Hadoop 案例11----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、说明**

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1-1所示。

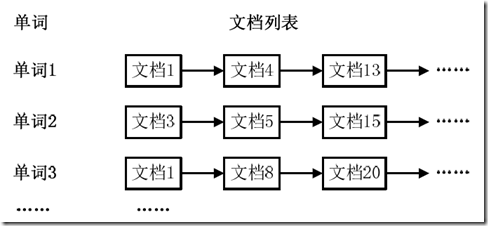
[](http://images.cnblogs.com/cnblogs_com/xia520pi/201206/201206041339489358.png)

图1-1 倒排索引结构

    从图1-1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1-2所示。

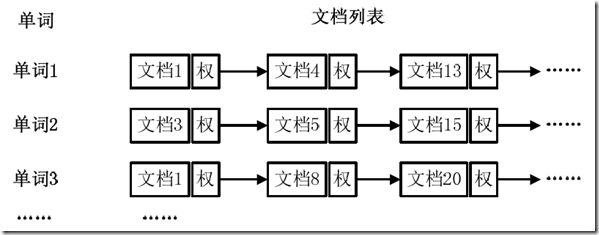
[](http://images.cnblogs.com/cnblogs_com/xia520pi/201206/201206041339495704.png)

图1-2 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。

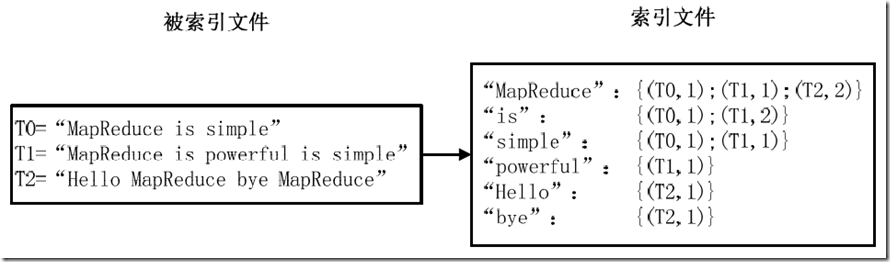
[](http://images.cnblogs.com/cnblogs_com/xia520pi/201206/201206041339495081.png)

图1-3 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**，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2-1所示。

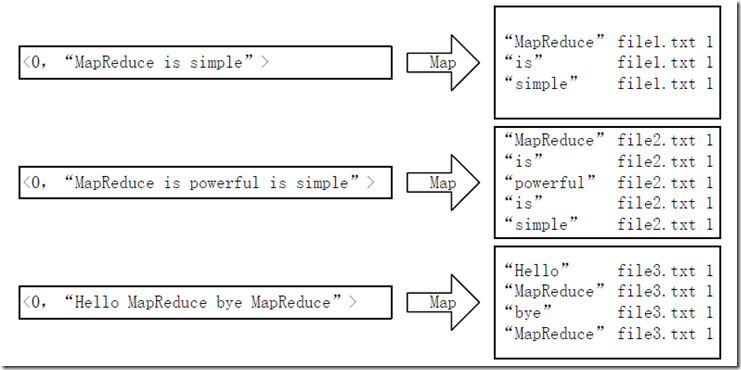
[](http://images.cnblogs.com/cnblogs_com/xia520pi/201206/201206041339508047.png)

图2-1 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2-2所示。**如果**直接将图2-2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**。

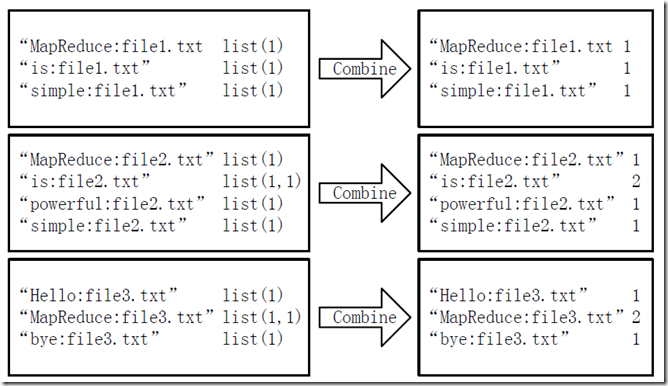
[](http://images.cnblogs.com/cnblogs_com/xia520pi/201206/201206041339518504.png)

图2-2 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2-3所示。索引文件的内容除分隔符外与图1-3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。

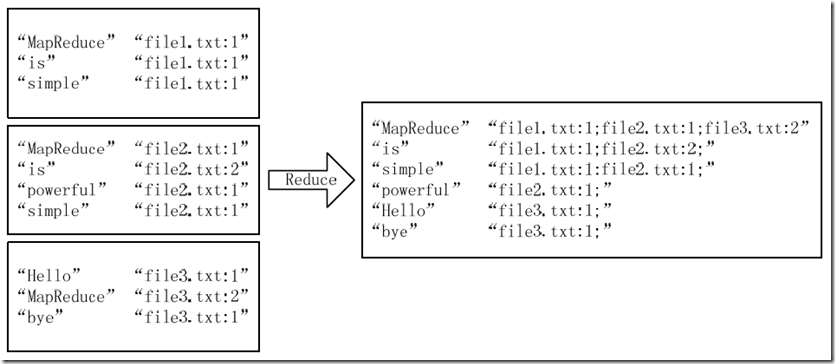
[](http://images.cnblogs.com/cnblogs_com/xia520pi/201206/201206041339528961.png)

图2-3 Reduce过程输入/输出

**3、代码实现**

**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.IntWritable;

**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(); // 存储单词和URL组合

**private** Text valueInfo = **new** Text(); // 存储词频

**private** FileSplit split; // 存储Split对象

        // 实现map函数

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

**throws** IOException, InterruptedException {

            // 获得<key,value>对所属的FileSplit对象

            split = (FileSplit) context.getInputSplit();

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

**while** (itr.hasMoreTokens()) {

                // key值由单词和URL组成，如"MapReduce：file1.txt"

                // 获取文件的完整路径

                // keyInfo.set(itr.nextToken()+":"+split.getPath().toString());

                // 这里为了好看，只获取文件的名称。

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

                keyInfo.set(itr.nextToken() + ":"

                    + split.getPath().toString().substring(splitIndex));

                // 词频初始化为1

                valueInfo.set("1");

                context.write(keyInfo, valueInfo);

            }

        }

    }

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

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

        // 实现reduce函数

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

**throws** IOException, InterruptedException {

            // 统计词频

**int** sum = 0;

**for** (Text value : values) {

                sum += Integer.parseInt(value.toString());

            }

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

            // 重新设置value值由URL和词频组成

            info.set(key.toString().substring(splitIndex + 1) + ":" + sum);

            // 重新设置key值为单词

            key.set(key.toString().substring(0, splitIndex));

            context.write(key, info);

        }

    }

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

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

        // 实现reduce函数

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

**throws** IOException, InterruptedException {

            // 生成文档列表

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

**for** (Text value : values) {

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

            }

            result.set(fileList);

            context.write(key, result);

        }

    }

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

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

        // 这句话很关键

        conf.set("mapred.job.tracker", "192.168.1.2:9001");

        String[] ioArgs = **new** String[] { "index\_in", "index\_out" };

        String[] otherArgs = **new** GenericOptionsParser(conf, ioArgs)

                .getRemainingArgs();

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

            System.err.println("Usage: Inverted Index <in> <out>");

            System.exit(2);

        }

        Job job = **new** Job(conf, "Inverted Index");

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

        // 设置Map、Combine和Reduce处理类

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

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

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

        // 设置Map输出类型

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

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

        // 设置Reduce输出类型

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

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

        // 设置输入和输出目录

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

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

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

    }

}