**FullFileInputFormat.java**

**import** java.io.IOException;

**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;

**public** **class** FullFileInputFormat **extends**

FileInputFormat<NullWritable, BytesWritable> {

@Override

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

**return** **false**;

}

@Override

**public** RecordReader<NullWritable, BytesWritable> createRecordReader(

InputSplit split, TaskAttemptContext context) **throws** IOException, InterruptedException {

FullFileRecordReader reader = **new** FullFileRecordReader();

reader.initialize(split, context);

**return** reader;

}

}

**FullFileRecordReader.java**

**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.BytesWritable;

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

**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;

**class** FullFileRecordReader **extends** RecordReader<NullWritable, BytesWritable> {

**private** FileSplit fileSplit;

**private** Configuration conf;

**private** BytesWritable value = **new** BytesWritable();

**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);

IOUtils.*readFully*(in, contents, 0, contents.length);

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** BytesWritable 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

}

}

……………………………………………………………………….

**SequenceFileKeyExtractor.java**

**import** org.apache.hadoop.conf.Configuration;

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

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

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

**import** org.apache.hadoop.io.SequenceFile.Reader;

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

**public** **class** SequenceFileKeyExtractor {

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

Configuration conf = **new** Configuration();

Path path = **new** Path(args[0]);

SequenceFile.Reader reader = **null**;

**try** {

reader = **new** SequenceFile.Reader(conf, Reader.*file*(path));

Text key = **new** Text();

**while** (reader.next(key)) { System.*out*.println(key);

}

} **finally** {

IOUtils.*closeStream*(reader);

}

}

}

SmallFilesToSequenceFile.java

**import** java.io.IOException;

**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.NullWritable;

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

**import** org.apache.hadoop.mapreduce.InputSplit;

**import** org.apache.hadoop.mapreduce.Mapper;

**import** org.apache.hadoop.mapreduce.Job;

**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.mapreduce.lib.output.SequenceFileOutputFormat;

**import** org.apache.hadoop.util.Tool;

**import** org.apache.hadoop.util.ToolRunner;

**public** **class** SmallFilesToSequenceFile **extends** Configured **implements** Tool {

**static** **class** SequenceFileMapper **extends**

Mapper<NullWritable, BytesWritable, Text, BytesWritable> {

**private** Text filename;

@Override

**protected** **void** setup(Context context) **throws** IOException,

InterruptedException {

InputSplit split = context.getInputSplit();

Path path = ((FileSplit) split).getPath();

filename = **new** Text(path.toString());

}

@Override

**protected** **void** map(NullWritable key, BytesWritable value,

Context context) **throws** IOException, InterruptedException {

context.write(filename, value);

}

}

@Override

**public** **int** run(String[] args) **throws** Exception {

Configuration conf = **new** Configuration();

Job job = Job.*getInstance*(conf);

job.setJarByClass(SmallFilesToSequenceFile.**class**);

job.setJobName("smallfilestoseqfile");

job.setInputFormatClass(FullFileInputFormat.**class**);

job.setOutputFormatClass(SequenceFileOutputFormat.**class**);

job.setNumReduceTasks(1);

FileInputFormat.*setInputPaths*(job, **new** Path(args[0]));

FileOutputFormat.*setOutputPath*(job, **new** Path(args[1]));

job.setOutputKeyClass(Text.**class**);

job.setOutputValueClass(BytesWritable.**class**);

job.setMapperClass(SequenceFileMapper.**class**);

**return** job.waitForCompletion(**true**) ? 0 : 1;

}

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

**int** exitCode = ToolRunner.*run*(**new** SmallFilesToSequenceFile(), args);

System.*exit*(exitCode);

}

}