Hadoop案例之倒排索引

 "**倒排索引**"是**文档检索系统**中**最常用**的**数据结构**，被广泛地应用于**全文搜索引擎**。它**主要**是用来**存储**某个**单词（或词组）在**一个**文档或一组文档**中的**存储位置**的**映射**，即提供了一种**根据内容来查找文档**的**方式**。由于不是根据文档来确定文档所包含的内容，而是进行相反的操作，因而称为倒排索引（Inverted Index）。

# 1 实例描述

    通常情况下，倒排索引由一个单词（或词组）以及相关的文档列表组成，文档列表中的文档或者是标识文档的ID号，或者是指文档所在位置的URL，如图1所示。

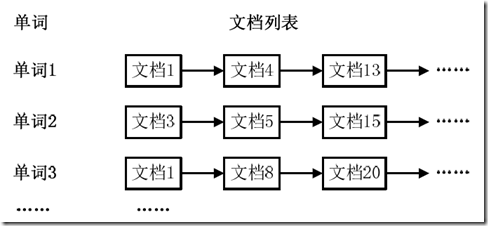
[](http://images.cnblogs.com/cnblogs_com/xia520pi/201206/201206041339489358.png)

图1倒排索引结构

 从图1可以看出，单词1出现在{文档1，文档4，文档13，……}中，单词2出现在{文档3，文档5，文档15，……}中，而单词3出现在{文档1，文档8，文档20，……}中。在**实际应用**中，**还需要**给**每个文档**添加一个**权值**，用来**指出**每个文档与搜索内容的**相关度**，如图2所示。

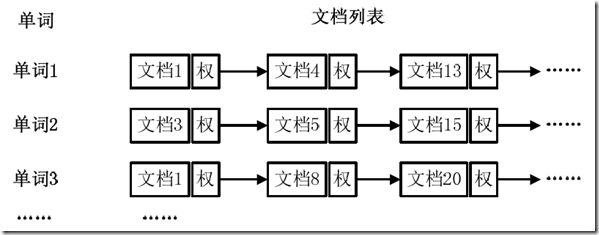
[](http://images.cnblogs.com/cnblogs_com/xia520pi/201206/201206041339495704.png)

图2 添加权重的倒排索引

最常用的是使用**词频**作为**权重**，即记录单词在文档中出现的次数。以英文为例，如图6.1-3所示，索引文件中的"MapReduce"一行表示："MapReduce"这个单词在文本T0中出现过1次，T1中出现过1次，T2中出现过2次。当搜索条件为"MapReduce"、"is"、"Simple"时，对应的集合为：{T0，T1，T2}∩{T0，T1}∩{T0，T1}={T0，T1}，即文档T0和T1包含了所要索引的单词，而且只有T0是连续的。

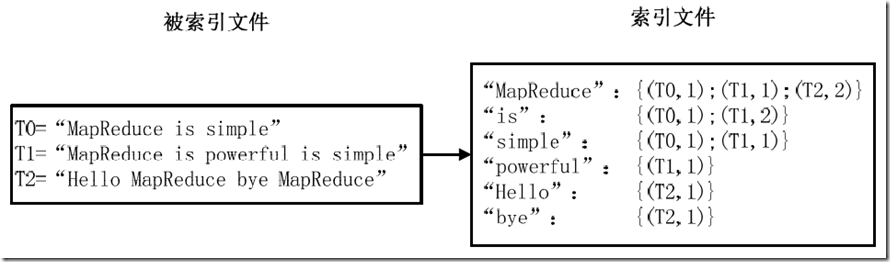
[](http://images.cnblogs.com/cnblogs_com/xia520pi/201206/201206041339495081.png)

图3 倒排索引示例

更复杂的权重还可能要记录单词在多少个文档中出现过，以实现TF-IDF（Term Frequency-Inverse Document Frequency）算法，或者考虑单词在文档中的位置信息（单词是否出现在标题中，反映了单词在文档中的重要性）等。

    样例**输入**如下所示。

**1）file1：**

MapReduce is simple

**2）file2：**

MapReduce is powerful is simple

**3）file3：**

Hello MapReduce bye MapReduce

    样例**输出**如下所示。

MapReduce      file1.txt:1;file2.txt:1;file3.txt:2;

is        　　　　file1.txt:1;file2.txt:2;

simple        　  file1.txt:1;file2.txt:1;

powerful   　　 file2.txt:1;

Hello       　　 file3.txt:1;

bye       　　   file3.txt:1;

# 2 设计思路

    实现"**倒排索引**"只要关注的信息为：**单词**、**文档URL**及**词频**，如图3-11所示。但是在实现过程中，索引文件的格式与图6.1-3会略有所不同，以避免重写OutPutFormat类。下面根据**MapReduce的处理过程**给出**倒排索引**的**设计思路**。

**1）Map过程**

    首先使用默认的**TextInputFormat**类对**输入文件**进行处理，得到文本中**每行**的**偏移量**及其**内容**。显然，Map过程首先必须分析输入的<key,value>对，得到倒排索引中需要的三个信息：单词、文档URL和词频，如图4所示。

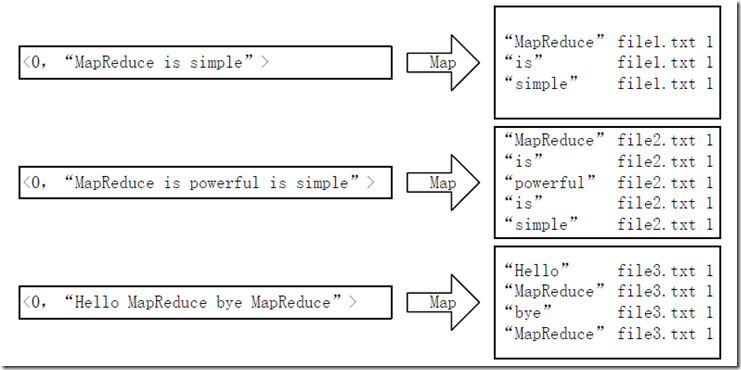
[](http://images.cnblogs.com/cnblogs_com/xia520pi/201206/201206041339508047.png)

图4 Map过程输入/输出

　这里**存在**两个**问题**：**第一**，<key,value>对只能有两个值，在不使用Hadoop自定义数据类型的情况下，需要根据情况将其中**两个值合并**成一个值，作为key或value值；**第二**，通过一个**Reduce**过程**无法同时完成词频统计**和**生成文档列表**，所以必须增加一个**Combine**过程**完成词频统计**。

    这里讲单词和URL组成key值（如"MapReduce：file1.txt"），将词频作为value，这样做的好处是可以利用MapReduce框架自带的Map端排序，将**同一文档**的**相同单词**的**词频**组成**列表**，传递给**Combine**过程，实现类似于WordCount的功能。

**2）Combine过程**

    经过map方法处理后，Combine过程将key值相同的value值累加，得到一个单词在文档在文档中的词频，如图5所示。**如果**直接将图5所示的输出作为Reduce过程的输入，在Shuffle过程时将**面临一个问题**：所有具有**相同单词**的记录（由单词、URL和词频组成）应该交由同一个Reducer处理，但当前的**key值**无法保证这一点，所以必须**修改**key值和value值。这次将**单词**作为**key**值，**URL和词频组**成**value**值（如"file1.txt：1"）。这样做的好处是可以利用MapReduce框架默认的HashPartitioner类完成Shuffle过程，将**相同单词**的**所有记录**发送给**同一个Reducer**进行**处理**。

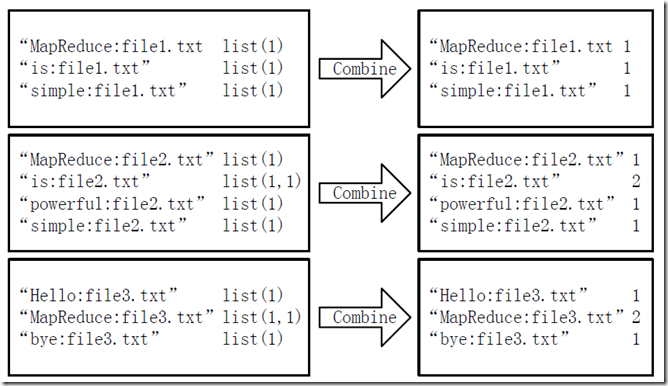
[](http://images.cnblogs.com/cnblogs_com/xia520pi/201206/201206041339518504.png)

图5 Combine过程输入/输出

**3）Reduce过程**

    经过上述两个过程后，Reduce过程只需将相同key值的value值组合成倒排索引文件所需的格式即可，剩下的事情就可以直接交给MapReduce框架进行处理了。如图6所示。索引文件的内容除分隔符外与图3解释相同。

**4）需要解决的问题**

    本实例设计的倒排索引在**文件数目**上**没有限制**，但是**单词文件不宜过大**（具体值与默认HDFS块大小及相关配置有关），要**保证每个文件对应一个split**。否则，由于**Reduce**过程**没有进一步统计词频**，最终结果**可能**会**出现词频未统计完全**的**单词**。可以通过**重写**InputFormat类将每个文件为一个split，避免上述情况。或者**执行两次MapReduce**，**第一次**MapReduce用于**统计词频**，**第二次**MapReduce用于**生成倒排索引**。除此之外，还可以利用复合键值对等实现包含更多信息的倒排索引。

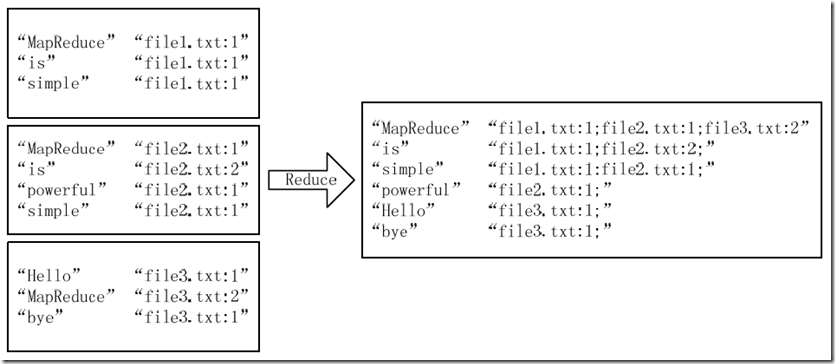
[](http://images.cnblogs.com/cnblogs_com/xia520pi/201206/201206041339528961.png)

图6 Reduce过程输入/输出

# 3.程序代码

**package** Hadoop\_InvertedIndex;

**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.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.FileSplit;

**import** org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

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

**public** **class** InvertedIndex {

**public** **static** **class** Map **extends** Mapper<Object, Text, Text, Text>

{

**private** Text keyInfo = **new** Text();

**private** Text valueInfo = **new** Text();

**private** FileSplit split;

@Override

**protected** **void** map(Object **key**, Text **value**, **Context** **context**)

**throws** IOException, InterruptedException {

//简单起见，只获取文件名

split = (FileSplit)context.getInputSplit();

**int** **index** = split.getPath().toString().indexOf("file");

String **fileName** = split.getPath().toString().substring(index);

StringTokenizer **st** = **new** StringTokenizer(value.toString());

**while**(st.hasMoreTokens())

{

//单词：文件名 作为key

//1 为 value

keyInfo.set(st.nextToken()+":"+fileName);

valueInfo.set("1");

context.write(keyInfo, valueInfo);

}

}

}

//combine

**public** **static** **class** Combine **extends** Reducer<Text, Text, Text, Text>

{

@Override

**protected** **void** reduce(Text **key**, Iterable<Text> **values**, **Context** **context**)

**throws** IOException, InterruptedException {

// 统计词频

**int** **sum** = 0;

**for**(Text **temp** : values)

{

sum+= Integer.*parseInt*(temp.toString());

}

**int** **index** = key.toString().indexOf(":");

// 重新设置key值为单词

String **sKey** = key.toString().substring(0, index);

Text **outKey** = **new** Text(sKey);

// 重新设置value值由URL和词频组成

String **sValue** = key.toString().substring(index+1);

Text **outValue** = **new** Text(sValue+":"+sum);

context.write(outKey, outValue);

}

}

//reduce

**public** **static** **class** Reduce **extends** Reducer<Text, Text, Text, Text>

{

@Override

**protected** **void** reduce(Text **key**, Iterable<Text> **values**,**Context** **context**)

**throws** IOException, InterruptedException {

String **fileList** = **new** String();

**for**(Text **temp** : values)

{

fileList += temp.toString() + ";";

}

context.write(key, **new** Text(fileList));

}

}

//main

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

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

String[] **otherArgs** = **new** GenericOptionsParser(conf,args).getRemainingArgs();

**if** (otherArgs.length != 2) {

System.***err***.println("Usage: InvertedIndex <in> <out>");

System.*exit*(2);

}

Job **job** = **new** ~~Job~~(conf, "InvertedIndex");

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

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

job.setCombinerClass(Combine.**class**);

job.setReducerClass(Reduce.**class**);

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

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

**FileInputFormat**.*addInputPath*(job, **new** Path(otherArgs[0]));

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

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

}

}

# 4.程序执行

root@node1:/usr/local/hadoop/hadoop-2.5.2/myJar# hadoop jar InvertedIndex.jar Hadoop\_InvertedIndex.InvertedIndex /usr/local/hadooptempdata/input/inverted /usr/local/hadooptempdata/output/inverted

16/12/29 23:29:04 INFO client.RMProxy: Connecting to ResourceManager at node1/192.168.233.129:8032

16/12/29 23:29:07 INFO input.FileInputFormat: Total input paths to process : 3

16/12/29 23:29:07 INFO mapreduce.JobSubmitter: number of splits:3

16/12/29 23:29:08 INFO mapreduce.JobSubmitter: Submitting tokens for job: job\_1483024810031\_0001

16/12/29 23:29:09 INFO impl.YarnClientImpl: Submitted application application\_1483024810031\_0001

16/12/29 23:29:10 INFO mapreduce.Job: The url to track the job: http://node1:8088/proxy/application\_1483024810031\_0001/

16/12/29 23:29:10 INFO mapreduce.Job: Running job: job\_1483024810031\_0001

16/12/29 23:29:55 INFO mapreduce.Job: Job job\_1483024810031\_0001 running in uber mode : false

16/12/29 23:29:55 INFO mapreduce.Job: map 0% reduce 0%

16/12/29 23:34:16 INFO mapreduce.Job: map 22% reduce 0%

16/12/29 23:34:25 INFO mapreduce.Job: map 44% reduce 0%

16/12/29 23:34:31 INFO mapreduce.Job: map 56% reduce 0%

16/12/29 23:34:32 INFO mapreduce.Job: map 78% reduce 0%

16/12/29 23:34:35 INFO mapreduce.Job: map 89% reduce 0%

16/12/29 23:34:46 INFO mapreduce.Job: map 100% reduce 0%

16/12/29 23:35:44 INFO mapreduce.Job: map 100% reduce 100%

16/12/29 23:35:49 INFO mapreduce.Job: Job job\_1483024810031\_0001 completed successfully

16/12/29 23:35:50 INFO mapreduce.Job: Counters: 49

File System Counters

FILE: Number of bytes read=215

FILE: Number of bytes written=396231

FILE: Number of read operations=0

FILE: Number of large read operations=0

FILE: Number of write operations=0

HDFS: Number of bytes read=478

HDFS: Number of bytes written=165

HDFS: Number of read operations=12

HDFS: Number of large read operations=0

HDFS: Number of write operations=2

Job Counters

Launched map tasks=3

Launched reduce tasks=1

Data-local map tasks=3

Total time spent by all maps in occupied slots (ms)=902667

Total time spent by all reduces in occupied slots (ms)=33136

Total time spent by all map tasks (ms)=902667

Total time spent by all reduce tasks (ms)=33136

Total vcore-seconds taken by all map tasks=902667

Total vcore-seconds taken by all reduce tasks=33136

Total megabyte-seconds taken by all map tasks=924331008

Total megabyte-seconds taken by all reduce tasks=33931264

Map-Reduce Framework

Map input records=4

Map output records=12

Map output bytes=226

Map output materialized bytes=227

Input split bytes=393

Combine input records=12

Combine output records=10

Reduce input groups=6

Reduce shuffle bytes=227

Reduce input records=10

Reduce output records=6

Spilled Records=20

Shuffled Maps =3

Failed Shuffles=0

Merged Map outputs=3

GC time elapsed (ms)=82879

CPU time spent (ms)=14870

Physical memory (bytes) snapshot=474574848

Virtual memory (bytes) snapshot=7537999872

Total committed heap usage (bytes)=382656512

Shuffle Errors

BAD\_ID=0

CONNECTION=0

IO\_ERROR=0

WRONG\_LENGTH=0

WRONG\_MAP=0

WRONG\_REDUCE=0

File Input Format Counters

Bytes Read=85

File Output Format Counters

Bytes Written=165

# 5.输出结果

root@node1:/usr/local/hadoop/hadoop-2.5.2/myJar# hdfs dfs -cat /usr/local/hadooptempdata/output/inverted/\*

Hello file3.txt:1;

MapReduce file3.txt:2;file1.txt:1;file2.txt:1;

bye file3.txt:1;

is file1.txt:1;file2.txt:2;

powerful file2.txt:1;

simple file2.txt:1;file1.txt:1;