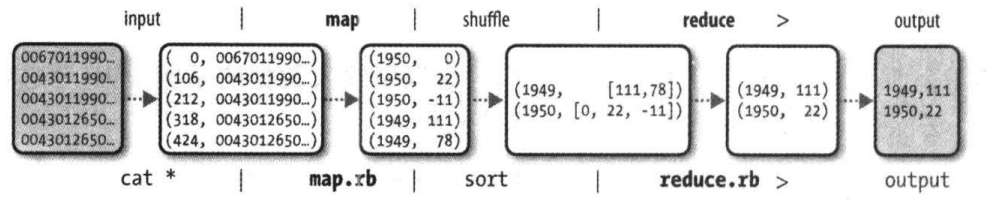
# 2.1 MapReduce类型

1.MapReduce数据处理模型非常简单：map和reduce函数的输入和输出是键值对。

2.Hadoop的MapReduce中，map函数和reduce函数遵循如下常规格式：

map: (K1, V1) -> list(K2, V2)reduce: (K2, list(V2)) -> list(K3, V3)

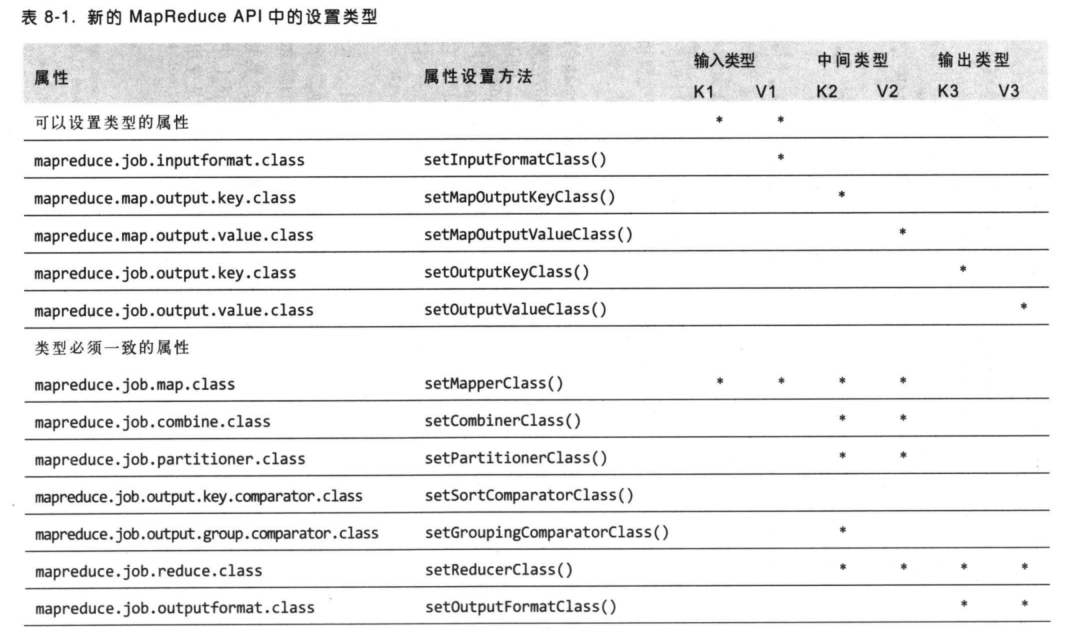


如果定义了**combiner函数，每个map输出数据会发生预聚合**，**最后combiner输出到reduce统一聚合**。一般来讲它与reduce函数(是Reducer的一个实现)的形式相同，不同之处是它输出类型是中间的键-值对类型(K2和V2)，这些中间值可以输入reduce函数：

map: (K1, V1) -> list(K2, V2)combiner: (K2, list(V2)) -> (K2, V2)reduce: (K2, list(V2)) -> (K3, V3)

3.一般来说，**map函数输入的键/值类型(K1 和V1)不同于输出类型(K2和V2)**。**然而，reduce函数的输入类型必须与map函数的输出类型相同**，但reduce 函数的输出类型(K3和V3)可以不同于输入类型。

4.partition函数对中间结果的键值对(K2 和 V2)进行处理，并且返回一个分区索引。实际上，分区由键单独决定(值被忽略)。



5.默认的partitioner是HashPartitioner，它对每条记录的键进行哈希操作以决定以决定该**记录应该属于哪个分区**。**每个分区由一个reduce任务处理，所以分区数等于作业的reduce任务个数**。

6.键的哈希码被转换为一个非负整数，它由哈希值与最大的整型值做一次按位与操作而获得，然后用分区数进行数模操作，来决定该记录属于哪个分区索引。

7.**默认情况下，只有一个reducer**，因此，**也就只有一个分区**，在这种情况下，由于所有数据都放入同一个分区，**partitioner操作将变得无关紧要了**。然而，如果有多个reduce任务，了解 HashPartitioner的作用就非常重要*。*假设基于键的散列函数足够好，那么记录将被均匀分到若干个reduce任务中，这样，具有相同键的记录将由同一个reduce任务进行处理。

8.**map任务数量并没有手动设置**。**原因是该数量等于输入文件被划分成的分块数，这取决于输入文件大小以及文件块的大小**（如果此文件块在HDFS中）。

9.选择reducer的个数：

对 Hadoop新手而言，单个 reducer 的默认配置很容易上手。但**在真实的应用中，几乎所有作业都把它设置成一个较大的数字**，**否则由于所有的中间数据都会放到一个reduce任务中,作业处理极其低效**。

为一个作业选择多少个reducer与其说是一门技术,不如说更多是一门艺术。由于并行化程度提高，增加reducer 的数量能缩短reduce过程。然而,如果做过了,小文件将会更多，这又不够优化。一条经验法则是，目标reducer保持在每个运行5分钟左右、且产生至少一个 HDFS块的输出比较合适。

# 2.2 输入格式

## 2.2.1 输入分片和记录

1.**一个输入分片(split)就是一个由单个map操作来处理的输入块**。**每一个map操作只处理一个输入分片**。**每个分片被划分为若干个记录，每条记录就是一个键-值对，map一个接一个地处理记录**。输入分片和记录都是逻辑概念，不必将它们对应到文件，尽管其常见形式都是文件。在数据库的场景中，一**个输入分片可以对应于一个表上的若干行**，**而一条记录对应到一行**(如同DBInputFormat，这种输入格式用于从关系型数据库读取数据)。

2.输入分片在Java中表示为InputSplit接口。**InputSplit包含一个以字节为单位的长度和一组存储位置(即一组主机名)**。注意，分片并不包含数据本身，而是指向数据的引用(reference)。存储位置供MapReduce系统使用以便将map任务尽量放在分片数据附近，而分片大小用来排序分片，以便优先处理最大的分片，从而最小化作业运行时间(这也是贪婪近似算法的一个实例)。

3.MapReduce应用开发人员不必直接处理InputSplit。因为它是InputFormat创建的(InputFormat负责创建输入分片并将它们分割成记录)。

4.**运行作业的客户端通过调用getSplits()计算分片**，**然后将它们发送到application master**，application master使用其**存储位置信息来调度map任务从而在集群上处理这些分片数据**。map任务把输入分片传给InputFormat的createRecordReader方法来获得这个分片的RecordReader。**RecordReader就像是记录上的迭代器**，**map任务用一个RecordReader来生成记录的键-值对**，**然后再传递给map函数**。查看Mapper的run()方法可以看到这些情况。

5.运行setup()之后，**再重复调用Context上的nextKeyvalue()**(委托给RecordReader的同名方法)为**mapper产生键-值对象**。通过Context，键/值从RecordReader中被检索出并传递给map()方法。当reader读到stream 的结尾时，nextKeyvalue()方法返回false，map任务运行其cleanup()方法，然后结束。

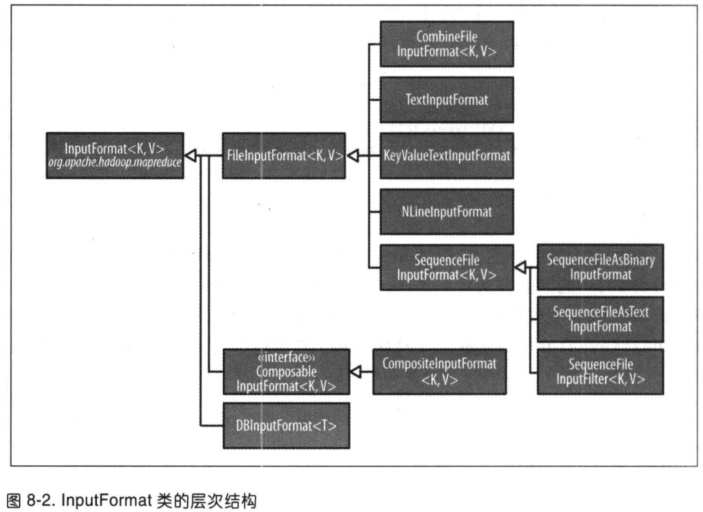
6.最后，**注意Mapper的run()方法是公共的**，**可以由用户定制**。MultithreadedMapRunner是另一个MapRunnable接口的实现，它可以使用可配置个数的线程来并发运行多个mapper(用mapreduce.mapper. multithreadedmapper.threads设置)。对于大多数数据处理任务来说，默认的执行机制没有优势。但是，对于因为需要连接外部服务器而造成单个记录处理时间比较长的 mapper来说，它允许多个mapper在同一个JVM下以尽量避免竞争的方式执行。

### FileInputFormat类

1.FileInputFormat**是所有使用文件作为其数据源的InputFormat 实现的基类**(参见图8-2)。它提供两个功能：

* **一个用于指出作业的输入文件位置**
* **一个是为输入文件生成分片的代码实现**。

把分片分割成记录的作业由其子类来完成。



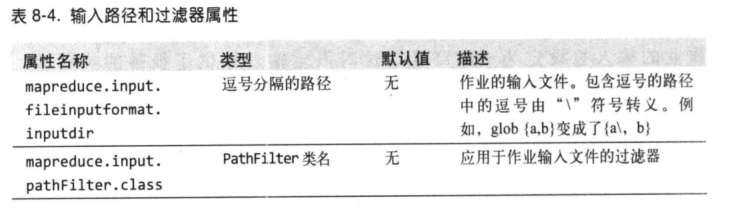
2.作业的输入被设定为一组路径，这对限定输入提供了很强的灵活性。FileInputFormat提供四种静态方法来设定Job的输入路径。

3.其中，**addInputPath()和 addInputPaths()方法可以将一个或多个路径加入路径列表**。可以分别调用这两种方法来建立路径列表。**setInputPaths()方法一次设定完整的路径列表**(替换前面调用中在Job 上所设置的所有路径)。

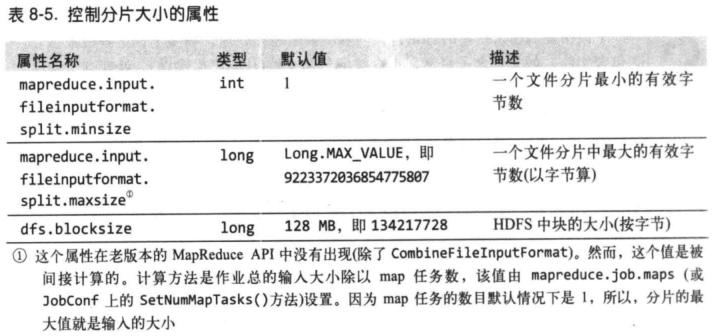
4.**一条路径可以表示一个文件、一个目录或是一个glob**，**即一个文件和目录的集合**。路径是目录的话，表示要包含这个目录下所有的文件，这些文件都作为作业的输入。

5.**add方法和 set方法允许指定包含的文件**。如果需要**排除特定文件，可以使用FileInputFormat的 setInputPathFilter()方法设置一个过滤器**。

6.**即使不设置过滤器，FileInputFormat也会使用一个默认的过滤器来排除隐藏文件**(名称中以“.”和“\_”开头的文件)。**如果通过调用setInputPathFilter()设置了过滤器，它会在默认过滤器的基础上进行过滤。换句话说，自定义的过滤器只能看到非隐藏文件**。



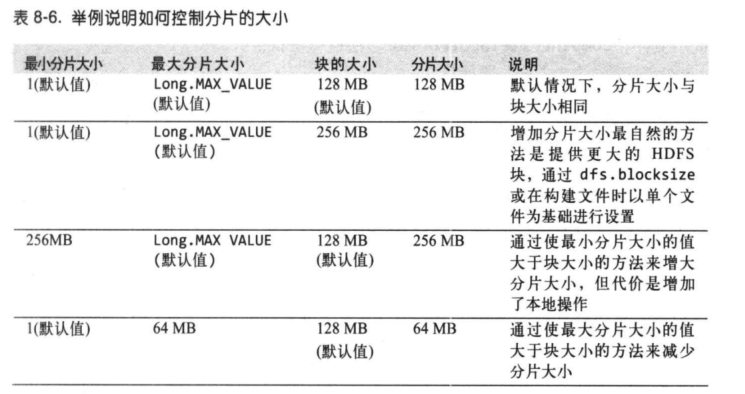
7.FileInputFormat只分割大文件。这里大文件指的是超过HDFS块的大小。分块通常与HDFS块大小一样，这在大多应用都是合理的；然而，这个值也可以通过设置不同的Hadoop属性来改变。



8.**最小的分片大小通常是1个字节，不过某些格式可以使分片大小有一个更低的下界。**例如，顺序文件在流中每次插入一个同步入口，所以，最小的分片大小不得不足够大以确保每个分片有一个同步点，以便reader 根据记录边界进行重新同步。

9.**应用程序可以强制设置一个最小的输入分片大小：通过设置一个比 HDFS块更大一些的值**，**强制分片比文件块大**。如果数据存储在 HDFS 上，那么这样做是没有好处的，因为这样做会增加对map任务来说不是本地文件的文件块数。

10.最大的分片大小默认是由Java 的 long类型表示的最大值。只有把它的值被设置成小于块大小才有效果，这将强制分片比块小。



### CombineFileInputFormat

1.**相对于大批量的小文件，Hadoop更合适处理少量的大文件。一个原因是FileInputFormat生成的分块是一个文件或该文件的一部分。如果文件很小(“小”意味着比 HDFS的块要小很多)，并且文件数量很多，那么每次 map任务只处理很少的输入数据，(一个文件)就会有很多map任务，每次 map 操作都会造成额外的开销**。请比较一下把1GB的文件分割成8个128 MB块与分成10000个左右100 KB的文件。10000个文件每个都需要使用一个map任务，作业时间比一个输入文件上用8个map任务慢几十倍甚至几百倍。

2.**CombineFileInputFormat可以缓解这个问题，它是针对小文件而设计的**。FileInputFormat为每个文件产生1个分片，而CombineFileInputFormat把多个文件打包到一个分片中以便每个mapper可以处理更多的数据。关键是，决定哪些块放入同一个分片时，CombineFileInputFormat会考虑到节点和机架的因素，所以在典型 MapReduce作业中处理输入的速度并不会下降。

3.当然，**如果可能的话应该尽量避免许多小文件的情况，因为 MapReduce处理数据的最佳速度最好与数据在集群中的传输速度相同，而处理小文件将增加运行作业而必需的寻址次数**。还有，**在 HDFS集群中存储大量的小文件会浪费namenode的内存**。一个可以减少大量小文件的方法是使用顺序文件(sequence file)将这些小文件合并成一个或多个大文件(参见范例8-4):可以将文件名作为键(如果不需要键，可以用Nullwritable 等常量代替)，文件的内容作为值。但如果 HDFS中已经有大批小文件，CombineFileInputFormat方法值得一试。

### 避免切分

1.**有些应用程序可能不希望文件被切分，而是用一个mapper完整处理每一个输入文件**。例如，检查一个文件中所有记录是否有序，一个简单的方法是顺序扫描每一条记录并且比较后一条记录是否比前一条要小。如果将它实现为一个map任务，那么只有一个map操作整个文件时，这个算法才可行。

2.有两种方法可以保证输入文件不被切分。

* **第一种(最简单但不怎么漂亮)方法就是增加最小分片大小，将它设置成大于要处理的最大文件大小**。把它设置为最大值long.MAX\_VALUE 即可。
* **第二种方法就是使用FileInputFormat具体子类，并且重写isSplitable()方法**。把返回值设置为false。

### 把整个文件作为一条记录处理

1.有时，**mapper需要访问一个文件中的全部内容**。即使不分割文件，仍然需要一个RecordReader来读取文件内容作为record的值。wholeFileInputFormat就可以实现此需求。

2.**wholeFileRecordReader负责将FileSplit转换成一条记录，该记录的键是null，值是这个文件的内容**。因为只有一条记录，**wholeFileRecordReader要么处理这条记录，要么不处理，所以它维护一个名称为 processed 的布尔变量来表示记录是否被处理过**。如果当nextKeyvalue()方法被调用时，文件没有被处理过，就打开文件，产生一个长度是文件长度的字节数组，并用Hadoop 的IOutils类把文件的内容放入字节数组。然后再被传递到next()方法的Byteswritable实例上设置数组，返回值为true则表示成功读取记录。

## 2.2.2 文本输入

### TextInputFormat

1.**TextInputFormat是默认的InputFormat**。**每条记录是一行输入。键是Longwritable 类型，存储该行在整个文件中的字节偏移量**。值是这行的内容，不包括任何行终止符(换行符和回车符)，它被打包成一个Text对象。所以，包含如下文本的文件被切分为包含4条记录的一个分片:

On the top of the Crumpetty TreeThe Quangle wangle sat,But his face you could not see,on account of his Beaver Hat.

2.每条记录转化为以下键值对：

(0, on the top of the Crumpetty Tree)(33，The Quangle wangle sat,)(57,But his face you could not see, )(89，on account of his Beaver Hat. )

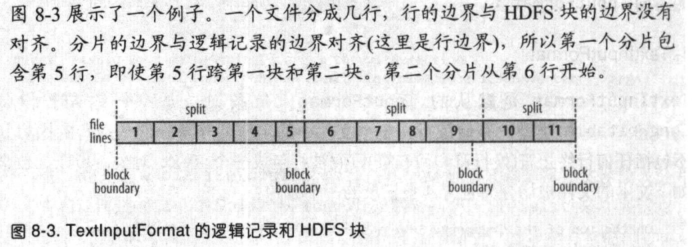
3.**很明显，键并不是行号。一般情况下，很难取得行号，因为文件按字节而不是按行切分为分片**。每个分片单独处理。行号实际上是一个顺序的标记，即每次读取一行的时候需要对行号进行计数。因此，在分片内知道行号是可能的，但在文件中是不可能的。

4.然而，**每一行在文件中的偏移量是可以在分片内单独确定的，而不需要知道分片的信息，因为每个分片都知道上一个分片的大小，只需要加到分片内的偏移量上，就可以获得每行在整个文件中的偏移量了**。

5.通常，**对于每行需要唯一标识的应用来说，有偏移量就足够了。如果再加上文件名，那么它在整个文件系统内就是唯一的。当然，如果每一行都是定长的，那么这个偏移量除以每一行的长度即可算出行号**。

6.输入分片与HDFS块之间的关系：

**FileInputFormat定义的逻辑记录有时并不能很好地匹配HDFS的文件块**。例如TextInputFormat的**逻辑记录是以行为单位的，那么很有可能某一行会跨文件块存放**。虽然这对程序的功能没有什么影响，如行不会丢失或出错，但这种现象应该引起注意，**因为这意味着那些“本地的”map**(即 map运行在输入数据所在的主机上)**会执行一些远程的读操作**。**由此而来的额外开销一般不是特别明显**。



7.控制一行的最大长度。

可以为预期的行长设一个最大值，**对付被损坏的文件。文件的损坏可以表现为一个超长行**，这会导致内存溢出错误，进而任务失败。通过将mapreduce.input.linerecordreader.line.maxlength**设置为用字节数表示的、在内存范围内的值(适当超过输入数据中的行长)，可以确保记录reader跳过(长的)损坏的行，不会导致任务失败**。

### KeyValueTextInputFormat

1.TextInputFormat的键，**即每一行在文件中的字节偏移量**，**通常并不是特别有用**。通常情况下，**文件中的每一行是一个键-值对**，**使用某个分界符进行分隔，比如制表符**。例如由TextOutputFormat(即 Hadoop 默认 OutputFormat)产生的输出就是这种。如果要正确处理这类文件，KeyvalueTextInputFormat比较合适。

可以通过mapreduce.input.keyvaluelinerecordreader.key.value.separator**属性来指定分隔符**。它的默认值是一个制表符。以下是一个范例，其中→表示一个(水平方向的)制表符:

linel ->on the top of the Crumpetty Treeline2 ->The Quangle wangle sat,line3 ->But his face you could not see,line4 ->on account of his Beaver Hat.

与TextInputFormat类似，输入是一个包含4条记录的分片，不过此时的键是每行排在制表符之前的Text序列：

(line1，on the top of the crumpetty Tree)(line2，The Quangle wangle sat,)(line3,But his face you could not see,)(line4，on account of his Beaver Hat.)

### NLineInputFormat

1.**通过TextInputFormat和KeyvalueTextInputFormat，每个mapper收到的输入行数不同**。**行数取决于输入分片的大小和行的长度**。如果希望**mapper 收到固定行数的输入，需要将NLineInputFormat作为InputFormat使用**。与TextInputFormat一样，键是文件中行的字节偏移量，值是行本身。

2.**N是每个mapper收到的输入行数**。**N设置为1(默认值)时，每个mapper 正好收到一行输入**。mapreduce.input.lineinputformat.linespermap属性控制N值的设定。仍然以刚才的4行输入为例:

on the top of the Crumpetty TreeThe Quangle wangle sat,But his face you could not see,On account of his Beaver Hat.

3.例如，如果N是2，则每个输入分片包含两行。一个mapper收到前两行键-值对:

(e,on the top of the crumpetty Tree)(33,The Quangle wangle sat,)

另一个mapper则收到后两行：

(57,But his face you could not see,)(89,on account of his Beaver Hat.)

键和值与TextInputFormat生成的一样。不同的是输入分片的构造方法。

4.通常来说，**对少量输入行执行map任务是比较低效的**(任务初始化的额外开销造成的)，**但有些应用程序会对少量数据做一些扩展的**(也就是**CPU 密集型的**)**计算任务**，然后产生输出。仿真是一个不错的例子。通过生成一个指定输入参数的输入文件，每行一个参数，便可以执行一个参数扫描分析(parameter sweep):并发运行一组仿真试验，看模型是如何随参数不同而变化的。

### 关于XML

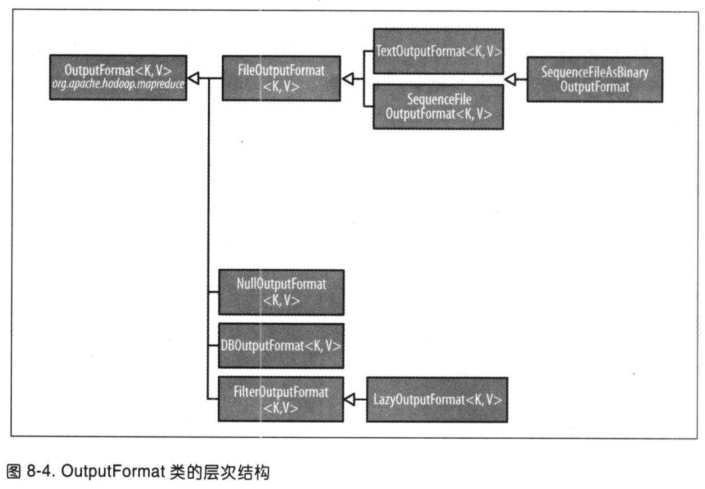
1.**大多数XML解析器会处理整个XML文档**，所以如果一个**大型XML文档由多个输入分片组成**，那么单独解析每个分片就相当有挑战。当然，可以在一个mapper上(如果这个文件不是很大)，可以用8.2.1节介绍的方法来处理整个XML文档。

2.**由很多“记录”(此处是XML文档片断)组成的XML文档**，**可以使用简单的字符串匹配或正则表达式匹配的方法来查找记录的开始标签和结束标签**，而得到很多记录。这可以解决由MapReduce框架进行分割的问题，因为一条记录的下一个开始标签可以通过简单地从分片开始处进行扫描轻松找到，就像TextInputFormat确定新行的边界一样。

3.Hadoop提供了streamxmlRecordReader类（在org.apache.hadoop.streaming.mapreduce包中，还可以在Streaming 之外使用)。通过把输入格式设为StreamInputFormat，把 stream.recordreader.class属性设为org.apache.hadoop.streaming.mapreduce.StreamXmlRecordReader来用streamXm1RecordReader类。reader的配置方法是通过作业配置属性来设reader开始标签和结束标签(详情参见这个类的帮助文档)。”

# 2.3 输出格式

1.OutputFormat类层次结构如下图所示：



## 2.3.1 文本输出

1.**默认的输出格式是 TextoutputFormat，它把每条记录写为文本行**。它的键和值可以是任意类型，因为TextoutputFormat调用toString()方法把它们转换为字符串。**每个键-值对由制表符进行分隔，当然也可以设定mapreduce.output.textoutputformat.separator属性改变默认的分隔符**。与TextOutputFormat对应的输入格式是KeyValueTextInputFormat，它通过可配置的分隔符将键-值对文本行分隔，详情参见8.2.2节。

2.可以**使用Nullwritable来省略输出的键或值(或两者都省略，相当于NulloutputFormat输出格式，后者什么也不输出)**。这也会导致无分隔符输出，以使输出适合用TextInputFormat读取。