小文件解决方案

小文件解决方案

1. 概述

小文件是指文件size小于HDFS上block大小的文件。这样的文件会给hadoop的扩展性和性能带来严重问题。首先，在HDFS中，任何block，文件或者目录在内存中均以对象的形式存储，每个对象约占150byte，如果有1百万个小文件，每个文件占用一个block，则namenode大约需要2G空间。如果存储1亿个文件，则namenode需要20G空间,这样namenode内存容量严重制约了集群的扩展。 其次，访问大量小文件速度远远小于访问几个大文件。HDFS最初是为流式访问大文件开发的，如果访问大量小文件，需要不断的从一个datanode跳到另一个datanode，严重影响性能。每一个小文件要占用一个slot，而task启动将耗费大量时间甚至大部分时间都耗费在启动task和释放task上。

2. Hadoop的解决方案

对于小文件问题，Hadoop提供了几个解决方案，分别为：Hadoop Archive，Sequence file和CombineFileInputFormat。

2.1. Hadoop Archive

Hadoop Archive或者HAR，是一个高效地将小文件放入HDFS块中的文件存档工具，它能够将多个小文件打包成一个HAR文件，这样在减少namenode内存使用的同时，仍然允许对文件进行透明的访问。

2.1.1. 创建har文件

语法：

hadoop archive –archiveName <name> <src> <dest>

事例：

对某个目录/ceshi下的所有小文件存档成/ceshi/out1：

hadoop archive -archiveName ceshi.har -p hdfs://master:9000/ceshi2 hdfs://master:9000/ceshi

2.1.2. 查看har文件

语法：

Hadoop Archives的URI是har://scheme-hostname:port/archivepath/fileinarchive。

如果没提供scheme-hostname，它会使用默认的文件系统。这种情况下URI是这种形式har:///archivepath/fileinarchive。

例如：

hadoop fs -ls har://hdfs-master:9000/ceshi/ceshi.har

hadoop fs -ls har:///ceshi/ceshi.har

[hadoop@master ~]$ hadoop fs -ls har:///ceshi/ceshi.har

Warning: $HADOOP_HOME is deprecated.

Found 3 items

-rw-r--r-- 2 hadoop supergroup 22930015 2015-06-08 19:43 /ceshi/ceshi.har/partition.txt

-rw-r--r-- 2 hadoop supergroup 25 2015-06-09 12:03 /ceshi/ceshi.har/mini.txt

-rw-r--r-- 2 hadoop supergroup 14340 2015-06-10 06:32 /ceshi/ceshi.har/access.log

[hadoop@master ~]$ hadoop fs -cat har:///ceshi/ceshi.har/mini.txt

Warning: $HADOOP_HOME is deprecated.

hello world

2.1.3. mapreduce操作har

wordcount

```
package com.itcast.wordcount;

import java.io.IOException;
import java.util.StringTokenizer;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
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.FileOutputFor
mat;
import org.apache.hadoop.util.GenericOptionsParser;

public class WordCount {

public static class TokenizerMapper
extends Mapper<Object, Text, Text, IntWritable>{

private final static IntWritable one = new IntWritable(1);
private Text word = new Text();

public void map(Object key, Text value, Context context
) throws IOException, InterruptedException {
//value：一行一行数据，对数据做切分
String[] split = value.toString().split(" ");
//来源url
String url = split[6];
//输出
word.set(url);
context.write(word, one);
}
}

public static class IntSumReducer
extends Reducer<Text,IntWritable,Text,IntWritable> {
private IntWritable result = new IntWritable();

public void reduce(Text key, Iterable<IntWritable> values,
Context context
) throws IOException, InterruptedException {
//循环相加每一个int类型的值
int sum = 0;
for (IntWritable val : values) {
sum += val.get();
}
//设置url
result.set(sum);
context.write(key, result);
}
}

public static void main(String[] args) throws Exception {
Configuration conf = new Configuration();
conf.set("df.default.name", "hdfs://master:9000/");//设置hdfs的默
认路径
conf.set("hadoop.job.ugi", "hadoop,hadoop");//用户名，组
conf.set("mapred.job.tracker", "master:9001");//设置jobtracker在
哪
String[] otherArgs = { "har://hdfs-master:9000/ceshi/ceshi.har",//
输入文件路径
"hdfs://master:9000/ceshi/out9/" };//输入输出路径，可以设置，也
可以通过main方法传进来
// String[] otherArgs = new GenericOptionsParser(conf, args).ge
tRemainingArgs();//输入和输出参数
if (otherArgs.length != 2) {
System.err.println("Usage: wordcount <in> <out>");
System.exit(2);
}
Job job = new Job(conf, "word count");
job.setJarByClass(WordCount.class);
job.setMapperClass(TokenizerMapper.class);
job.setCombinerClass(IntSumReducer.class);
job.setReducerClass(IntSumReducer.class);
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(IntWritable.class);
FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));
System.exit(job.waitForCompletion(true) ? 0 : 1);
}
}
```

2.1.4. 注意事项

使用HAR时需要两点，第一，对小文件进行存档后，原文件并不会自动被删除，需要用户自己删除；第二，创建HAR文件的过程实际上是在运行一个mapreduce作业，因而需要有一个hadoop集群运行此命令。

此外，HAR还有一些缺陷：第一，一旦创建，Archives便不可改变。要增加或移除里面的文件，必须重新创建归档文件。第二，要归档的文件名中不能有空格，否则会抛出异常

2.2. Sequence file

sequence file由一系列的二进制key/value组成，如果为key小文件名，value为文件内容，则可以将大批小文件合并成一个大文件。

Hadoop中提供了SequenceFile，包括Writer，Reader和SequenceFileSorter类进行写，读和排序操作。

2.2.1. 序列化文件读写

```
package com.itcast.userdefined.sequence.makedata;

import java.io.IOException;
import java.io.InputStream;
import java.net.URI;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileStatus;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IOUtils;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.SequenceFile;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.Writable;
import org.apache.hadoop.util.ReflectionUtils;
import org.junit.Test;

public class SeqComSmallFile {
@Test
public void testWriter(){

Configuration conf = new Configuration();//创建配置信息
conf.set("fs.default.name", "hdfs://master:9000");// 不写走默认
conf.set("hadoop.job.ugi", "hadoop,hadoop");// 如果不写系统将
按照默认的用户进行操作
String url = "hdfs://master:9000/ceshi";//读取的文件
Path path = new Path("hdfs://master:9000/ceshi2/test.seq");//序
列化文件名
FileSystem fs = null;
InputStream is = null;
SequenceFile.Writer w = null;//创建writer流
Text k = new Text();//key，相当于string
Text v = new Text();//value，相当于String
try {
//代表一个统一资源标识符(URI)
fs = FileSystem.get(URI.create(url), conf);//创建filesystem
w = SequenceFile.createWriter(fs, conf, path, k.getClass(), v.get
Class());//创建writer
for (FileStatus a : fs.listStatus(new Path(url))) {
is = fs.open(a.getPath());//通过filesystem打开文件读取流
k.set(a.getPath().getName());
byte buf[] = new byte[1024*64];
int bytesRead = is.read(buf);
System.out.println();
v.set(new String(buf));
w.append(k, v);
IOUtils.copyBytes(is, System.out, 1024, false);
}

} catch (Exception e) {
e.printStackTrace();
}finally{
try {
IOUtils.closeStream(is);
fs.close();
} catch (IOException e1) {
fs=null;
e1.printStackTrace();
}
}

}
@Test
public void TestReader(){

Configuration conf = new Configuration();//创建配置信息
conf.set("fs.default.name", "hdfs://master:9000");//默认路径
conf.set("hadoop.job.ugi", "hadoop,hadoop");// 用户和组的信息
String uriin = "hdfs://master:9000/ceshi2/";//文件位置
FileSystem fs = null;
try {
fs = FileSystem.get(URI.create(uriin), conf);
} catch (IOException e2) {
e2.printStackTrace();
}//创建filesystem
Path path = new Path("hdfs://master:9000/ceshi/out1/part-r-000
00");//序列化文件的路径
SequenceFile.Reader r = null;
```

```
try {
r = new SequenceFile.Reader(fs, path, conf);//通过序列化文件类SequenceFile创建readerliu
Writable k = (Writable) ReflectionUtils.newInstance(r.getKeyClass(), conf);//创建key
Writable v = (Writable) ReflectionUtils.newInstance(r.getValueClass(), conf);//创建value
while (r.next(k, v)) {//顺序读取
System.out.println(k + "\n" + v);
}
} catch (Exception e) {
e.printStackTrace();
} finally {
IOUtils.closeStream(r);
try {
fs.close();
} catch (IOException e1) {
fs=null;
e1.printStackTrace();
}
}

}
}
```

2.2.2. mapreduce输入序列化文件

```
package com.itcast.userdefined.sequence.input;

import java.io.IOException;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
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.input.SequenceFileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
/**
* 处理序列化文件
* @author wilson
*
*/
public class SequenceFileInputFormatDemo {
public static class TokenizerMapper extends
Mapper<Text, Text, Text, IntWritable> {

private final static IntWritable one = new IntWritable(1);
private Text word = new Text();
//key为序列化文件的key，也就是文件的名字，value是序列化文件的value，也就是文件的内容
public void map(Text key, Text value, Context context)
throws IOException, InterruptedException {
// 用'\n'将value分割成每一行
String[] split = value.toString().split("\n");
for (String string : split) {
//用" "将每一行分割
String[] split2 = string.split(" ");
//有空字符串存在,判空
if (split2.length>=6) {
// 来源url
String url = split2[6];
// 设置、输出
word.set(url);
context.write(word, one);
}else {
word.set(string);
context.write(word, one);
}
}



}
}

public static class IntSumReducer extends
Reducer<Text, IntWritable, Text, IntWritable> {
private IntWritable result = new IntWritable();
```

```
public void reduce(Text key, Iterable<IntWritable> values,
Context context) throws IOException, InterruptedException {
// 循环相加每一个int类型的值
int sum = 0;
for (IntWritable val : values) {
sum += val.get();
}
// 设置url
result.set(sum);
context.write(key, result);
}
}

public static void main(String[] args) throws Exception {
Configuration conf = new Configuration();
conf.set("df.default.name", "hdfs://master:9000/");// 设置hdfs的
默认路径
conf.set("hadoop.job.ugi", "hadoop,hadoop");// 用户名，组
conf.set("mapred.job.tracker", "master:9001");// 设置jobtracker
在哪
String[] otherArgs = { "hdfs://master:9000/ceshi2/test.seq",// 输
入文件路径
"hdfs://master:9000/ceshi/out1/" };// 输入输出路径，可以设置，
也可以通过main方法传进来
// String[] otherArgs = new GenericOptionsParser(conf,
// args).getRemainingArgs();//输入和输出参数
Job job = new Job(conf, "sequence file input demo");
job.setJarByClass(SequenceFileInputFormatDemo.class);

job.setMapperClass(TokenizerMapper.class);
job.setReducerClass(IntSumReducer.class);
job.setNumReduceTasks(1);
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(IntWritable.class);
job.setMapOutputKeyClass(Text.class);
job.setMapOutputValueClass(IntWritable.class);
job.setInputFormatClass(SequenceFileInputFormat.class);
FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));
System.exit(job.waitForCompletion(true) ? 0 : 1);
}
}
```

2.2.3. mapreduce输出序列化文件

```
package com.itcast.userdefined.sequence.output;

import java.io.IOException;

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.SequenceFile.CompressionType;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.compress.DefaultCodec;
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.FileOutputFor
mat;
import org.apache.hadoop.mapreduce.lib.output.SequenceFile
OutputFormat;

public class SequenceOutputFormat {
public static class TokenizerMapper extends
Mapper<Object, Text, Text, IntWritable> {

private final static IntWritable one = new IntWritable(1);
private Text word = new Text();

public void map(Object key, Text value, Context context)
throws IOException, InterruptedException {
// value：一行一行数据，对数据做切分
String[] split = value.toString().split(" ");
// 来源url
String url = split[6];
// 输出
word.set(url);
context.write(word, one);
}
}
```

```
public static class IntSumReducer extends
Reducer<Text, IntWritable, Text, IntWritable> {
private IntWritable result = new IntWritable();

public void reduce(Text key, Iterable<IntWritable> values,
Context context) throws IOException, InterruptedException {
// 循环相加每一个int类型的值
int sum = 0;
for (IntWritable val : values) {
sum += val.get();
}
// 设置url
result.set(sum);
context.write(key, result);
}
}

public static void main(String[] args) throws Exception {
Configuration conf = new Configuration();
conf.set("df.default.name", "hdfs://master:9000/");// 设置hdfs的
默认路径
conf.set("hadoop.job.ugi", "hadoop,hadoop");// 用户名，组
conf.set("mapred.job.tracker", "master:9001");// 设置jobtracker
在哪
String[] otherArgs = { "hdfs://master:9000/ceshi/access.log",// 输
入文件路径
"hdfs://master:9000/ceshi/out3/" };// 输入输出路径，可以设置，
也可以通过main方法传进来
// String[] otherArgs = new GenericOptionsParser(conf,
// args).getRemainingArgs();//输入和输出参数
if (otherArgs.length != 2) {
System.err.println("Usage: wordcount <in> <out>");
System.exit(2);
}
Job job = new Job(conf, "sequence file output demo ");
job.setJarByClass(SequenceOutputFormat.class);
job.setMapperClass(TokenizerMapper.class);
job.setReducerClass(IntSumReducer.class);
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(IntWritable.class);
// 设置输出类
job.setOutputFormatClass(SequenceFileOutputFormat.class);
// 设置sequecnfile的格式，对于sequencefile的输出格式，有多种
组合方式,
// 从下面的模式中选择一种，并将其余的注释掉
// 组合方式1：不压缩模式
//SequenceFileOutputFormat.setOutputCompressionType(job,C
ompressionType.NONE);

// 组合方式2：record压缩模式(只压缩值)，
//并指定采用的压缩方式 ：默认:gzip压缩等

// SequenceFileOutputFormat.setOutputCompressionType(job,
// CompressionType.RECORD);
// SequenceFileOutputFormat.setOutputCompressorClass(job,
// DefaultCodec.class);


// 组合方式3：block压缩模式(压缩序列记录到一个块)，
//并指定采用的压缩方式 ：默认、gzip压缩等

// SequenceFileOutputFormat.setOutputCompressionType(job,
// CompressionType.BLOCK);
// SequenceFileOutputFormat.setOutputCompressorClass(job,
// DefaultCodec.class);

FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));
System.exit(job.waitForCompletion(true) ? 0 : 1);
}
}
```

2.3. CombineFileInputFormat

2.3.1. 介绍

CombineFileInputFormat是一种新的inputformat，用于将多个文件合并成一个单独的split，另外，它会考虑数据的存储位置。

CombineFileInputFormat的大致原理是，他会将输入多个数据文件（小文件）的元数据全部包装到CombineFileSplit类里面。也就是说，因为小文件的情况下，在HDFS中都是单Block的文件，即一个文件一个Block，一个CombineFileSplit包含了一组文件Block，包括每个文件的起始偏移（offset），长度（length），Block位置（localtions）等元数据。如果想要处理一个CombineFileSplit，很容易想

到，对其包含的每个InputSplit（实际上这里面没有这个，你需要读取一个小文件块的时候，需要构造一个FileInputSplit对象）。在执行MapReduce任务的时候，需要读取文件的文本行（简单一点是文本行，也可能是其他格式数据）。那么对于CombineFileSplit来说，你需要处理其包含的小文件Block，就要对应设置一个RecordReader，才能正确读取文件数据内容。通常情况下，我们有一批小文件，格式通常是相同的，只需要在为CombineFileSplit实现一个RecordReader的时候，内置另一个用来读取小文件Block的RecordReader，这样就能保证读取CombineFileSplit内部聚积的小文件。

2.3.2. 代码实现

mapreduce代码

```
package com.itcast.userdefined.CombineInputFormat;

import java.io.IOException;

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.FileOutputFor
mat;

public class CombineSmallfiles {
public static class TokenizerMapper extends
Mapper<LongWritable, Text, Text, IntWritable> {

private final static IntWritable one = new IntWritable(1);
private Text word = new Text();

public void map(LongWritable key, Text value, Context context)
throws IOException, InterruptedException {
// value：一行一行数据，对数据做切分
String[] split = value.toString().split(" ");
// 来源url
String url = split[6];
// 输出
word.set(url);
context.write(word, one);
}
}

public static class IntSumReducer extends
Reducer<Text, IntWritable, Text, IntWritable> {
private IntWritable result = new IntWritable();

public void reduce(Text key, Iterable<IntWritable> values,
Context context) throws IOException, InterruptedException {
// 循环相加每一个int类型的值
int sum = 0;
for (IntWritable val : values) {
sum += val.get();
}
// 设置url
result.set(sum);
context.write(key, result);
}
}

public static void main(String[] args) throws IOException,
ClassNotFoundException, InterruptedException {

Configuration conf = new Configuration();
conf.set("df.default.name", "hdfs://master:9000/");// 设置hdfs的
默认路径
conf.set("hadoop.job.ugi", "hadoop,hadoop");// 用户名，组
conf.set("mapred.job.tracker", "master:9001");// 设置jobtracker
在哪
/*
* 在同一个节点上的Blocks合并，超过maxSplitSize就生成新分片
。如果没有指定，则只汇总本节点BLock，暂不分片。
*/
conf.setLong("mapreduce.input.fileinputformat.split.maxsize",
64 * 1024 * 1024);
/*
* 把maxsize中处理剩余的Block，进行合并，如果超过minSizeNod
e，那么全部作为一个分片。
*/
conf.setLong("mapreduce.input.fileinputformat.split.minsize.per.
node",
```

```
1024);

// conf.setInt("mapred.min.split.size", 1);
// conf.setLong("mapred.max.split.size", 26214400); // 25m
String[] otherArgs = { "hdfs://master:9000/ceshi",// 输入文件路径
"hdfs://master:9000/ceshi/out1/" };// 输入输出路径，可以设置，
也可以通过main方法传进来
// String[] otherArgs = new GenericOptionsParser(conf,
// args).getRemainingArgs();//输入和输出参数
if (otherArgs.length != 2) {
System.err.println("Usage: wordcount <in> <out>");
System.exit(2);
}
Job job = new Job(conf, "combine smallfiles");
job.setJarByClass(CombineSmallfiles.class);
job.setMapperClass(TokenizerMapper.class);
job.setReducerClass(IntSumReducer.class);
job.setMapOutputKeyClass(Text.class);
job.setMapOutputValueClass(IntWritable.class);
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(IntWritable.class);
job.setInputFormatClass(CombineSmallfileInputFormat.class);
FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));
System.exit(job.waitForCompletion(true) ? 0 : 1);

}

}
```

inputformat代码

```
package com.itcast.userdefined.CombineInputFormat;

import java.io.IOException;

import org.apache.hadoop.io.BytesWritable;
import org.apache.hadoop.io.LongWritable;
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.CombineFileInput
tFormat;
import org.apache.hadoop.mapreduce.lib.input.CombineFileRec
ordReader;
import org.apache.hadoop.mapreduce.lib.input.CombineFileSplit
;
/**
* 继承CombineFileInputFormat，采用combineFileSplit，实现自
己的recordReader
* @author wilson
*
*/
public class CombineSmallfileInputFormat extends CombineFileI
nputFormat<LongWritable, Text> {

@Override
public RecordReader<LongWritable, Text> createRecordReade
r(InputSplit split, TaskAttemptContext context) throws IOExceptio
n {

CombineFileRecordReader<LongWritable, Text> recordReader
= new CombineFileRecordReader<LongWritable, Text>((Combi
neFileSplit) split, context, CombineSmallfileRecordReader.class)
;
try {
recordReader.initialize((CombineFileSplit) split, context);
} catch (InterruptedException e) {
new RuntimeException("Error to initialize CombineSmallfileRecor
dReader.");
}
return recordReader;
}

}
```

recordreader代码

```
package com.itcast.userdefined.CombineInputFormat;

import java.io.IOException;

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

```
import org.apache.hadoop.io.LongWritable;
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.CombineFileSplit
;
import org.apache.hadoop.mapreduce.lib.input.FileSplit;
import org.apache.hadoop.mapreduce.lib.input.LineRecordRea
der;

public class CombineSmallfileRecordReader extends RecordRe
ader<LongWritable, Text> {

private CombineFileSplit combineFileSplit;//分片
private LineRecordReader lineRecordReader = new LineRecord
Reader();//读文件
private Path[] paths;//路径
private int totalLength;//总长度
private int currentIndex;
private float currentProgress = 0;
private LongWritable currentKey;
private Text currentValue = new Text();;

public CombineSmallfileRecordReader(CombineFileSplit combin
eFileSplit, TaskAttemptContext context, Integer index) throws IO
Exception {
super();
this.combineFileSplit = combineFileSplit;
// 当前要处理的小文件Block在CombineFileSplit中的索引
this.currentIndex = index;
}
/**
* 循环初始化，此方法会被循环调用。
*/
@Override
public void initialize(InputSplit split, TaskAttemptContext context)
throws IOException, InterruptedException {
//分片
this.combineFileSplit = (CombineFileSplit) split;
//初始化每一片的io
FileSplit fileSplit = new FileSplit(combineFileSplit.getPath(current
Index), combineFileSplit.getOffset(currentIndex), combineFileSpl
it.getLength(currentIndex), combineFileSplit.getLocations());
lineRecordReader.initialize(fileSplit, context);
//获取所有路径
this.paths = combineFileSplit.getPaths();
//几个文件块，如果文件中包含大于64M的文件，算两个文件快，
如241M的文件还会被视为4个文件块
totalLength = paths.length;
//设置当前处理的文件的文件名
context.getConfiguration().set("map.input.file.name", combineFil
eSplit.getPath(currentIndex).getName());
}
@Override
public boolean nextKeyValue() throws IOException, InterruptedE
xception {
//如果当前文件的索引大于0，并且没有超过所有文件的总长度
if (currentIndex >= 0 && currentIndex < totalLength) {
//读一行，记录到lineRecordReader的成员变量key、value中
return lineRecordReader.nextKeyValue();
} else {
return false;
}
}
@Override
public LongWritable getCurrentKey() throws IOException, Interru
ptedException {
//返回之前读取的key
currentKey = lineRecordReader.getCurrentKey();
return currentKey;
}

@Override
public Text getCurrentValue() throws IOException, InterruptedEx
ception {
//返回之前读取的value
currentValue = lineRecordReader.getCurrentValue();
return currentValue;
}


/**
* 执行流程
*/
```

```
@Override
public float getProgress() throws IOException {
if (currentIndex >= 0 && currentIndex < totalLength) {
currentProgress = (float) currentIndex / totalLength;
return currentProgress;
}
return currentProgress;
}
/**
* 关闭流
*/
@Override
public void close() throws IOException {
lineRecordReader.close();
}
}
```