Day08_Avro的序列化及代码实现

大数据-张军锋 Day08 Avro的序列化 代码实现

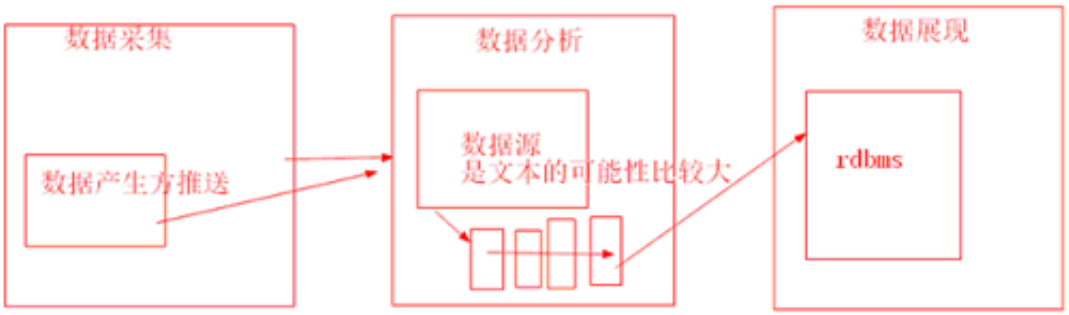
Avro

> Avro是序列化和反序列化工具和RPC库组成，用于改进MapReduce中的数据交换频率、互操作性和版本控制。Avro使用压缩二进制数据格式（可以通过配置选项设置压缩，压缩可以快速提高序列化时间）。Avro可以直接在MR中该处理，可以使用通用的数据模型处理schema数据。

Avro 提供的基本服务

1. 提供更加丰富的数据类型
2. 提供快速压缩的二进制数据格式
3. Avro是一个文件容器，可以解决小文件的问题
4. RPC(remote procedure call) 远程过程调用
5. 支持代码自动生成策略，并且能够和动态语言进行结合

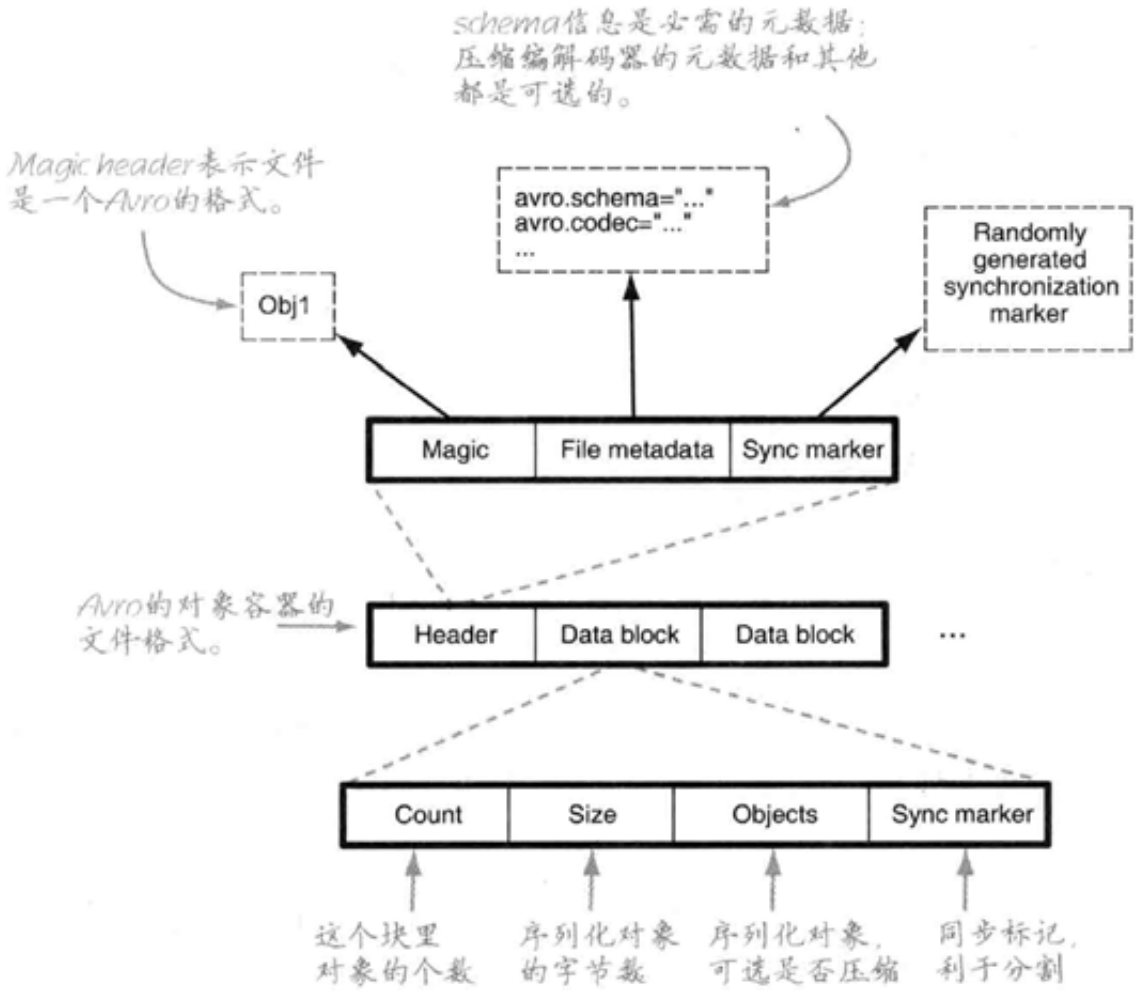
> 在大数据中，大数据的业务主要分成三个部分，数据采集、数据分析、数据展现
> 数据采集可以通过爬虫等手段获取，但是在实际生活中，大部分数据都是有数据产商提供的，所以获取数据源并不是我们的重点，我们的重点是进行数据分析，提到数据分析，就会牵扯到io的读写操作。在大数据产业链中，数据源大部分都是文本类型，但是文本类型的数据读取比较消耗资源，因为数据资源是数据产商提供的，我们不能改变，但是我们可以在我们分析的阶段将数据之间的传输格式定义为其他的方式。例如：avro，sequenceFile等等，在数据展现时一般是将数据存入到关系型数据库(rdbms)中.



SequenceFile：二进制文件格式,能够保存数据类型，是hadoop自带的数据类型
Avro是hadoop的创始人开发出的，avro能够将每个字段的数据类型进行存储
Arvo：序列化反序列化，序列化过程中提供复杂的数据结构(多个字段，多个字段类型)，类似数据库中的表，也就是模式
Avro使用json的形式定义schema，json在数据交互时，扮演的角色是数据存储格式，在webservice和rpc一般希望使用json来作为数据的传输格式

Avro序列化后文件详解

> Arvo对象的文件格式是有header和data block两部分组成。header中包含Magic、File metadata、Sync marker。Data block 中包含Count、Size、Objects、Sync marker



Avro的代码实现

Arvo在大数据中扮演两个角色，一是读写文件效果比较好，二是arvo是一个容器，能够将小文件合并成大文件，起到优化程序的角色

读写操作

> arvo读取文件时，有两种不同的方式，一是通过模式读取，一个不需要模式的读取，下面一一进行阐述

模式读取

写操作

1. 编写schema

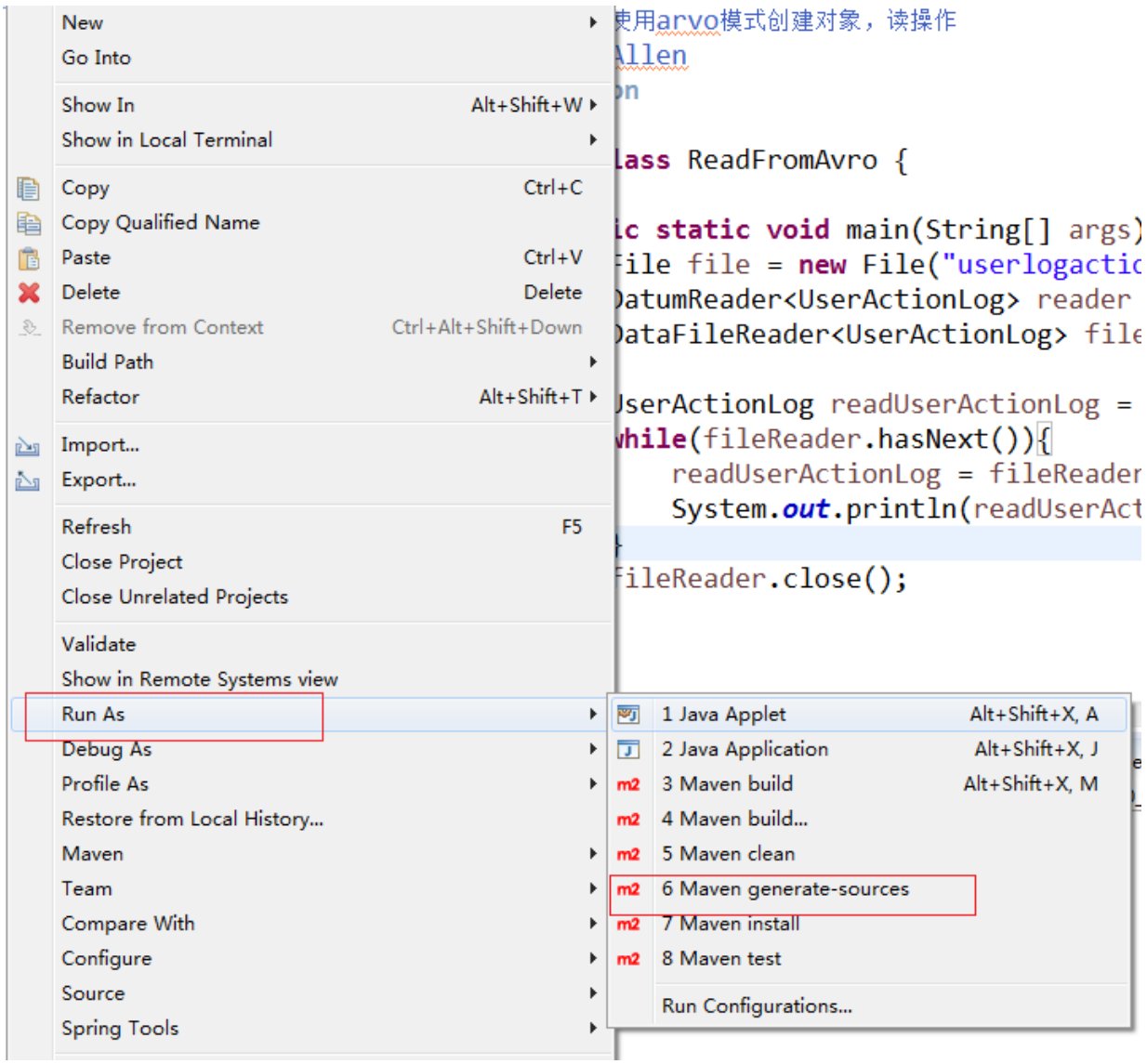
> 定义schema，文件名必须是以.avsc结尾的，并且目录位置设置需要与maven项目中pom文件的 `<sourceDirectory>${project.basedir}/src/main/avro/</sourceDirectory>` 对应，json中表示的具体参数类型 ，请参考 http://avro.apache.org/docs/1.8.2/gettingstartedjava.html中defining schema部分

```
{
"type":"record",
"name":"UserActionLog",
"namespace":"top.xiesen.avro.schema",
"fields":[
    {"name":"userName","type":"string"},
    {"name":"actionType","type":"string"},
    {"name":"ipAddress","type":"string"},
    {"name":"gender","type":"int"},
    {"name":"provience","type":"string"}
]
}
```

编写完成schema约束文件之后，我们需要生成对应的模式，arvo提供了自动生成模式的代码，我们只需要在maven中配置相对应的插件即可

```
<plugin>
    <groupId>org.apache.avro</groupId>
    <artifactId>avro-maven-plugin</artifactId>
    <version>1.8.2</version>
    <executions>
        <execution>
        <!-- generate-sources maven指令，生成代码 ， 使用mvn generate-so
urces指令生成代码-->
            <phase>generate-sources</phase>
            <goals>
            <!-- 执行生成schema -->
                <goal>schema</goal>
            </goals>
            <configuration>
                <!-- 生成代码的依据(定义schema文件) -->
                <sourceDirectory>${project.basedir}/src/main/avr
o/</sourceDirectory>
                <!-- 生成代码的输出位置 -->
                <outputDirectory>${project.basedir}/src/main/jav
a/</outputDirectory>
            </configuration>
        </execution>
    </executions>
</plugin>
<plugin>
    <!-- mvn  compiler 指令-->
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-compiler-plugin</artifactId>
    <configuration>
        <source>1.6</source>
        <target>1.6</target>
    </configuration>
</plugin>
```

配置完成之后我们只需要执行`mvn generate-sources`即可，也可以通过eclipse进行执行，道理是一样的，具体操作如图所示


New
Go Into
Show In Alt+Shift+W
Show in Local Terminal
Copy Ctrl+C
Copy Qualified Name
Paste Ctrl+V
Delete Delete
Remove from Context Ctrl+Alt+Shift+Down
Build Path
Refactor Alt+Shift+T
Import...
Export...
Refresh F5
Close Project
Close Unrelated Projects
Validate
Show in Remote Systems view
Run As
Debug As
Profile As
Restore from Local History...
Maven
Team
Compare With
Configure
Source
Spring Tools
1 Java Applet Alt+Shift+X, A
2 Java Application Alt+Shift+X, J
3 Maven build Alt+Shift+X, M
4 Maven build...
5 Maven clean
6 Maven generate-sources
7 Maven install
8 Maven test
Run Configurations...
使用arvo模式创建对象，读操作
Allen
lass ReadFromAvro {
ic static void main(String[] args)
File file = new File("userlogactic
DatumReader<UserActionLog> reader
DataFileReader<UserActionLog> file
UserActionLog readUserActionLog =
while(fileReader.hasNext()){
readUserActionLog = fileReader
System.out.println(readUserAct
FileReader.close();

2. 编写操作类

```
import java.io.File;
import org.apache.avro.file.DataFileWriter;
import org.apache.avro.io.DatumWriter;
import org.apache.avro.specific.SpecificDatumWriter;
import top.xiesen.avro.schema.UserActionLog;

/**
* 项目名称：avrotest
* 类名称：WriterAsAvro
* 类描述：使用arvo模式创建对象，写操作
* @version
*/
public class WriterAsAvro {

    public static void main(String[] args) throws Exception {
        UserActionLog userActionLog = new UserActionLog();
        userActionLog.setActionType("login");
        userActionLog.setGender(1);
        userActionLog.setIpAddress("192.168.1.1");
        userActionLog.setProvience("henan");
        userActionLog.setUserName("lisi");

        UserActionLog userActionLog2 = UserActionLog.newBuilder().s
etActionType("logout")
                .setGender(0).setIpAddress("192.168.6.110").setProv
ience("hunan").setUserName("allen").build();
        // 把两条记录写入到文件(序列化)
        DatumWriter<UserActionLog> writer = new SpecificDatumWrite
r<UserActionLog>();
        DataFileWriter<UserActionLog> fileWriter = new DataFileWrit
er<UserActionLog>(writer);

        // 创建序列化
        fileWriter.create(UserActionLog.getClassSchema(), new Fil
e("userlogaction.avro"));
        // 写入内容
        fileWriter.append(userActionLog);
        fileWriter.append(userActionLog2);

        fileWriter.flush();
        fileWriter.close();
    }
}
```

读操作

```
/**
* 项目名称：avrotest
* 类名称：ReadFromAvro
* 类描述：使用arvo模式创建对象，读操作
* @version
*/
public class ReadFromAvro {

    public static void main(String[] args) throws Exception {
        File file = new File("userlogaction.avro");
        DatumReader<UserActionLog> reader = new SpecificDatumReade
r<UserActionLog>();
        DataFileReader<UserActionLog> fileReader = new DataFileRead
er<UserActionLog>(file, reader);

        UserActionLog readUserActionLog = null;
        while(fileReader.hasNext()){
            readUserActionLog = fileReader.next();
            System.out.println(readUserActionLog);
        }
        fileReader.close();
    }
}
```

无模式读取

```
import java.io.File;
import java.io.IOException;

import org.apache.avro.Schema;
import org.apache.avro.file.DataFileWriter;
import org.apache.avro.generic.GenericData;
import org.apache.avro.generic.GenericDatumWriter;
import org.apache.avro.generic.GenericRecord;
import org.apache.avro.io.DatumWriter;

/**
* 项目名称：avrotest
* 类名称：AvroWriter
* 类描述：不使用插件生成模式对象,写操作
* @version
*/
public class AvroWriter {
    private Schema schema;
    // parser是专门用来把字符串或者asvc文件转换成Schema对象的一个工具
    private Schema.Parser parser = new Schema.Parser();

    // 初始化schema，根据要序列化的数据而定
    public AvroWriter(String schemaFile) throws Exception {
        this.schema = parser.parse(new File(schemaFile));
    }

    /**
    * writeData 写数据到文件中
    * @param @param record
    * @param @throws Exception 参数
    * @return void 返回类型
    * @Exception 异常对象
    */
    public void writeData(GenericRecord record) throws Exception {
        DatumWriter<GenericRecord> writer = new GenericDatumWrite
r<GenericRecord>();
        DataFileWriter<GenericRecord> fileWriter = new DataFileWrit
er<GenericRecord>(writer);
        fileWriter.create(schema, new File("noobjectuseraction.avr
o"));
        fileWriter.append(record);
        fileWriter.flush();
    }

    public static void main(String[] args) throws Exception {
        // 用模式文件位置作为参数初始化writer序列化类
        AvroWriter avroWriter = new AvroWriter("src/main/avro/use
r_action_log.avsc");
```

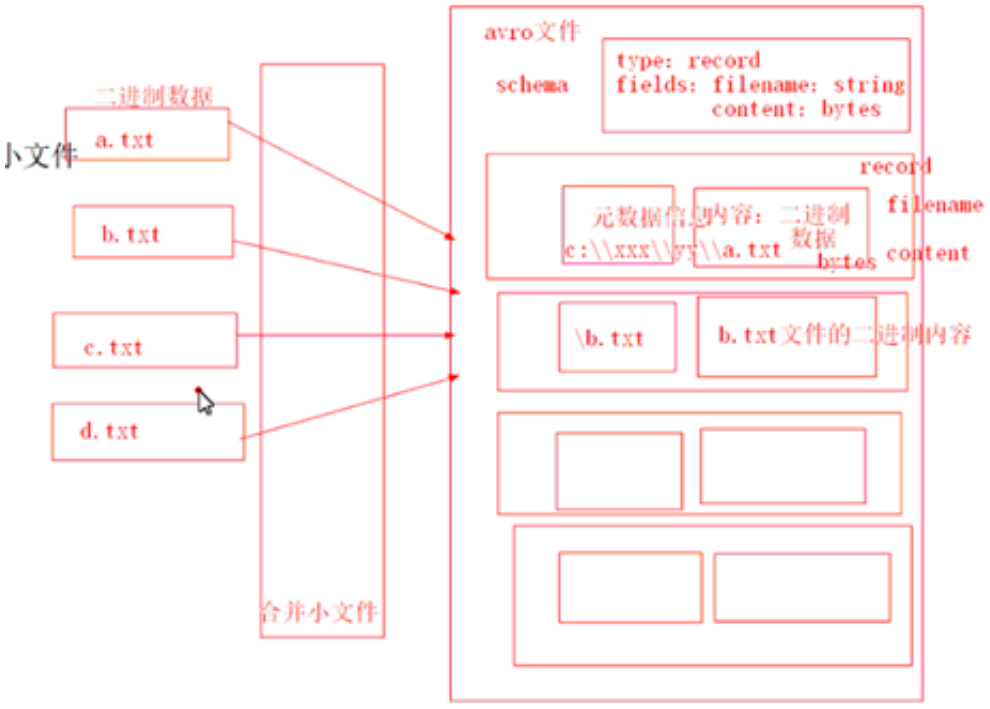
```
        // 创建GenericRecord对象
        GenericRecord record = new GenericData.Record(avroWriter.sc
hema);
        record.put("userName", "zhangsan");
        record.put("actionType", "new_tweet");
        record.put("ipAddress", "192.168.6.76");
        record.put("gender", 0);
        record.put("provience", "bj");

        avroWriter.writeData(record);
    }
}
```

写操作

```
import java.io.File;

import org.apache.avro.file.DataFileReader;
import org.apache.avro.generic.GenericDatumReader;
import org.apache.avro.generic.GenericRecord;
import org.apache.avro.io.DatumReader;

public class AvroRead {

    public static void main(String[] args) throws Exception {
        DatumReader<GenericRecord> reader = new GenericDatumReade
r<GenericRecord>();
        DataFileReader<GenericRecord> fileReader = new DataFileRead
er<GenericRecord>(new File("noobjectuseraction.avro"), reader);
        GenericRecord record = null;
        while(fileReader.hasNext()){
            record = fileReader.next();
            System.out.println(record);
        }
        fileReader.close();
    }
}
```

小文件的合并操作

小文件的合并是将某个目录下的文件,以记录的形式存读取到avro文件中,下面是分析流程示意图



1. 定义schema

```
{
"type":"record",
"name":"SmallFile",
"namespace":"top.xiesen.avro.schema",
"fields":[
    {"name":"fileName","type":"string"},
    {"name":"content","type":"bytes"}
]
}
```

2. 编写程序

```
import java.io.File;
import java.nio.ByteBuffer;
import java.util.ArrayList;
import java.util.List;

import org.apache.avro.Schema;
import org.apache.avro.file.DataFileWriter;
import org.apache.avro.io.DatumWriter;
import org.apache.avro.specific.SpecificDatumWriter;
import org.apache.commons.io.FileUtils;
import org.apache.hadoop.fs.FileUtil;


import top.xiesen.avro.schema.SmallFile;

/**
* 项目名称：avromr
* 类名称：AvroMergeSmallFile
* 类描述：avro把一个文件下的小文件合并成大文件
* @version
*/
public class AvroMergeSmallFile {

    private Schema.Parser parser = new Schema.Parser();
    private Schema schema;
    // 用来设置添加被合并的文件的名称
    private List<String> inputFilePaths = new ArrayList<String>();

    // 在构造方法中初始化schema
    public AvroMergeSmallFile() {
        schema = SmallFile.getClassSchema();
    }

    // 添加要合并的文件夹
    public void addInputFileDir(String inputDir) throws Exception {
        // 获取文件下的所有文件
        File[] files = FileUtil.listFiles(new File(inputDir));
        // 把文件路径添加到inputFilePaths中
        for (File file : files) {
            inputFilePaths.add(file.getPath());
        }
    }

    // 把inputFilePaths中的所有文件合并到一个avro文件中
    public void mergeFiles(String outputPath) throws Exception {
        DatumWriter<SmallFile> writer = new SpecificDatumWriter<Sma
llFile>();
        DataFileWriter<SmallFile> fileWriter = new DataFileWriter<S
```

```
mallFile>(writer);
        // 创建avro文件
        fileWriter.create(SmallFile.getClassSchema(), new File(outp
utPath));
        // 把inputFilePaths的文件一个一个读取出来根据模式放入到avro文件中
        for (String filePath : inputFilePaths) {
            File inputFile = new File(filePath);
            // 把文件读取成字节数组，放入content
            byte[] content = FileUtils.readFileToByteArray(inputFil
e);
            // ByteBuffer调用wrap方法把字节数组封装成一个ByteBuffer对
象，作为参数设置到oneSmallFile的content属性中
            SmallFile oneSmallFile = SmallFile.newBuilder().setFile
Name(inputFile.getAbsolutePath())
                    .setContent(ByteBuffer.wrap(content)).build();
            fileWriter.append(oneSmallFile);
            System.out.println("写入" + inputFile.getAbsolutePath()
+ "成功");
        }
        fileWriter.flush();
        fileWriter.close();
    }

    public static void main(String[] args) throws Exception {
        AvroMergeSmallFile avroMergeSmallFile = new AvroMergeSmallF
ile();
        avroMergeSmallFile.addInputFileDir("D:\\inputpath");
        avroMergeSmallFile.mergeFiles("C:\\Users\\Administrator\\De
sktop\\bigfile.avro");
    }
}
```

对于小文件合并后的大文件进行词频统计

```
import java.io.File;
import java.io.IOException;
import java.nio.ByteBuffer;

import org.apache.avro.Schema;
import org.apache.avro.Schema.Parser;
import org.apache.avro.generic.GenericData;
import org.apache.avro.generic.GenericRecord;
import org.apache.avro.mapred.AvroKey;
import org.apache.avro.mapreduce.AvroJob;
import org.apache.avro.mapreduce.AvroKeyInputFormat;
import org.apache.avro.mapreduce.AvroKeyOutputFormat;
import org.apache.avro.mapreduce.AvroKeyValueInputFormat;
import org.apache.avro.mapreduce.AvroKeyValueOutputFormat;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
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.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import top.xiesen.avro.schema.SmallFile;

// 计算合并后的大的avro文件的词频
public class AvroFilemr {

    /**
    * 项目名称：avromr
    * 类名称：AvroFilemrMap
    * 类描述：读取avro文件，每读取一条记录，其实是一个小文件，对齐wordcount解
析，并以(单词,1)的形式发送到reduce上
    * @version
    */
    public static class AvroFilemrMap extends Mapper<AvroKey<SmallF
ile>, NullWritable, Text, IntWritable> {
        private IntWritable ONE = new IntWritable(1);
        private Text outKey = new Text();
        private String[] infos;
        private ByteBuffer content;

        @Override
        protected void map(AvroKey<SmallFile> key, NullWritable val
ue,
                Mapper<AvroKey<SmallFile>, NullWritable, Text, IntW
```

```
ritable>.Context context)
                throws IOException, InterruptedException {
            content = key.datum().getContent();
            infos = new String(content.array()).split("\\s");
            for (String word : infos) {
                outKey.set(word);
                context.write(outKey, ONE);
            }
        }
    }
    /**
    * 项目名称：avromr
    * 类名称：AvroFilemrReducer
    * 类描述：把wordcount的计算结果，以word_count.avsc的模式输出成avro文件
    * @version
    */
    public static class AvroFilemrReducer extends Reducer<Text, Int
Writable, AvroKey<GenericRecord>, NullWritable> {
        private int sum;
        private Schema writeSchema;
        private GenericRecord record;
        private AvroKey<GenericRecord> outKey = new AvroKey<Generic
Record>();
        private NullWritable outValue = NullWritable.get();

        // 初始化writeSchema和record
        @Override
        protected void setup(Reducer<Text, IntWritable, AvroKey<Gen
ericRecord>, NullWritable>.Context context)
                throws IOException, InterruptedException {
            Parser parser = new Parser();
            writeSchema = parser.parse(new File("src/main/avro/wor
d_count.avsc"));
            record = new GenericData.Record(writeSchema);
        }

        @Override
        protected void reduce(Text key, Iterable<IntWritable> value
s,
                Reducer<Text, IntWritable, AvroKey<GenericRecord>,
NullWritable>.Context context)
                throws IOException, InterruptedException {
            sum = 0;
            for (IntWritable value : values) {
                sum += value.get();
            }
            record.put("word", key.toString());
            record.put("count", sum);
            outKey.datum(record);
```

```
            context.write(outKey, outValue);
        }
    }

    public static void main(String[] args) throws Exception {
        Configuration configuration = new Configuration();
        Job job = Job.getInstance(configuration);
        job.setJarByClass(AvroFilemr.class);
        job.setJobName("计算合并后的大的avro文件的词频");

        job.setMapperClass(AvroFilemrMap.class);
        job.setReducerClass(AvroFilemrReducer.class);

        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(IntWritable.class);
        job.setOutputKeyClass(AvroKey.class);
        job.setOutputValueClass(NullWritable.class);

        // 设置上输入输出的inputformat
        job.setInputFormatClass(AvroKeyInputFormat.class);
        job.setOutputFormatClass(AvroKeyOutputFormat.class);

        // 设置输入输出的schema
        AvroJob.setInputKeySchema(job, SmallFile.getClassSchema());
        Parser parser = new Parser();
        AvroJob.setOutputKeySchema(job, parser.parse(new File("sr
c/main/avro/word_count.avsc")));

        FileInputFormat.addInputPath(job, new Path("/bigfile.avr
o"));
        Path outputDir = new Path("/bd14/AvroFilemr");
        outputDir.getFileSystem(configuration).delete(outputDir,tru
e);
        FileOutputFormat.setOutputPath(job, outputDir);

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