultipleOutputMRMapper.class);
		job.setMapOutputKeyClass(Text.class);
		job.setMapOutputValueClass(NullWritable.class);

		job.setOutputFormatClass(LongOutputFormat.class);

		FileInputFormat.setInputPaths(job, "d:/score/input");

		Path outPath = new Path("d:/score/output_success");
		FileSystem fs = FileSystem.get(conf);
		if (fs.exists(outPath)) {
			fs.delete(outPath, true);
		}
		FileOutputFormat.setOutputPath(job, outPath);

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

	static class MultipleOutputMRMapper extends
			Mapper<LongWritable, Text, Text, NullWritable> {

		@Override
		protected void map(LongWritable key, Text value, Context context)
				throws IOException, InterruptedException {

			// value
			// 参考次数大于 7 次算合格
			String[] splits = value.toString().split("\t");
			if (splits.length > 9) {
				context.write(new Text("1::" + value.toString()), NullWritable.get());
			} else {
				context.write(new Text("2::" + value.toString()), NullWritable.get());
			}
		}
	}
}
```

2、自定义 InputFormat--小文件合并

2.1、需求

无论 HDFS 还是 MapReduce，对于小文件都有损效率，实践中，又难免面临处理大量小文件的场景，此时，就需要有相应解决方案

2.2、分析

小文件的优化无非以下几种方式：
1、 在数据采集的时候，就将小文件或小批数据合成大文件再上传 HDFS
2、 在业务处理之前，在 HDFS 上使用 MapReduce 程序对小文件进行合并
3、 在 MapReduce 处理时，可采用 CombineFileInputFormat 提高效率

2.3、实现

在此，我们采用第二种方式使用 MapReduce 程序来对小文件进行合并。注意： 并不是说编写一个 MR 程序来实现对这小文件的计算，只是做合并

核心实现思路：
1、编写自定义的 InputFormat
2、改写 RecordReader，实现一次 maptask 读取一个小文件的完整内容封装了一个 KV 对
3、在 Driver 类中一定要设置使用自定义的 InputFormat：
job.setInputFormatClass(WholeFileInputFormat.class)

看具体实现：
第一步，编写自定义的 InputFormat

```
package com.ghgj.mr.format.input;

import java.io.IOException;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.InputSplit;
import org.apache.hadoop.mapreduce.JobContext;
import org.apache.hadoop.mapreduce.RecordReader;
import org.apache.hadoop.mapreduce.TaskAttemptContext;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

public class WholeFileInputFormat extends FileInputFormat<NullWritable, Text> {
	// 设置每个小文件不可分片,保证一个小文件生成一个 key-value 键值对
	@Override
	protected boolean isSplitable(JobContext context, Path file) {
		return false;
	}

	@Override
	public RecordReader<NullWritable, Text> createRecordReader(InputSplit split,
TaskAttemptContext context) throws IOException, InterruptedException {
		WholeFileRecordReader reader = new WholeFileRecordReader();
		reader.initialize(split, context);
		return reader;
	}
}
```

第二步，编写自定义的 RecordReader

```
package com.ghgj.mr.format.input;

import java.io.IOException;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FSDataInputStream;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IOUtils;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.InputSplit;
import org.apache.hadoop.mapreduce.RecordReader;
import org.apache.hadoop.mapreduce.TaskAttemptContext;
import org.apache.hadoop.mapreduce.lib.input.FileSplit;

class WholeFileRecordReader extends RecordReader<NullWritable, Text> {
	private FileSplit fileSplit;
	private Configuration conf;
	private Text value = new Text();
	private boolean processed = false;

	@Override
	public void initialize(InputSplit split, TaskAttemptContext context)
			throws IOException, InterruptedException {
		this.fileSplit = (FileSplit) split;
		this.conf = context.getConfiguration();
	}
```

```
    @Override
    public boolean nextKeyValue() throws IOException, InterruptedException {
        if (!processed) {
            byte[] contents = new byte[(int) fileSplit.getLength()];
            Path file = fileSplit.getPath();
            FileSystem fs = file.getFileSystem(conf);
            FSDataInputStream in = null;
            try {
                in = fs.open(file);
                // 把输入流上的数据全部读取到 contents 字节数组里
                IOUtils.readFully(in, contents, 0, contents.length);
                // 把读取到的数据设置到 value 里
                value.set(contents, 0, contents.length);
            } finally {
                IOUtils.closeStream(in);
            }
            processed = true;
            return true;
        }
        return false;
    }

    @Override
    public NullWritable getCurrentKey() throws IOException, InterruptedException {
        return NullWritable.get();
    }

    @Override
    public Text getCurrentValue() throws IOException, InterruptedException {
        return value;
    }

    @Override
    public float getProgress() throws IOException {
        return processed ? 1.0f : 0.0f;
    }

    @Override
    public void close() throws IOException {
        // do nothing
    }
}
```

第三步，编写 MapReduce 程序

```
package com.ghgj.mr.format.input;

import java.io.IOException;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.InputSplit;
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.input.FileSplit;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;

public class SmallFilesConvertToBigMR extends Configured implements Tool {

	public static void main(String[] args) throws Exception {
		int exitCode = ToolRunner.run(new SmallFilesConvertToBigMR(), args);
		System.exit(exitCode);
	}

static class SmallFilesConvertToBigMRMapper extends Mapper<NullWritable, Text, Text, Text> {
		private Text filenameKey;

		@Override
		protected void setup(Context context) throws IOException, InterruptedException {
			InputSplit split = context.getInputSplit();
			Path path = ((FileSplit) split).getPath();
			filenameKey = new Text(path.toString());
		}

		@Override
		protected void map(NullWritable key, Text value, Context context)
				throws IOException, InterruptedException {
			context.write(filenameKey, value);
		}
	}
static class SmallFilesConvertToBigMRReducer extends Reducer<Text, Text, NullWritable, Text> {
		@Override
		protected void reduce(Text filename, Iterable<Text> bytes,
				Context context) throws IOException, InterruptedException {
			context.write(NullWritable.get(), bytes.iterator().next());
		}
	}

	@Override
	public int run(String[] args) throws Exception {
		Configuration conf = new Configuration();

		conf.set("fs.defaultFS", "hdfs://hadoop02:9000");
		System.setProperty("HADOOP_USER_NAME", "hadoop");

		Job job = Job.getInstance(conf, "combine small files to bigfile");
		job.setJarByClass(SmallFilesConvertToBigMR.class);

		job.setMapOutputKeyClass(Text.class);
		job.setMapOutputValueClass(Text.class);
		job.setMapperClass(SmallFilesConvertToBigMRMapper.class);

		job.setReducerClass(SmallFilesConvertToBigMRReducer.class);
		job.setOutputKeyClass(NullWritable.class);
		job.setOutputValueClass(Text.class);

		job.setInputFormatClass(WholeFileInputFormat.class);
		// job.setOutputFormatClass(SequenceFileOutputFormat.class);

		Path input = new Path("/smallfiles");
		Path output = new Path("/bigfile");
		FileInputFormat.setInputPaths(job, input);
		FileSystem fs = FileSystem.get(conf);
		if (fs.exists(output)) {
			fs.delete(output, true);
		}
		FileOutputFormat.setOutputPath(job, output);

		int status = job.waitForCompletion(true) ? 0 : 1;
		return status;
	}
}
```