大实验程序设计报告

组长：陈智麒 151220015 邮箱：csczq123456@gmail.com

组员：陈博钏 151220007；陈彧 151220013；张帅 151220162

(南京大学 计算机科学与技术系, 南京 210093)

# **背景知识**

## **知识1（PageRank）**

。

## **知识2（标签传播算法）**

。

# **实验过程说明**

本次实验主要由以下内容构成：。

以下为本次实验的主要流程图：

。

## **数据预处理**

### 实验要求

本任务的主要工作是从原始的金庸小说文本（未分词）中，抽取出与人物互动相关的数据，而屏蔽掉与人物关系无关的文本内容，为之后的基于人物共现算法的分析做准备。任务输入为全本的金庸武侠小说文集（未分词）和金庸武侠小说人名列表，而输出为分词后仅保留人名的金庸武侠小说全集。

### 实验过程

1）整体设计

我们利用Hadoop的MapReduce编程框架解决任务1数据预处理。任务1中，我们设计了5个类，包括Task1主类、Task1Mapper类、Tasks1Reducer类、Task1CustomOutputFormat类和Task1CustomRecordWriter类。

在Task1主类中，我们做了一些初始化的操作，主要包括指定输入输出文件路径，设定使用的Mapper类和Reducer类，设定输入输出格式以及键值对格式。在主类中指定了相关初始化设置之后，任务1进入map和reduce阶段。

map阶段主要通过Task1Mapper类实现。在Task1Mapper类中，我们将每次输入的一段（即一行，下同）小说内容进行分词、小说人名的提取，并将这些人名组成的字符串输出给Reducer节点。reduce阶段主要通过Task1Reducer类实现。在Task1Reducer类中，我们将多个输入的小说人名构成的字符串汇总在一起，并输出到输出文件中。

2）细节要点

①Distributed Cache File的使用

在任务1中，我们需要在map阶段使用小说人名列表。我们在Task1主类中通过以下代码将小说人名列表文件放置到Distributed Cache File中，其中args[0]为小说人名列表文件的路径。

|  |
| --- |
| job**.**addCacheFile**(new** Path**(**args**[**0**]).**toUri**());** |

之后，我们在Task1Mapper类的setup方法中通过以下代码获取Distributed Cache File：

|  |
| --- |
| Path**[]** cacheFiles **=** context**.**getLocalCacheFiles**();**  BufferedReader joinReader **=** **new** BufferedReader**(new** FileReader**(**cacheFiles**[**0**].**toUri**().**getPath**()));**  String line**=** joinReader**.**readLine**();** |

另外，我们也在Task1Mapper类中创建了静态私有成员names，用来保存从Distributed Cache File中读取的小说人名，以供所有Mapper节点进行使用。

②自定义数据输出类型

在实验过程中，为了保证任务1的输出能够直接作为任务2的输入进行进一步的操作，我们需要确保输出的文件中，每一行包含的人名由单个空格隔开，且每一行最后一个人名之后有一个空格；但是hadoop中提供的TextOutputFormat类使得输出文件中每一行最后包含一个制表符“\t”，所以我们需要自定义输出格式类型。我们自定义了输出格式类型，即根据需要自行定制了OutputFormat和RecordWriter。我们主要实现了Task1CustomOutputFormat类和Task1CustomRecordWriter类。

Task1CustomOutputFormat类继承自FileOutputFormat类。我们重写Task1CustomOutputFormat类中的getRecordWriter方法，在该方法中我们自定义输出文件名及其路径，并在HDFS上进行文件的创建。最后，它通过创建Task1CustomRecordWriter类，将输出内容写入到输出文件中。

Task1CustomRecordWriter类继承自RecordWriter类。我们重写Task1CustomRecordWriter类中的write方法，在该方法中，我们自定义将输出内容写入到输出文件中的格式，实现代码如下：

|  |
| --- |
| @Override  public void write**(**Text arg0**,** Text arg1**)** **throws** IOException**,** InterruptedException **{**  //custom write format  fileOutstream**.**write**((**arg0**.**toString**()** **+** "\n"**).**getBytes**());**  **}** |

③分词

在任务1中，我们使用了Ansj工具进行分词。我们通过引入Ansj的相关类，并调用其提供的API，对小说文本内容进行分类。

我们在Task1Mapper类的setup方法中将小说人名导入到Ansj的用户自定义词典中；之后在map方法中，每次读入小说的一行内容，我们利用Ansj提供的API进行分词（由于小说人名在setup阶段已导入Ansj的用户自定义词典中，故它们不会在分词时被分割开），并逐个判断分好的词是否属于小说人名列表中，若属于小说人名列表则保留，否则去除。

最终，每次map过程结束后，我们将小说每一行包含的人名组成的字符串输出给Reducer节点。

### 实验结果

数据预处理结果如下图：

【图片】

## **过程2**

### 实验要求

。

### 实验过程

。

### 实验结果

。

## **过程3**

### 实验要求

。

### 实验过程

。

### 实验结果

。

## **过程4**

### 实验要求

。

### 实验过程

。

### 实验结果

。

## **过程5**

### 实验要求

。

### 实验过程

。

### 实验结果

。

## **过程6**

### 实验要求

。

### 实验过程

。

### 实验结果

。

# **实验中遇到的问题及改进**

## **Ansj提供的API调用**

关于Ansj工具的使用，我们最初是选择的是这样的方案：将小说人名列表中的人名添加至Ansj默认的字典中，并在添加时设定添加的词语的属性为“userDefine”；当我们得到分词结果后，判断每个词的属性是否为“userDefine”，从而确定是否将该词保留下来。但是当我们添加的词在Ansj默认的字典中存在的情况时，我们添加的词的属性无法改为“userDefine”，所以在分词阶段出现了一些问题。

因此我们选择了另外的方案：将小说人名列表中的人名添加至Ansj默认的字典中，在添加时设定添加的词语的属性为任意值（该属性之后不被使用）；当我们得到分词结果后，判断每个词是否属于小说人名列表，从而确定是否将该词保留下来。通该方案我们可以获得较理想的分词效果。

## **第三方依赖包的打包**

在Hadoop的Mapper类中调用了第三方库Ansj，使用maven进行打包的过程非常顺利；但是运行打包出来的jar文件时报错：找不到Ansj中相关类的定义。

我们经过相关资料的查询，通过以下命令将第三方依赖包打包进maven生成的jar包中：

|  |
| --- |
| jar uf ./target/MPLab-1.0.jar lib/\*.jar |

最终我们在集群上运行包含第三方依赖包的jar包，从而获得最终的实验结果。

## **改进3**

。

# **小组分工**

：。

# **实验体会**

。