实验报告

报告标题：MR-Hbase-Hive

学号：21190630

姓名：黄艺杰

日期：2022年12月11日

# 一、实验环境

1. 操作系统：Windows 10、Linux

2. 相关软件（含版本号）：VMWare15pro、FinalShell3.9、Intellij IDEA2019Professional

3. 其它工具：JDK1.8

# 二、实验内容及其完成情况

（针对上述实验内容逐一详述实验过程）

1. MapReduce编程题

要求：基于天气数据文件（weather.txt），计算月平均气温。要求有上传气象数据的代码，MR 程序代码，运行过程，下载并查看结果的代码验证方法：在虚拟机中查看运行过程与结果。

先在IDEA上使用put函数将weather.txt文件上传到自定义文件夹homework中

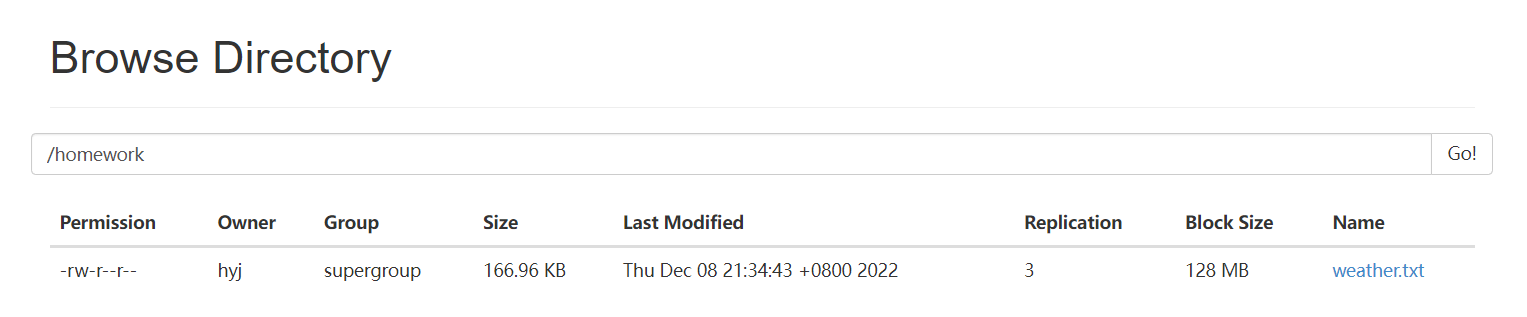


图1.1HDFS中homework文件夹

上传代码如下：

1. put("/D:/qiyeshixun/wenjian/weather.txt","/homework/weather.txt");
2. **private** **static** **void** put(String src,String dest) throws IOException {
3. fileSystem.copyFromLocalFile(**new** Path(src),**new** Path(dest));
4. System.out.println("put ok");
5. }

MR代码如下：

WeatherMapper：

1. **public** **class** WeatherMapper extends Mapper<LongWritable, Text, Text, LongWritable> {
2. @Override
3. **protected** **void** map(
4. LongWritable key,
5. Text value,
6. Mapper<LongWritable, Text, Text, LongWritable>.Context context
7. ) throws IOException, InterruptedException {
8. String[] ss = value.toString().split(",");
9. **if** (ss[0].equals("DATE")) {
10. **return**;
11. }
13. String yymm = ss[0].substring(0, 6);
14. **try** {
15. context.write(
16. **new** Text(yymm),
17. **new** LongWritable(Long.parseLong(ss[5]))
18. );
19. } **catch** (Exception ex) {
20. }
21. }
22. }

WeatherReducer：

1. **public** **class** WeatherReducer extends Reducer<Text, LongWritable, Text, DoubleWritable> {
2. @Override
3. **protected** **void** reduce(
4. Text key,
5. Iterable<LongWritable> values,
6. Reducer<Text, LongWritable, Text, DoubleWritable>.Context context
7. ) throws IOException, InterruptedException {
8. **long** sum = 0L;
9. **long** cnt = 0L;
10. **for** (LongWritable lw : values) {
11. cnt++;
12. sum += lw.get();
13. }
14. context.write(
15. key,
16. **new** DoubleWritable(sum \* 1.0 / cnt)
17. );
18. }
19. }

WeatherApp：

1. **public** **class** WeatherApp {
2. **public** **static** **void** main(String[] args) throws IOException {
3. Configuration conf = **new** Configuration();
4. Job job = Job.getInstance(conf);
5. job.setJarByClass(WeatherApp.**class**);
6. job.setMapperClass(WeatherMapper.**class**);
7. job.setReducerClass(WeatherReducer.**class**);
8. job.setMapOutputKeyClass(Text.**class**);
9. job.setMapOutputValueClass(LongWritable.**class**);
10. job.setOutputKeyClass(Text.**class**);
11. job.setOutputValueClass(DoubleWritable.**class**);
12. FileInputFormat.setInputPaths(job, **new** Path(args[0]));
13. FileOutputFormat.setOutputPath(job, **new** Path(args[1]));
14. **try** {
15. job.waitForCompletion(**true**);
16. } **catch** (Exception ex) {
17. System.out.println(ex);
18. }
19. }
20. }

将以上文件打包后，上传到虚拟机中，并在虚拟机中运行结果，将结果保存到HDFS的out1文件夹中

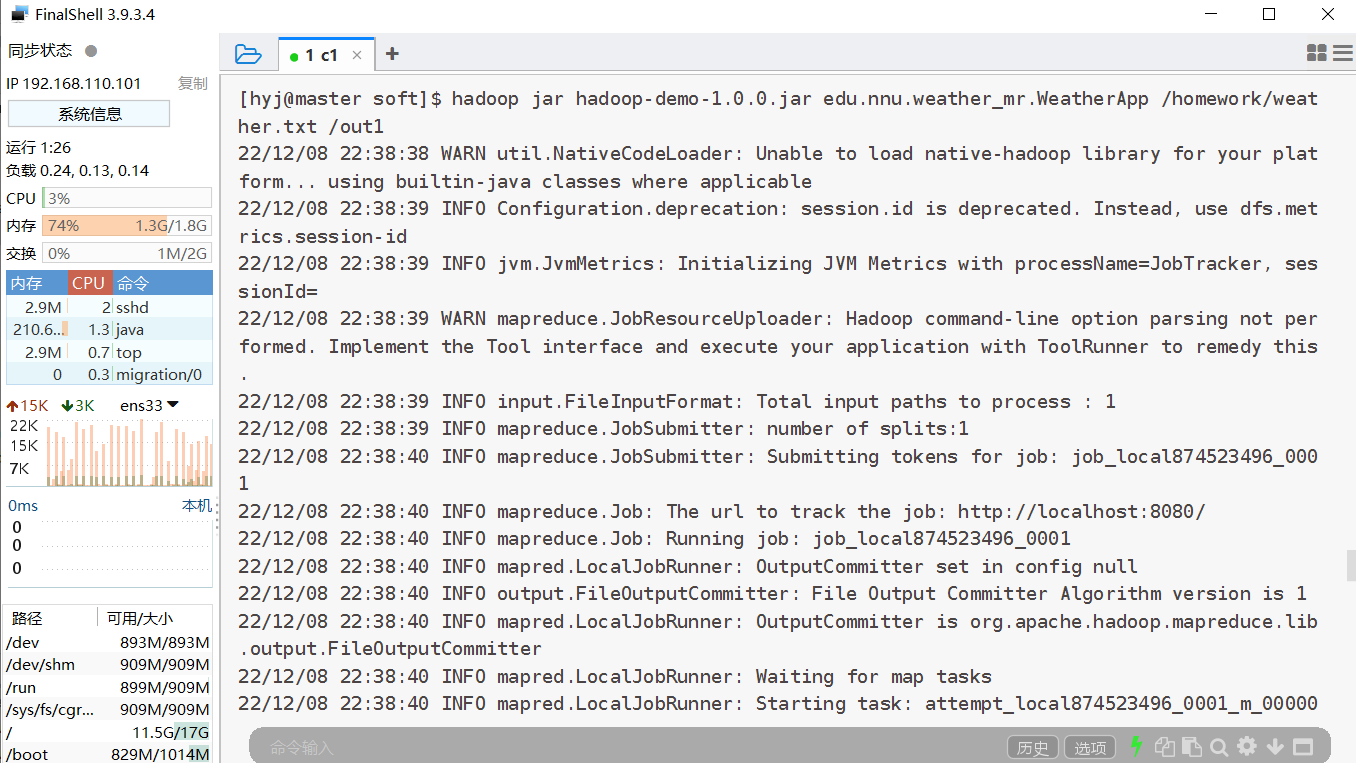


图1.2在虚拟机上运行WeatherApp

可在虚拟机上远程查看结果内容

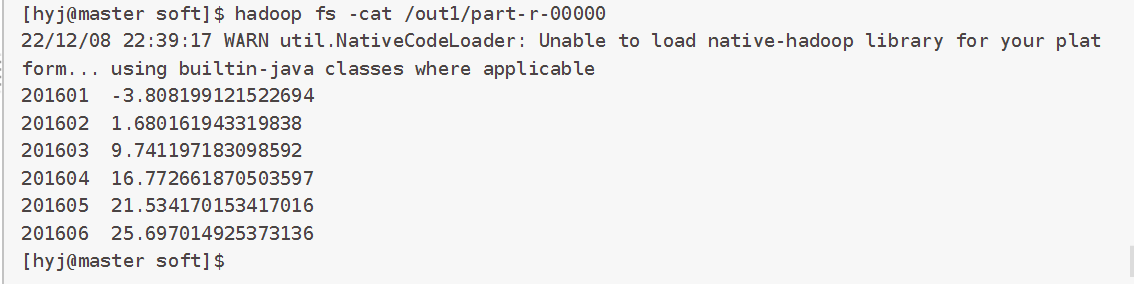


图1.3虚拟机上远程查看结果

将实验结果下载到本地result文件中的weather.txt，并在本地查看，下载代码如下：

1. get("/out1/part-r-00000","/D:/qiyeshixun/result/weather.txt");
2. **private** **static** **void** get(String src, String dest) throws IOException {
3. fileSystem.copyToLocalFile(**false**,**new** Path(src),**new** Path(dest),**true**);
4. System.out.println("get ok!");
5. }

在本地打开下载的weather.txt文件查看结果：

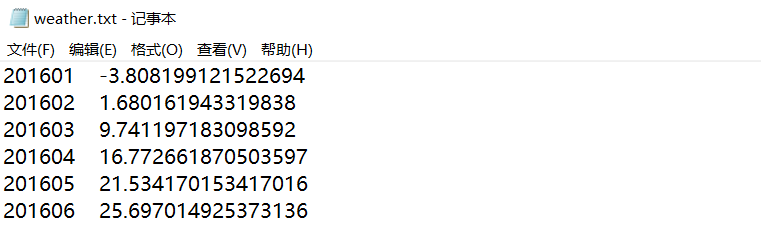


图1.4在本地查看下载的实验结果

1. MapReduce 编程题

要求：统计在四大名著中出现总次数最多的 20 个词（2个字或以上的词）。要求有文件上传的代码、MR 程序、运行过程，下载结果的代码，排序数据并打印最终结果（输出：词和出现次数）的代码。验证方法：在虚拟机中查看运行过程，在 IDEA 中查看最终结果。

将四大名著上传到HDFS的demo文件夹中并合并成一个文件 Four\_masterpieces.txt

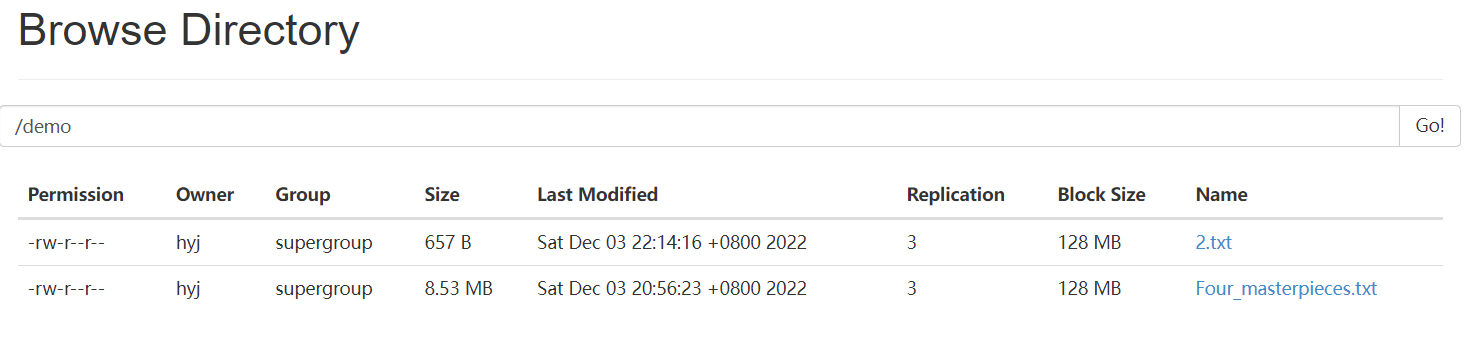


图2.1上传四大名著并合并成一个文件

上传代码如下：

1. writeAllFilesOfDir("/D:/qiyeshixun/book","/demo/Four\_masterpieces.txt");
2. **private** **static** **void** writeAllFilesOfDir(String dir,String dest) {
3. File path = **new** File(dir);
4. **if**(path.exists() && path.isDirectory()){
5. File[] files=path.listFiles();
6. byte[] bts=**new** byte[4096];
7. **int** size = 0;
8. FSDataOutputStream outputStream=null;
9. **try** {
10. outputStream = fileSystem.create(**new** Path(dest));
11. **for** (**int** i = 0; i < files.length; i++) {
12. FileInputStream fis = null;
13. **if** (files[i].isFile()) {
14. **try** {
15. fis = **new** FileInputStream(files[i]);
16. **while** ((size = fis.read(bts)) > 0) {
17. outputStream.write(bts,0,size);
18. }
19. } **catch** (Exception ex) {
20. System.out.println(ex);
21. } finally {
22. **try** {
23. fis.close();
24. } **catch** (Exception e) {
25. }
26. }
27. }
28. }
29. }
30. **catch**(Exception ex){
31. System.out.println(ex);
32. }
33. finally{
34. }
35. }
36. **else**{
37. System.out.println("not found");
38. }
39. }

MR代码如下：

WordCountMapper代码：

1. **public** **class** WordCountMapper extends Mapper<LongWritable, Text,Text,LongWritable> {
2. @Override
3. **protected** **void** map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
4. JiebaSegmenter seg=**new** JiebaSegmenter();
5. List<SegToken> tokenList=seg.process(value.toString(),JiebaSegmenter.SegMode.SEARCH);
6. LongWritable one=**new** LongWritable(1);
7. **for**(SegToken segToken : tokenList){
8. **if**(segToken.word.matches("[\u4e00-\u9fa5]{2,}")){
9. context.write(**new** Text(segToken.word),one);
10. }
11. }
12. }
13. }

WordCountReducer代码：

1. **public** **class** WordCountReducer extends Reducer<Text, LongWritable,Text,LongWritable> {
2. @Override
3. **protected** **void** reduce(Text key, Iterable<LongWritable> values, Context context) throws IOException, InterruptedException {
4. //super.reduce(key, values, context);
5. **long** sum=0;
6. **for** (LongWritable value : values) {
7. sum+=value.get();
8. }
9. context.write(key,**new** LongWritable(sum));
10. }
11. }

WordCountApp代码：

1. **public** **class** WordCountApp {
2. **public** **static** **void** main(String[] args) throws Exception {
3. Configuration config=**new** Configuration();
4. Job job=Job.getInstance(config);
6. job.setMapperClass(WordCountMapper.**class**);
7. job.setReducerClass(WordCountReducer.**class**);
8. job.setJarByClass(WordCountApp.**class**);
10. job.setMapOutputKeyClass(Text.**class**);
11. job.setMapOutputValueClass(LongWritable.**class**);
12. job.setOutputKeyClass(Text.**class**);
13. job.setOutputValueClass(LongWritable.**class**);
15. FileInputFormat.setInputPaths(job, **new** Path(args[0]));
16. FileOutputFormat.setOutputPath(job,**new** Path(args[1]));
18. job.waitForCompletion(**true**);
19. }
20. }

将以上代码打包在虚拟机上运行，将结果保存在HDFS的out2文件夹，运行过程如下：

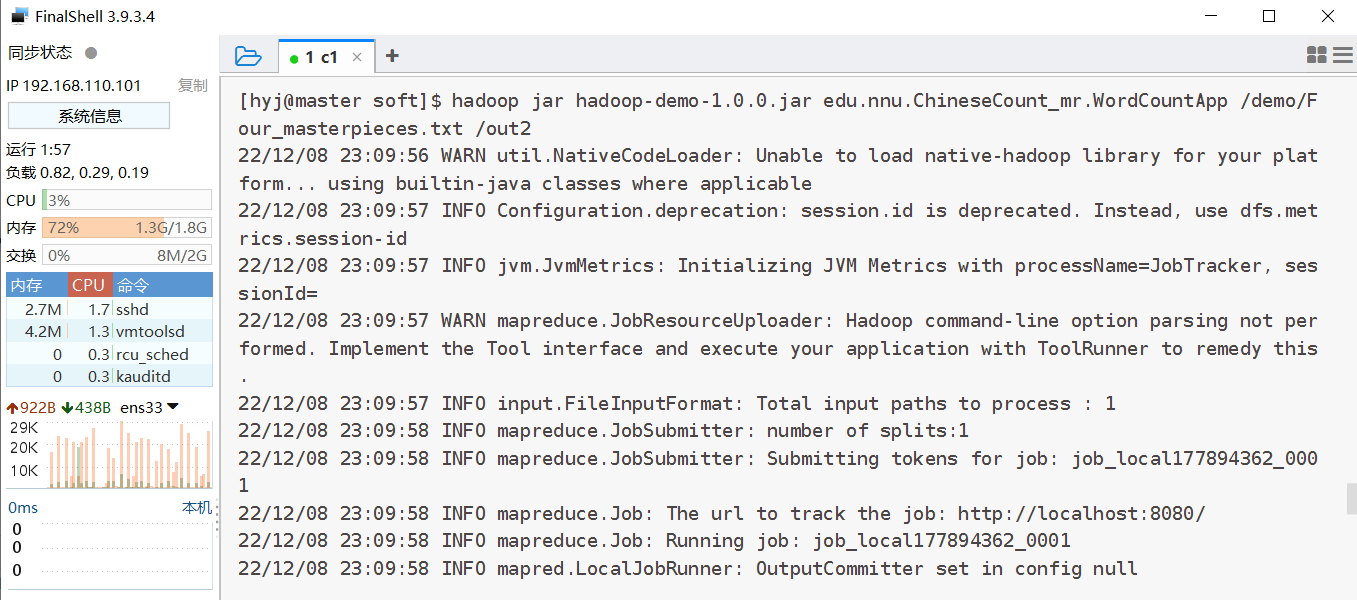


图2.2WordCountApp在虚拟机上运行过程

下载结果的代码如下：

1. get("/out2/part-r-00000","/D:/qiyeshixun/result/Four\_masterpieces.txt");
2. 1.**private** **static** **void** get(String src, String dest) throws IOException {
3. 2.        fileSystem.copyToLocalFile(**false**,**new** Path(src),**new** Path(dest),**true**);
4. 3.        System.out.println("get ok!");
5. 4.    }

最后在IDEA排序数据并打印最终的结果，其代码如下：

1. **public** **class** WordCountResultApp {
2. **static** **class** Word{
3. **public** String key;
4. **public** Integer count;
5. }
6. **public** **static** **void** main(String[] args){
7. File file =**new** File("D:/qiyeshixun/result/Four\_masterpieces.txt");
8. **if**(!file.exists()){
9. System.out.println("file not found");
10. **return**;
11. }
12. InputStream is=null;
13. Reader reader=null;
14. BufferedReader br=null;
15. List<Word> words=**new** ArrayList<Word>();
16. **try**{
17. is=**new** FileInputStream(file);
18. reader =**new** InputStreamReader(is);
19. br=**new** BufferedReader(reader);
20. String line=null;
21. **while**((line=br.readLine())!=null){
22. String []tmp=line.split("\t");
23. Word word=**new** Word();
24. word.key=tmp[0];
25. word.count=Integer.parseInt(tmp[1]);
26. words.add(word);
27. }
28. }**catch**(Exception ef){
29. System.out.println(ef);
30. }finally{
31. **try**{
32. br.close();
33. }**catch** (Exception e){}
34. **try**{
35. reader.close();
36. }**catch**(Exception e){}
37. **try**{
38. is.close();
39. }**catch**(Exception e){}
40. }
41. words.sort((a,b)->b.count-a.count);
42. **for**(**int** i=0;i<20;i++){
43. System.out.println(words.get(i).key+":"+words.get(i).count);
44. }
45. }
46. }

运行后输出四大名著出现总次数最多的20个词：

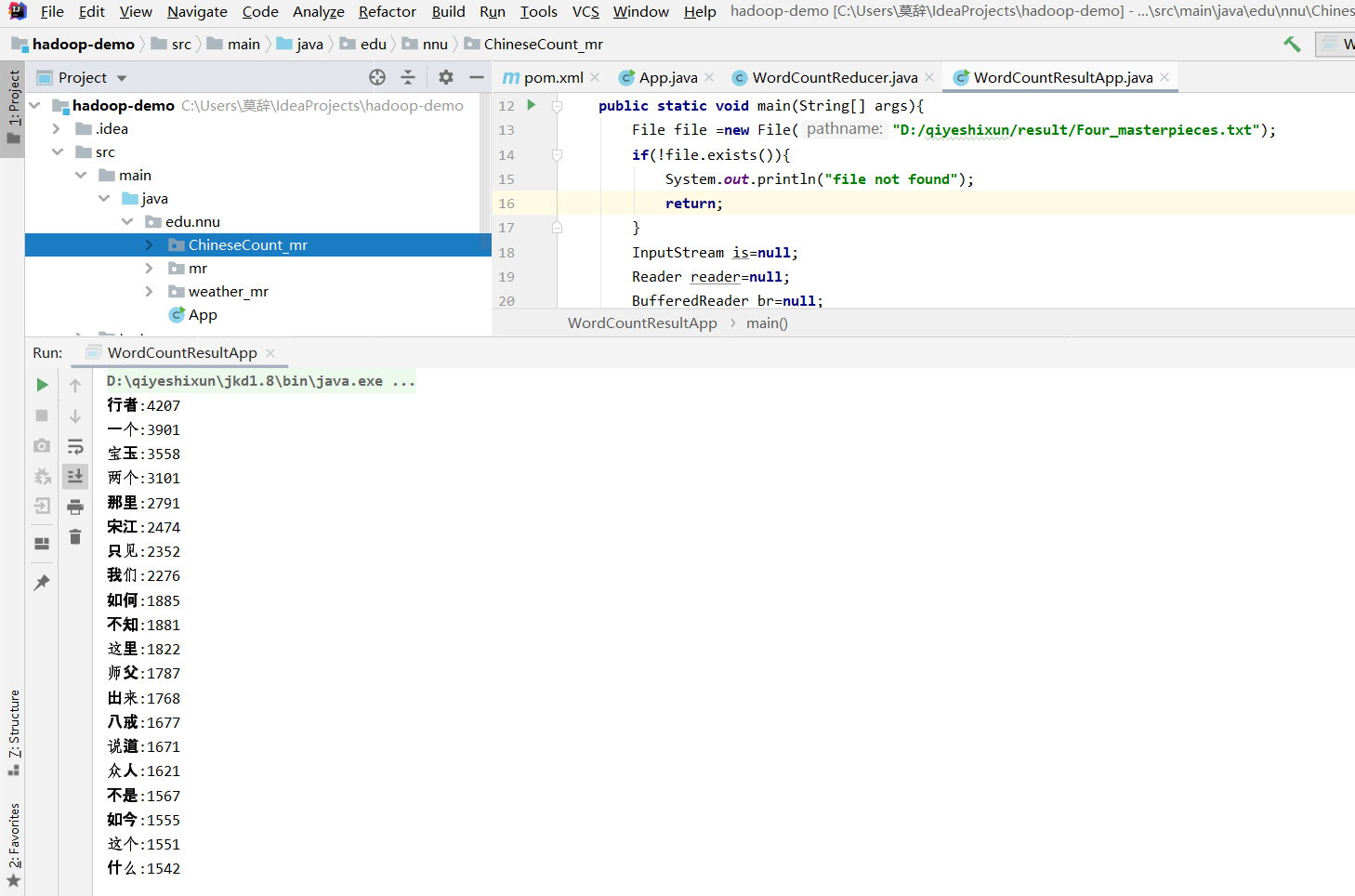


图2.3统计四大名著热词

1. Hbase 集群的安装配置（包括 Zookeeper 的分布式配置）：

验证方法：在宿主机中通过浏览器查看 CentOS 7 的 60010 端口

先在虚拟机上将zookeeper的压缩包解压到根目录并改名为zookeeper

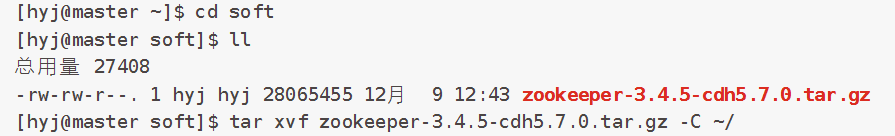


图3.1解压zookeeper的压缩包

接下来在.bash\_profile中配置环境，配置zoo.cfg文件

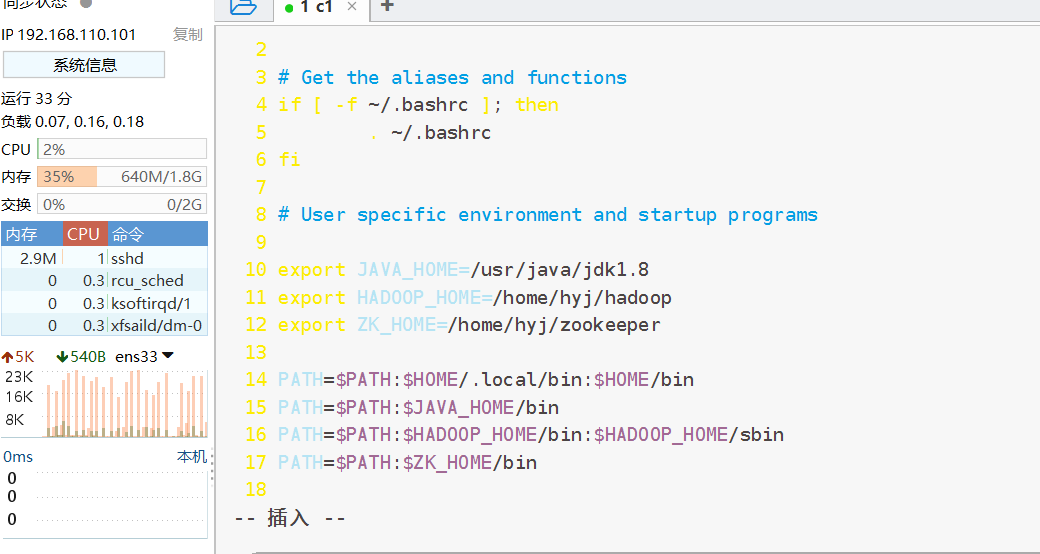


图3.2编辑.bash\_profile环境配置文件

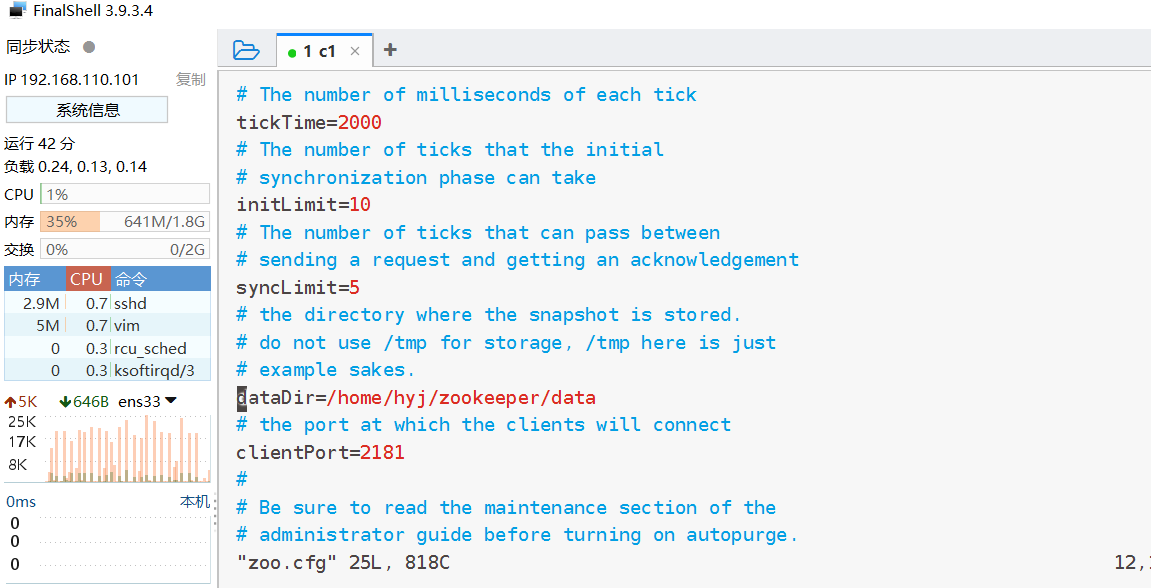


图3.3编辑zoo.cfg文件

接下来上传Hbase的文件，与上述过程基本类似，先上传文件并解压缩，改名，添加环境变量，修改hbase-env.sh文件。

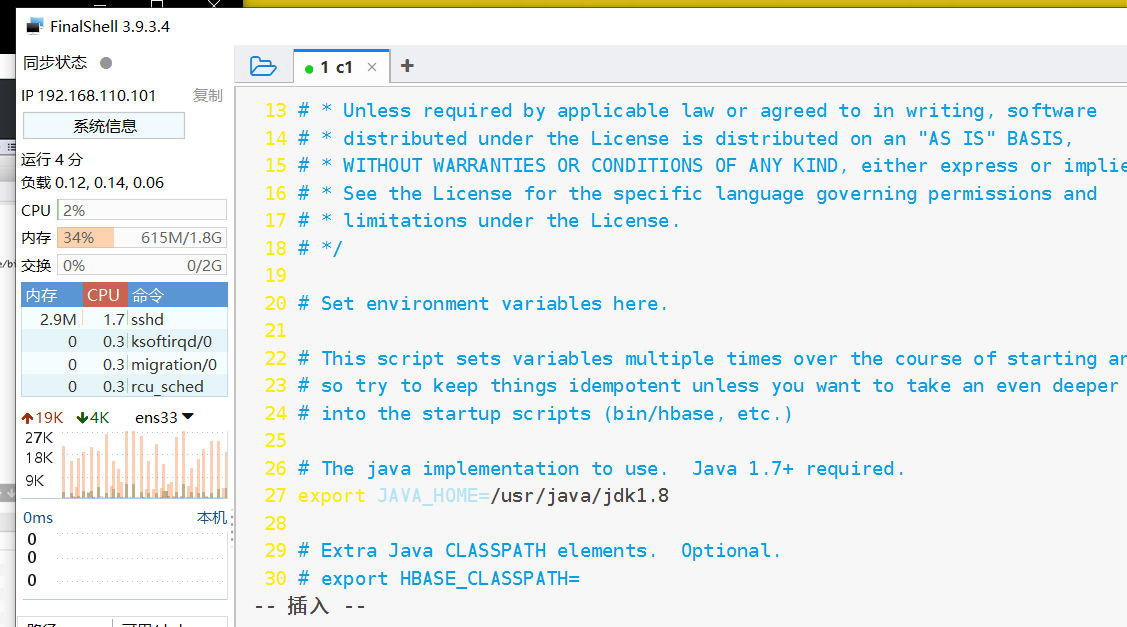


图3.4配置hbase-env.sh文件

配置hbase-site.xml



图3.5配置hbase-site.xml文件

打开hbase.sh，在jps中看到HMaster

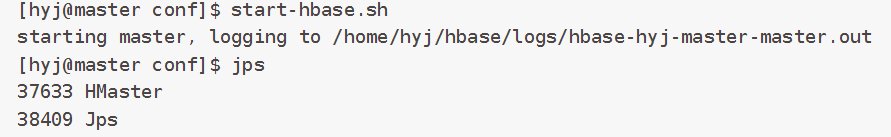


图3.6开启hbase.sh

最后进入浏览器中60010端口

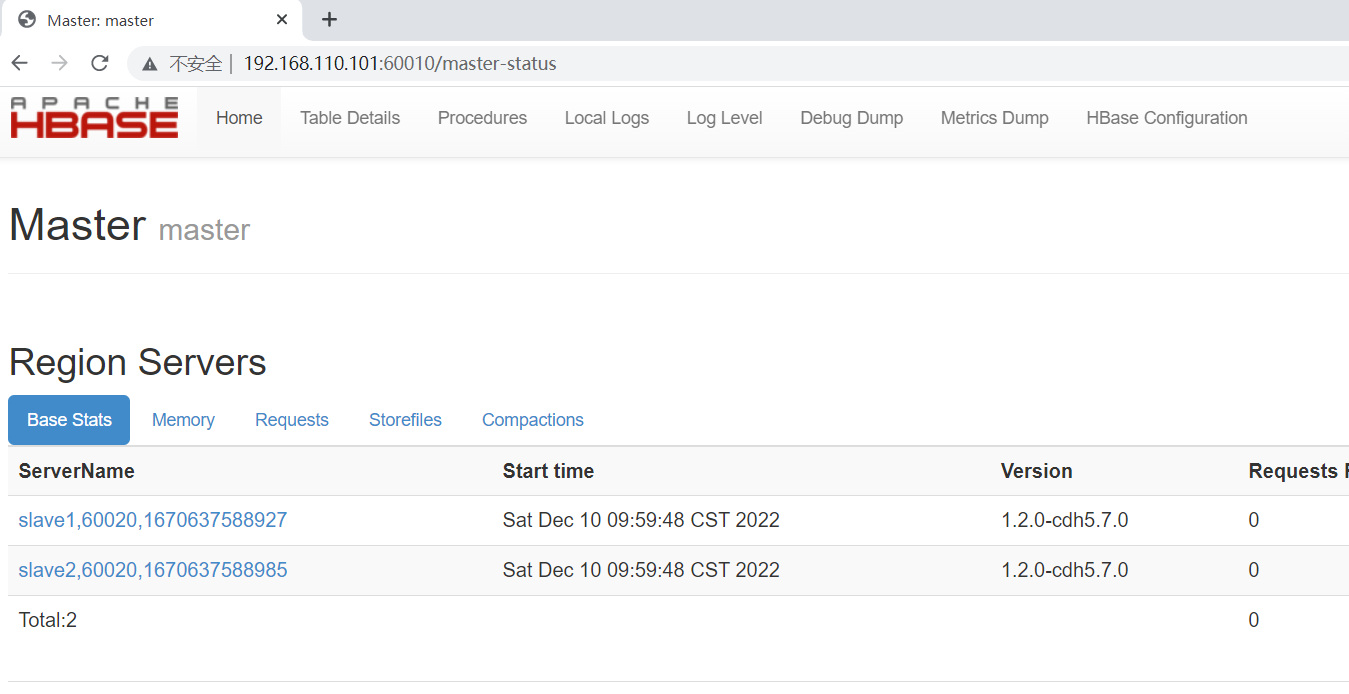


图3.7浏览器查看60010端口

1. Hbase编程题

基于课堂案例，定义一个方法，参数为课程名称，功能为完成根据该课程名称查询出该课程所有学生的成绩总分和均分，在 main 方法中测试该方法（课程名自行指定，如果是不存在的课程名，则输出提示信息后结束）。要求有完整的代码。

验证方法：在 IDEA 中输出运行结果

该方法代码如下：

1. **private** **static** **void** CourseScores(String coursename) throws Exception {
2. String single="";
3. **switch**(coursename){
4. **case** "Java":single="C001";
5. **break**;
6. **case** "大数据":single="C002";
7. **break**;
8. **case** "MySQL":single="C003";
9. **break**;
10. **case** "数学":single="C004";
11. **break**;
12. }
13. **if**(single.equals("")){System.out.println("无此课程");**return**;}
14. Table table=connection.getTable(TableName.valueOf("student"));
15. Scan scan=**new** Scan();
16. Filter filter=**new** ColumnPrefixFilter(single.getBytes());
17. scan.setFilter(filter);
18. ResultScanner result= table.getScanner(scan);
19. Result rs=null;
20. **int** sum=0;
21. **int** cnt=0;
22. **while**((rs=result.next())!=null) {
23. Cell[] cells = rs.rawCells();
24. **for** (**int** i = 0; i < cells.length; i++) {
25. Cell cell = cells[i];
26. String val = **new** String(CellUtil.cloneValue(cell));
27. sum+=Integer.parseInt(val);
28. cnt++;
29. }
30. }
31. System.out.println(coursename+"课程学生的总分为:"+sum+"\t平均分为:"+(sum\*1.0/cnt));
32. table.close();
33. }

运行完整代码，查询MySQL课程，结果如下：



图4.1MySQL课程学生总分和平均分

1. Hive 的安装配置（包括 MySQL 的安装配置）

验证方法：在 CentOS 7 中使用 beeline 方式连接 Hive

首先是上传并解压hive，之后改名为hive

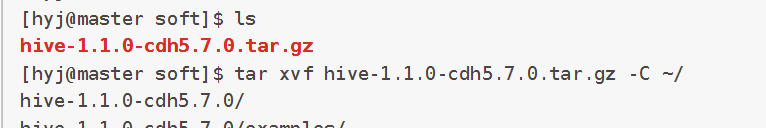


图5.1上传并解压hive的压缩文件

然后为hive配置环境路径



图5.2为hive设置环境变量

之后配置hive-env.sh文件和hive-site.xml文件，上传mysql的jar包，在虚拟机上安装相关依赖

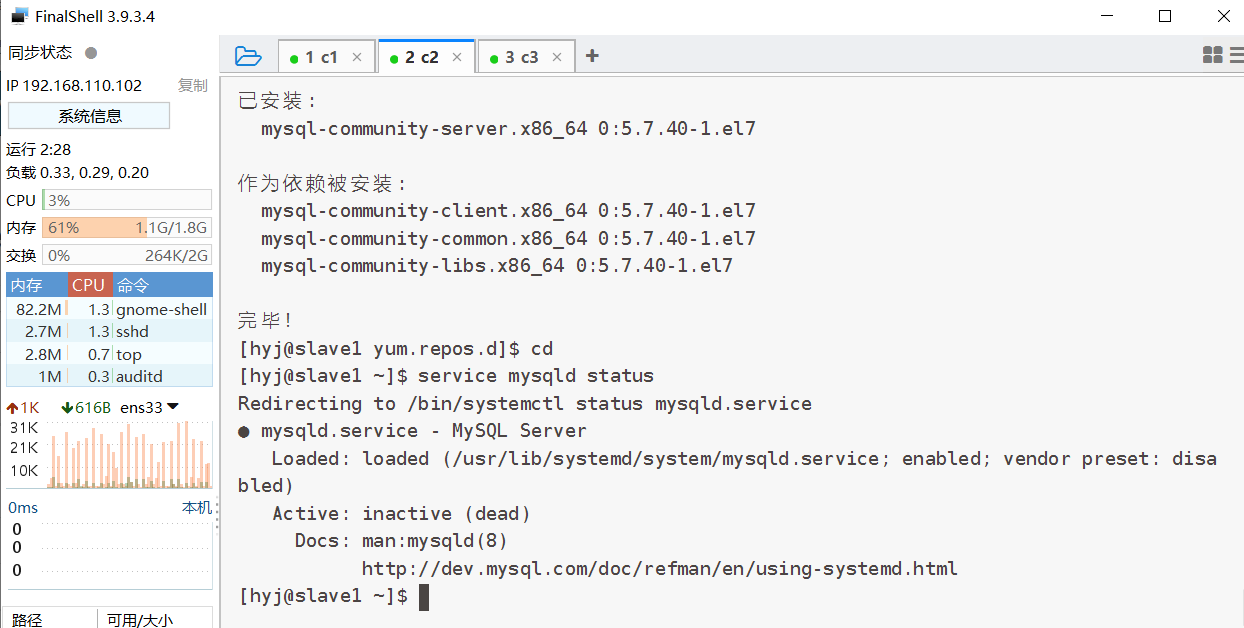


图5.3查看数据库状态

接着激活数据库并修改密码，修改为Hyjnb@102

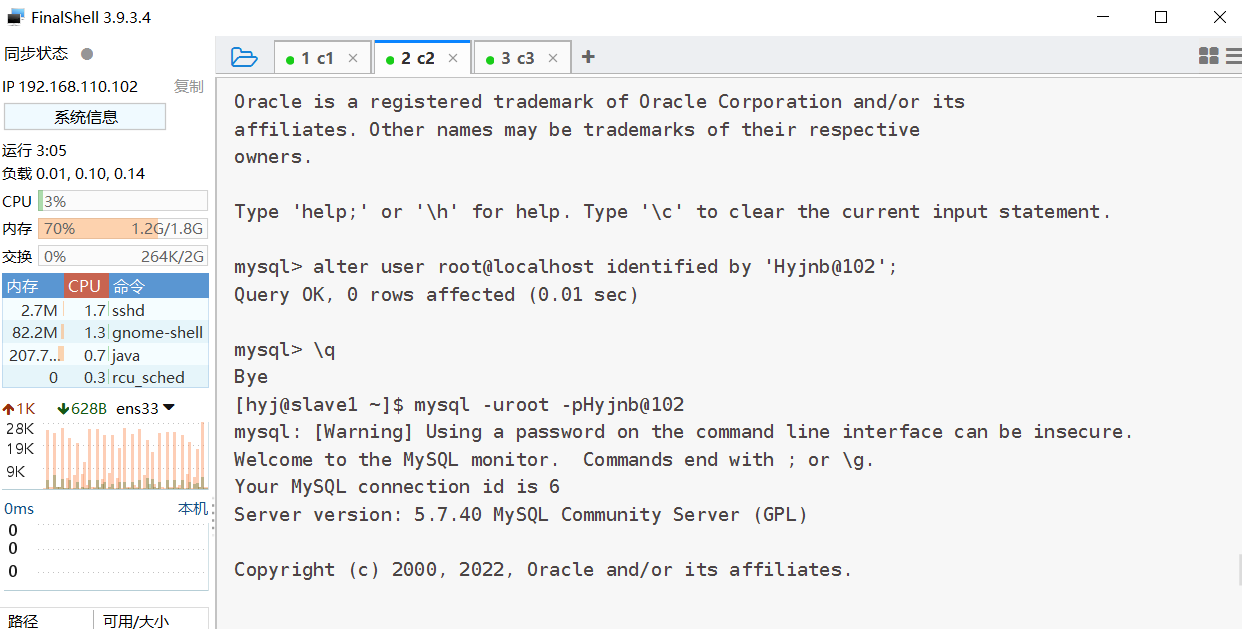


图5.4进入数据库并修改密码

最后用beeline命令进入beeline模式，利用！connect jdbc:hive2://master:10000连接到Hive

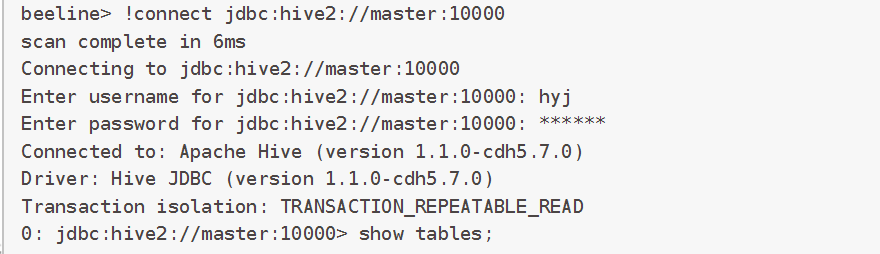


图5.5使用beeline方式连接Hive

1. Hive 编程题

在beeline模式下创建数据库，命名为：hive\_+自己姓名拼音缩写，将该数据文件上传到hdfs后，导入music表；编写SQL语句，完成以下查询：学号尾数为单数的同学统计查询每首歌的播放次数学号尾数为双数的同学统计查询每种播放端的播放次数每个步骤均要求完整截图。

先通过beeline连接Hive，进入beeline模式

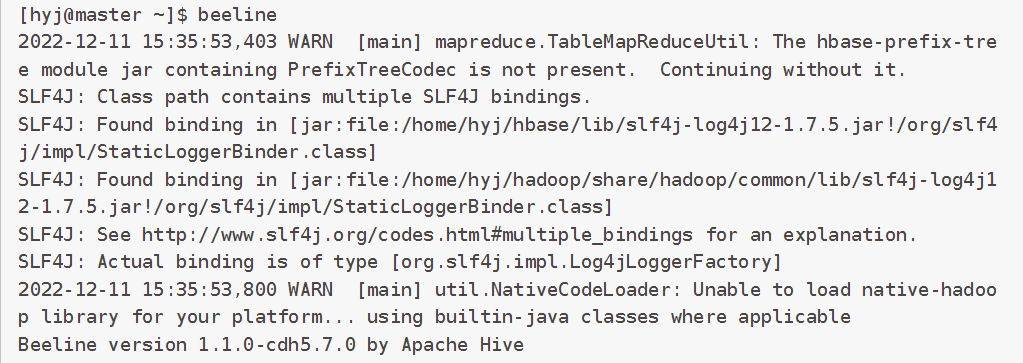


图6.1使用beeline连接

之后在beeline模式通过nohup hiveserver2 > hs2.log 2>&1 &语句连接数据库

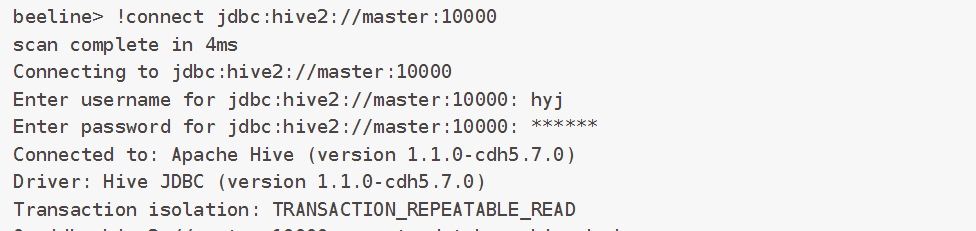


图6.2在beeline模式中连接数据库

进入数据库后按要求创建并查看数据库

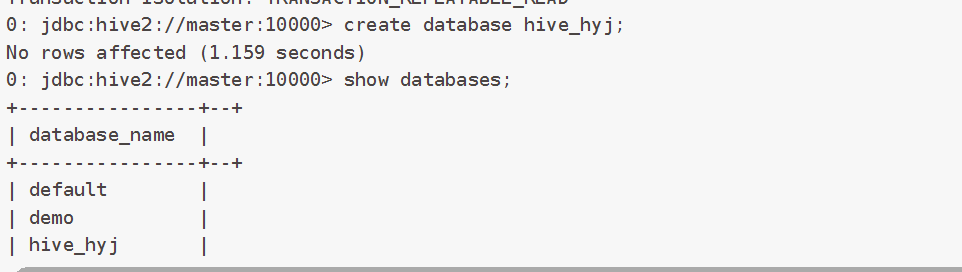


图6.3查看新建的hive\_hyj数据库

接着进入创建的hive\_hyj数据库，按照给定数据类型和题目要求创建表格，将事先上传在HDFS上的music.txt导入到新创建的数据库中

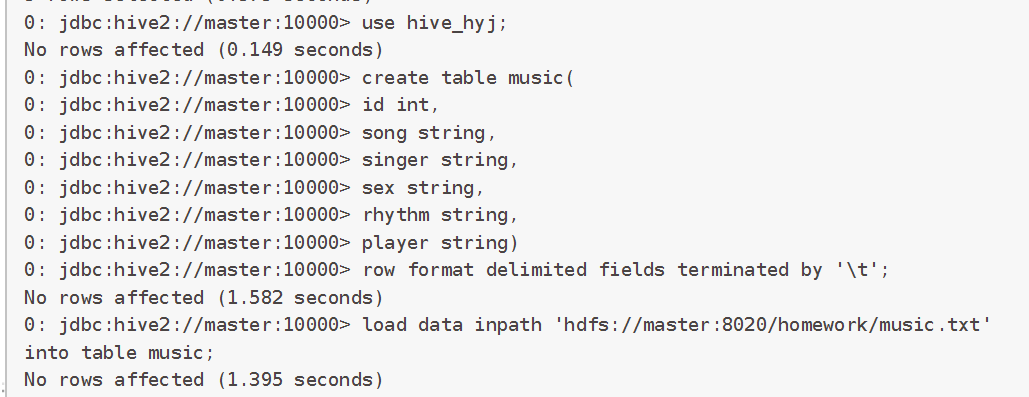


图6.4创建并导入music表

使用SQL语句查看表内容

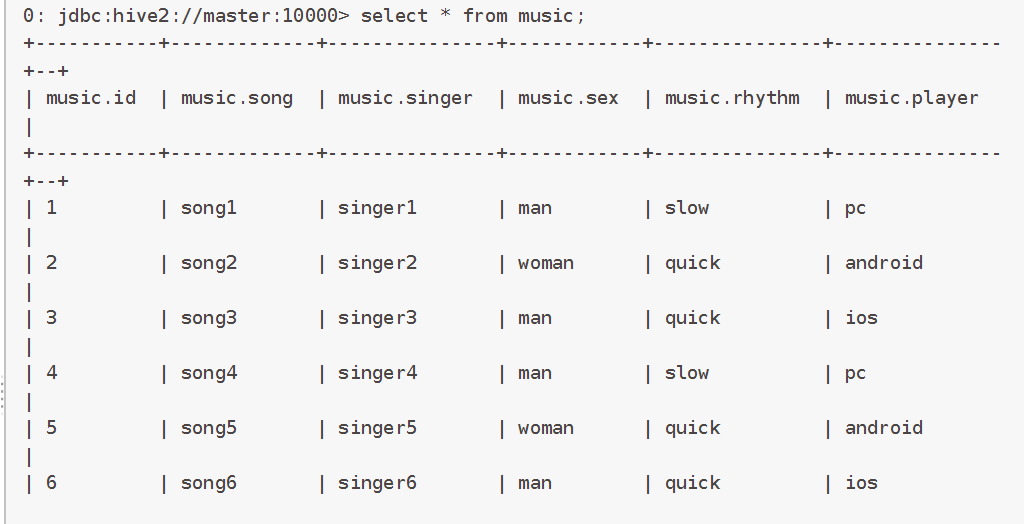


图6.5查看music表中内容

利用SQL语句查询每个播放器的使用次数

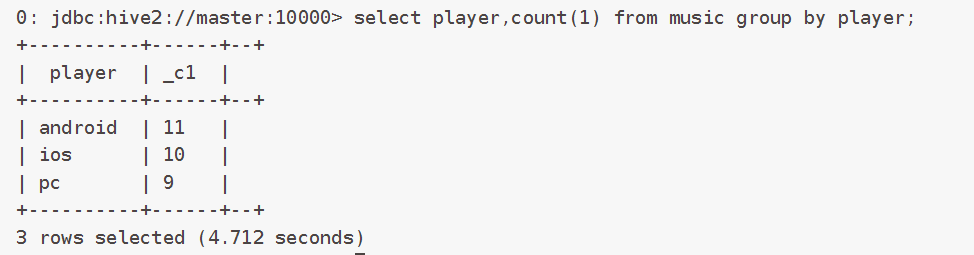


图6.6查询不同播放器的使用次数

# 三、实验总结

（可以总结实验中出现的问题以及解决的思路，也可以列出没有解决的问题）

1. 在第一次用mysql -uroot -p\*\*\*\*\*命令进入数据库时，密码中的符号前要加’\’转义，否则会报错
2. 在将IDEA上写的代码到虚拟机上运行时，每次都要先clean，然后在package，最后将打包好的文件上传