

# 哈爾濱工業大學

## 大数据计算基础课程报告

题 目：“纽约时报”词频统计与倒排索引

(选修-7)

专 业 人工智能

学 号 2021112845

姓 名 张智雄

课 程 大数据计算基础（选修）

日 期 2024.1.5

## 摘要

随着现在社会的发展所带来的数据量庞大化问题复杂化，在处理许多现实问题时可能会涉及巨量信息，但对实时性和响应时间要求越来越高，因此需要将问题分解以减轻负载，利用集群的并行计算能力加速问题的求解。

在大数据背景下，词频统计和建立倒排索引是信息检索、文本挖掘和数据分析领域中非常重要的技术手段。它们有助于更高效地管理和处理大规模文本数据，并提供了丰富的信息以支持各种应用。本实验主要基于 Hadoop MapReduce 框架对《纽约时报》评论的数据进行词频统计并建立倒排索引。

首先，我们简要介绍了 Hadoop 框架，并阐述了其核心组件 MapReduce 的工作原理，理解了其在分布式计算中的作用。随后，对 HDFS 进行了详细的讲解，探讨了其分布式文件系统的设计原理、数据存储方式以及容错机制。

其次，根据实际实验过程，较为细致的编写了 Hadoop 的安装与伪分布式配置文档。

然后，我们基于 Hadoop MapReduce 平台，对《纽约时报》评论数据其中的 snippet 字段进行了词频统计。这一步骤涉及了 Map 阶段的数据切分、分布式计算以及 Reduce 阶段的结果合并等关键步骤。通过实际操作，我们深入理解了 MapReduce 编程模型，并学会了如何将其应用于实际数据处理任务中。

最终，我们基于词频统计成功建立了一个倒排索引，以 snippet 字段中的词语为关键词，统计了它们在评论数据中的出现频率。

通过本次实验，我们简单熟悉了 Hadoop 框架的具体应用，还在实践中体会到了大规模数据处理的挑战和优势。

**关键词：**分布式计算 词频统计 倒排索引

## 目 录

摘 要 .....	I
第 1 章 绪 论 .....	4
1.1 问题背景 .....	4
1.2 问题描述 .....	4
1.5 本文的贡献 .....	5
1.6 章节安排 .....	5
第 2 章 HADOOP 框架 .....	6
2.1 HADOOP 简介 .....	6
2.2 HADOOP MAPREDUCE 原理 .....	6
2.3 HADOOP HDFS 原理 .....	7
2.4 HADOOP 的优点与局限性 .....	8
第 3 章 HADOOP 分布式配置 .....	8
3.1 前期准备(新用户创建和 SSH 配置) .....	8
3.1.1 新用户创建 .....	8
3.1.2 配置 SSH .....	9
3.1.3 总结 .....	10
3.2 JAVA 环境配置 .....	10
3.3 HADOOP 3.3.5 安装 .....	10
3.3.1 下载安装 .....	11
3.3.2 伪分布式配置 .....	11
3.4 项目创建与导出 .....	13

3.4.1	在 Eclipse 中创建项目 .....	13
3.4.2	编译打包程序 .....	15
<b>第 4 章 数据预处理 .....</b>		<b>15</b>
4.1	数据集简介 .....	15
4.2	预处理过程 .....	16
4.3	预处理结果 .....	17
<b>第 5 章 词频统计 .....</b>		<b>17</b>
5.1	统计过程 .....	17
5.1.1	Mapper.....	17
5.1.2	Reducer.....	18
5.1.3	Driver.....	18
5.1.4	总结 .....	18
5.2	统计结果 .....	19
<b>第 6 章 建立倒排索引 .....</b>		<b>19</b>
6.1	倒排索引概念 .....	19
6.2	倒排索引建立过程 .....	20
6.3	索引建立结果 .....	21
<b>第 7 章 结论 .....</b>		<b>21</b>
7.1	实验总结与收获 .....	21
7.2	不足与改进之处 .....	21
<b>参考文献 .....</b>		<b>22</b>

# 第1章 绪论

## 1.1 问题背景

随着现在社会的发展所带来的数据量庞大化问题复杂化，在处理许多现实问题时涉及巨量信息，当数据量达到 GB 或 TB 规模时，计算量指数级上升，但用户的应用分析结果呈整合趋势，对实时性和响应时间要求越来越高，需要将问题分解，减轻负载，利用集群的并行计算能力加速问题的求解。

在大数据背景下，词频统计和建立倒排索引是信息检索、文本挖掘和数据分析领域中非常重要的技术手段。它们有助于更高效地管理和处理大规模文本数据，并提供了丰富的信息以支持各种应用。

通过分析大量文本数据中的词频，可以识别常见词汇和短语，从而抽取文本的关键信息和概要，有助于识别文本中频繁出现的关键词，这对于理解文本内容和主题非常关键。同时，通过分析词在文本中的相对频率，可以帮助理解词在不同上下文中的含义和语境。

倒排索引是一种非常高效的数据结构，可以加速文本数据的检索操作。通过记录每个词在哪些文档中出现，可以快速定位相关文档，实现关键词的搜索。并且，倒排索引还有助于对文本进行聚类分析，将相似内容的文档聚集在一起，从而提供更有组织和结构的数据。

通过结合词频统计和倒排索引，可以构建强大的信息检索系统，支持用户在大规模文本数据中快速、准确地找到相关信息。同时还可以进行文本挖掘任务，如情感分析、主题建模和实体识别，从而深入挖掘文本数据的内在信息。

## 1.2 问题描述

《纽约时报》拥有广泛的读者群，对塑造人们对时事的看法和观点，尤其是在美国塑造公共话语的方面起着重要作用。

现有一组《纽约时报》评论的数据，数据来源 <https://www.kaggle.com/aashita/nyt-comments/home>，请按要求完成如下问题：

- 1) 请编写 Hadoop 安装与配置文档，并简述 Hadoop 基本原理；
- 2) 请利用“纽约时报”数据集，在 Hadoop 环境下编写 wordcount 程序，对数据集中 snippet 字段出现的所有词进行词频统计；
- 3) 在第 2 题的基础上，在 Hadoop 环境下对数据集建立倒排索引，统计不同词在不同评论 ID 中出现的位置及次数；

## 1.5 实验内容

本实验简要介绍了 Hadoop 框架，包括 MapReduce 和 HDFS，较为细致的编写了 Hadoop 的安装与伪分布式配置文档。并基于 Hadoop MapReduce 实现了对《纽约时报》评论的数据 snippet 字段出现的所有词的词频统计并建立了倒排索引。

首先，我们简要介绍了 Hadoop 框架，并阐述了其核心组件 MapReduce 的工作原理，理解了其在分布式计算中的作用。随后，对 HDFS 进行了详细的讲解，探讨了其分布式文件系统的设计原理、数据存储方式以及容错机制。

其次，根据实际实验过程，较为细致的编写了 Hadoop 的安装与伪分布式配置文档。通过这一步，我们建立了一个基本的 Hadoop 环境，为后续的数据处理任务创造了必要的条件。

然后，我们基于 Hadoop MapReduce 平台，对《纽约时报》评论数据其中的 snippet 字段进行了词频统计。这一步骤涉及了 Map 阶段的数据切分、分布式计算以及 Reduce 阶段的结果合并等关键步骤。通过实际操作，我们深入理解了 MapReduce 编程模型，并学会了如何将其应用于实际数据处理任务中。

最终，我们基于词频统计成功建立了一个倒排索引，以 snippet 字段中的词语为关键词，统计了它们在评论数据中的出现频率。

## 1.6 章节安排

本文第 2 章简要介绍了 Hadoop 框架，并阐述了其核心组件 MapReduce 和 HDFS 的工作原理。第 3 章为分布式计算平台 Hadoop 的安装与伪分布式配置文档。第 4 章是基于 Hadoop MapReduce 对《纽约时报》评论数据进行词频统计的实验原理、过程及结果。第 5 章是基于 Hadoop MapReduce 对《纽约时报》评论数据建立倒排索引的实验原理、过程及结果。最后，第 7 章简要总结了本次实验的收获以及不足和改进之处。

## 第 2 章 Hadoop 框架

### 2.1 Hadoop 简介

Hadoop 是一种分析和处理大数据的软件平台，是一个用 Java 语言实现的 Apache 的开源软件框架，在大量计算机组成的集群中实现了对海量数据的分布式计算。其主要采用 MapReduce 分布式计算框架，包括根据 GFS 原理开发的分布式文件系统 HDFS、根据 BigTable 原理开发的数据存储系统 HBase 以及资源管理系统 YARN。

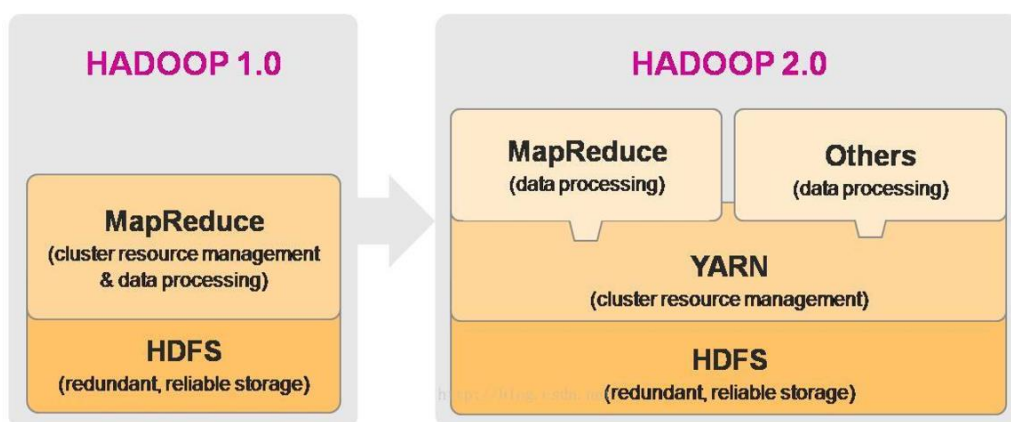


图 1 Hadoop 基本框架

### 2.2 Hadoop MapReduce 原理

MapReduce 最早由 Google 于 2004 年在一篇名为《MapReduce: Simplified Data Processing on Large Clusters》的论文中提出，把分布式数据处理的过程拆分为 Map 和 Reduce 两个操作函数，随后被 Apache Hadoop 参考并提供开源版本。

MapReduce 将复杂的、运行于大规模集群上的并行计算过程高度抽象到了两个函数：Map 和 Reduce，并极大地方便了分布式编程工作，其主要包含以下过程：

- 1) Map（映射）：对一些独立元素组成的列表的每一个元素进行制定的操作，可以高度并行。
- 2) Shuffle（重组）：对 Map 输出的数据会经过分区、排序、分组等动作进行重组，使得 key 相同的分在同一个分区，同一个分区被同一个 reduce 处理。
- 3) Reduce（归约）：归约过程，把若干组映射结果进行汇总并输出。

用户编写的程序分成三个部分：Mapper, Reducer, Driver(提交运行程序的客户端驱动)。需要注意的是，整个 MapReduce 程序中，数据都是以 < key, value > 键值对的形式流转的。

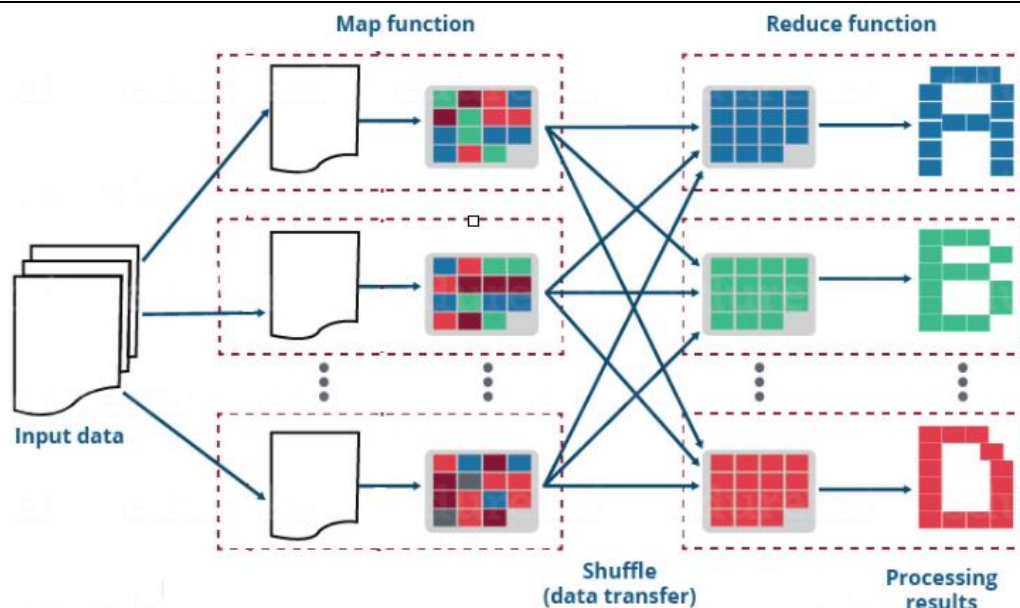


图 2 MapReduce基本流程

## 2.3 Hadoop HDFS 原理

HDFS 最初是模仿 GFS 开发的开源系统，适合存储大文件并提供高吞吐量的顺序读/写访问。其整体架构如图所示，其由 NameNode, DataNode, Secondary NameNode 以及客户端构成。

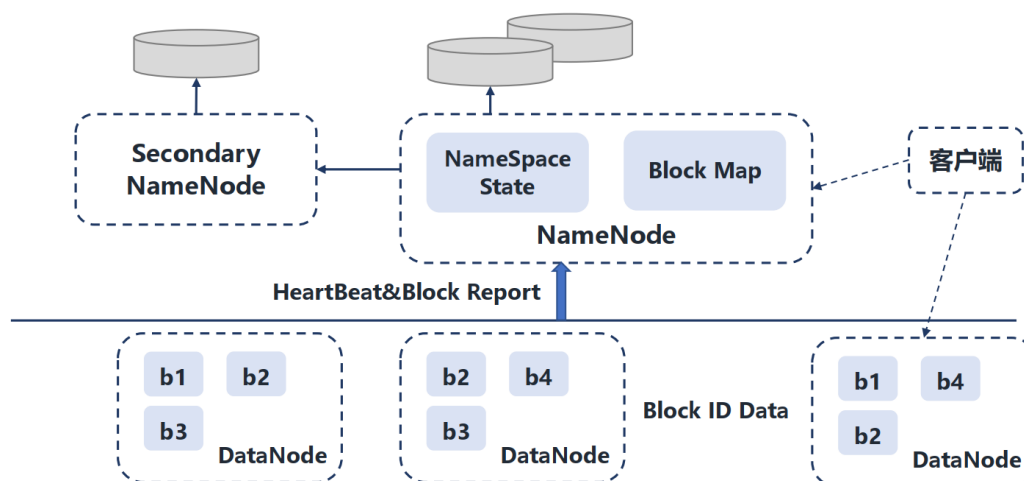


图 3 HDFS整体架构

NameNode 负责管理整个分布式文件系统的元数据，包括文件目录树结构、文件到数据块 Block 的映射关系、Block 副本及其存储位置等各种管理数据。其磁盘保存两个元数据管理文件 fsimage 和 editlog：

- 1) fsimage 是内存命名空间元数据在外存的镜像文件；
- 2) editlog 是各种元数据操作的 write-ahead-log 文件。



Secondary NameNode 提供检查点功能服务，职责是定期从 NameNode 拉取 fsimage 和 editlog 文件进行合并，形成新的 fsimage 文件并传回给 NameNode;

DataNode 负责数据块的实际存储和读/写工作，为保证数据可用性，每个 Block 以多备份的形式存储。

同时，NameNode 与 DataNode 通过短时间间隔的心跳来传递管理信息和数据信息，从而实现 DataNode 的状态监控。如果某个 DataNode 发生故障，NameNode 会将其负责的 Block 在其他 DataNode 机器增加相应备份以维护数据可用性。

## 2.4 Hadoop 的优点与局限性

Hadoop 是一个基础框架，具有低成本、高可靠、高扩展、高有效、高容错等特性，能够进行海量数据的离线处理。

Hadoop 允许用简单的编程模型在计算机集群上对大型数据集进行分布式处理。用户可以在不了解分布式底层细节的情况下，轻松地在 Hadoop 上开发和运行处理海量数据的应用程序。

同时其计算能力可以随节点数目增长保持近似于线性的增长，它的设计规模从单一服务器到数千台机器，每个服务器都能提供本地计算和存储功能，框架本身提供的是计算机集群高可用的服务，不依靠硬件来提供高可用性。

但 MapReduce 主要应用于离线作业，无法作到秒级或者是亚秒级得数据响应。且主要是针对静态数据集，不能进行流式计算。

# 第 3 章 Hadoop 分布式配置

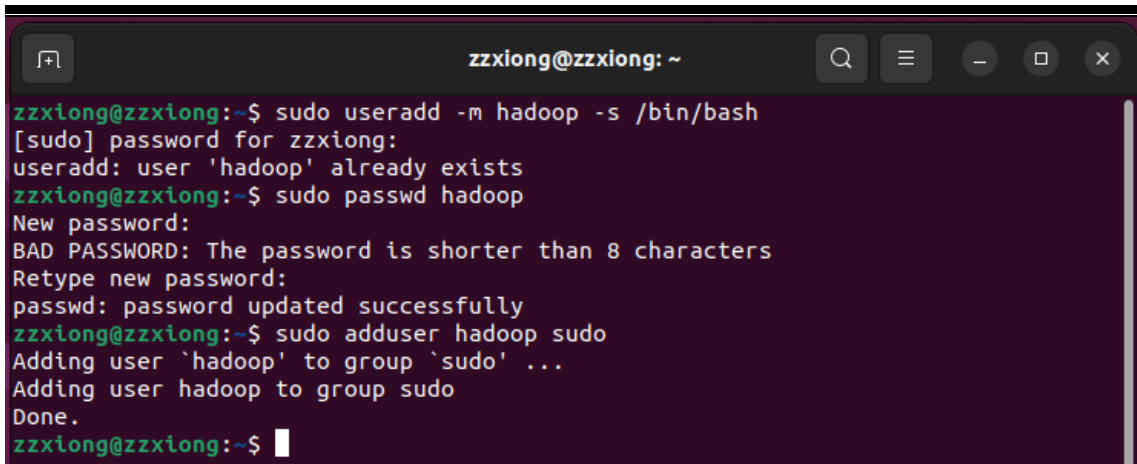
本实验使用 Vmware Workstation 17 + Ubuntu22.04 LTS 64 位操作系统作为系统环境配置 Hadoop 3.3.5。

## 3.1 前期准备(新用户创建和 SSH 配置)

### 3.1.1 新用户创建

首先在 Ubuntu 下，通过"**sudo useradd -m hadoop -s /bin/bash**"命令增加一个名为 hadoop 的用户，并使用/bin/bash 作为 shell。而后通过"**sudo passwd hadoop**"和"**sudo adduser hadoop sudo**"设置密码并为 hadoop 用户增加管理员权限。

最后注销当前用户（点击屏幕右上角的齿轮，选择注销 Log Out），返回登陆界面。在登陆界面中选择刚创建的 hadoop 用户进行登陆。



```

zzxiong@zzxiong: ~
zzxiong@zzxiong:~$ sudo useradd -m hadoop -s /bin/bash
[sudo] password for zzxiong:
useradd: user 'hadoop' already exists
zzxiong@zzxiong:~$ sudo passwd hadoop
New password:
BAD PASSWORD: The password is shorter than 8 characters
Retype new password:
passwd: password updated successfully
zzxiong@zzxiong:~$ sudo adduser hadoop sudo
Adding user 'hadoop' to group 'sudo' ...
Adding user hadoop to group sudo
Done.
zzxiong@zzxiong:~$

```

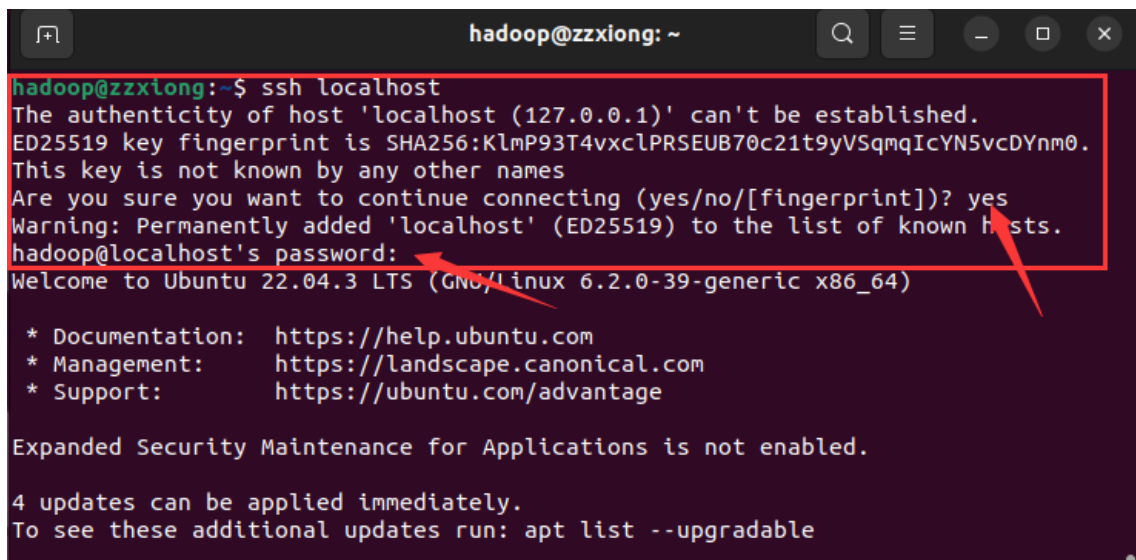
图 4 创建新用户过程截图

### 3.1.2 配置 SSH

安全外壳协议(Secure Shell, SSH), 是一种加密的网络传输协议, 可在不安全的网络中网络服务提供安全的传输环境。它通过在网络中创建安全隧道来实现 SSH 客户端和服务端之间的连接。

集群、单节点模式都需要用到 SSH 登陆, 以实现远程登陆, Ubuntu 默认已安装了 SSH client, 此外还需要通过命令"**sudo apt-get install openssh-server**"来安装 SSH server。安装后, 可以使用命令"**ssh localhost**"登陆本机。

首次登录时, 会出现下图所示提示, 按提示输入密码即可登陆到本机。



```

hadoop@zzxiong: ~
hadoop@zzxiong:~$ ssh localhost
The authenticity of host 'localhost (127.0.0.1)' can't be established.
ED25519 key fingerprint is SHA256:Klmp93T4vxclPRSEUB70c21t9yVSqmQIcYN5vcDYNm0.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'localhost' (ED25519) to the list of known hosts.
hadoop@localhost's password:
Welcome to Ubuntu 22.04.3 LTS (GNU/Linux 6.2.0-39-generic x86_64)

* Documentation:  https://help.ubuntu.com
* Management:    https://landscape.canonical.com
* Support:       https://ubuntu.com/advantage

Expanded Security Maintenance for Applications is not enabled.

4 updates can be applied immediately.
To see these additional updates run: apt list --upgradable

```

图 5 登陆提示

此时还可以利用 `ssh-keygen` 生成密钥, 并将密钥加入到授权中, 从而实现无密码的登陆。

### 3.1.3 总结

本节创建了一个新用户 `hadoop`，并配置了 `SSH`，能够无密码模拟实现本机的远程登陆。主要 `shell` 命令汇总如下：

```
1. sudo useradd -m hadoop -s /bin/bash      # 创建新用户
2. sudo passwd hadoop
3. sudo adduser hadoop sudo
4. sudo apt-get update
5. sudo apt-get install openssh-server      # 配置 SSH
6. ssh localhost
7. exit
8. cd ~/.ssh/                               # 生成密钥并加入授权
9. ssh-keygen -t rsa
10. cat ./id_rsa.pub >> ./authorized_keys
```

## 3.2 Java 环境配置

Hadoop 3.3.5 需要 `JDK` 版本在 1.8 及以上。本实验通过手动安装配置。首先将压缩格式的文件 `jdk-8u371-linux-x64.tar.gz` 下载保存 `"/home/hadoop/Desktop/"` 目录下，而后执行如下 `Shell` 命令将文件解压缩至 `"/usr/lib/jvm"` 目录下。

```
1. cd /usr/lib
2. sudo mkdir jvm
3. cd ~/Desktop
4. sudo tar -zxvf ./jdk-8u371-linux-x64.tar.gz -C /usr/lib/jvm
```

而后通过 `"sudo gedit ~/.bashrc"` 命令打开环境变量配置文件，添加以下内容配置 `java` 环境的路径。

```
1. export JAVA_HOME=/usr/lib/jvm/jdk1.8.0_371
2. export JRE_HOME=${JAVA_HOME}/jre
3. export CLASSPATH=.:${JAVA_HOME}/lib:${JRE_HOME}/lib
4. export PATH=${JAVA_HOME}/bin:$PATH
```

保存 `.bashrc` 文件，然后执行 `"source ~/.bashrc"` 命令让 `.bashrc` 文件的配置立即生效。这时，可以使用 `"java -version"` 查看是否安装成功。

如果能够在屏幕上返回如下信息，则说明安装成功：

```
hadoop@zzxiong:~$ source ~/.bashrc
hadoop@zzxiong:~$ java -version
java version "1.8.0_371"
Java(TM) SE Runtime Environment (build 1.8.0_371-b11)
Java HotSpot(TM) 64-Bit Server VM (build 25.371-b11, mixed mode)
```

图 6 Java 环境信息截图

## 3.3 Hadoop 3.3.5 安装

### 3.3.1 下载安装

同样地，首先将压缩格式的文件 `hadoop-3.3.5.tar.gz` 文件下载保存 `"/home/hadoop/Desktop/"` 目录下，而后通过以下命令将 Hadoop 安装至 `"/usr/local/"` 中。

1. `sudo tar -zxvf ~/Desktop/hadoop-3.3.5.tar.gz -C /usr/local`
2. `cd /usr/local/`
3. `sudo mv ./hadoop-3.3.5/ ./hadoop`
4. `sudo chown -R hadoop ./hadoop`

在命令行输入命令 `"/usr/local/hadoop/bin/hadoop version"` 来检查 Hadoop 是否可用，成功则会显示 Hadoop 版本信息如下：

```
hadoop@zzxiong:~/usr/local$ cd /usr/local/hadoop
./bin/hadoop version
Hadoop 3.3.5
Source code repository https://github.com/apache/hadoop.git -r 706d88266abcee09e
d78fbaa0ad5f74d818ab0e9
Compiled by stevel on 2023-03-15T15:56Z
Compiled with protoc 3.7.1
From source with checksum 6bbd9afcf4838a0eb12a5f189e9bd7
This command was run using /usr/local/hadoop/share/hadoop/common/hadoop-common-3
.3.5.jar
```

图 7 Hadoop 版本信息截图

Hadoop 附带了丰富的例子（在 `"/usr/local/hadoop "` 目录下运行 `"/bin/hadoop jar" "/share/hadoop/mapreduce/hadoop-mapreduce-examples-3.3.5.jar"` 命令可以看到所有例子），包括 `wordcount`、`terasort`、`join`、`grep` 等。

### 3.3.2 伪分布式配置

Hadoop 默认模式为非分布式模式（即单 Java 进程），无需进行其他配置即可运行。同时，Hadoop 也可以在单节点上以伪分布式的方式运行，即 Hadoop 进程以分离的 Java 进程来运行，节点既作为 `NameNode` 也作为 `DataNode`，而进程读取的是 HDFS 中的文件。

进行伪分布式配置需要修改 2 个配置文件 `core-site.xml` 和 `hdfs-site.xml` (xml 格式下，每个配置以声明 `property` 的 `name` 和 `value` 的方式来实现)。具体包括配置 `fs.defaultFS` 和 `dfs.replication`、临时目录参数 `hadoop.tmp.dir` 和节点临时目录参数 `dfs.namenode.name.dir` 和 `dfs.datanode.data.dir`。

#### a) core-site.xml

```
<configuration>
  <property>
    <name>hadoop.tmp.dir</name>
    <value>file:/usr/local/hadoop/tmp</value>
    <description>Abase for other temporary directories.</description>
  </property>
```

```

<property>
  <name>fs.defaultFS</name>
  <value>hdfs://localhost:9000</value>
</property>
</configuration>

```

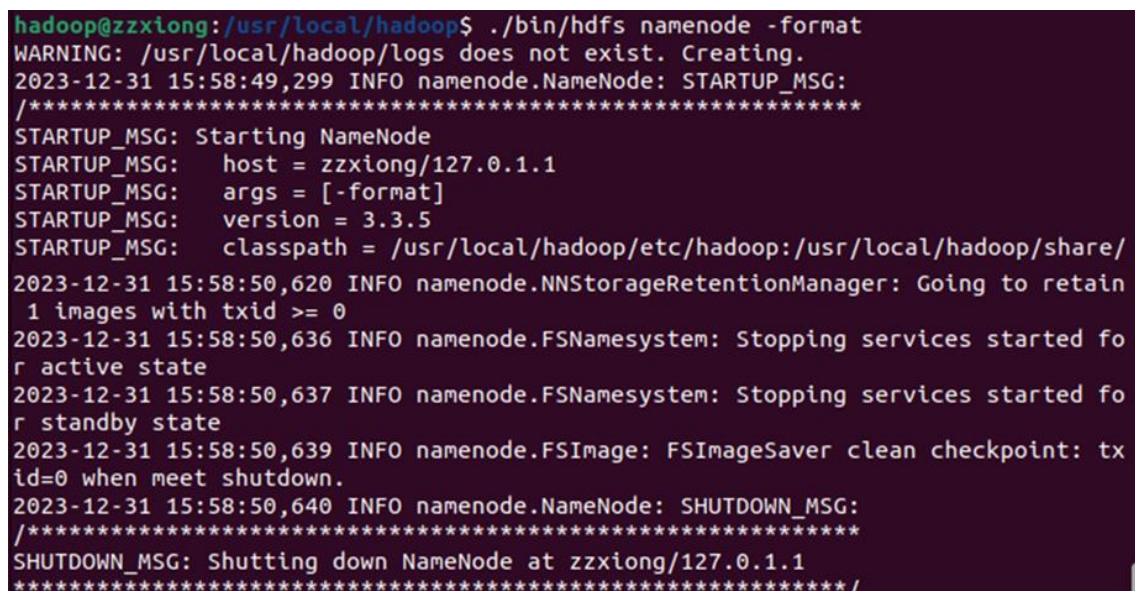
### b) hdfs-site.xml

```

<configuration>
  <property>
    <name>dfs.replication</name>
    <value>1</value>
  </property>
  <property>
    <name>dfs.namenode.name.dir</name>
    <value>file:/usr/local/hadoop/tmp/dfs/name</value>
  </property>
  <property>
    <name>dfs.datanode.data.dir</name>
    <value>file:/usr/local/hadoop/tmp/dfs/data</value>
  </property>
</configuration>

```

配置完成后，执行命令"**./bin/hdfs namenode -format**"以进行 NameNode 的格式化，具体返回信息如下：



```

hadoop@zzxiong:/usr/local/hadoop$ ./bin/hdfs namenode -format
WARNING: /usr/local/hadoop/logs does not exist. Creating.
2023-12-31 15:58:49,299 INFO namenode.NameNode: STARTUP_MSG:
/*****
STARTUP_MSG: Starting NameNode
STARTUP_MSG:   host = zzxiong/127.0.1.1
STARTUP_MSG:   args = [-format]
STARTUP_MSG:   version = 3.3.5
STARTUP_MSG:   classpath = /usr/local/hadoop/etc/hadoop:/usr/local/hadoop/share/
2023-12-31 15:58:50,620 INFO namenode.NNStorageRetentionManager: Going to retain
  1 images with txid >= 0
2023-12-31 15:58:50,636 INFO namenode.FSNamesystem: Stopping services started fo
r active state
2023-12-31 15:58:50,637 INFO namenode.FSNamesystem: Stopping services started fo
r standby state
2023-12-31 15:58:50,639 INFO namenode.FSImage: FSImageSaver clean checkpoint: tx
id=0 when meet shutdown.
2023-12-31 15:58:50,640 INFO namenode.NameNode: SHUTDOWN_MSG:
/*****
SHUTDOWN_MSG: Shutting down NameNode at zzxiong/127.0.1.1
*****/

```

图 8 NameNode初始化返回信息截图（省略冗余信息）

接着执行"**./sbin/start-dfs.sh**"脚本文件开启 NameNode 和 DataNode 守护进程。通过命令"**jps**"来判断是否成功启动如下：

```
hadoop@zzxiong:/usr/local/hadoop$ jps
7824 DataNode
8019 SecondaryNameNode
7684 NameNode
8902 Jps
```

图 9 Hadoop启动检查截图

和非分布式模式下 `grep` 例子读取本地数据不同，伪分布式读取的则是 HDFS 上的数据。要使用 HDFS，首先需要通过 `./bin/hdfs dfs -mkdir -p /user/hadoop` 在 HDFS 中创建用户目录。

下面用一个简单的例子来说明在 Hadoop 上提交编译好的 JAR 文件来伪分布式运行 MapReduce 作业。即以配置文件为输入，筛选当中符合正则表达式 `dfs[a-z.]+` 的单词并统计出现的次数，最后输出结果到 `output` 文件夹中。

1. `./bin/hdfs dfs -mkdir input`
2. `./bin/hdfs dfs -put ./etc/hadoop/*.xml input`
3. `./bin/hadoop jar ./share/hadoop/mapreduce/hadoop-mapreduce-examples-3.3.5.jar grep input output 'dfs[a-z.]+'`
4. `./bin/hdfs dfs -cat output/*`
5. `./bin/hdfs dfs -get output ./output`
6. `cat ./output/*`

执行成功后如下所示，输出了作业的相关信息，输出的结果是符合正则的单词及出现次数。

```
WRONG_REDUCE=0
File Input Format Counters
  Bytes Read=219
File Output Format Counters
  Bytes Written=77
hadoop@zzxiong:/usr/local/hadoop$ ./bin/hdfs dfs -cat output/*
1 dfsadmin
1 dfs.replication
1 dfs.namenode.name.dir
1 dfs.datanode.data.dir
```

图 10 测试结果截图

需要注意在 Hadoop 运行程序时，输出目录不能存在，否则会提示错误 `"org.apache.hadoop.mapred.FileAlreadyExistsException: Output directory hdfs://localhost:9000/user/hadoop/output already exists"`，因此若要再次执行，需要执行 `./bin/hdfs dfs -rm -r output` 命令删除 `output` 文件夹。

最后通过 `./sbin/stop-dfs.sh` 命令关闭 Hadoop，即可结束本次运行。

## 3.4 项目创建与导出

### 3.4.1 在 Eclipse 中创建项目



首先，启动 Eclipse，打开当前的工作空间。选择“File-->New-->Java Project”菜单，开始创建一个 Java 工程，弹出如下图所示界面。

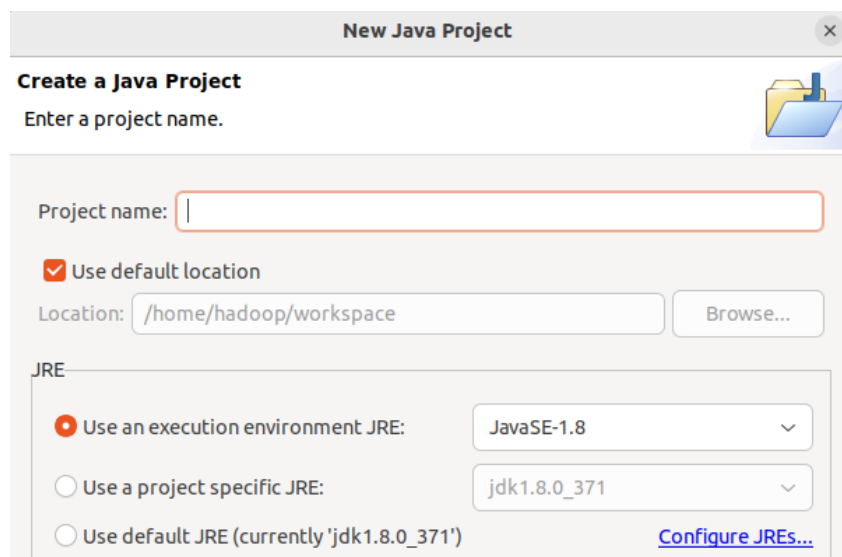


图 11 Eclipse中创建项目

在"Project name"后面输入工程名称，选中"Use default location"，让这个 Java 工程的所有文件都保存到"/home/hadoop/workspace/<工程名>"目录下。在"JRE"这个选项卡中，可以选择当前的 Linux 系统中已经安装好的 JDK，比如 jdk1.8.0\_371。然后，点击界面底部的"Next>"按钮，进入下一步的设置。

然后，需要在这个界面中加载 Java 工程所需要用到的 JAR 包，也就是 Hadoop 相关的 Java API。点击界面中的"Libraries"选项卡，然后点击界面右侧的"Add External JARs..."按钮，在弹出界面中选择需要的 JAR 包，然后点击界面右上角的"OK"按钮，就可以把选中的 JAR 包增加到当前 Java 工程中。

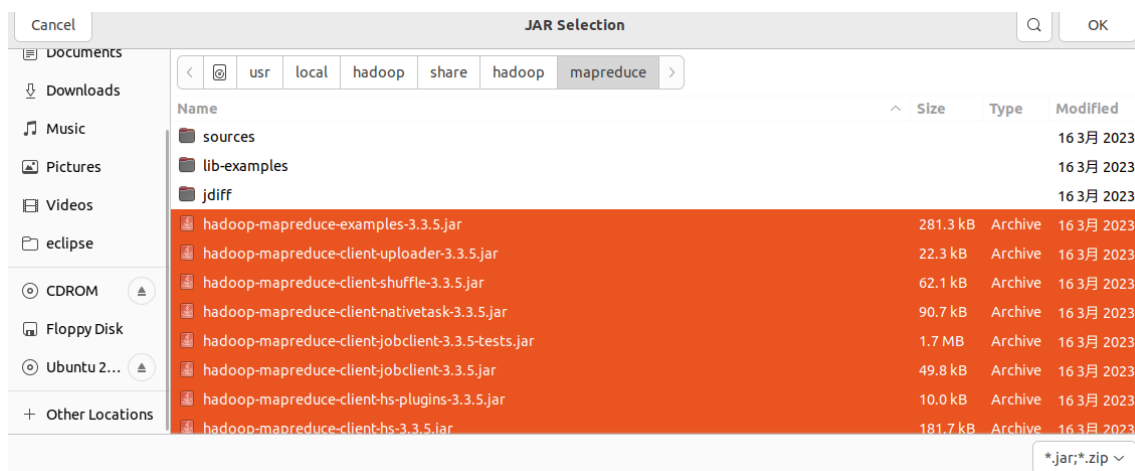


图 12 导入JAR包

而后在 Eclipse 工作界面左侧的"Package Explorer"面板中找到刚才创建好的工程，右键选择"New-->Class"菜单根据提示创建 Java 类文件进行代码编写。

### 3.4.2 编译打包程序

代码编写完成后，需要将工程打包生成 JAR 包。首先点击 Eclipse 工作界面上部的运行程序的快捷按钮，选中"Run as--> Java Application"运行代码。

而后在 Eclipse 工作界面左侧的"Package Explorer"面板中找到刚才创建好的工程，右键选择"Export--> Runnable JAR file"菜单根据提示将 JAR 包生成到指定目录下，部署到 Hadoop 平台上运行。

如下图所示，"Launch configuration"用于设置生成的 JAR 包被部署启动时运行的主类，需要在下拉列表中选择代码配置的类。在"Export destination"中需要设置 JAR 包要输出保存到哪个目录。同时在"Library handling"下面选择"Extract required libraries into generated JAR"。

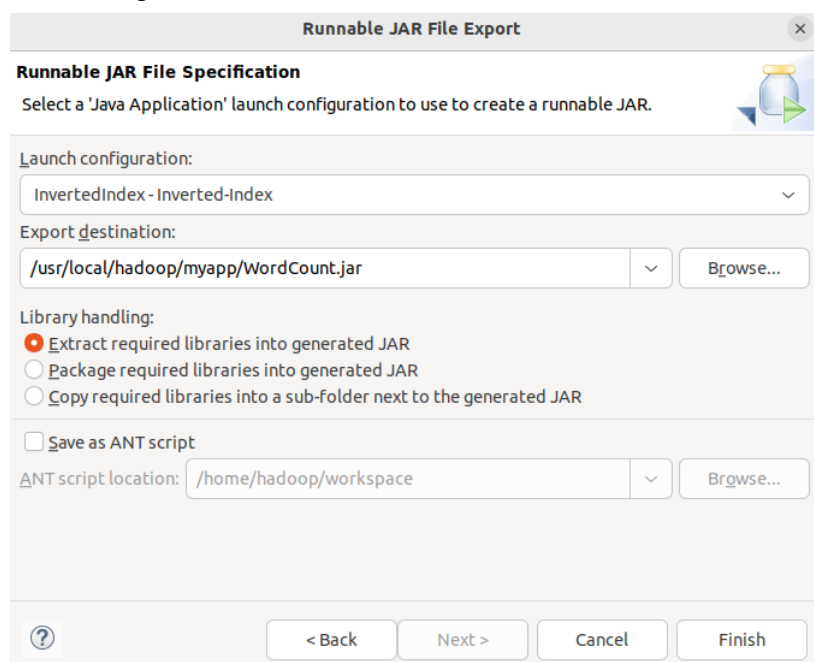


图 13 Eclipse 中项目导出

## 第 4 章 数据预处理

### 4.1 数据集简介

《纽约时报》拥有广泛的读者群，对塑造人们对时事的看法和观点，尤其是在美国塑造公共话语的方面起着重要作用。



"纽约时报"评论数据集记录了有关《纽约时报》2017年1月至5月和2018年1月至4月发表的文章上的评论的信息。月度数据分为两个csv文件：一个用于包含发表评论的文章，另一个用于评论本身。评论的csv文件总共包含超过200万条评论，有34个特征，而文章的csv文件包含超过9000篇文章，有16个特征。

本实验需要提取其中的articleID和snippet字段进行词频统计和建立倒排索引，但文章的csv文件在不同月份的列标签分布存在差异，不能简单直接的进行MapReduce处理，需要对数据进行预处理。

## 4.2 预处理过程

数据在采集和存储之后，往往还需要进行必要的加工整理后才能用于分析建模。预处理一般包括数据抽样和过滤、数据标准化和归一化、数据清洗三个步骤。

在本实验中，文章的csv文件在不同月份的列标签分布存在差异，需要标准化处理，结合实验只涉及其中的articleID和snippet字段，那么我们便将其中所有该字段提取出来即可，这样可以减少后续统计时的数据读入量。此外，由于英文中单词会出现同一词汇大小写不一致的情况，因此本实验将所有字符全部转换为小写以标准化处理。

那么总结一下，此预处理过程就是从指定的输入数据目录中读取所有csv文件，提取其中指定列的数据并进行标准化，然后将提取的数据写入输出目录下的文本文件当中。具体伪代码如下：

---

### PSEUDO-CODE 1 PreProcess(预处理)

---

```
1: input inputFolder ← 输入数据目录, outputFolder ← 输出目录;
2: for inputFolder 中每一个 csv 文件:
3:   获取列索引(articleIDIndex, snippetIndex);
4:   for 此 csv 文件中每条记录:
5:     if 记录长度 > max(articleIDIndex, snippetIndex) then:
6:       获取articleID从记录的articleIDIndex;
7:       获取snippet从记录的snippetIndex并转换为小写;
8:       向outputFolder中写入(articleID + "\t" + snippet + "\n");
9:     end if
10:  end for
11: end for
```

---

## 4.3 预处理结果

处理结束后，得到的数据如下所示：



图 14 预处理结果

## 第 5 章 词频统计

### 5.1 统计过程

在 Hadoop 中，输入文件通常会通过 InputFormat 被分成一系列的逻辑分片，分片是输入文件的逻辑划分，每个分片由一个 Mapper 处理。

本实验中，WordCount 通过 MapReduce 统计 snippet 字段中每个单词出现的总次数。程序主要包括 Mapper, Reducer, Driver 三个部分。

自定义的 Mapper 和 Reducer 都要继承各自的父类。Mapper 中的业务逻辑写在 map()方法中，Reducer 的业务逻辑写在 reduce()方法中。整个程序还需要一个 Driver 来进行提交，提交的是一个描述了各种必要信息的 job 对象。

程序总体流程如下图所示。

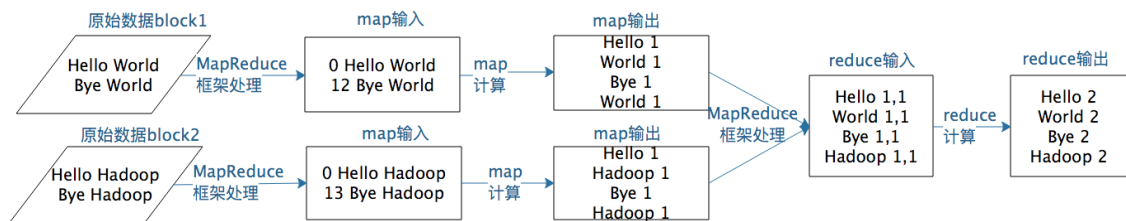


图 15 WordCount 计算流程

#### 5.1.1 Mapper

Mapper 的主要任务是处理输入分片并生成中间键值对，这些键值对将被传递给 Reducer 进行进一步处理，也就是对应的 Map 的过程。

在本实验中，Mapper 需要将这行文本中的单词提取出来，针对每个单词输出一个<word, 1>的<Key, Value>对。之后 MapReduce 会对这些<word,1>进行排序重组，将相同的 word 放在一起，形成<word, [1,1,1,1,1,1,1...]>的<Key,Value >结构并传递给 Reducer。

### 5.1.2 Reducer

Reducer 则以中间键值对为输入，将其按照键进行分组，并将每个组的值按一定规则合并成最终的输出。

注意在此阶段前，Hadoop 框架会自行将中间键值对经过默认的排序分区分组，Key 相同的单词会作为一组数据构成新的<Key, Value>对。

在本实验中，Reducer 将集合里的 1 求和，再将单词（word）与这个和（sum）组成一个<Key, Value>，也就是<word, sum>输出。每一个输出就是一个单词和它的词频统计总和了。

### 5.1.3 Driver

Driver 是一个程序的主入口，负责配置和启动整个 MapReduce 任务。Driver 类通常包含了整个 MapReduce 作业的配置信息、作业的输入路径、输出路径等信息，并启动 MapReduce 作业的执行。

### 5.1.4 总结

该程序基于 Hadoop MapReduce 框架实现了简单的单词计数功能，适用于大规模文本数据的并行处理。

---

#### PSEUDO-CODE 2 WordCount(词频统计)

---

```
/* Map 函数，处理每一行的文本 */
1: input < Key, Value >;           //Value 使用 Text 类型表示文本行
2: 从文本中提取文档 ID 和实际文本内容 snippet;
3: 使用空格、单引号和破折号作为分隔符，将文本 snippet 分词;
4: for 文本 snippet 中的每个单词:
5:   去除特殊字符后将<word,1>写入 context, 发射给 Reducer;
6: end for
/* Reduce 函数，处理相同键的所有值 */
1: input < Key, Value >, sum← 0;    //来自 Map 的<word,[1,1,1...]>
2: for Value 的每个 1:
3:   累加计数 sum += 1;
```

---

## 5.2 统计结果

统计结束后，下载 HDFS 上得到的数据如下所示：



图 16 词频统计结果

# 第 6 章 建立倒排索引

## 6.1 倒排索引概念

倒排索引（Inverted Index）是信息检索领域中的一种数据结构，它是一种反转（倒排）文档-词项关系的数据结构，以支持通过词项来查找相关文档。

在倒排索引中，每个词项都被映射到包含该词项的文档列表。并且在实际应用中，还需要给每个文档添加一个权值，用以指出每个文档与搜索内容的相关度。

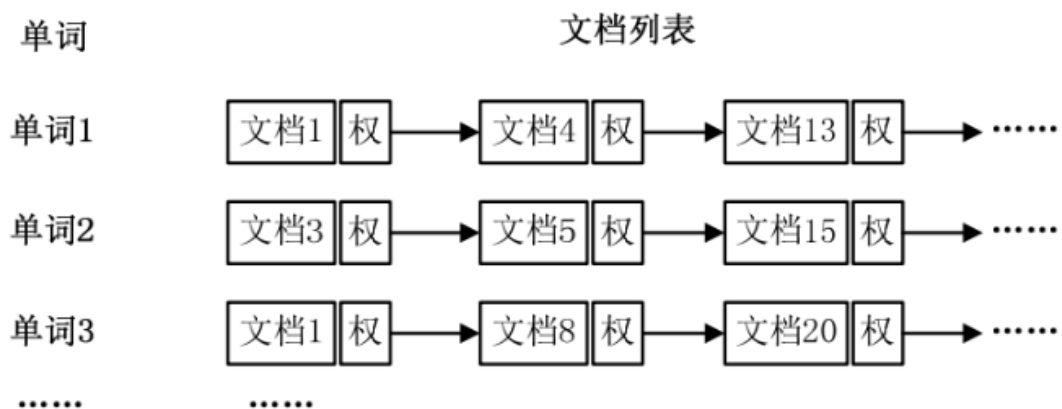


图 17 倒排索引示意图

与倒排索引相对应的是正向索引，即文档-词项关系的数据结构。当用户发起查询关键词时，需要扫描索引库中的所有文档，找出所有包含关键词的文档，在检索过程中效率较低。

倒排索引在搜索引擎等信息检索系统中得到广泛应用。通过倒排索引，检索系统能够以更高效的方式在大规模文档集合中定位包含特定词项的文档，从而为用户提供快速准确的搜索结果。

## 6.2 倒排索引建立过程

在本实验中，在上一章 WordCount 的基础上，使用词频作为权重，即记录单词在文档中出现的次数。因此相较于上一章，需要统计信息增加了文档 ID，即统计单词、文档 ID 和词频。因此大致流程与上一章基本相同，本实验在 WordCount 的基础上进行简单改动即可。

由于需要统计单词的来源文章，因此对于 Mapper 需要将生成的中间键值对由<word,1>改为 <word,articleID>;

而 Reducer 和之前不同的是，需要根据排序重组后的中间键值对<word,[articleID1, articleID2, ...]>建立一个 HashMap，用于存储每个 articleID 对应的出现次数。整体流程伪代码如下：

---

### PSEUDO-CODE 3 Inverted Index(倒排索引)

---

```
/* Map 函数，处理每一行的文本 */
1: input < Key, Value >;           //Value 使用 Text 类型表示文本行
2: 从文本中提取文档 ID 和实际文本内容 snippet;
3: 使用空格、单引号和破折号作为分隔符，将文本 snippet 分词;
4: for 文本 snippet 中的每个单词:
5:   去除特殊字符后将<word, articleID>写入 context，发射给 Reducer;
6: end for
/* Reduce 函数，处理相同键的所有值 */
1: input < Key, Value >;           //来自 Map 的<word,[ articleID1, articleID2,...]>
2: 创建一个 HashMap，用于存储每个 articleID 对应的出现次数;
3: for Value 的每个 articleID:
4:   if articleID in HashMap then map(articleID)++;
5:   else 在 HashMap 中添加该 articleID，map(articleID) ← 1;
6: end for
7: 将Key 和 HashMap 中的 articleID 和对应出现次数拼接并输出;
```

---

## 6.3 索引建立结果

统计结束后，下载 HDFS 上得到的数据如下所示：



```
10521 missing 5a744ab410f40f00018bf294:2 591221447c459f24986de95e:1 5a832bf810f40f00018c8c1c:1 5a995ff9410cf7000162ee3a:1 5a9c469a47de81a901209d6f:1 58db71657c459f24986d7208:1 5ab3718847de81a9012151ac:1
10522 mission 5899009095d0e0392607ebbb:1 58f9b4c10f40f00018c28c4:1 592005b07c459f24986deb37:1 58b61d2095d0e024902fc8c0:1 58a5e10e95d0e0247463ebc:1 5a738ce210f40f00018bef2b:1
10523 missions 58b740cd7c459f24986dcb3:1 5ab12e5b47de81a9012134fa:1 5ac372d0808401528a2a163f:1 590c27ec7c459f24986ddca:1
10524 missing 5a70bec9060401528a2901aa:1 5ab12e5b47de81a9012134fa:1 5ac372d0808401528a2a163f:1 590c27ec7c459f24986ddca:1
10525 missing 5a9ab57a10f40f00018c32fa:1 5aaaab947de81a90121096a:1 5ad084770608401528a2a1d5:1 5ad7c2e0608401528a2a9773:1 5ad071ab0608401528a2a87c5:1 5a66f99010f40f00018bd471:1
10526 missing 58b740cd7c459f24986dcb3:1 5ab12e5b47de81a9012134fa:1 5ac372d0808401528a2a163f:1 590c27ec7c459f24986ddca:1
10527 missing 5a70bec9060401528a2901aa:1 5ab12e5b47de81a9012134fa:1 5ac372d0808401528a2a163f:1 590c27ec7c459f24986ddca:1
10528 missing 58b740cd7c459f24986dcb3:1 5ab12e5b47de81a9012134fa:1 5ac372d0808401528a2a163f:1 590c27ec7c459f24986ddca:1
10529 missing 5a70bec9060401528a2901aa:1 5ab12e5b47de81a9012134fa:1 5ac372d0808401528a2a163f:1 590c27ec7c459f24986ddca:1
10530 missing 58b740cd7c459f24986dcb3:1 5ab12e5b47de81a9012134fa:1 5ac372d0808401528a2a163f:1 590c27ec7c459f24986ddca:1
10531 missing 58b740cd7c459f24986dcb3:1 5ab12e5b47de81a9012134fa:1 5ac372d0808401528a2a163f:1 590c27ec7c459f24986ddca:1
10532 missing 58b740cd7c459f24986dcb3:1 5ab12e5b47de81a9012134fa:1 5ac372d0808401528a2a163f:1 590c27ec7c459f24986ddca:1
10533 missing 58b740cd7c459f24986dcb3:1 5ab12e5b47de81a9012134fa:1 5ac372d0808401528a2a163f:1 590c27ec7c459f24986ddca:1
10534 missing 58b740cd7c459f24986dcb3:1 5ab12e5b47de81a9012134fa:1 5ac372d0808401528a2a163f:1 590c27ec7c459f24986ddca:1
10535 missing 58b740cd7c459f24986dcb3:1 5ab12e5b47de81a9012134fa:1 5ac372d0808401528a2a163f:1 590c27ec7c459f24986ddca:1
10536 missing 58b740cd7c459f24986dcb3:1 5ab12e5b47de81a9012134fa:1 5ac372d0808401528a2a163f:1 590c27ec7c459f24986ddca:1
10537 missing 58b740cd7c459f24986dcb3:1 5ab12e5b47de81a9012134fa:1 5ac372d0808401528a2a163f:1 590c27ec7c459f24986ddca:1
10538 missing 58b740cd7c459f24986dcb3:1 5ab12e5b47de81a9012134fa:1 5ac372d0808401528a2a163f:1 590c27ec7c459f24986ddca:1
10539 missing 58b740cd7c459f24986dcb3:1 5ab12e5b47de81a9012134fa:1 5ac372d0808401528a2a163f:1 590c27ec7c459f24986ddca:1
10540 missing 58b740cd7c459f24986dcb3:1 5ab12e5b47de81a9012134fa:1 5ac372d0808401528a2a163f:1 590c27ec7c459f24986ddca:1
10541 missing 58b740cd7c459f24986dcb3:1 5ab12e5b47de81a9012134fa:1 5ac372d0808401528a2a163f:1 590c27ec7c459f24986ddca:1
10542 missing 58b740cd7c459f24986dcb3:1 5ab12e5b47de81a9012134fa:1 5ac372d0808401528a2a163f:1 590c27ec7c459f24986ddca:1
10543 missing 58b740cd7c459f24986dcb3:1 5ab12e5b47de81a9012134fa:1 5ac372d0808401528a2a163f:1 590c27ec7c459f24986ddca:1
10544 missing 58b740cd7c459f24986dcb3:1 5ab12e5b47de81a9012134fa:1 5ac372d0808401528a2a163f:1 590c27ec7c459f24986ddca:1
10545 missing 58b740cd7c459f24986dcb3:1 5ab12e5b47de81a9012134fa:1 5ac372d0808401528a2a163f:1 590c27ec7c459f24986ddca:1
10546 missing 58b740cd7c459f24986dcb3:1 5ab12e5b47de81a9012134fa:1 5ac372d0808401528a2a163f:1 590c27ec7c459f24986ddca:1
10547 missing 58b740cd7c459f24986dcb3:1 5ab12e5b47de81a9012134fa:1 5ac372d0808401528a2a163f:1 590c27ec7c459f24986ddca:1
10548 missing 58b740cd7c459f24986dcb3:1 5ab12e5b47de81a9012134fa:1 5ac372d0808401528a2a163f:1 590c27ec7c459f24986ddca:1
10549 missing 58b740cd7c459f24986dcb3:1 5ab12e5b47de81a9012134fa:1 5ac372d0808401528a2a163f:1 590c27ec7c459f24986ddca:1
10550 missing 58b740cd7c459f24986dcb3:1 5ab12e5b47de81a9012134fa:1 5ac372d0808401528a2a163f:1 590c27ec7c459f24986ddca:1
10551 missing 58b740cd7c459f24986dcb3:1 5ab12e5b47de81a9012134fa:1 5ac372d0808401528a2a163f:1 590c27ec7c459f24986ddca:1
10552 missing 58b740cd7c459f24986dcb3:1 5ab12e5b47de81a9012134fa:1 5ac372d0808401528a2a163f:1 590c27ec7c459f24986ddca:1
```

图 18 倒排索引建立结果

## 第 7 章 结论

### 7.1 实验总结与收获

本次实验实现了对"纽约时报"评论的数据集特定列的数据清洗、词频统计以及建立倒排索引。在此过程中学习熟悉了分布式计算平台 Hadoop MapReduce 的实际使用与调试，加深了对课程内容并行计算模型以及大数据系统中分布式文件系统等的理解。

同时，通过本次实验，熟悉了基本的 MapReduce 编程方法，能够使用 JAVA 语言和 MapReduce 解决一些常见的数据处理问题，包括数据去重、数据排序和数据挖掘等。

最后，通过对本课程的学习以及实验的切身体会，对大数据场景下的问题分析和建模有了新的理解，体会到了在海量的数据背景下，好的算法与好的系统相互"配合"对实际问题解决带来的帮助。

### 7.2 不足与改进之处

本实验使用使用空格、单引号和破折号作为分隔符，将文本 snippet 分词，对一些特殊的单词比如说"it's, pre-tuning"等复合单词的处理仍不够细致，可以使用正则表达式的方案进一步细化匹配。

同时，预处理过程可以通过重写 InputFormat 和 RecordReader 来适应 csv 文件间列数据分布不同的情况，将预处理过程交由 MapReduce 框架加速处理。

## 参考文献

- [1] Hadoop3.3.5 安装教程\_单机/伪分布式配置\_Hadoop3.3.5/Ubuntu22.04(20.04/18.04/16.04)\_厦大数据实验室博客 <https://dblab.xmu.edu.cn/blog/4193/>
- [2] MapReduce 编程实践(Hadoop3.3.5)\_厦大数据实验室博客 <https://dblab.xmu.edu.cn/blog/4289/>
- [3] 信息检索——简单易懂的倒排索引（原理+例子）[https://blog.csdn.net/qq\\_43403025/article/details/114779166](https://blog.csdn.net/qq_43403025/article/details/114779166)
- [4] 林子雨.大数据技术原理与应用[M].人民邮电出版社,2017.
- [5] 张俊林.大数据日知录[M].电子工业出版社,2014.