金融大数据处理技术：实验 3

计算机科学与技术系

161220156

杨浩然

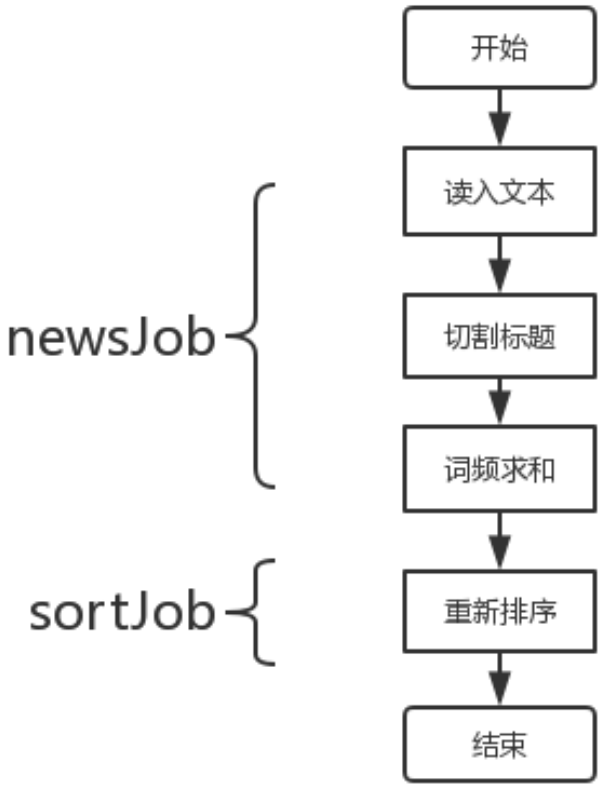
1 统计新闻标题中的词频数

1.1 实验目的

针对股票新闻数据集中的新闻标题，编写 WordCount 程序，统计所有除 Stop-word（如“的”，“得”，“在”等）出现次数 k 次以上的单词计数，最后的结果按照词频从高到低排序输出。

1.2 实验设计

1.2.1 流程图



1.2.2 具体过程描述

由上述流程图可知，完成整个实验需要启用两个 Job，首先说明 newsJob：

步骤 1、按行读取文本；

步骤 2、去除每行文本中的数字与字母，然后分割文本，提取新闻标题部分；

步骤 3、调用第三方工具包 word 进行中文分词（word 工具包会自动去除停用词）；

步骤 4、Mapper 端以<word, 1>的形式把分词后的结果发送给 Reducer 端；

步骤 5、Reducer 计算各个词语出现的次数；

步骤 6、将计算结果保存在 HDFS 上的一个临时文件中。

newsJob 执行完毕后我们把输出结果存在一个临时文件中，该文件将会是 sortJob 的输入：

步骤 1、读入临时文件中的内容；

步骤 2、调换键值对的先后顺序；

步骤 3、对读取到的文本进行重新排序；

步骤 4、将结果输出至最终文件路径，并删除临时文件。

该实验最关键之处在于 sortJob 中重新排序，首先我们要将原来的键值对的键值顺序进行交换，用“词频”作为键，然后，通过查找资料可知，MapReduce 默认排序是“由大到小”，因此我们必须修改这样的排序方式，查看 API 手册可知，我们需要给 sortJob 设定一个自定义的“比较器”（调用 setSortComparatorClass()方法进行设置），代码如下：

```
public static class IntWritableDecreasingComparator
        extends IntWritable.Comparator{
        public int compare (WritableComparable a, WritableComparable b){
                return -super.compare(a, b);
        }

        public int compare (byte[] b1, int s1, int l1, byte[] b2, int s2,
        int l2){
                return -super.compare(b1, s1, l1, b2, s2, l2);
        }
}
```

该类继承了 IntWritable 类中的 Comparator 类，并对 compare 方法进行了重载，重载非常简单，即对原有的 compare 方法的返回值取相反数，这样处理是相当直观的，能让 MapReduce 框架按照与原来的排序原则相反的原则进行排序。

1.2.3 类功能简介

类名	功能
PUBLIC STATIC CLASS NEWSMAPPER EXTENDS MAPPER<LONGWRITABLE, TEXT, TEXT, INTWRITABLE>	继承自 Mapper 类，对新闻标题进行分割
PUBLIC STATIC CLASS NEWSREDUCER EXTENDS REDUCER<TEXT, INTWRITABLE, TEXT, INTWRITABLE>	继承自 Reducer 类，对各个词出现的频数进行求和并输出至临时文件
PUBLIC STATIC CLASS INTWRITABLEDECREASINGCOMPARATOR EXTENDS INTWRITABLE.COMPARATOR	继承自 IntWritable.Comparator 类，重载 compare 方法，以适应新的排序规则

1.3 实验总结

1.3.1 实验结果分析

参见 wordcount_output 文件，可知本程序很好地满足了需求 1，输出的结果符合“按照词频从高到低排序输出”的要求，由于结果较为简单，不进行过多分析。

1.3.2 可改进之处

结合 MapReduce 程序框架特点，总结如下几点程序可改进之处：

1、通过参考 Hadoop-2.9.1 版本的 API 手册，自定义“比较器“已经不推荐从 WritableComparable.Comparator 继承并重载 compare 方法；

2、执行 sortJob 时只设定了一个 Reduce 任务，没有充分发挥 Hadoop 的并行计算优势。

2 倒排索引

2.1 实验目的

针对股票新闻数据集，以新闻标题中的词组为 key，编写带 URL 属性和词频的文档倒排索引程序，并按照词频从大到小排序，将结果输出到指定文件。输出格式可以如下：

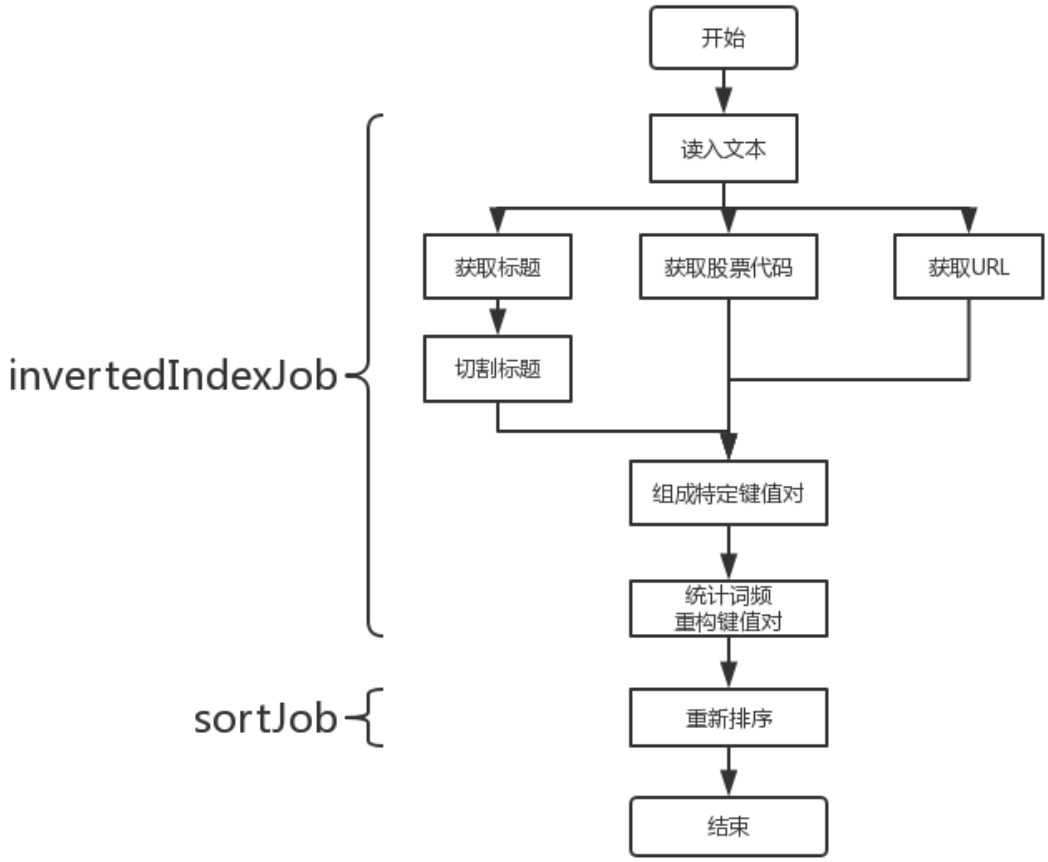
高速公路， 10， 股票代码，[url0, url1,...,url9]

高速公路， 8， 股票代码，[url0, url1,...,url7]

注：可以用提供的 Stop-word 列表，也可以自行建立一个 Stop-word 列表，其中包含部分停词即可，不需要列出全部停词；参数 k 作为输入参数动态制定（如 k=10）

2.2 实验设计

2.2.1 流程图



2.2.2 具体过程描述

为了达到目的，该过程也需要两个 Job 来完成，首先说明 invertedIndexJob：

步骤 1、按行读取文本；

步骤 2、获取标题词语（调用 word 包）、获取股票代码、获取单行文本中的单个 URL；

步骤 3、组成键值对，发送给 Reducer，***形式为<(word, stock code), (1, url)>***；

步骤 4、统计词频并重组键值对，***形式为<(word, frequent), (stock code, url0, url1, …)***。

invertedIndexJob 输出的形式如上步骤 4 所述，并将所有结果存储在 HDFS 上的一个临时文件中，sortJob 读取该临时文件并进行排序处理：

步骤 1、按行读取文本；

步骤 2、根据主键进行排序;

步骤 3、将结果输出至最终的输出路径。

该过程有两个关键点，一是键值对的构造，二是排序。键值对的构造已经在上述具体过程描述中给出，现在只讨论“比较器“的构造。为了达到需求 2 中给出的输出样例的效果，给出如下的比较规则:

1. 主键由两部分构成，即词语（word）和词频（frequent）；

2. 首先比较“词语“，调用 Java 自带的 compareTo 方法即可；

3. 若两主键的“词语“部分相等，再比较”词频“部分；

4. 将词频转为整型来比较大小；

5. 为了按从大到小的排序，规定：

key_1 < key_2，函数返回 1；

key_1 = key_2，函数返回 0；

key_1 > key_2，函数返回-1；

另外，在上文需求 1 的报告中我们提到，Hadoop-2.9.1 不推荐继承类 WritableComparable.Comparator 来构造“比较器“，为了适应 2.9.1 版本，此处采用继承 WritableComparator 类并重载 compare 方法的方式来构建自定义”比较器“，自定义“比较器“代码如下:

```
public static class TextDecreasingComparator extends WritableComparator{
        public TextDecreasingComparator(){
                super(Text.class, true);
        }

        public int compare(WritableComparable a, WritableComparable b){
                String thisText = a.toString();
                String thatText = b.toString();
                String thisKey = thisText.split(",")[0];
                String thatKey = thatText.split(",")[0];
                int thisValue = Integer.parseInt(thisText.split(",")[1]);
                int thatValue = Integer.parseInt(thatText.split(",")[1]);

                if (thisKey.equals(thatKey) ||
                        thisKey.compareTo(thatKey) == 0){
                        if (thisValue < thatValue)
                                return 1;
                        else if (thisValue == thatValue)
```

```
                    return 0;
                else if (thisValue > thatValue)
                    return -1;
            }
            else
                return -thisKey.compareTo(thatKey);

            return 1;
        }
    }
```

其中 compare 方法的代码即是前文中规定的比较规则。

2.2.3 类功能简介

类名	功能
PUBLIC STATIC CLASS MAP EXTENDS MAPPER<LONGWRITABLE, TEXT, TEXT, TEXT>	继承自 Mapper 类，从文本中提取所需信息并构造前文中提及的键值对
PUBLIC STATIC CLASS REDUCE EXTENDS REDUCER<TEXT, TEXT, TEXT, TEXT>	继承自 Reducer 类，统计词频，并重新构造键值对存入临时文件中
PUBLIC STATIC CLASS TEXTDECREASINGCOMPARATOR EXTENDS WRITABLECOMPARATOR	继承自 WritableComparator 类，自定义排序方法满足需求
PUBLIC CLASS SORT	方便 sortJob 正常执行

2.3 实验总结

2.3.1 实验结果分析

参见 invertedindex_output 文件，可知本程序很好地满足了需求 2，输出的结果符合“相同词语聚集在一起，且按照词频从高到低排序输出”的要求，由于结果较为简单，不进行过多分析。

2.3.2 可改进之处

结合 MapReduce 程序框架以及具体需求特点，总结如下几点程序可改进之处：

1、可选用更轻量级的中文分词第三方工具包，此次实验采用的 word 中文分词包比较庞大，初始化和处理过程均较慢；

2、可自定义一个 Combiner，以减少 Mapper 向 Reducer 传输的数据量；

3、针对本次实验中形式较为特殊的键，我们还可以创建一个 WritableComparable 实例，这样应该能在排序、数据传输和其他处理上有更好的效果。