2017.09

Hadoop离线大数据分析

MapReduce类型与格式

Mapreduce map输入输出格式

```
public class Mapper<KEYIN, VALUEIN, KEYOUT, VALUEOUT> {

  public class Context extends MapContext<KEYIN, VALUEIN, KEYOUT, VALUEOUT> {
    // ...
  }

  protected void map(KEYIN key, VALUEIN value,
                     Context context) throws IOException, InterruptedException {
    // ...
  }
}
```

Mapreduce reduce输入输出格式

```
public class Reducer<KEYIN, VALUEIN, KEYOUT, VALUEOUT> {

  public class Context extends ReducerContext<KEYIN, VALUEIN, KEYOUT, VALUEOUT> {
    // ...
  }

  protected void reduce(KEYIN key, Iterable<VALUEIN> values, Context context

                        Context context) throws IOException, InterruptedException {
    // ...
  }
}
```

Mapreduce combiner输入输出格式

```
map: (K1, V1) → list(K2, V2)
combine: (K2, list(V2)) → list(K2, V2)
reduce: (K2, list(V2)) → list(K3, V3)
```

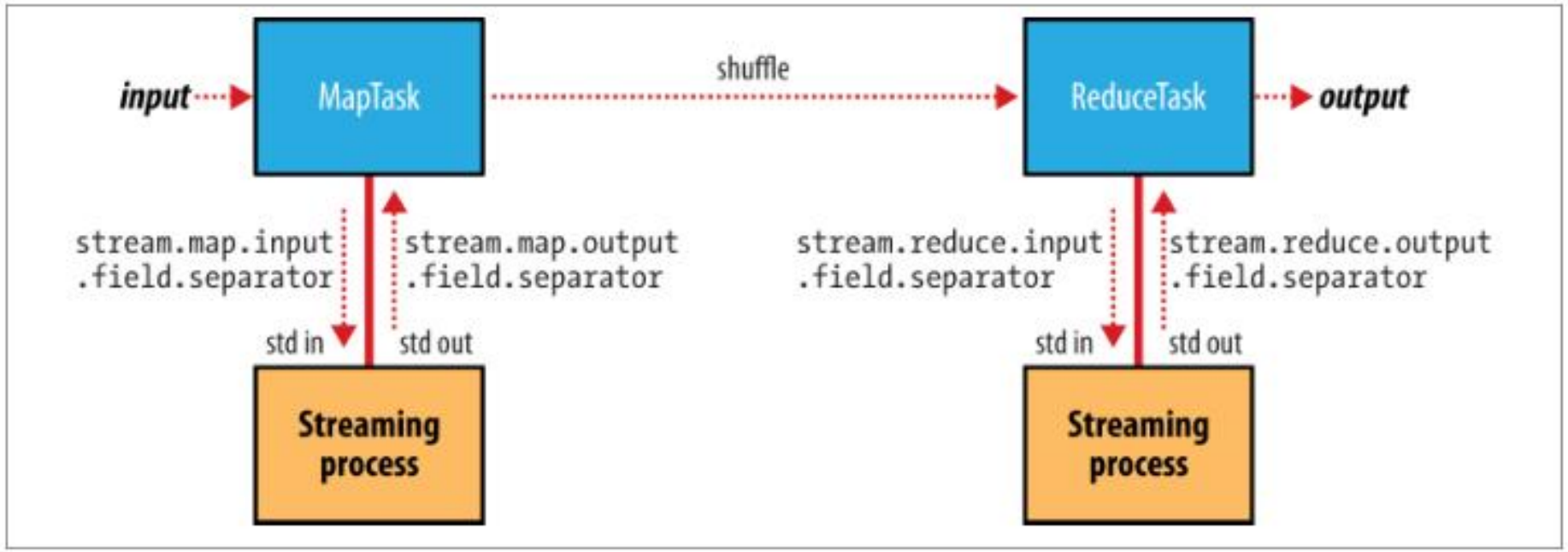
缺省Mapreduce 作业

```
public class MinimalMapReduce extends Configured implements Tool {

  @Override
  public int run(String[] args) throws Exception {
    if (args.length != 2) {
      System.err.printf("Usage: %s [generic options] <input> <output>\n
          getClass().getSimpleName());
      ToolRunner.printGenericCommandUsage(System.err);
      return -1;
    }

    Job job = new Job(getConf());
    job.setJarByClass(getClass());
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    return job.waitForCompletion(true) ? 0 : 1;
  }

  public static void main(String[] args) throws Exception {
    int exitCode = ToolRunner.run(new MinimalMapReduce(), args);
    System.exit(exitCode);
  }
}
```

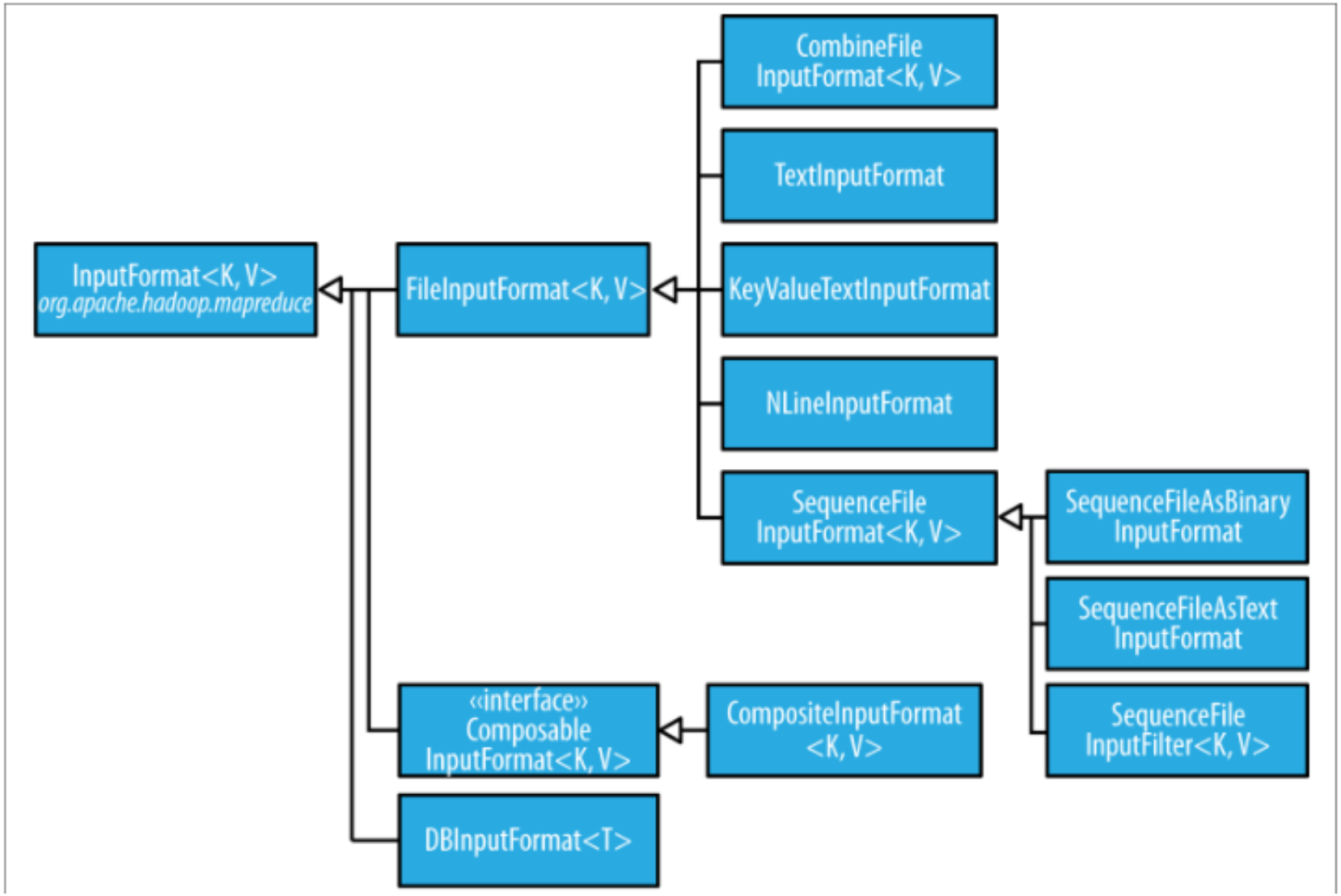
7 MR类型与格式

```java
public class MinimalMapReduceWithDefaults extends Configured implements Tool {

  @Override
  public int run(String[] args) throws Exception {
    Job job = JobBuilder.parseInputAndOutput(this, getConf(), args);
    if (job == null) {
      return -1;
    }

    job.setInputFormatClass(TextInputFormat.class);

    job.setMapperClass(Mapper.class);

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

    job.setPartitionerClass(HashPartitioner.class);

    job.setNumReduceTasks(1);
    job.setReducerClass(Reducer.class);

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

    job.setOutputFormatClass(TextOutputFormat.class);

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

  public static void main(String[] args) throws Exception {
    int exitCode = ToolRunner.run(new MinimalMapReduceWithDefaults(), args);
    System.exit(exitCode);
  }
}
```

Mapreduce 流作业



FileInputFormat



FileSplit

FileSplit method	Property name	Type	Description
`getPath()`	`mapreduce.map.input.file`	`Path/String`	The path of the input file being processed
`getStart()`	`mapreduce.map.input.start`	`long`	The byte offset of the start of the split from the beginning of the file
`getLength()`	`mapreduce.map.input.length`	`long`	The length of the split in bytes

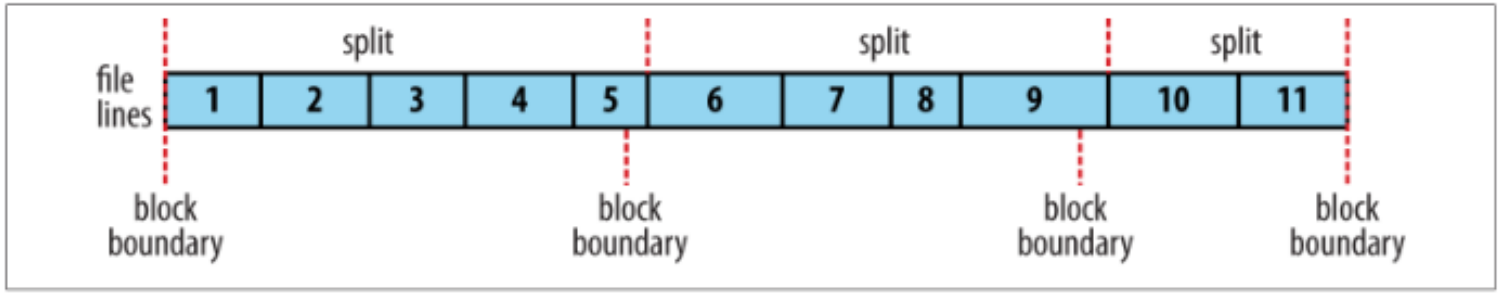


Figure 8-3. Logical records and HDFS blocks for TextInputFormat

WholeFileInputFormat

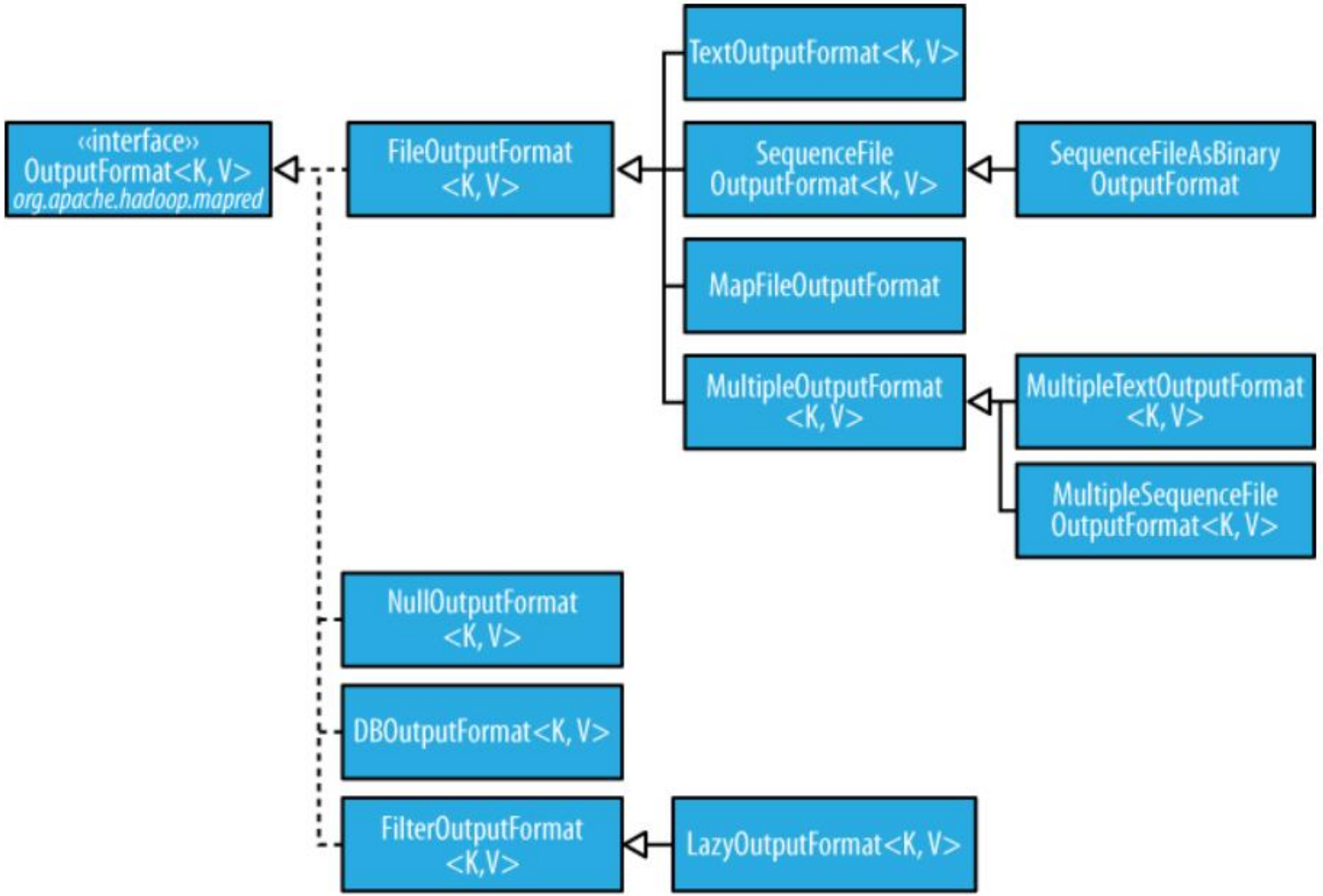
```
public class WholeFileInputFormat
    extends FileInputFormat<NullWritable, BytesWritable> {

  @Override
  protected boolean isSplitable(JobContext context, Path file) {
    return false;
  }

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

```
class WholeFileRecordReader 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, Interrupte
    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
  }
}
```

FileInputFormat输入路径

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

OutputFormat



2017.08

THE END