

# 文档的倒排索引与二级索引实现方案

## 小组成员

组员姓名	学号	教学班
洪子翔	1820211062	07812101
何秉翰	1820211061	07812101
陈晓如	1820211064	07812101
杨珍奇	1820211050	07832101
金泰显	1820191052	07832101

## 1. 实验要求

运用 MapReduce 算法计算，构建一个倒排索引，将倒排索引存储在 HBase 中。

## 2. 数据准备

下载数据之后，按照句子数量（例如 10000 个句子）构成一个文件，形成一个拥有 940 个文件的集合。此处使用 Python 脚本程序来实现文件分割的工作。

## 3. 环境的安装与配置

### Hadoop 集群搭建

本次实验搭建的环境版本

VMware WorkStation 16 Pro

hadoop-3.3.6

hbase-2.5.5

jdk1.8.0\_241

zookeeper-3.8.2

CentOS Linux release 7.7.1908 (Core)

Hadoop 集群分布式安装

集群规划

主机	hadoop102	hadoop103	hadoop104
角色	NameNode DataNode  NodeManager  JobHistoryServer  QuorumPeerMain  HMaster HRegionServer	DataNode  ResourceManager NodeManager  QuorumPeerMain  HRegionServer	SecondaryNameNode DataNode  NodeManager  QuorumPeerMain  HRegionServer

集群节点设置

- 1. 修改主机名
- 2. host 映射
- 3. 关闭防火墙
- 4. 开启 ssh 免密登录

Hadoop 安装

- 1. 下载并解压 Hadoop 安装包：tar -zxvf hadoop-3.3.6.tar.gz -C /opt/module/
- 2. 修改以下配置文件：hadoop-env.sh、core-site.xml、hdfs-site.xml、mapred-site.xml、yarn-site.xml、workers。因为配置文件很多，要修改的内容也很多，故不再把这些配置文件的所有内容在此列出
- 3. 添加 hadoop 到环境变量

4. 首次启动之前格式化 namenode

## HBase 安装

### ZooKeeper 搭建

1. 下载并解压 Zookeeper 安装包: `tar -zxvf apache-zookeeper-3.8.2-bin.tar.gz -C /opt/module/`
2. 进入 zookeeper 的安装目录下创建 zkData 目录, 存放我们的数据: `mkdir zkData`
3. 进入 conf 目录, 把 zoo\_sample.cfg 改名为 zoo.cfg :

```
cd conf/           //进入 conf 目录
```

```
cp zoo_sample.cfg zoo.cfg    //复制一份, 名为 zoo.cfg
```

4. 修改 zoo.cfg 文件: `vim zoo.cfg`
5. 进入 zoo.cfg 文件添加和修改 dataDir 和 server node:

```
dataDir=/opt/module/zookeeper-3.8.2/zkData    //更改 Data 的  
Directory
```

```
server.2=hadoop102:2888:3888    //更改 ServerNode
```

```
server.3=hadoop103:2888:3888    //更改 ServerNode
```

```
server.4=hadoop104:2888:3888    //更改 ServerNode
```

6. 配置 myid, 设置当前服务器的编号 :

```
cd /opt/module/zookeeper-3.8.2/zkData/
```

```
echo 2 > myid
```

7. 使用集群分发脚本 xsync, 将 zookeeper 文件夹分发到集群上的其他节点

```
//集群分发到 hadoop102, hadoop103, hadoop104
```

```
xsync /opt/module/zookeeper-3.8.2
```

## Hbase 搭建

1. 下载并解压 hbase: `tar- zxvf hbase-2.5.5-bin.tar.gz -C /opt/module/`
2. 配置: `cd /opt/module/hbase-2.5.5/conf`
3. 修改 hbase-env.sh : `cp hbase-env.sh hbase-env.sh.bak`

4. 告诉 hbase 使用外部的 zookeeper :

```
export JAVA_HOME=/opt/module/jdk1.8.0_241 ,
```

```
export HBASE_MANAGES_ZK=false
```

5. 备份并修改 conf 下 hbase-site.xml 为如下内容:

```
<property>
```

```
<name>hbase.cluster.distributed</name>
```

```
<value>true</value>
```

```
</property>
```

```
<property>
```

```
<name>hbase.zookeeper.quorum</name>
```

```
<value>hadoop102,hadoop103,hadoop104</value>
```

```
</property>
```

```
<property>
```

```
<name>hbase.zookeeper.property.dataDir</name>
```

```
<value>/opt/module/zookeeper-3.8.2/zkData</value>
```

```
</property>
```

```
<property>
```

```
<name>hbase.rootdir</name>

<value>hdfs://hadoop102:8020/hbase</value>

</property>

<property>

  <name>hbase.wal.provider</name>

  <value>filesystem</value>

</property>

<property>

  <name>hbase.server.keyvalue.maxsize</name>

  <value>57671680</value>

</property>

<property>

  <name>hbase.client.keyvalue.maxsize</name>

  <value>57671680</value>

</property>
```

6. 配置 conf 路径下 regionserver，添加内容：

```
hadoop102  #配置 conf 路径下 regionserver，添加内容
hadoop103  #配置 conf 路径下 regionserver，添加内容
hadoop104  #配置 conf 路径下 regionserver，添加内容
```

7. 将使用集群分发脚本 xsync，将 hbase 文件夹分发到集群上的其他节点

```
//集群分发到 hadoop102, hadoop103, hadoop104
```

```
xsync /opt/module/hbase-2.5.5
```

## Shell 脚本程序

在开发的过程中，为了轻松自动化地执行各种常规任务，提高工作效率，我们团队一共编写了 4 个方便且快捷的 Shell 脚本程序，它们分别是：

### 1. 集群分发脚本：xsync

作用：循环复制文件到所有节点的相同目录下

```
#!/bin/bash

#1. 判断参数个数
if [ $# -lt 1 ]
then
    echo Not Enough Arguement!
    exit;
fi

#2. 遍历集群所有机器
for host in hadoop102 hadoop103 hadoop104
do
    echo ===== $host =====
    #3. 遍历所有目录，挨个发送

    for file in $@
    do
        #4. 判断文件是否存在
        if [ -e $file ]
        then
            #5. 获取父目录
            pdir=$(cd -P $(dirname $file); pwd)

            #6. 获取当前文件的名称
            fname=$(basename $file)
            ssh $host "mkdir -p $pdir"
            rsync -av $pdir/$fname $host:$pdir
        else
            echo $file does not exists!
        fi
    done
done
```

### 2. Hadoop 集群启停脚本（包含 HDFS、YARN、HistoryServer）：myhadoop.sh

```
#!/bin/bash

if [ $# -lt 1 ]
then
    echo "No Args Input..."
    exit ;
fi

case $1 in
"start")
    echo "===== 启动 hadoop 集群 ===== "

    echo "----- 启动 hdfs ----- "
    ssh hadoop102 "/opt/module/hadoop-3.3.6/sbin/start-dfs.sh"
```

```

echo " ----- 启动 yarn ----- "
ssh hadoop103 "/opt/module/hadoop-3.3.6/sbin/start-yarn.sh"
echo " ----- 启动 historyserver ----- "
ssh hadoop102 "/opt/module/hadoop-3.3.6/bin/mapred --daemon start historyserver"
;;
"stop")
echo " ===== 关闭 hadoop 集群 ===== "

echo " ----- 关闭 historyserver ----- "
ssh hadoop102 "/opt/module/hadoop-3.3.6/bin/mapred --daemon stop historyserver"
echo " ----- 关闭 yarn ----- "
ssh hadoop103 "/opt/module/hadoop-3.3.6/sbin/stop-yarn.sh"
echo " ----- 关闭 hdfs ----- "
ssh hadoop102 "/opt/module/hadoop-3.3.6/sbin/stop-dfs.sh"
;;
*)
echo "Input Args Error..."
;;
esac

```

3. Zookeeper 集群启动停止脚本: zk.sh

```

#!/bin/bash

case $1 in
"start") {
for i in hadoop102 hadoop103 hadoop104
do
echo ----- zookeeper $i 启动 -----
ssh $i "/opt/module/zookeeper-3.8.2/bin/zkServer.sh start"
done
}
;;
"stop") {
for i in hadoop102 hadoop103 hadoop104
do
echo ----- zookeeper $i 停止 -----
ssh $i "/opt/module/zookeeper-3.8.2/bin/zkServer.sh stop"
done
}
;;
"status") {
for i in hadoop102 hadoop103 hadoop104
do
echo ----- zookeeper $i 状态 -----
ssh $i "/opt/module/zookeeper-3.8.2/bin/zkServer.sh status"
done
}
;;
esac

```

4. 查看三台服务器 Java 进程脚本: jpsall

```

#!/bin/bash

for host in hadoop102 hadoop103 hadoop104
do
echo ===== $host =====
ssh $host jps

```

done

## 4. 算法与实现

### 文件分割

```
fileInput = "D:\\bigData\\sentences\\sentences.txt"
root = "D:\\PythonProject\\splitSentencesFile\\sentencesMFile"

SPLIT_NUM = 10000

f = open(fileInput)
lines = f.readlines()

fileId = 1
outputFileName = str(fileId).zfill(3) + ".txt"
outputFilePath = root + "\\ " + outputFileName

output = open(outputFilePath, 'a')

sentencesNum = 1
row = 1

for line in lines:

    output.write(line)

    if row % SPLIT_NUM == 0 and sentencesNum != len(lines):

        print(f"file_{fileId} has {row} rows, Done!")

        output.close()

        fileId += 1
        outputFileName = str(fileId).zfill(3) + ".txt"
        outputFilePath = root + "\\ " + outputFileName
```



```

        output = open(outputFilePath, 'a')

        sentencesNum += 1
        row = 1

        continue

        sentencesNum += 1
        row += 1

sentencesNum -= 1
row -= 1

print(f"file_{fileId} has {row} rows, Done!\n")

print(f"sentencesNum = {sentencesNum}, splitNum = {SPLIT_NUM}")
print(f"fileNum = {fileId}\n")

output.close()

f.close()

```

## 倒排索引

以下是代码实现的部分

InvertedIndexMapper.java

```

package com.bingh.mapreduce.invertedindex;

import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

import java.io.IOException;

```

```
public class InvertedIndexMapper extends
Mapper<LongWritable, Text, Text, LongWritable> {

    private Text outKey = new Text();
    private LongWritable outValue = new LongWritable();

    @Override
    protected void map(LongWritable key, Text value,
Mapper<LongWritable, Text, Text, LongWritable>.Context
context) throws IOException, InterruptedException {

        // 1. 获取一行内容
        String line = value.toString();

        // 2. 分割行内容为两个部分
        // 第一部分：句子编号
        // 第二部分：句子内容
        String[] linePart = line.split(" ", 2);

        // 3. 转换句子编号的类型，String -> long
        long sentenceId = Long.parseLong(linePart[0]);

        // 4. 以空格作为分隔符，切割句子内容中的各个单词，把
        切割结果存储到 words 中
        String[] words = linePart[1].split(" ");

        // 5. 遍历各个单词，封装，写出
        for (String word : words) {

            // 5.1 封装
```

```

        outKey.set(word);
        outValue.set(sentenceId);

        // 5.2 写出
        context.write(outKey, outValue);

    }

}

}

```

InvertedIndexReducer.java

```

package com.bingh.mapreduce.invertedindex;

import org.apache.hadoop.hbase.client.Mutation;
import org.apache.hadoop.hbase.client.Put;
import org.apache.hadoop.hbase.mapreduce.TableReducer;
import org.apache.hadoop.hbase.util.Bytes;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

import java.io.IOException;
import java.util.HashSet;

public class InvertedIndexReducer extends TableReducer<Text,
LongWritable, NullWritable> {

    private NullWritable outKey = NullWritable.get();

```

```
private Put outValue;

@Override
protected void reduce(Text key, Iterable<LongWritable>
values, Reducer<Text, LongWritable, NullWritable,
Mutation>.Context context) throws IOException,
InterruptedException {

    // 1. 创建一个 HashSet 对象: HashSetId, 用于存储该单词
    出现的所有句子编号
    // 集合中的句子编号都是唯一不重复的 (在同一个句子中可
    能出现多次该单词)
    HashSet<Long> HashSetId = new HashSet<>();

    // 2. 出现该单词的句子编号不重复地添加进集合
    for (LongWritable value : values) {
        HashSetId.add(value.get());
    }

    // 3. 创建一个 StringBuilder 对象: StringId, 用于拼接
    集合元素的内容
    StringBuilder StringId = new StringBuilder();

    // 3.1 插入 '(' 到 StringId
    StringId.append("(");

    // 3.2 遍历集合中的句子编号, 追加到 StringId
    for (Long sentenceId : HashSetId) {
        StringId.append(sentenceId).append(",");
    }
}
```

```

// 3.3 以 '))' 替代最后一个逗号
if (StringId.length() > 0) {
    StringId.setCharAt(StringId.length() - 1, '))');
}

// 4. 设置单词为 rowKey, 创建 Put 对象
outValue = new Put(Bytes.toBytes(key.toString()));

// 5. 指定插入的列族、列名和值
outValue.addColumn(Bytes.toBytes("info"),
Bytes.toBytes("sentenceFrequency"),
Bytes.toBytes(String.valueOf(HashSetId.size())));
outValue.addColumn(Bytes.toBytes("info"),
Bytes.toBytes("sentenceId"),
Bytes.toBytes(StringId.toString()));

// 6. 写出
// reduce 输出的 key 类型为 null, 写入 HBase 中 reduce 的
// 输出 key 并不重要, 重要的是 value, value 的数据会被写入 HBase
// 表
context.write(outKey, outValue);

}

}

```

InvertedIndexDriver.java

```

package com.bingh.mapreduce.invertedindex;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;

```

```
import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.hbase.mapreduce.TableMapReduceUtil;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import
org.apache.hadoop.mapreduce.lib.input.CombineTextInputFormat
;
import
org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.util.GenericOptionsParser;

import java.io.IOException;

public class InvertedIndexDriver {

    public static void main(String[] args) throws
IOException, InterruptedException, ClassNotFoundException {

        // 1. 创建 HBase 配置 conf
        Configuration conf = HBaseConfiguration.create();

        // 2. 设置调大客户端和服务端参数为 55MB，避免当实际插
        入的 keyValue 的大小超过默认大小 10MB 限制阈值时，发生报错
        conf.set("hbase.client.keyvalue.maxsize",
"57671680");
        conf.set("hbase.server.keyvalue.maxsize",
"57671680");

        // 3. 辨别命令行参数
```

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

// 4. 获取 job
Job job = Job.getInstance(conf);

// 5. 设置 jar 包路径
job.setJarByClass(InvertedIndexDriver.class);

// 6. 关联 mapper 和 reducer
job.setMapperClass(InvertedIndexMapper.class);
job.setReducerClass(InvertedIndexReducer.class);

// 7. 设置 map 输出的 kv 类型
job.setMapOutputKeyClass(Text.class);
job.setMapOutputValueClass(LongWritable.class);

// 8. 设置读取输入文件的格式
// 如果不设置 InputFormat, 默认使用的是
TextInputFormat.class

job.setInputFormatClass(CombineTextInputFormat.class);

// 9. 虚拟存储切片最大值设置为 12MB
CombineTextInputFormat.setMaxInputSplitSize(job,
12582912);

// 10. 根据首字符分区存储
job.setPartitionerClass(Partition.class);
job.setNumReduceTasks(36);
```

```

        // 11. 设置结果输出到 HBase 表
        // "bingh:bigdata"是已经在 HBase 中建好的表

TableMapReduceUtil.initTableReducerJob("bingh:inverted",
InvertedIndexReducer.class, job, Partition.class);

        // 12. 设置输入路径
        // 这是上传到 HDFS 上多个小文件的父目录
        FileInputFormat.addInputPath(job, new
Path(otherArgs[0]));

        // 13. 提交 job
        boolean result = job.waitForCompletion(true);

        System.exit(result ? 0 : 1);

    }

}

```

## 分区类

Partition.java

```

package com.bingh.mapreduce.invertedindex;

import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Partitioner;

import java.util.HashMap;

public class Partition extends Partitioner<Text,
LongWritable> {

```



```
HashMap<String, Integer> partition = new HashMap<>();

public Partition() {
    setPartition();
}

public void setPartition() {

    for (int i = 0; i < 10; i++) {
        String key = String.valueOf(i);
        partition.put(key, i);
    }

    for (int i = 0; i < 26; i++) {
        char letter = (char)('a' + i);
        String key = String.valueOf(letter);
        partition.put(key, i + 10);
    }

}

@Override
public int getPartition(Text text, LongWritable
longWritable, int NumPartition) {

    String singleWord = text.toString();

    String preWord = singleWord.substring(0, 1);

    return partition.get(preWord);
}
```

```
}  
  
}
```

## Mapper 阶段

-主要任务是将输入数据拆分成单词，并将每个单词与对应的句子编号（文件 id）进行关联。这个过程包括了获取输入行、分割行内容、将句子编号转换为长整型、切分句子内容中的单词，并最终遍历单词，封装成键值对并输出。

## Reducer 阶段

-核心任务是汇总相同单词的句子编号（文件 ID），并将它们有效地存储在 HBase 表中。这个过程包括了创建 HashSet 以确保句子编号的唯一性、遍历输入的句子编号、构建包含句子编号的字符串、将单词作为 rowKey 并创建 Put 对象、指定列族、列名和值，并最终将结果写入 HBase 表中。值得强调的是，我们的亮点在于使用哈希表来实现句子编号的去重，并在最初的算法中记录单词在各个句子编号中的频率、具体位置以及单词的整体出现频率，这些信息都被显现在最终的代码中。此外，我们还对生成的倒排文件进行了一些优化修改，以减小文件大小，从而提高检索速度。

## Driver 阶段

-它是整个 MapReduce 作业的主控制中心，负责配置和运行作业。这个过程包括创建 HBase 配置、设置参数、解析命令行参数、创建 Job 实例、关联 Mapper 和 Reducer、设置输入格式、配置虚拟存储切片、设置分区器、初始化 TableReducerJob、指定输入路径以及最终提交作业。

## Partition 分区器

-这是一个自定义组件，用于根据单词的首字母将数据进行分区存储，以确保相同首字母的单词被分配到相同的 Reduce 任务中。这个过程包括初始化分区映射以及实现 getPartition 方法来确定分区编号。值得特别提到的是，这个功能是在满足实验要求的基础上实现的，通过首字母分区存储文件，大大提高了检索的效率。

## 二级索引

以下是代码实现的部分

WordOffsetMapper.java

```
package WordOffset;  
  
import org.apache.hadoop.io.*;  
import org.apache.hadoop.mapreduce.*;
```

```

import java.io.IOException;

public class WordOffsetMapper extends Mapper<LongWritable, Text, Text, LongWritable> {

    //创建 Text 对象来存储单词
    private Text word = new Text();
    //创建 LongWritable 对象来存储偏移量
    private LongWritable offset = new LongWritable();

    @Override
    protected void map(LongWritable key, Text value, Context context) throws IOException,
    InterruptedException {

        //将输入的 Text 值转换为字符串
        String line = value.toString();

        //将字符串按\t 分割成两部分
        String[] parts = line.split("\t");

        //检查是否成功分割为两部分
        if (parts.length == 2) {

            //提取分割后的单词部分
            String wordText = parts[0];
            //提取分割后的单词信息部分
            String info = parts[1];

            //封装
            //将单词文本设置到 Text 对象
            word.set(wordText);
            //将偏移量设置到 LongWritable 对象
            offset.set(Long.parseLong(key.toString()));

            //将单词和偏移量写入上下文以供 Reduce 阶段使用
            context.write(word, offset);
        }
    }
}

```

WordOffsetReducer. java

```

package WordOffset;

import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;

```

```

import java.io.IOException;

public class WordOffsetReducer extends Reducer<Text, LongWritable, Text, Text> {

    //创建 Text 对象用于存储结果
    private Text result = new Text();

    @Override
    protected void reduce(Text key, Iterable<LongWritable> values, Context context) throws
IOException, InterruptedException {

        StringBuilder offsets = new StringBuilder();

        for (LongWritable offset : values) {
            offsets.append(offset.toString());
        }

        result.set(offsets.toString());

        context.write(key, result);

    }
}

```

WordOffsetDriver.java

```

package WordOffset;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WordOffsetDriver {

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

        // 创建 Hadoop 配置对象
        Configuration conf = new Configuration();

        // 创建一个新的作业
        Job job = Job.getInstance(conf, "Word Offset Job");

        // 设置主类
        job.setJarByClass(WordOffsetDriver.class);
    }
}

```

```

// 设置 Mapper 和 Reducer 类
job.setMapperClass(WordOffsetMapper.class);
job.setReducerClass(WordOffsetReducer.class);

// 设置 Mapper 的输出键值类型
job.setMapOutputKeyClass(Text.class);
job.setMapOutputValueClass(LongWritable.class);

// 设置 Reducer 的输出键值类型
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(Text.class);

// 设置输入和输出路径
FileInputFormat.addInputPath(job, new Path("D:\\bigData\\output_sentence\\part-r-00000"));
FileOutputFormat.setOutputPath(job, new Path("D:\\bigData\\output_Sentences"));

// 提交作业并等待完成
boolean success = job.waitForCompletion(true);

// 根据作业是否成功完成返回适当的退出码
System.exit(success ? 0 : 1);

}
}

```

## Mapper 阶段

将倒排索引的每一行文本中的单词和该单词在倒排索引文本中的偏移量按“\t”制表符分割后，提取单词作为键（word），偏移量作为值（offset），封装成一组键值对输出。

## Reducer 阶段

根据 Mapper 阶段输出的键值对，合并具有相同单词（key）的所有偏移量值（offsets），将它们拼接成一个字符串（result），即对于每个相同的键，将它们的值合并起来后封装成一组键值对输出。

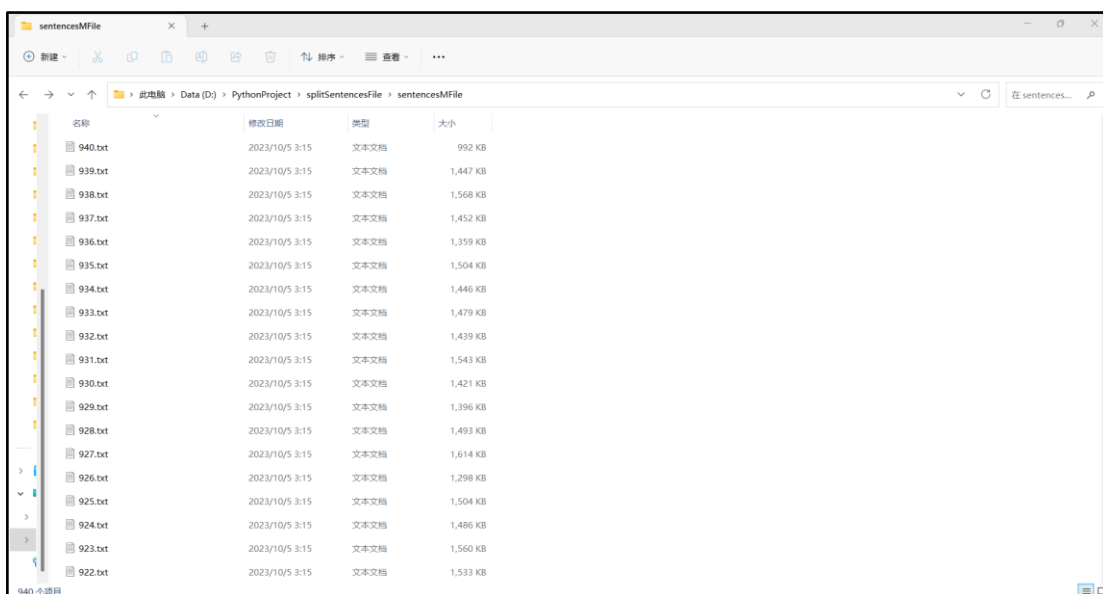
## Driver 阶段

是 MapReduce 作业必要的驱动程序，配置了 Hadoop 运行的参数和属性（Configuration）、新作业（Word Offset Job）、主类（WordOffsetDriver）、Mapper 类（WordOffsetMapper）、Reducer 类（WordOffsetReducer）、输出键值类型、输入输出路径和提交作业。

## 5. 运行结果

→ 倒排索引

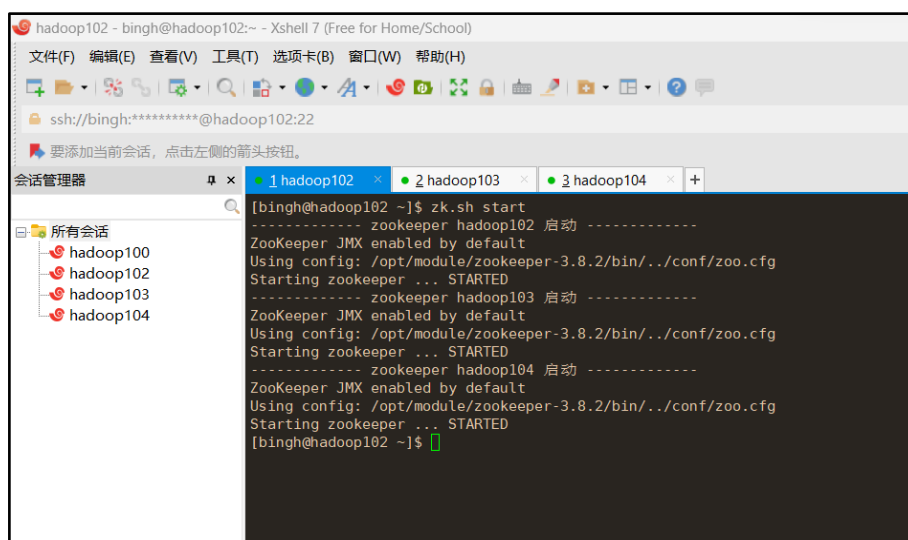
- A. 以 10000 个句子为一个小文件单位，利用 Python 脚本对源文件 sentences.txt 内容切割成 940 个小文件



## B. 启动集群

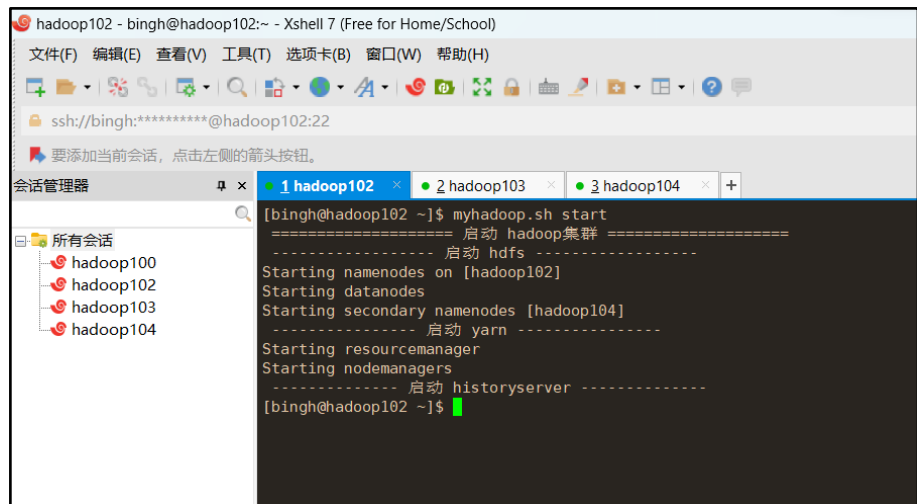
- a. 启动 Zookeeper 【执行自己编写的 shell 脚本：zk.sh】；

```
zk.sh start
```



- b. 启动 Hadoop 【执行自己编写的 shell 脚本：myhadoop.sh】；

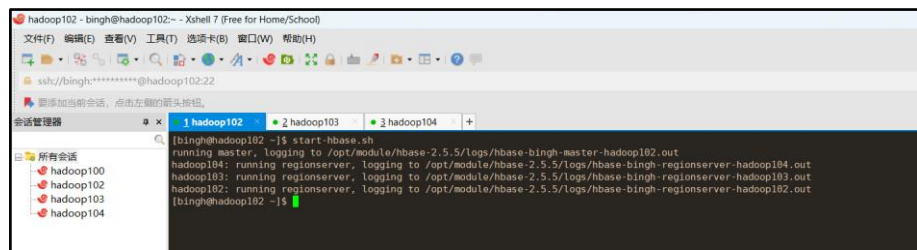
```
myhadoop.sh start
```



```
hadoop102 - bingh@hadoop102:~ - Xshell 7 (Free for Home/School)
文件(F) 编辑(E) 查看(V) 工具(T) 选项卡(B) 窗口(W) 帮助(H)
ssh://bingh:*****@hadoop102:22
会话管理器
所有会话
hadoop100
hadoop102
hadoop103
hadoop104
[bingh@hadoop102 ~]$ myhadoop.sh start
===== 启动 hadoop集群 =====
----- 启动 hdfs -----
Starting namenodes on [hadoop102]
Starting datanodes
Starting secondary namenodes [hadoop104]
----- 启动 yarn -----
Starting resource manager
Starting node managers
----- 启动 historyserver -----
[bingh@hadoop102 ~]$
```

c. 启动 HBase;

```
start-hbase.sh
```



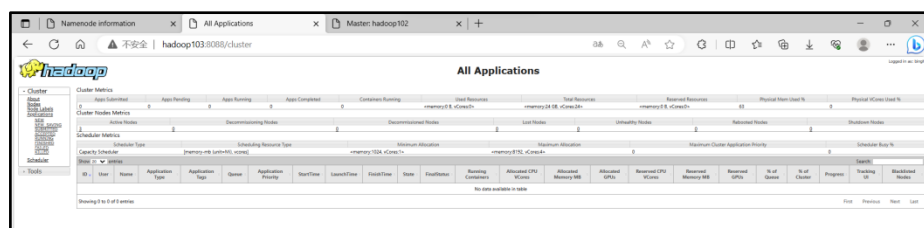
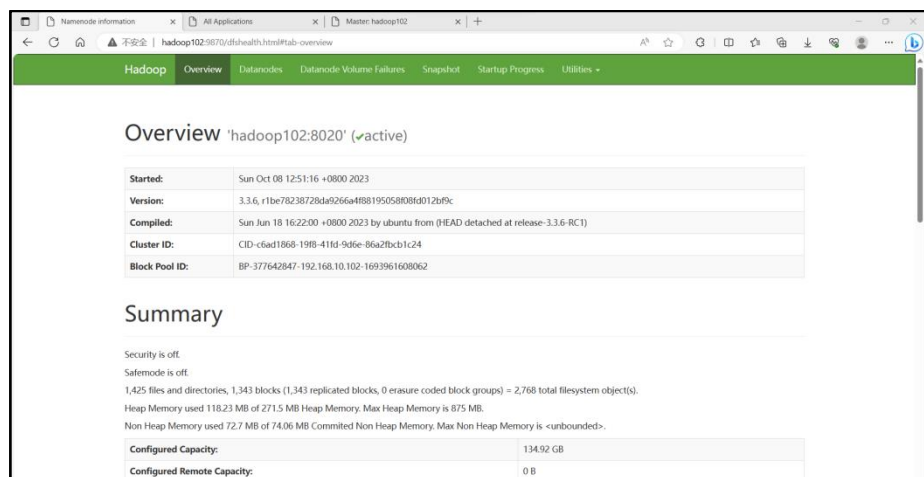
```
hadoop102 - bingh@hadoop102:~ - Xshell 7 (Free for Home/School)
文件(F) 编辑(E) 查看(V) 工具(T) 选项卡(B) 窗口(W) 帮助(H)
ssh://bingh:*****@hadoop102:22
会话管理器
所有会话
hadoop100
hadoop102
hadoop103
hadoop104
[bingh@hadoop102 ~]$ start-hbase.sh
running master, logging to /opt/module/hbase-2.5.5/logs/hbase-bingh-master-hadoop102.out
hadoop104: running regionserver, logging to /opt/module/hbase-2.5.5/logs/hbase-bingh-regionserver-hadoop104.out
hadoop103: running regionserver, logging to /opt/module/hbase-2.5.5/logs/hbase-bingh-regionserver-hadoop103.out
hadoop102: running regionserver, logging to /opt/module/hbase-2.5.5/logs/hbase-bingh-regionserver-hadoop102.out
[bingh@hadoop102 ~]$
```

d. 查看当前所有进程状态【执行自己编写的 shell 脚本: jpsall】;

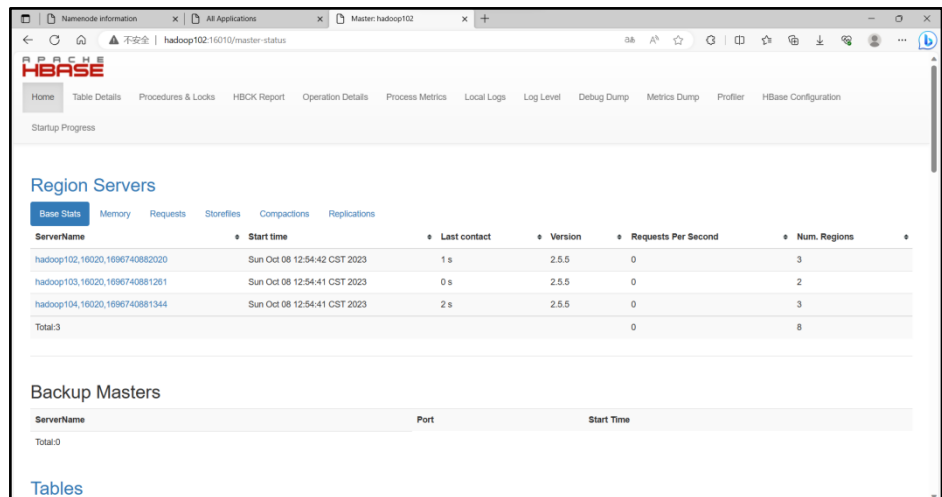
```
jpsall
```

```
hadoop102 - bingh@hadoop102:~ - Xshell 7 (Free for Home/School)
文件(F) 编辑(E) 查看(V) 工具(T) 选项卡(B) 窗口(W) 帮助(H)
ssh://bingh:*****@hadoop102:22
要添加当前会话，点击左侧的箭头按钮。
会话管理器
所有会话
hadoop100
hadoop102
hadoop103
hadoop104
[bingh@hadoop102 ~]$ jpsall
===== hadoop102 =====
2337 QuorumPeerMain
2610 NameNode
3318 JobHistoryServer
5690 Jps
3117 NodeManager
2750 DataNode
4846 HMaster
5086 HRegionServer
===== hadoop103 =====
2467 DataNode
2314 QuorumPeerMain
2795 NodeManager
4060 Jps
2669 ResourceManager
3695 HRegionServer
===== hadoop104 =====
2402 QuorumPeerMain
2643 SecondaryNameNode
3444 HRegionServer
2550 DataNode
3911 Jps
2779 NodeManager
[bingh@hadoop102 ~]$
```

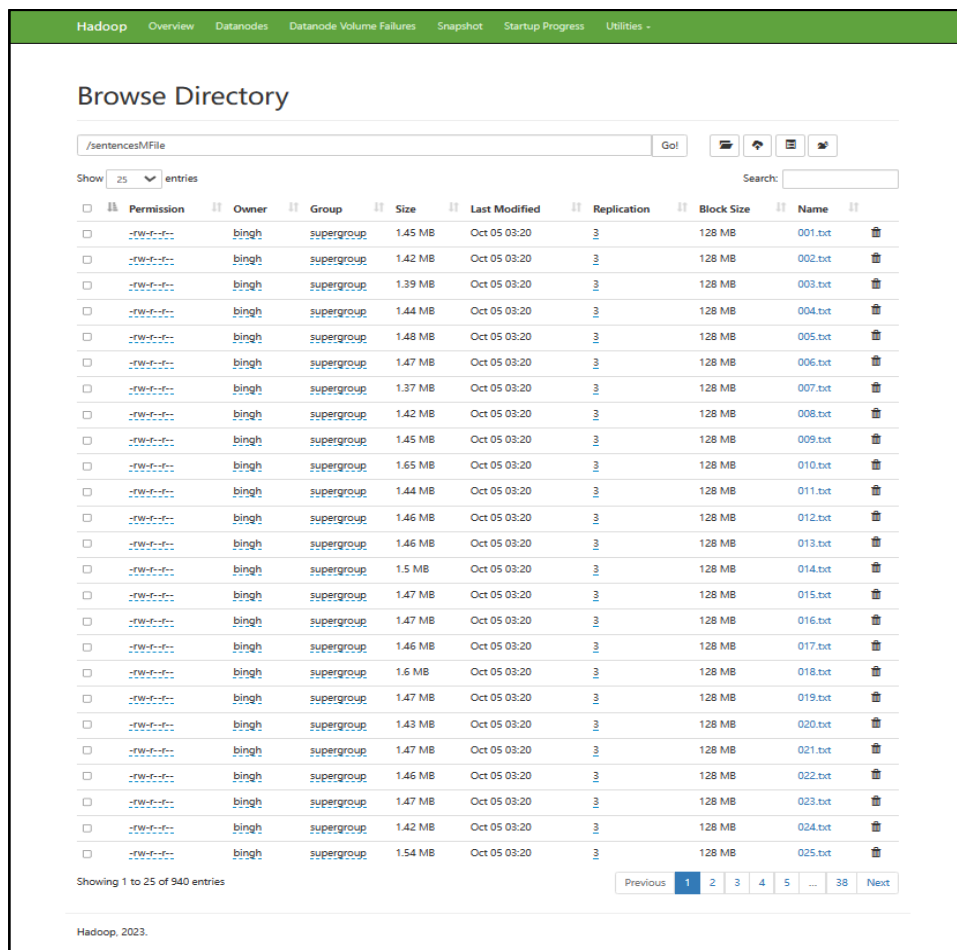
e. 打开 HDFS 网页界面、YARN 管理界面、HBase 网页界面。







C. 把 940 个小文件上传至 HDFS，父目录路径为 “/sentencesMFile”



D. 在 Hbase 预先创建好表格

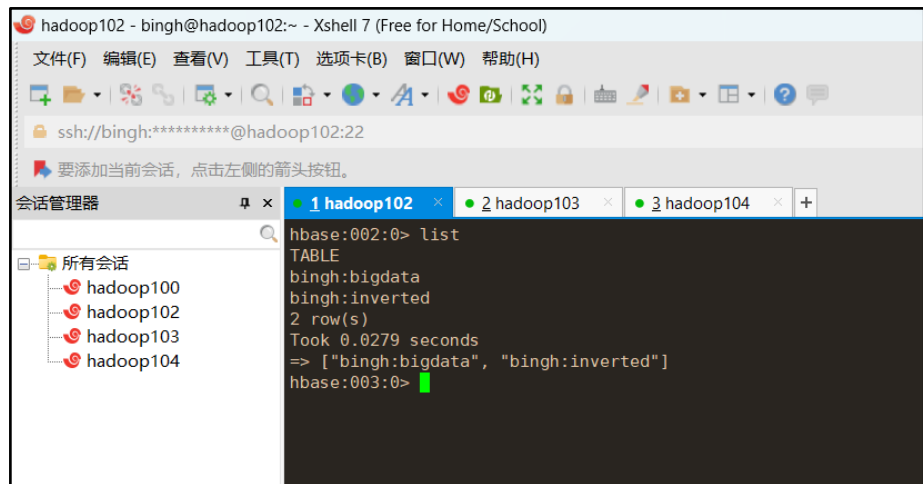
a. 在 HBase 的 HMaster 主机上（即 hadoop102）通过命令行输入 “hbase shell”，即可进入 HBase 命令行环境；

[illegible][illegible]

```
create_namespace 'bingh'
```

```
create 'bingh:bigdata', 'info'
```

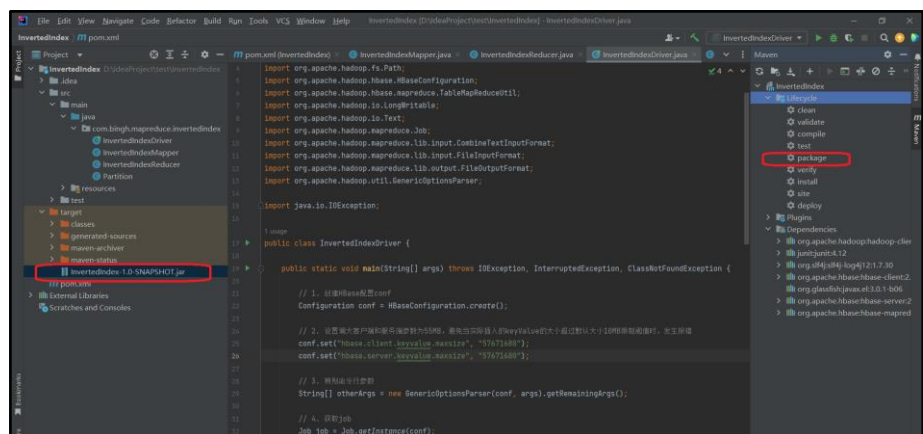
list



e. 退出 HBase 命令行环境。

E. 在集群运行倒排索引代码

a. 把本地 java 文件打包 (Package) 成 jar 包



b. 更改 jar 包名称为 “InvertedIndex.jar”

c. 复制 jar 包放到 Linux 集群上

```
hadoop102 - bingh@hadoop102:/opt/module/hadoop-3.3.6 - Xshell 7 (Free for Home/School)
文件(F) 编辑(E) 查看(V) 工具(T) 选项卡(B) 窗口(W) 帮助(H)
ssh://bingh:*****@hadoop102:22
要添加当前会话，点击左侧的箭头按钮。
会话管理器
1 hadoop102 2 hadoop103 3 hadoop104
[bingh@hadoop102 hadoop-3.3.6]$ ll
总用量 132
drwxr-xr-x. 2 bingh bingh 203 6月 18 17:08 bin
drwxr-xr-x. 4 bingh bingh 37 9月 6 08:55 data
drwxr-xr-x. 3 bingh bingh 20 6月 18 16:24 etc
drwxr-xr-x. 2 bingh bingh 106 6月 18 17:08 include
-rw-r--r--. 1 bingh bingh 7883 10月 5 17:02 InvertedIndex.jar
-rw-r--r--. 3 bingh bingh 20 6月 18 17:08 Lib
drwxr-xr-x. 4 bingh bingh 288 6月 18 17:08 libexec
-rw-r--r--. 1 bingh bingh 24276 6月 14 08:16 LICENSE-binary
drwxr-xr-x. 2 bingh bingh 4096 6月 18 17:08 licenses-binary
-rw-r--r--. 1 bingh bingh 15217 6月 10 07:41 LICENSE.txt
-rw-r--r--. 1 bingh bingh 7 9月 6 16:25 liubei.txt
drwxr-xr-x. 3 bingh bingh 4096 10月 8 15:40 logs
-rw-r--r--. 1 bingh bingh 29473 6月 10 07:41 NOTICE-binary
-rw-r--r--. 1 bingh bingh 1541 6月 10 07:33 NOTICE.txt
-rw-r--r--. 1 bingh bingh 175 6月 10 07:33 README.txt
drwxr-xr-x. 3 bingh bingh 4096 6月 18 16:24 sbin
drwxr-xr-x. 4 bingh bingh 31 6月 18 17:37 share
-rw-r--r--. 1 bingh bingh 14 9月 6 16:34 shuguo2.txt
-rw-r--r--. 1 bingh bingh 14 9月 6 16:32 shuguo.txt
drwxr-xr-x. 2 bingh bingh 22 9月 5 13:13 wcinput
-rw-r--r--. 1 bingh bingh 9246 9月 8 02:33 wc.jar
drwxr-xr-x. 2 bingh bingh 88 9月 5 13:11 wcoutput
-rw-r--r--. 1 bingh bingh 7 9月 6 15:48 weiguo.txt
-rw-r--r--. 1 bingh bingh 6 9月 6 15:50 wuguo.txt
[bingh@hadoop102 hadoop-3.3.6]$
```

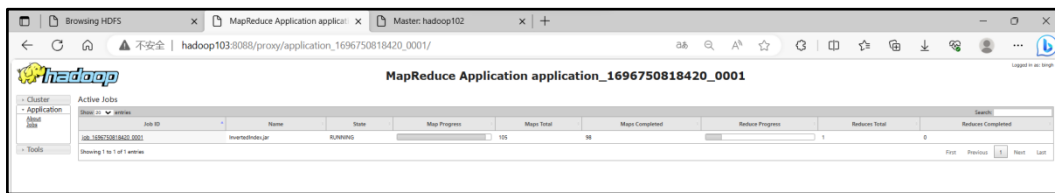
- d. 使用 “hadoop jar” 命令在 Hadoop 集群中运行 “InvertedIndex.jar” 程序

hadoop jar InvertedIndex.jar  
com.bingh.mapreduce.invertedindex.InvertedIndexDriver  
/sentencesMFile

```
hadoop102 - bingh@hadoop102:/opt/module/hadoop-3.3.6 - Xshell 7 (Free for Home/School)
文件(F) 编辑(E) 查看(V) 工具(T) 选项卡(B) 窗口(W) 帮助(H)
ssh://bingh:*****@hadoop102:22
要添加当前会话，点击左侧的箭头按钮。
会话管理器
1 hadoop102 2 hadoop103 3 hadoop104
[bingh@hadoop102 hadoop-3.3.6]$ hadoop jar InvertedIndex.jar com.bingh.mapreduce.invertedindex.InvertedIndexDriver /sentencesMFile
2023-10-08 16:19:56,732 INFO zookeeper.ZooKeeper: Client environment:java.library.path=/opt/module/hadoop-3.3.6/lib/native
2023-10-08 16:19:56,732 INFO zookeeper.ZooKeeper: Client environment:java.io.tmpdir=/tmp
2023-10-08 16:19:56,732 INFO zookeeper.ZooKeeper: Client environment:java.compiler=dmu
2023-10-08 16:19:56,738 INFO zookeeper.ZooKeeper: Client environment:os.arch=amd64
2023-10-08 16:19:56,738 INFO zookeeper.ZooKeeper: Client environment:os.name=Linux
2023-10-08 16:19:56,738 INFO zookeeper.ZooKeeper: Client environment:os.version=3.10.0-1160.71.1.el7.x86_64
2023-10-08 16:19:56,738 INFO zookeeper.ZooKeeper: Client environment:user.name=bingh
2023-10-08 16:19:56,738 INFO zookeeper.ZooKeeper: Client environment:user.home=/home/bingh
2023-10-08 16:19:56,738 INFO zookeeper.ZooKeeper: Client environment:user.dir=/opt/module/hadoop-3.3.6
2023-10-08 16:19:56,738 INFO zookeeper.ZooKeeper: Client environment:user.memory.free=300
2023-10-08 16:19:56,738 INFO zookeeper.ZooKeeper: Client environment:memory.max=47500
2023-10-08 16:19:56,738 INFO zookeeper.ZooKeeper: Client environment:memory.total=16400
2023-10-08 16:19:56,741 INFO zookeeper.ZooKeeper: Initiating client connection, connectString=127.0.0.1:2181 sessionTimeout=90000 watcherorg.apache.hadoop.hbase.zookeeper.ReadOnlyZKClient$1$1
2023-10-08 16:19:56,746 INFO common.XS09011: Setting -D jdk.tls.rejectClientInitiatedNegotiation=true to disable client-initiated TLS renegotiation
2023-10-08 16:19:56,792 INFO zookeeper.ClientCnxnSocket: jute.maxBuffer value is 1048576 bytes
2023-10-08 16:19:56,792 INFO zookeeper.ClientCnxn: zookeeper.request.timeout value is 8, feature enabled=false
2023-10-08 16:19:56,798 INFO zookeeper.ClientCnxn: Opening socket connection to server localhost/127.0.0.1:2181.
2023-10-08 16:19:56,798 INFO zookeeper.ClientCnxn: SASL config status: will not attempt to authenticate using SASL (unknown error)
2023-10-08 16:19:56,772 INFO zookeeper.ClientCnxn: Socket connection established, initializing session, client/127.0.0.1:4148, server/localhost/127.0.0.1:2181
2023-10-08 16:19:56,782 INFO zookeeper.ClientCnxn: Session establishment complete on server localhost/127.0.0.1:2181, session id = 0x20000756c6a002, negotiated timeout = 40000
2023-10-08 16:19:58,085 INFO zookeeper.ZooKeeper: Session: 0x20000756c6a002 close
2023-10-08 16:19:58,086 INFO zookeeper.ClientCnxn: EventThread shut down for session: 0x20000756c6a002
2023-10-08 16:20:02,180 INFO mapreduce.JobSourceCodeLoader: Job job_1696750818420_0001 jar path: /tmp/hadoop-yarn/staging/bingh/staging/job_1696750818420_0001
2023-10-08 16:20:02,476 INFO mapreduce.JobSubmitter: total input files to process : 940
2023-10-08 16:20:02,545 INFO Configuration.deprecation: io.bytes-per-checksum is deprecated. Instead, use dfs.bytes-per-checksum
2023-10-08 16:20:02,546 INFO Configuration.deprecation: yarn.resourcemanager.system-metrics-publisher.enabled is deprecated. Instead, use yarn.system-metrics-publisher.enabled
2023-10-08 16:20:02,869 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1696750818420_0001
2023-10-08 16:20:03,130 INFO conf.Configuration: resource-types.xml not found
2023-10-08 16:20:03,130 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'
2023-10-08 16:20:03,734 INFO impl.YarnClientImpl: Submitting application application_1696750818420_0001
2023-10-08 16:20:03,802 INFO mapreduce.Job: The url to track the job: http://hadoop102:8080/proxy/application_1696750818420_0001/
2023-10-08 16:20:03,801 INFO mapreduce.Job: Running job: job_1696750818420_0001
2023-10-08 16:20:16,615 INFO mapreduce.Job: Job job_1696750818420_0001 running in uber mode : false
2023-10-08 16:20:16,919 INFO mapreduce.Job: map 0% reduce 0%
[bingh@hadoop102 hadoop-3.3.6]$
```

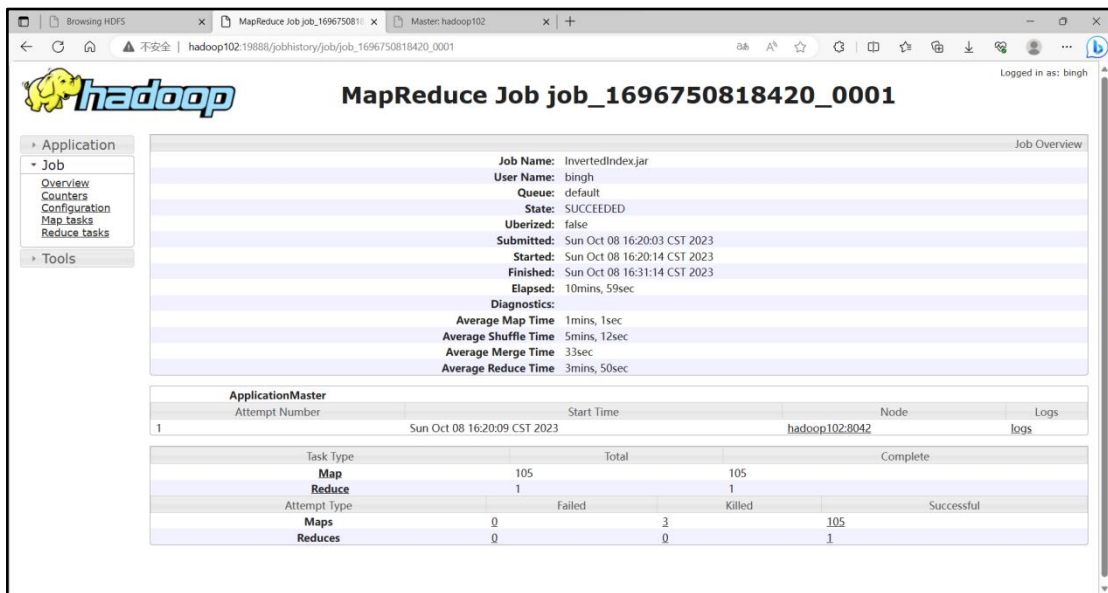
【Total input files to process: 940, number of splits: 105】

## F. 查看运行进度



Job ID	Name	Status	Map Progress	Map Total	Map Completed	Reduce Progress	Reduce Total	Reduce Completed
job_1696750818420_0001	InvertedIndexJar	RUNNING	105	98	98	1	0	0

## G. 查看运行结果



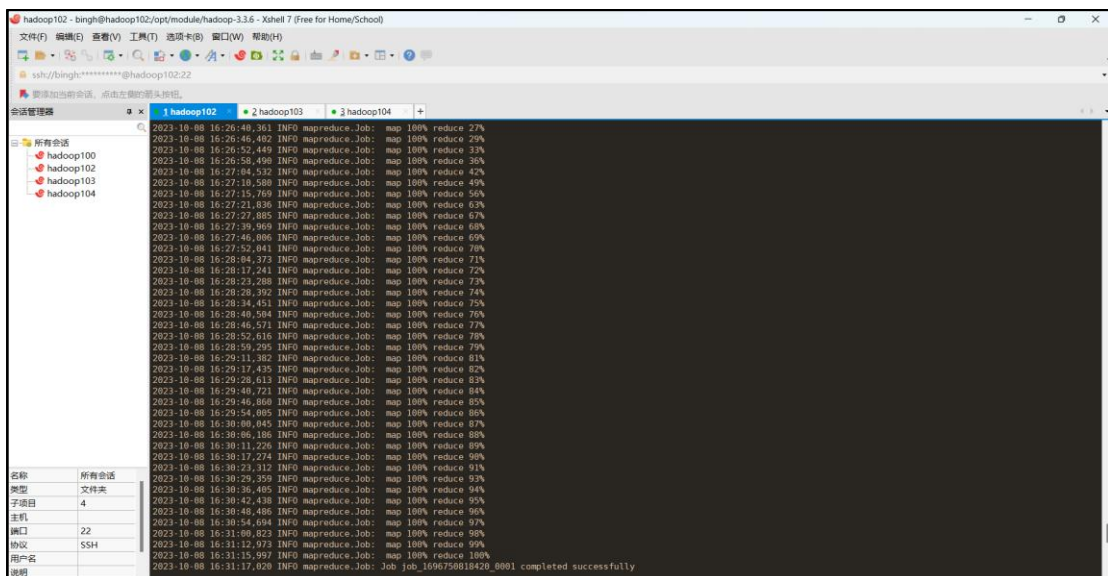
ApplicationMaster	Attempt Number	Start Time	Node	Logs
1	Sun Oct 08 16:20:09 CST 2023	hadoop102.8042	logs	

Task Type	Total	Complete
Map	105	105
Reduce	1	1

Attempt Type	Failed	Killed	Successful
Maps	0	3	105
Reduces	0	0	1



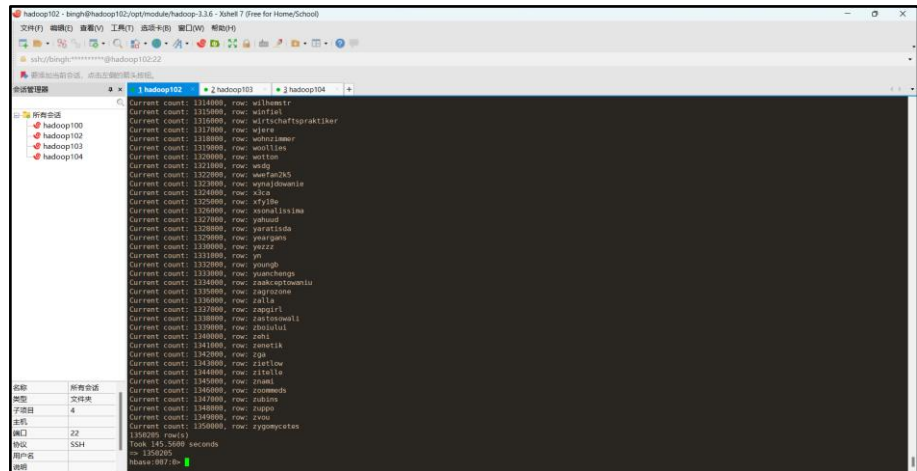
```
2023-10-08 16:26:40,361 INFO mapreduce.Job: map 100% reduce 27%
2023-10-08 16:26:46,492 INFO mapreduce.Job: map 100% reduce 29%
2023-10-08 16:26:52,449 INFO mapreduce.Job: map 100% reduce 33%
2023-10-08 16:26:58,498 INFO mapreduce.Job: map 100% reduce 36%
2023-10-08 16:27:04,532 INFO mapreduce.Job: map 100% reduce 42%
2023-10-08 16:27:10,588 INFO mapreduce.Job: map 100% reduce 49%
2023-10-08 16:27:15,769 INFO mapreduce.Job: map 100% reduce 56%
2023-10-08 16:27:21,836 INFO mapreduce.Job: map 100% reduce 63%
2023-10-08 16:27:27,885 INFO mapreduce.Job: map 100% reduce 67%
2023-10-08 16:27:39,969 INFO mapreduce.Job: map 100% reduce 68%
2023-10-08 16:27:46,086 INFO mapreduce.Job: map 100% reduce 69%
2023-10-08 16:27:52,041 INFO mapreduce.Job: map 100% reduce 70%
2023-10-08 16:28:04,373 INFO mapreduce.Job: map 100% reduce 71%
2023-10-08 16:28:17,243 INFO mapreduce.Job: map 100% reduce 72%
2023-10-08 16:28:23,288 INFO mapreduce.Job: map 100% reduce 73%
2023-10-08 16:28:28,392 INFO mapreduce.Job: map 100% reduce 74%
2023-10-08 16:28:34,451 INFO mapreduce.Job: map 100% reduce 75%
2023-10-08 16:28:40,504 INFO mapreduce.Job: map 100% reduce 76%
2023-10-08 16:28:46,571 INFO mapreduce.Job: map 100% reduce 77%
2023-10-08 16:28:52,616 INFO mapreduce.Job: map 100% reduce 78%
2023-10-08 16:28:59,295 INFO mapreduce.Job: map 100% reduce 79%
2023-10-08 16:29:11,382 INFO mapreduce.Job: map 100% reduce 81%
2023-10-08 16:29:17,435 INFO mapreduce.Job: map 100% reduce 82%
2023-10-08 16:29:28,613 INFO mapreduce.Job: map 100% reduce 83%
2023-10-08 16:29:40,721 INFO mapreduce.Job: map 100% reduce 84%
2023-10-08 16:29:46,868 INFO mapreduce.Job: map 100% reduce 85%
2023-10-08 16:29:54,805 INFO mapreduce.Job: map 100% reduce 86%
2023-10-08 16:30:00,845 INFO mapreduce.Job: map 100% reduce 87%
2023-10-08 16:30:06,186 INFO mapreduce.Job: map 100% reduce 88%
2023-10-08 16:30:11,226 INFO mapreduce.Job: map 100% reduce 89%
2023-10-08 16:30:17,274 INFO mapreduce.Job: map 100% reduce 90%
2023-10-08 16:30:23,312 INFO mapreduce.Job: map 100% reduce 91%
2023-10-08 16:30:29,359 INFO mapreduce.Job: map 100% reduce 93%
2023-10-08 16:30:36,405 INFO mapreduce.Job: map 100% reduce 94%
2023-10-08 16:30:42,438 INFO mapreduce.Job: map 100% reduce 95%
2023-10-08 16:30:48,486 INFO mapreduce.Job: map 100% reduce 96%
2023-10-08 16:30:54,694 INFO mapreduce.Job: map 100% reduce 97%
2023-10-08 16:31:00,823 INFO mapreduce.Job: map 100% reduce 98%
2023-10-08 16:31:12,073 INFO mapreduce.Job: map 100% reduce 99%
2023-10-08 16:31:15,997 INFO mapreduce.Job: map 100% reduce 100%
2023-10-08 16:31:17,020 INFO mapreduce.Job: Job job_1696750818420_0001 completed successfully
```

## a. 进入 HBase 命令行环境

```
hbase shell
```

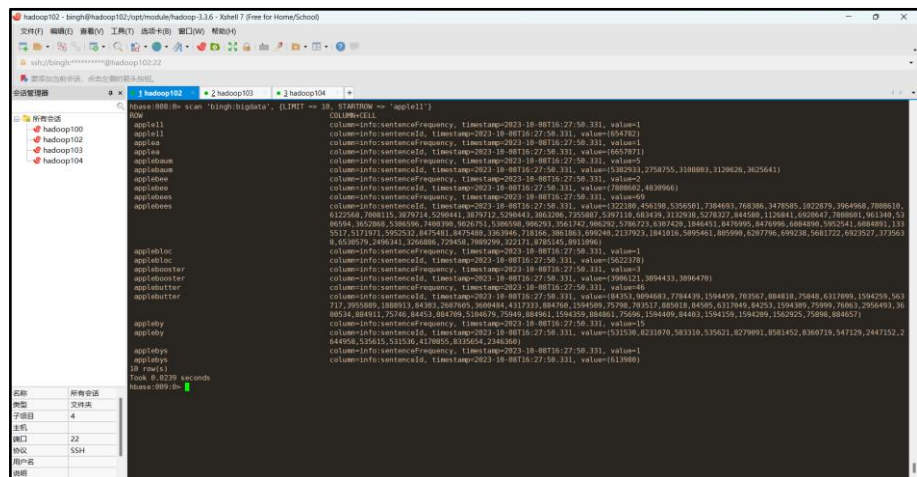
## b. 统计”bingh:bigdata”表的行数

count “bingh:bigdata”



## c. 批量查询

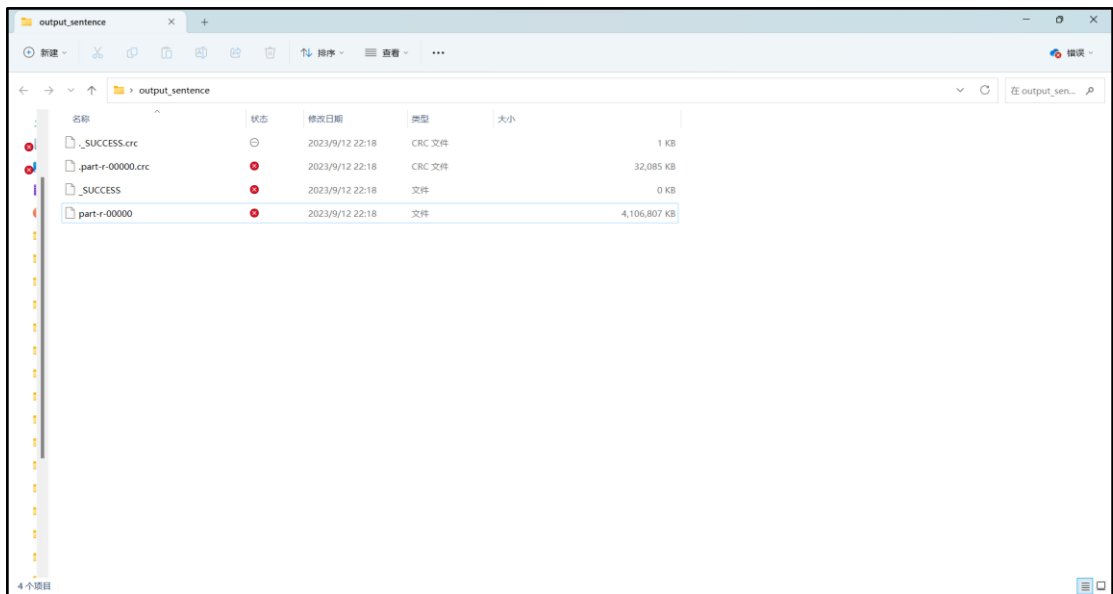
scan 'bingh:bigdata', {LIMIT => 10, STARTROW => 'apple11'}



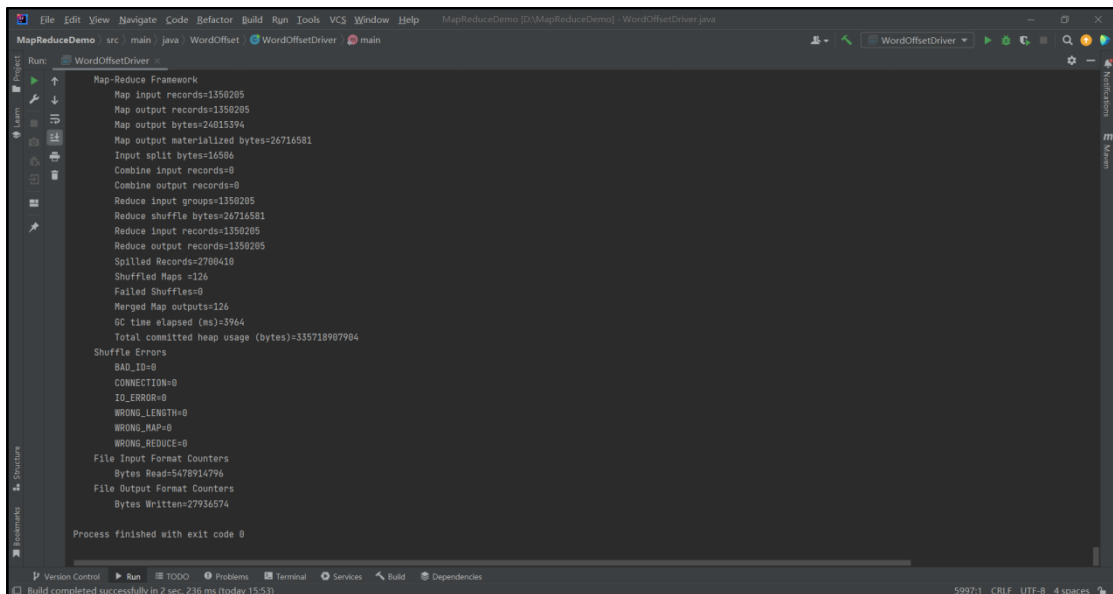
## → 二级索引

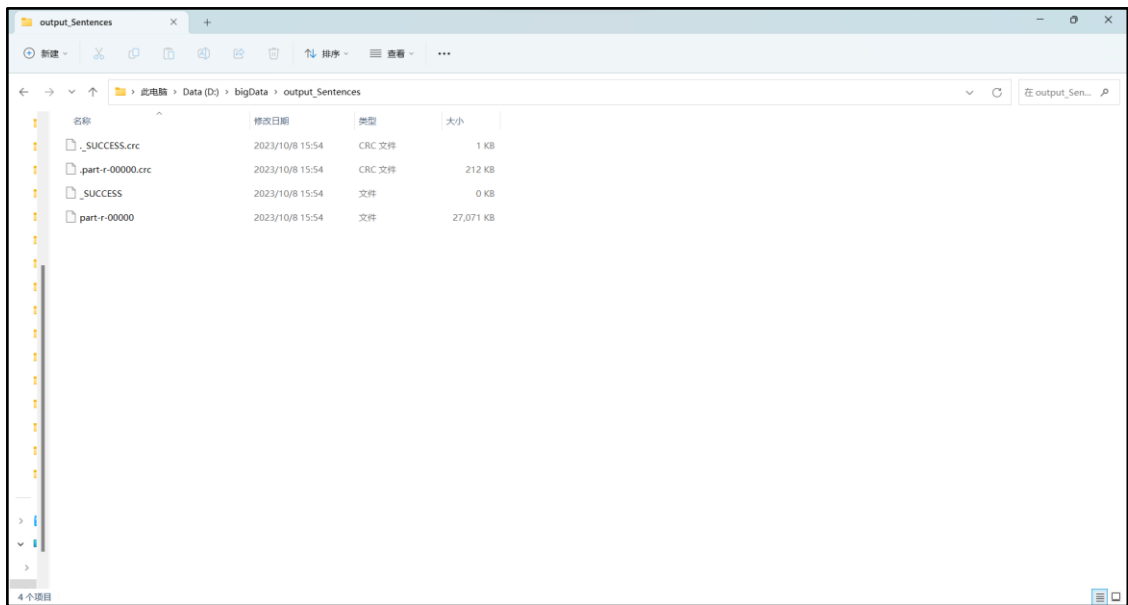
### A. 运行在本地的倒排索引结果



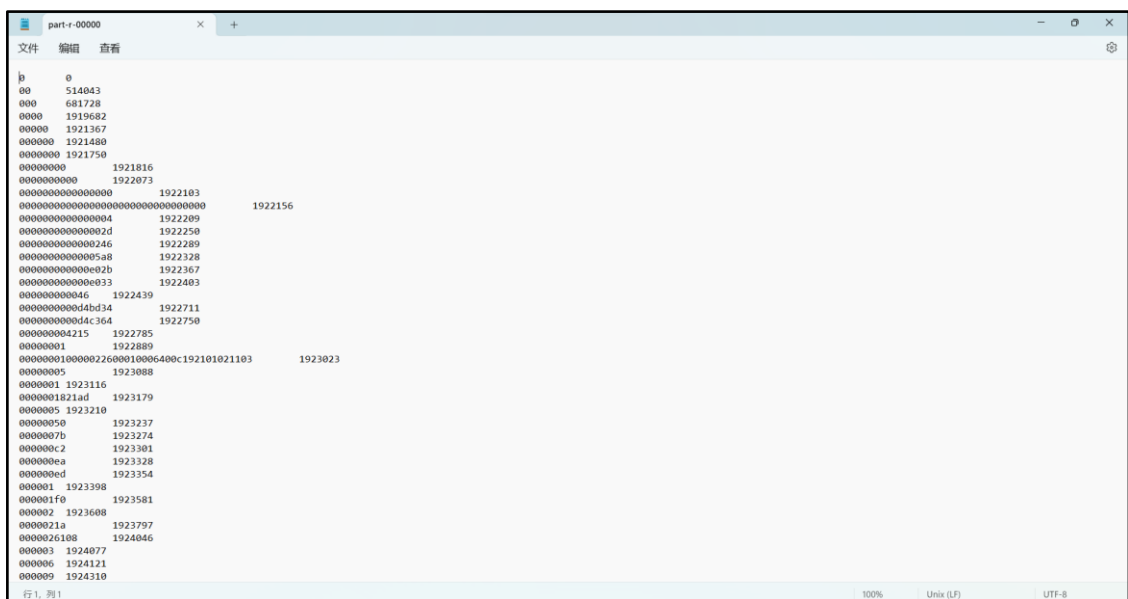


## B. 对倒排索引结果 “part-r-00000” 在本地上运行二级索引代码

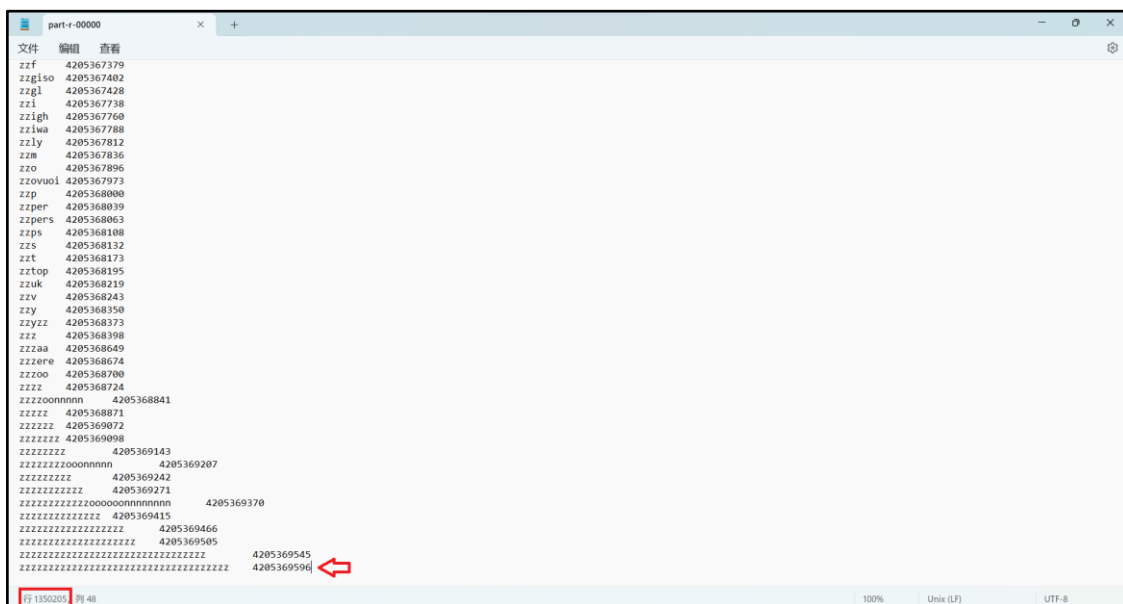




### C. 查看运行结果







在这个项目中，我们深入了解了 Hadoop 的核心架构，包括资源管理系统、分布式计算框架和分布式文件系统之间的运作关系。我们重点学习了 MapReduce 分布式计算框架的原理，并将其成功运用到离线搜索引擎的实现中。对于许多团队成员来说，这是第一次亲自使用 Hadoop 来管理数据库，我们充分体验了数据处理和分析的高效性，学到了处理大规模数据的能力，以及通过并行处理提高速度和性能的方法。此外，我们认识到 Hadoop 的可扩展性和灵活性，使其成为可应用于各种业务问题的强大工具。这个课程的实验对我们来说是一次巨大的挑战，需要全面理解知识并协调各个模块，以确保小组的进展顺利。这次实验给我们带来了宝贵的经验，也让我们认识到了自己的不

足之处，我们将努力在未来的机会中表现更出色。在搭建实验环境的过程中，我们面临了许多挑战，特别是在 HBase 和 Zookeeper 的配置方面，这部分工作耗费了大量的时间和精力。然而，最终我们成功克服了这些问题，这让我们感到非常满足。我们的团队在这次实验中按照要求一步步解决了问题，最终成功完成了任务。除了满足原始实验要求外，我们还加入了分区和二级索引的功能，以提高搜索效率。我们要感谢所有团队成员，他们积极投入了课后时间，共同完成了这个项目。