# 自定义inputFormat合并小文件

## 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, 将多个小文件合并成一个文件 SequenceFile.SequenceFile 里面存储着多个文件。存储的形式为文件名称为 key,文件内容为 value。

## 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combineInputFormat提高效率

## 3 实现

本节实现的是上述第二种方式

程序的核心机制：

自定义一个InputFormat

改写RecordReader，实现一次读取一个完整文件封装为KV

在输出时使用SequenceFileOutPutFormat输出合并文件

### 第一步：自定义InputFromat

**import org.apache.hadoop.fs.Path;**

**import org.apache.hadoop.io.BytesWritable;**

**import org.apache.hadoop.io.NullWritable;**

**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;**

**import java.io.IOException;**

**public class CustomFileInputFormat extends FileInputFormat<NullWritable, BytesWritable> {**

**//直接返回文件不可切割,保证一个文件是一个完整的一行**

**@Override**

**protected boolean isSplitable(JobContext context, Path filename) {**

**return false;**

**}**

**@Override**

**public RecordReader<NullWritable, BytesWritable> createRecordReader(InputSplit split, TaskAttemptContext context) throws IOException, InterruptedException {**

**CustomFileRecordReader customFileRecordReader = new CustomFileRecordReader();**

**customFileRecordReader.initialize(split, context);**

**return customFileRecordReader;**

**}**

**}**

### 第二步：自定义RecordReader

import org.apache.commons.io.IOUtils;

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.BytesWritable;

import org.apache.hadoop.io.NullWritable;

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;

import java.io.IOException;

public class CustomFileRecordReader extends RecordReader<NullWritable, BytesWritable> {

private FileSplit fileSplit;

private Configuration configuration;

private BytesWritable bytesWritable = new BytesWritable();

private boolean processed = false;

/\*\*

\* 初始化的方法 带了两个参数，第一个inputSplit,封装了我们的数据都在这里面

\* context与我们mapreduce当中的context类似 ，上下文对象

\*

\* @param split

\* @param context

\* @throws IOException

\* @throws InterruptedException

\*/

@Override

public void initialize(InputSplit split, TaskAttemptContext context) throws IOException, InterruptedException {

this.fileSplit = (FileSplit) split;

this.configuration = context.getConfiguration();

}

//往下读文件，如果返回true，表示已经读取过了

@Override

public boolean nextKeyValue() throws IOException, InterruptedException {

if (!processed) {

//获取文件切片的路径，可以通过filesystem读取这个文件，读成一个流

Path path = fileSplit.getPath();

//获取我们的FileSystem

FileSystem fileSystem = null;

//通过FileSystem来获取我们的文件输入流

FSDataInputStream inputStream = null;

try {

fileSystem = FileSystem.get(configuration);

inputStream = fileSystem.open(path);

//初始化一个字节数组，长度为读取内容的大小

byte[] contents = new byte[(int) fileSplit.getLength()];

//将我们文件的输入流读取到byte[]当中

IOUtils.readFully(inputStream, contents, 0, contents.length);

//将我们字节数组中的内容全部转换到我们的BytesWritable当中进行封装

bytesWritable.set(contents, 0, contents.length);

} catch (IOException e) {

e.printStackTrace();

} finally {

fileSystem.close();

if (null != inputStream) {

IOUtils.closeQuietly(inputStream);

}

}

processed=true;

return true;

} else {

return false;

}

}

//获取当前返回的key

@Override

public NullWritable getCurrentKey() throws IOException, InterruptedException {

return NullWritable.get();

}

//获取当前返回的value值

@Override

public BytesWritable getCurrentValue() throws IOException, InterruptedException {

return bytesWritable;

}

//获取读取的进度

@Override

public float getProgress() throws IOException, InterruptedException {

return processed ? 1.0f : 0.0f;

}

//关闭清理方法

@Override

public void close() throws IOException {

}

}

### 第三步：CustomMapper

import org.apache.hadoop.io.BytesWritable;

import org.apache.hadoop.io.NullWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

import org.apache.hadoop.mapreduce.lib.input.FileSplit;

import java.io.IOException;

public class CustomMapper extends Mapper<NullWritable,BytesWritable,Text,BytesWritable> {

@Override

protected void map(NullWritable key, BytesWritable value, Context context) throws IOException, InterruptedException {

//获取文件名，将我们的k2,v2 定义成Text文件名 BytesWritable 文件内容

FileSplit inputSplit = (FileSplit) context.getInputSplit();

String name = inputSplit.getPath().getName();

context.write(new Text(name),value);

}

}

### 第四步：CustomInputFormatDriver

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.conf.Configured;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.BytesWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.lib.output.SequenceFileOutputFormat;

import org.apache.hadoop.util.Tool;

import org.apache.hadoop.util.ToolRunner;

public class CustomInputFormatDriver extends Configured implements Tool{

@Override

public int run(String[] strings) throws Exception {

Job job = Job.getInstance(super.getConf(), "CustomInputFormatDriver");

job.setInputFormatClass(CustomFileInputFormat.class);

CustomFileInputFormat.addInputPath(job,new Path("E:\\wordcount\\CustomFileInputFormat\_input"));

job.setMapperClass(CustomMapper.class);

job.setMapOutputKeyClass(Text.class);

job.setMapOutputValueClass(BytesWritable.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(BytesWritable.class);

job.setOutputFormatClass(SequenceFileOutputFormat.class);

SequenceFileOutputFormat.setOutputPath(job,new Path("E:\\wordcount\\CustomFileInputFormat\_output"));

boolean b = job.waitForCompletion(true);

return b?0:1;

}

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

int run = ToolRunner.run(new Configuration(), new CustomInputFormatDriver(), args);

System.exit(run);

}

}

# 自定义outputFormat

## 1 需求

现在有一些订单的评论数据，需求，将订单的好评与差评进行区分开来，将最终的数据分开到不同的文件夹下面去，数据内容参见资料文件夹，其中数据第九个字段表示好评，中评，差评。0：好评，1：中评，2：差评

## 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

## 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()

### 第一步：自定义outputformat

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.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;

import java.io.IOException;

public class CustomFileOutputFormat extends FileOutputFormat<Text,NullWritable>{

@Override

public RecordWriter<Text, NullWritable> getRecordWriter(TaskAttemptContext taskAttemptContext) throws IOException, InterruptedException {

//获取我们的configuration

Configuration configuration = taskAttemptContext.getConfiguration();

FileSystem fileSystem = FileSystem.get(configuration);

//好评写入的路径

FSDataOutputStream fsDataOutputStream = fileSystem.create(new Path("E:\\wordcount\\CustomFileOutputFormat\_output\\out\_good\_comment\\1.txt"));

FSDataOutputStream fsDataOutputStream1 = fileSystem.create(new Path("E:\\wordcount\\CustomFileOutputFormat\_output\\out\_bad\_comment\\1.txt"));

//自定义RecordWriter

CustomRecordWriter customRecordWriter = new CustomRecordWriter(fsDataOutputStream, fsDataOutputStream1);

return customRecordWriter;

}

}

### 第二步：自定义RecordWriter

import org.apache.hadoop.fs.FSDataOutputStream;

import org.apache.commons.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 java.io.IOException;

public class CustomRecordWriter extends RecordWriter<Text,NullWritable> {

private FSDataOutputStream out1;

private FSDataOutputStream out2;

public CustomRecordWriter() {

}

public CustomRecordWriter(FSDataOutputStream out1, FSDataOutputStream out2) {

this.out1 = out1;

this.out2 = out2;

}

//通过这个方法把文件往外写，决定了我们的文件写入到哪个路径下

@Override

public void write(Text key, NullWritable value) throws IOException, InterruptedException {

//把在CustomFileOutputFormat里面定义的两个FSDataOutputStream传入到这里来，

//在这里对我们的文件进行判断，如果类型是0，写入到一个文件中，如果评论类型是1和2写入到另一个文件中

if (key.toString().split("\t")[9].equals("0")){

//好评

out1.write(key.toString().getBytes());

out1.write("\r\n".getBytes());

}else {

//中评和差评

out2.write(key.toString().getBytes());

out2.write("\r\n".getBytes());

}

}

@Override

public void close(TaskAttemptContext context) throws IOException, InterruptedException {

if (null!=out1){

IOUtils.closeQuietly(out1);

}

if (null!=out2){

IOUtils.closeQuietly(out2);

}

}

}

### 第三步：CustomMapper

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.NullWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

import java.io.IOException;

public class CustomMapper extends Mapper<LongWritable,Text,Text,NullWritable> {

@Override

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

context.write(value,NullWritable.get());

}

}

### 第四步：CustomOutputFormatDriver

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.conf.Configured;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.NullWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;

import org.apache.hadoop.util.Tool;

import org.apache.hadoop.util.ToolRunner;

public class CustomOutputFormatDriver extends Configured implements Tool{

@Override

public int run(String[] strings) throws Exception {

Job job=Job.getInstance(super.getConf(),"CustomOutputFormatDriver");

job.setInputFormatClass(TextInputFormat.class);

TextInputFormat.addInputPath(job,new Path("E:\\wordcount\\CustomFileOutputFormat\_input"));

job.setMapperClass(CustomMapper.class);

job.setMapOutputKeyClass(Text.class);

job.setMapOutputValueClass(NullWritable.class);

job.setOutputFormatClass(CustomFileOutputFormat.class);

CustomFileOutputFormat.setOutputPath(job, new Path("E:\\wordcount\\CustomFileOutputFormat\_output"));

boolean b = job.waitForCompletion(true);

return b?0:1;

}

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

int run = ToolRunner.run(new Configuration(), new CustomOutputFormatDriver(), args);

System.exit(run);

}

}