MapReduce3 InputFormat数据输入

输入文件路径指定

FileInputFormat类中提供了四种静态方法来设定Job的输入路径：

```
public static void addInputPath(Job job,Path path) throws IOException
public static void addInputPaths(Job job, String commaSeparatedPaths) throws
IOException
public static void setInputPaths(Job job, Path... inputPaths) throws IOException
public static void setInputPaths(Job job,String commaSeparatedPaths) throws
IOException
```

FileInputFormat切片机制

切片机制

- 简单按照文件的内容长度进行切片
- 切片大小，默认等于Block大小
- 切片时不考虑数据集整体，而是逐个针对每个文件进行单独的切片

案例分析

输入数据有两个文件

```
file1.txt      320M
file2.txt      10M
```

经过FileInputFormat的切片机制运算后，形成的切片信息如下：

```
file1.txt.split1-- 0~128
file1.txt.split2-- 128~256
file1.txt.split3-- 256~320
file2.txt.split1-- 0~10M
```

FileInputFormat切片大小的参数配置

源码中计算切片大小的公式：

```
Math.max(minSize, Math.min(maxSize, blockSize));
mapreduce.input.fileinputformat.split.minsize=1 默认值为1
mapreduce.input.fileinputformat.split.maxsize=Long.MAXValue  默认为Long.MAXValue
因此默认情况下，切片大小等于= block
```

切片大小设置

`maxsize`(切片最大值)：参数如果调得比`blockSize`小，则会让切片变小，而且就等于配置的这个参数的值

`minsize`(切片最小值)：参数调的比`blockSize`大，则可以让切片变得比`bolckSize`还大

获取切片信息API

```
//获取切片的文件名称
String name = inputSplit.getPath().getName();
//根据文件类型获取切片信息
FileSplit inputSplit = (FileSplit) context.getInputSplit();
```

FileInputFormat切片源码解析

- 程序先找到你数据存储的目录
- 开始遍历处理（规划切片）目录下的每一个文件
- 遍历第一个文件ss.txt
    - 获取文件大小fs.sizeOf(ss.txt)；
    - 计算切片大小

      computeSliteSize(Math.max(minSize,Math.min(maxSize, blocksize))) = blocksize = 128M
    - 默认情况下，切片大小=blocksize
    - 开始切，形成第1个切片：ss.txt->0:128M、第2个切片：ss.txt->128:256M、第3个切片：ss.txt->256:300M(每次切片时，都要判断剩下的部分是否大于块的1.1倍，不大于1.1倍就划分一块切片)
    - 将切片信息写到一个切片规划文件中
    - 整个切片的核心过程在getSplit()方法中
    - InputSplit只记录切片的元数据信息，比如起始位置、长度以及所在的节点列表
- 提交切片规划文件到YARN，YARN上的MrAppMaster就可以根据切片规划文件计算开启MapTask个数

FileInputFormat实现类

Map reduce任务的输入文件一般是存储在HDFS里面，输入的文件格式包括：

> 基于行的日志文件、二进制格式文件等。这些文件一般会很大，达到数十GB，甚至更大。那么Mapreduce是如何读取这些数据的呢?

FileInputFormat常见实现类

> 包括：TextInputFormat、Key value TextInput Format、 NlineInput Format、Combine、TextInputFormat和自定义InputFormat等

TextInputFormat切片机制

TextInput Format是默认的InputFormat

> 每条记录是一行输入，键是Long Writable类型，存储改行在整个文件中的起始字节偏移量。值是这行的内容，不包括任何终止符（换行符和回车符）

以下是一个示例，比如，一个分片包含了如下4条文本记录

```
Rich learning form
Intelligent learning engine
Learning more convenient
From the real demand for more close to enterprise

每条记录表示为以下键/值对：
(0, Rich learning form)
(19, Intelligent learning engine)
(47, Learning more convenient)
(72, Form the real demand for more close to the enterprise)
键并不是行号，因为文件是按照字节而不是行划分的
```

KeyValueTextInputFormat切片机制

> 每一行均一条记录，被分隔符分隔为key、value。可以通过在驱动类中设置conf.set(KeyValueLineRecordReader.KEY_VALUE_SEPERATOR, '\t')来设置分隔符，默认分隔符是tab(\t)；

```
以下是一个示例，输入是一个包含4条记录的分片，其中->表示一个(水平方向的)制表符
line1->Rich Learning form
line2->Intelling learning engine
line3->Learning more convenient
line4->From the real demand for more close to the enterprise

每条记录表示为以下键/值对
(line1,  Rich Learning form)
(line2,  Intelling learning engine)
(line3,  Learning more convenient)
(line4,  From the real demand for more close to the enterprise)
此时的键是排在制表符之前的Text序列
```

NLineInputFormat切片机制

> 如果使用NLineInputFormat，代表每个map进程处理的InputSplit不再按Bolck去分块，而是按NLineInputFormat指定的行数N来划分，即输入文件的总行数N=切片数，如果不整除，切片数=商+1

```
以下是一个示例：
Rich learning form
Intelligent learning engine
Learning more convenient
From the real demand for more close to enterprise

例如，如果N是2，则每个输入分片包含两行，开启2个MapTask
一个map接收
(0, Rich learning form)
(19, Intelligent learning engine)

另一个map接收
(47, Learning more convenient)
(72, Form the real demand for more close to the enterprise)
```

CombineTextInputFormat切片机制

问题背景：

> 框架默认的TextInputFormat切片机制是对任务按文件规划切片，不管文件多小，都会是一个单独的切片，都会交给一个MapTask，这样如果有大量小文件，就会产生大量的MapTask，处理效率极其低下

应用场景：

> CombineTextInputFormat用于小文件过多的场景，它可以将多个小文件从逻辑上规划到一个切片中，这样，多个小文件就可以交给一个MapTask处理

虚拟存储切片最大值设置

```
job.setInputFormatClass(CombineTextInputFormat.class)
CombineTextInputFormat.setMaxInputSplitSize(job, 4194304);
//4m虚拟存储切片最大值设置可以根据实际文件大小情况来设置具体的值
CombineTextInputFormat.setMinInputSplitSize(job, 2097152);// 2m

// 设置最大值即可 128M
```

切片机制

生成切片的过程包括：虚拟存储过程和切片过程两个部分

```
虚拟存储过程：
(1) 将输入目录下所有文件按找文件名称字典顺序依次读入，记录文件大小，并累加计算所有文件的总长度。
(2) 根据是否设置setMaxInputSplitSize值，将每个文件划分城一个个setMaxInputSplitSize值大小的文件
(3) 注意，当剩余数据大小超过setMaxInputSplitSize值且不大于两倍的setMaxInputSplitSize，此使将文件均分为两个虚拟存储块（防止出现切片太小）
大于怎么办
例如setMaxInputSplitSize值为4M，最后文件剩余大小为4.02M，如果按照4M逻辑划分，就会出现0.02M的小的虚拟存储文件，所讲剩余的4.02M文件切分成(2.01M和2.01M)两个文件
```

```
切片过程：
(1) 判断虚拟存储的文件大小是否大于setMaxInputSplitSize值，大于等于则单独形成一个切片
(2) 如果不大于根下一个虚拟存储进行合并，共同形成一个切片
(3) 例子：有4个小文件大小分别为1.7M,5.1M,3.4M以及6.8M这四个小文件，则虚拟存储之后形成6个文件块，大小分别为
1.7M，(2.55M，2.55M)，3.4,(3.4M和3.4M)
最终会相册3个切片
(1.7+2.55)M、(2.55+3.4M)、(3.4+3.4)M
```

自定义InputFormat

> 在企业开发中，Hadoop框架自带的InputFormat类型不能满足所有应用场景，需要自定义InputFormat来解决实际问题

自定义InputFormat步骤如下：

1. 自定义一个类继承FileInputFormat
2. 改写RecordReader，实现一次读取一个完整文件封装为KV
3. 在输出的使用SwquenceFileOutPutFormat输出合并输出文件

TextInputFormat实例

这里使用wordCount程序，故map和reduce编程就不写出来啦，主要是如何设置TextInputFormat

```
package com.ky;

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.mapreduce.Job;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;


public class WordCount {

    public static void main(String[] args) throws IOException,
ClassNotFoundException,InterruptedException{
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf);

        job.setJarByClass(WordCount.class);
        job.setMapperClass(TokenizerMapper.class);
        job.setCombinerClass(IntSumReducer.class);
        job.setReducerClass(IntSumReducer.class);

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

        //设置InputFormat，默认为TextInputFormat.class
        job.setInputFormatClass(TextInputFormat.class);
        TextInputFormat.setMinInputSplitSize(job, 1);
        TextInputFormat.setMaxInputSplitSize(job, 1024 * 1024 * 128);

        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        job.waitForCompletion(true);
    }
}
```

KeyValueTextInputFormat实例

首先看一下数据文件（空格为分隔符）

```
hello you
hello me
hello you
hello me
```

```
package com.ky;

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.mapreduce.Job;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class KVTextDriver {

    public static void main(String[] args) throws IOException,
ClassNotFoundException,InterruptedException{
        Configuration conf = new Configuration();
        //设置分隔符的类型
        conf.set(KeyValueLineRecordReader.KEY_VALUE_SEPERATOR,2)
        Job job = Job.getInastance(conf);
        //使用KeyValueTextInputFormat的输入类型
        job.setInputFormatClass(KeyValueTextInputFormat.class);

        job.setJarByClass(KVTextDriver.class);
        job.setMapperClass(KVTextMapper.class);
        job.setReducerClass(KVTextRedcuer.class);

        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(LongWritable.class);

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

        FileInputFormat.setInputPaths(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        job.waitForCompletion(true);
    }
}
```

NLineInputFormat实例

首先看一下数据文件（空格为分隔符）

```
hadoop hello
hadoop me
hadoop java
```

```
package com.ky;

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.mapreduce.Job;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class NLInputDriver {

    public static void main(String[] args) throws IOException,
ClassNotFoundException,InterruptedException{
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf);
        //设置NLineInputFormat行数为2
        NLineInputFormat.setNumLinePerSplit(job, Integer.parseInt("2"));
        job.setInputFormatClasee(NLineInputFormat.class);

        job.setJarByClass(NLInputDriver.class);
        job.setMapperClass(NLMapper.class);
        job.setReducerClass(NLReducer.class);

        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(LongWritable.class);

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

        FileInputFormat.setInputPaths(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        job.waitForCompletion(true);
    }
}
```

CombineTextInputFormat实例

查看一下文件信息：

```
-rw-r--r--  1 ylj  staff   176K  4  7 15:35 flie1.txt
-rw-r--r--  1 ylj  staff   879K  4  7 15:39 flie2.txt
-rw-r--r--  1 ylj  staff   1.1M  4  7 15:39 flie3.txt
-rw-r--r--  1 ylj  staff   2.1M  4  7 15:41 flie4.txt
-rw-r--r--  1 ylj  staff   3.3M  4  7 15:42 flie5.txt
```

不设置CombineTextInputFormat允许所形成的分片为5

仅设置CombineTextInputFormat时

```
//设置InputFormat
job.setInputFormatClass(CombineTextInputFormat.class);
```

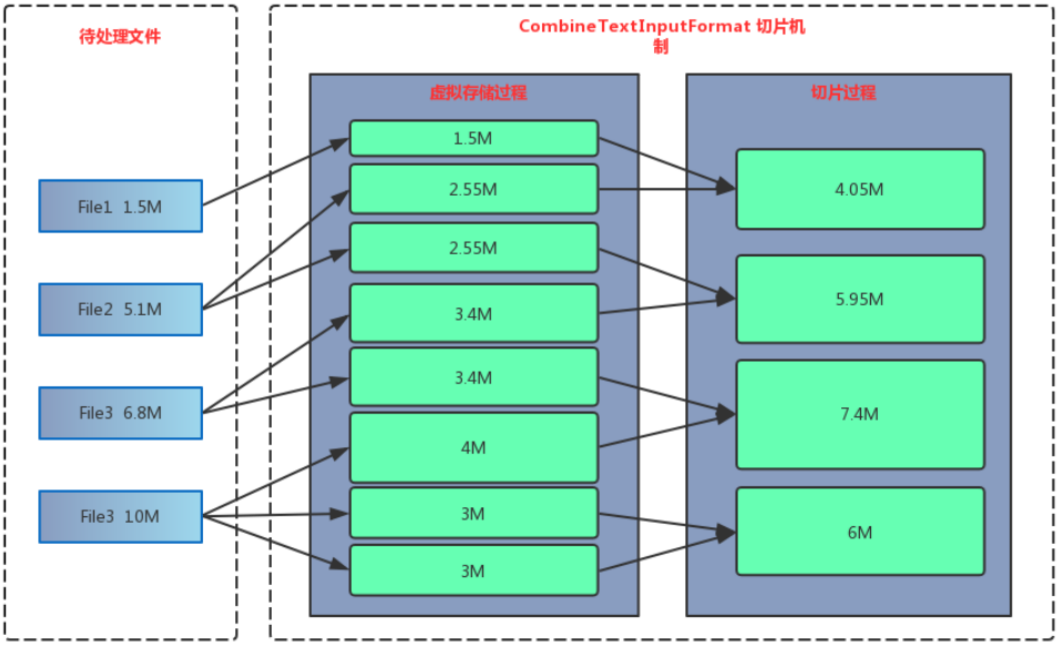
此使的配置如下：

```
SPLIT_MAXSIZE = Long.MAXValue
SPLIT_MINSIZE = 1
```

所以此时分区为 1（因为所有文件相加都在中间）

设置SPLIT_MAXSIZE、SPLIT_MINSIZE

```
//设置InputFormat
job.setInputFormatClass(CombineTextInputFormat.class);
CombineTextInputFormat.setMaxInputSplitSize(job, 4194304);// 4m
CombineTextInputFormat.setMinInputSplitSize(job, 2097152);// 2m
```



此使切片为数量为2

```
package com.ky;

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.mapreduce.Job;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class NLInputDriver {

    public static void main(String[] args) throws IOException,
ClassNotFoundException,InterruptedException{
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf);

        //设置CombineTextInputFormat格式，最大为4M，最小为2M
        job.setInputFormatClass(CombineTextInputFormat.class);
        CombineTextInputFormat.setMaxInputSplitSize(job, 4194304);// 4m
        CombineTextInputFormat.setMinInputSplitSize(job, 2097152);// 2m

        job.setJarByClass(NLInputDriver.class);
        job.setMapperClass(NLMapper.class);
```

```
        job.setReducerClass(NLReducer.class);

        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(LongWritable.class);

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

        FileInputFormat.setInputPaths(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        job.waitForCompletion(true);
    }
}
```

自定义InputFormat实例

> 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。
>
> 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

实例：多文件合并

1. 自定义一个类继承FileInputFormat
2. 改写RecordReader，实现一次读取一个完整文件封装为KV
3. 在输出的使用SwquenceFileOutPutFormat输出合并输出文件

自定义InputFormat

```
public class Custom_FileInputFormat extends FileInputFormat<NullWritable,
BytesWritable){
    /*
    直接返回整个文件不可分割，保证一个文件是完整的一行
    */
    @Override
    protected boolean isSpiltable(JobContext context, Path filename){
        return false;
    }

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

自定义RecordReader类

```
/*
```

```
 * RecordReader核心工作逻辑
 * 通过nextKeyValue()方法读取数据结构将返回的Key vlaue
 * 通过getCurrentKey 和 getCurrentValue来返回上面构造好的key 和value
 */

public class Custom_RecordReader extends RecordReader<NullWritable,
BytesWritable>{
    private FileSplit fileSplit;
    private Configuration conf;
    private BytesWritable bytesWritable = new BytesWritable();
    private boolean pressced = false;

    /*
     *初始化
     *@param split 封装的文件对象
     *@param context 上下文对象
     *@throw IOException
     *@throw InterruptedException
     */
    @Override
    public void initable(InputSplit split, TaskAttemptContext context) throws
IOException,InterruptedException{
        this.fileSplit = (FileSplit) split;
        this.conf = context.getConfiguration();
    }

    //读取下个文件(这里修改)
    public boolean nextKeyValue() throws throws IOException,
InterruptedException {
        if(!pressced){
            //获取文件路径
            Path path = fileSplit.getPath();

            //获取FileSystem对象
            FileSystem fileSystem = null;
            FSDataInputStream inputStream = null;
            try{
                fileSystem = FileSystem.get(conf);
                //读取文件
                inputStream = fileSystem.open(path);
                //初始化一个字节数据，大小为文件的长度
                byte[] bytes = new byte[(int)fileSplit.getLength()];
                //把数据流转化为字节数组
                IOUtils.readFully(inputStream, bytes, 0, bytes.length);

                //把字节数组转化为bytesWritable对象
                bytesWritable.set(bytes, 0, bytes.length);
            }catch(Exception e){
                e.printStackTrace();
            }finally{
                fileSystem.close();
                if(null != inputStream){
                    inputStream.close();
                }
            }
            pressced = true;
            return true;
        }else{
```

```
            return false;
        }

    }

    //获取当前的key值
    @Override
    public NullWritable getCurrentKey() throws IOException, InterruptedException
{
        return NullWritable.get();

    }

    //获取当前的value值
    @Override
    public BytesWritable getCurrentValue() throws IOException,
InterruptedException {
        return bytesWritable;
    }

    //获取当前的进程
    @Override
    public float getProgress() throws IOException, InterruptedException {
        return pressced?0:1;
    }

    //关流
    @Override
    public void close() throws IOException {


    }
}
```

map处理

```
public class Custom_Mapper extends Mapper<NullWritable, BytesWritable, Text,
BytesWritable> {


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

        FileSplit fileSplit = (FileSplit)context.getInputSplit();
        String name = fileSplit.getPath().getName();
        context.write(new Text(name),value);

    }
}
```

设置处理流程

```
public class Customer_Driver {

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

```
        //1.实例化job对象
        Job job = Job.getInstance(new Configuration(), "Customer_Driver");
        job.setJarByClass(Customer_Driver.class);

        //2.设置输入(这里做修改)
        job.setInputFormatClass(Custom_FileInputFormat.class);
        Custom_FileInputFormat.addInputPath(job,new Path(args[0]));

        //3.设置Map
        job.setMapperClass(Custom_Mapper.class);
        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(BytesWritable.class);

        //4.设置Reduce()
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(BytesWritable.class);

        //5.设置输出
        job.setOutputFormatClass(SequenceFileOutputFormat.class);
        //使用SequenceFileOutFoemat合并输出
        SequenceFileOutputFormat.setOutputPath(job,new Path(args[1]));

        boolean b = job.waitForCompletion(true);
        System.out.println(b?0:1);

    }
}
```