大数据平台技术课程实验报告

**实验四：Hbase的操作实践**

姓名：\_\_\_\_杨佳森\_\_\_\_

学号：\_\_\_\_2112080106

专业：\_\_\_ 数据科学与大数据技术

年级：\_ 2021

主讲教师：\_ 刘春 \_

实验时间： 2023年3月31日 至 2023年4月21日

实验内容与要求：

[1](#_Toc43287683) Hbase基本知识总结

2 Hbase的操作实践

2.1 Hbase的安装部署：记录部署过程以及相应问题和解决方法

2.2 Hbase的应用实践：按照PPT内容，实践Hbase的shell命令接口以及JAVA API接口

**特别提醒：**

**（1）基本知识点的总结请使用自己的语言，根据自己的理解去总结，就像写课程笔记一样，不要去网上抄写**

**（2）实践需要有截图以及相应的文字介绍，欢迎大家把自己在实践过程中碰到的问题以及解决方法也记录下来**

1. **Hbase基本知识的总结**
2. **Hbase数据库系统**

一种NoSQL数据库系统。

不支持SQL查询语言，也缺乏了传统关系型数据库所具有的特性和遵循的机制。

借鉴了谷歌的BigTable的设计，并通过Java语言进行开发，是BigTable的开源实现。

提供面向列、可伸缩的分布式存储。

运行于HDFS之上，是Hadoop的重要组件。

解决了HDFS只适合于批量访问不能随机访问的问题。

HBase 是 BigTable 的开源 Java 版本。**是建立在 HDFS 之上**，提供高可靠性、高性能、列存储、可伸缩、实时读写 NoSql 的数据库系统。

它介于 NoSql 和 RDBMS 之间，仅能通过主键(row key)和主键的 range 来检索数据，仅支持单行事务(可通过 hive 支持来实现多表 join 等复杂操作)。

主要用来存储结构化和半结构化的松散数据。

Hbase 查询数据功能很简单，不支持 join 等复杂操作，不支持复杂的事务（行级的事务） Hbase 中支持的数据类型：byte[] 与 hadoop 一样，Hbase 目标主要依靠**横向扩展**，通过不断增加廉价的商用服务器，来增加计算和存储能力。

HBase 中的表一般有这样的特点：

* 大：一个表可以有上十亿行，上百万列
* 面向列:面向列(族)的存储和权限控制，列(族)独立检索。
* 稀疏:对于为空(null)的列，并不占用存储空间，因此，表可以设计的非常稀疏。

1. **Hbase的数据模型**
2. 逻辑视图

反映数据逻辑上的组织结构。

逻辑结构便于人们理解。

同传统的关系型数据库系统一样，利用表格来组织数据。

每个表由行和列组成。

但又与传统的关系型数据库有区别。

1. **物理视图**

反映数据的存储结构。

**（三）Hbase基础架构**

* **HMaster**

**功能**：

1. 监控 RegionServer
2. 处理 RegionServer 故障转移
3. 处理元数据的变更
4. 处理 region 的分配或移除
5. 在空闲时间进行数据的负载均衡
6. 通过 Zookeeper 发布自己的位置给客户端

* **RegionServer**

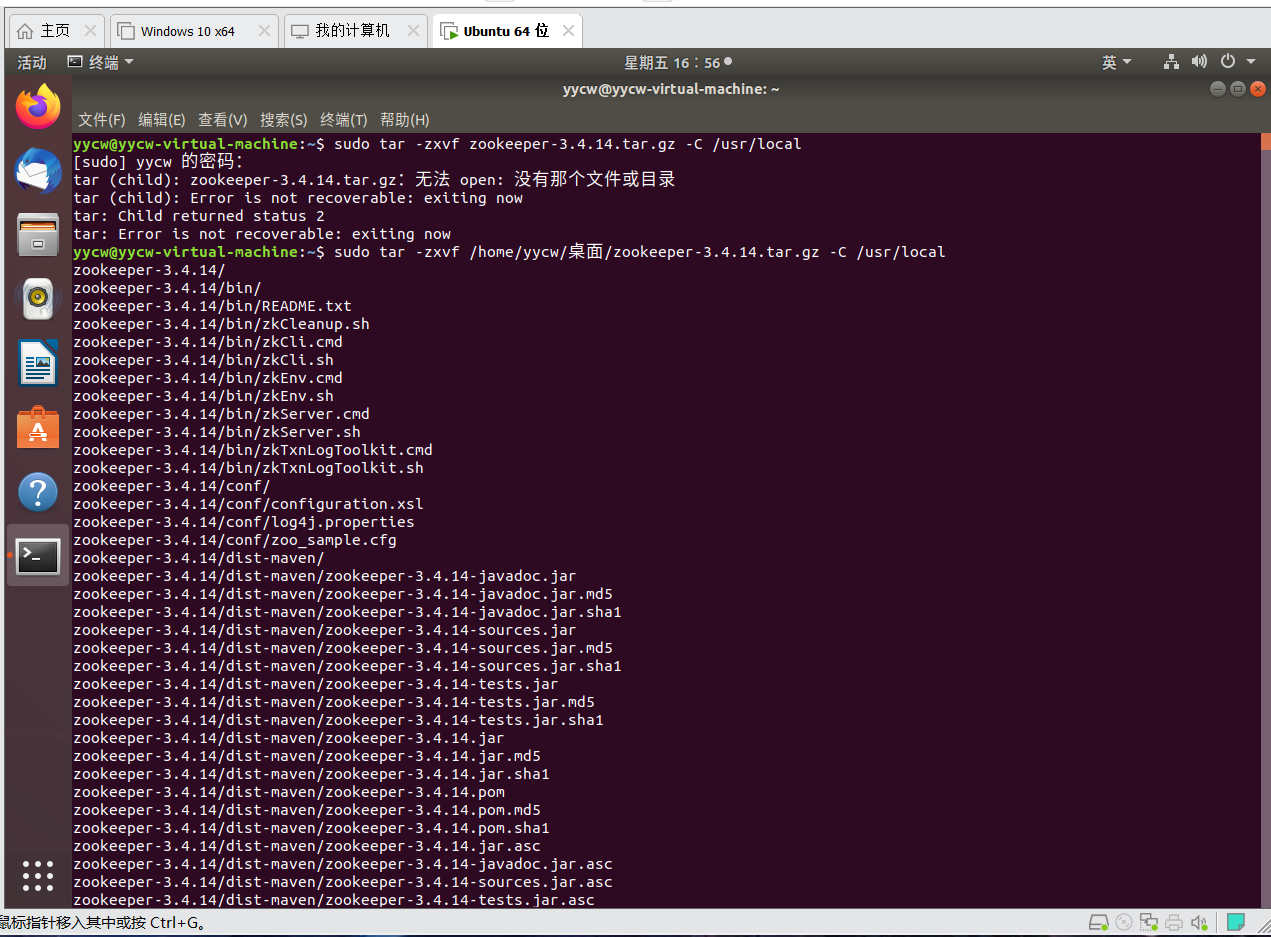
**功能**：

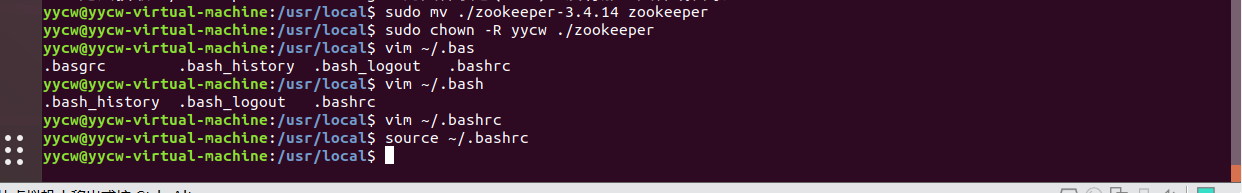
1. 负责存储 HBase 的实际数据
2. 处理分配给它的 Region
3. 刷新缓存到 HDFS
4. 维护 HLog
5. 执行压缩
6. 负责处理 Region 分片
7. **Hbase的操作实践**

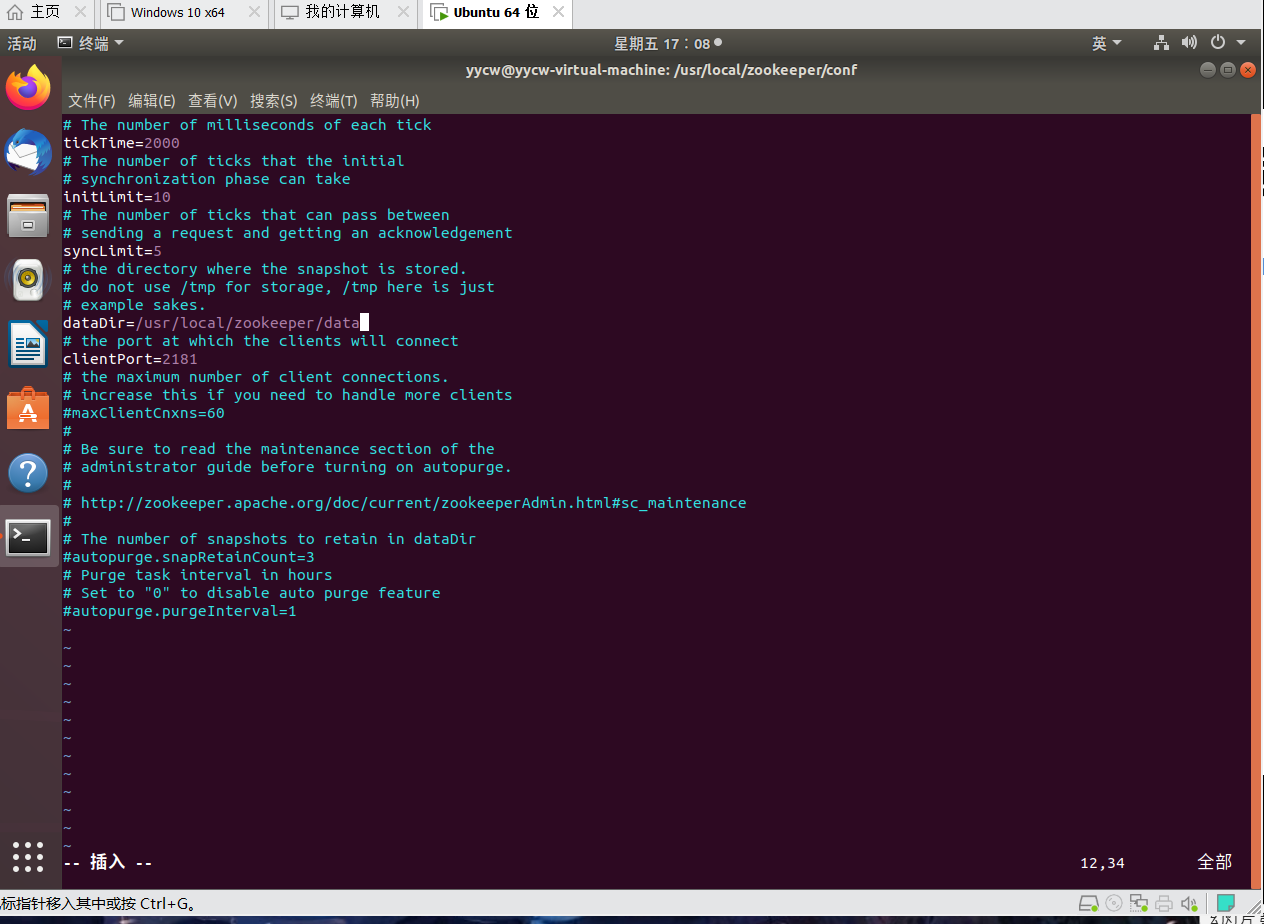
**2.1 Hbase的安装部署实践**

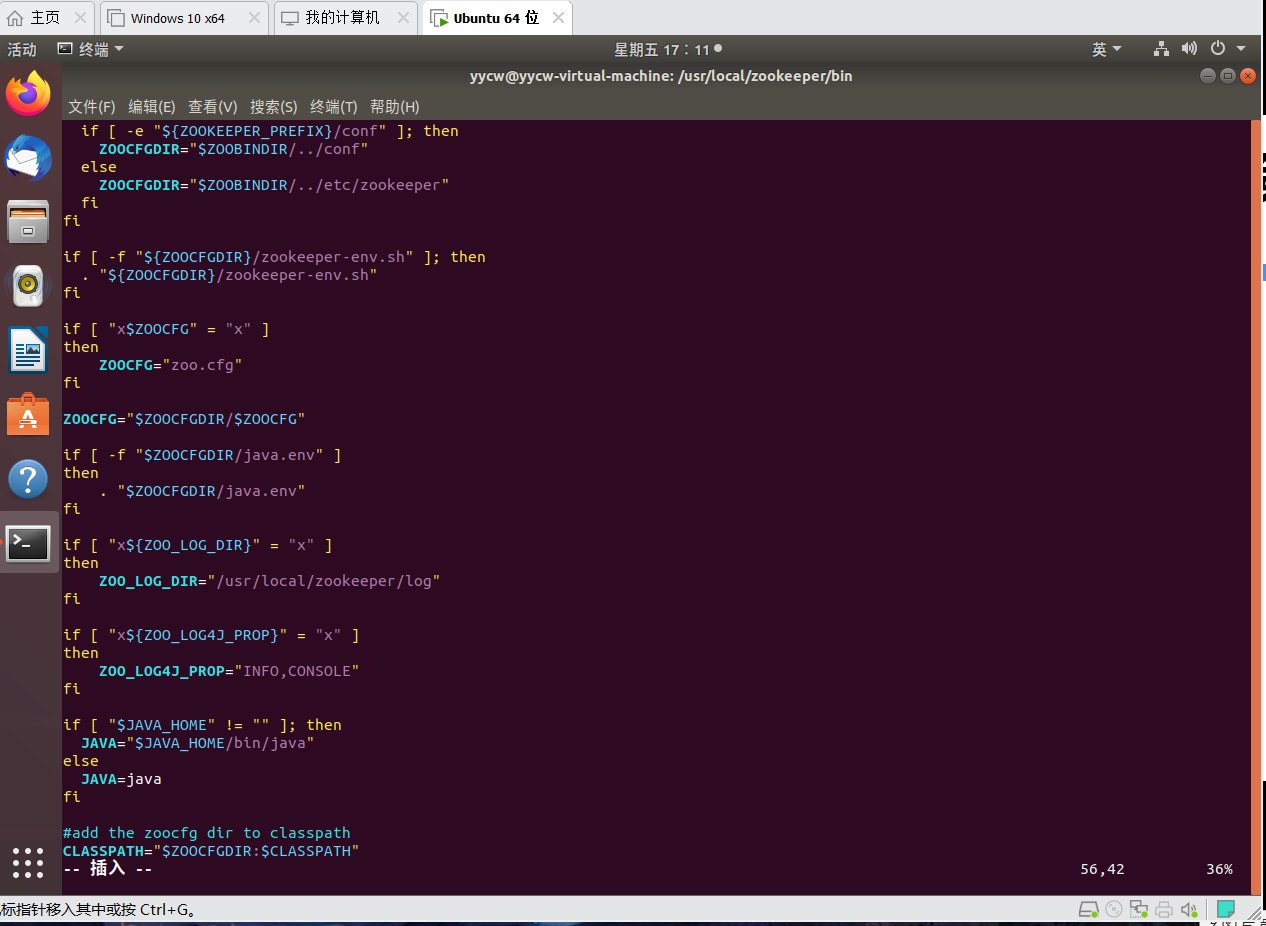
**1）安装zookeeper**

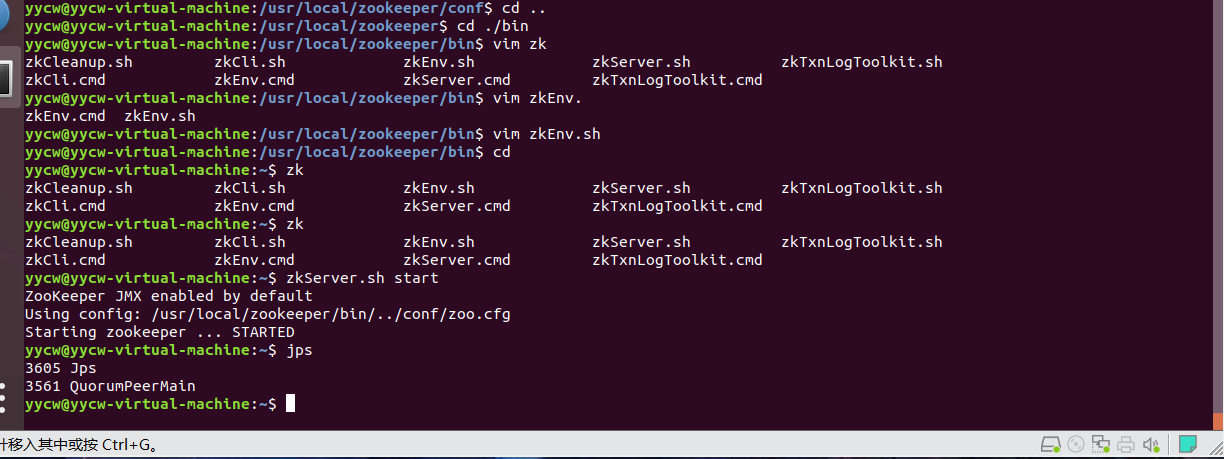
解压文件时应注意加上文件的绝对路径



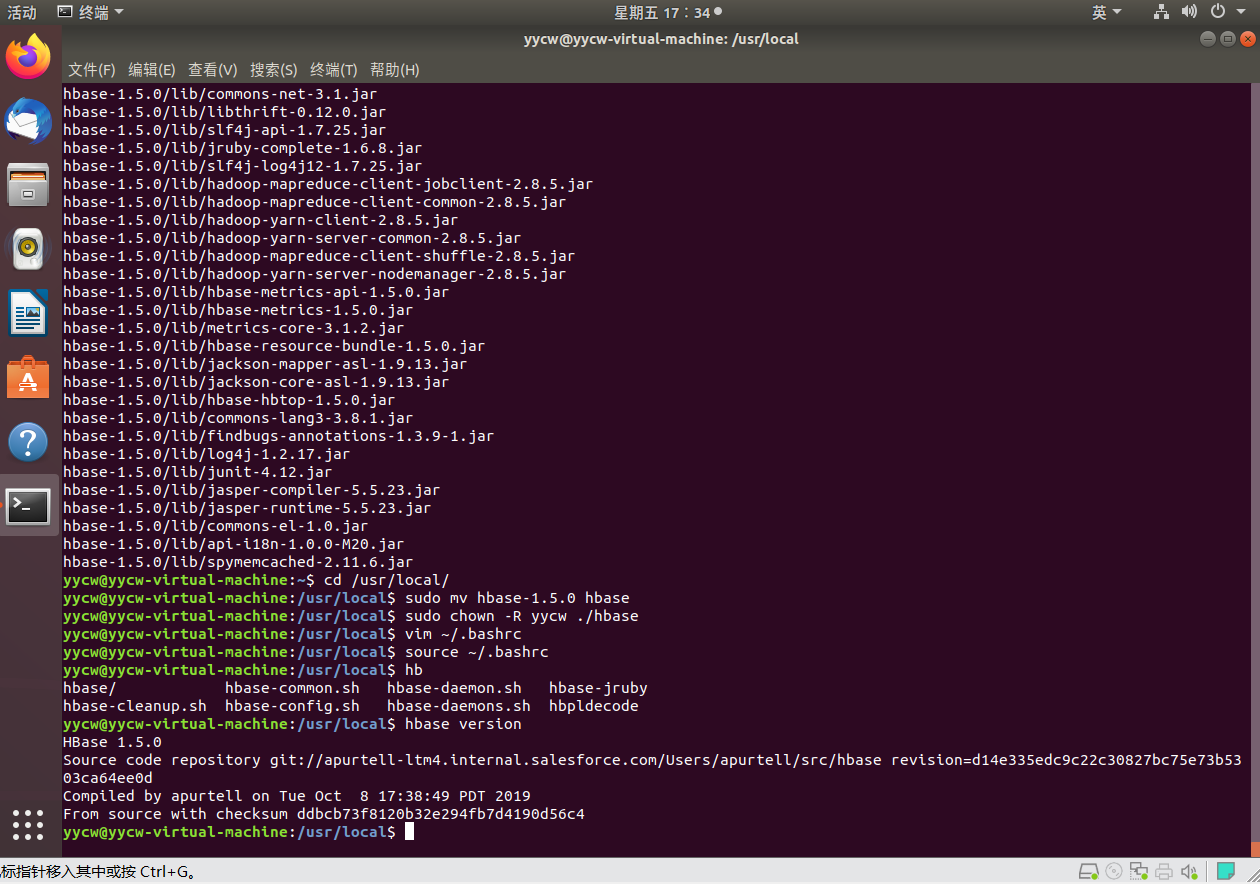
****

****

****

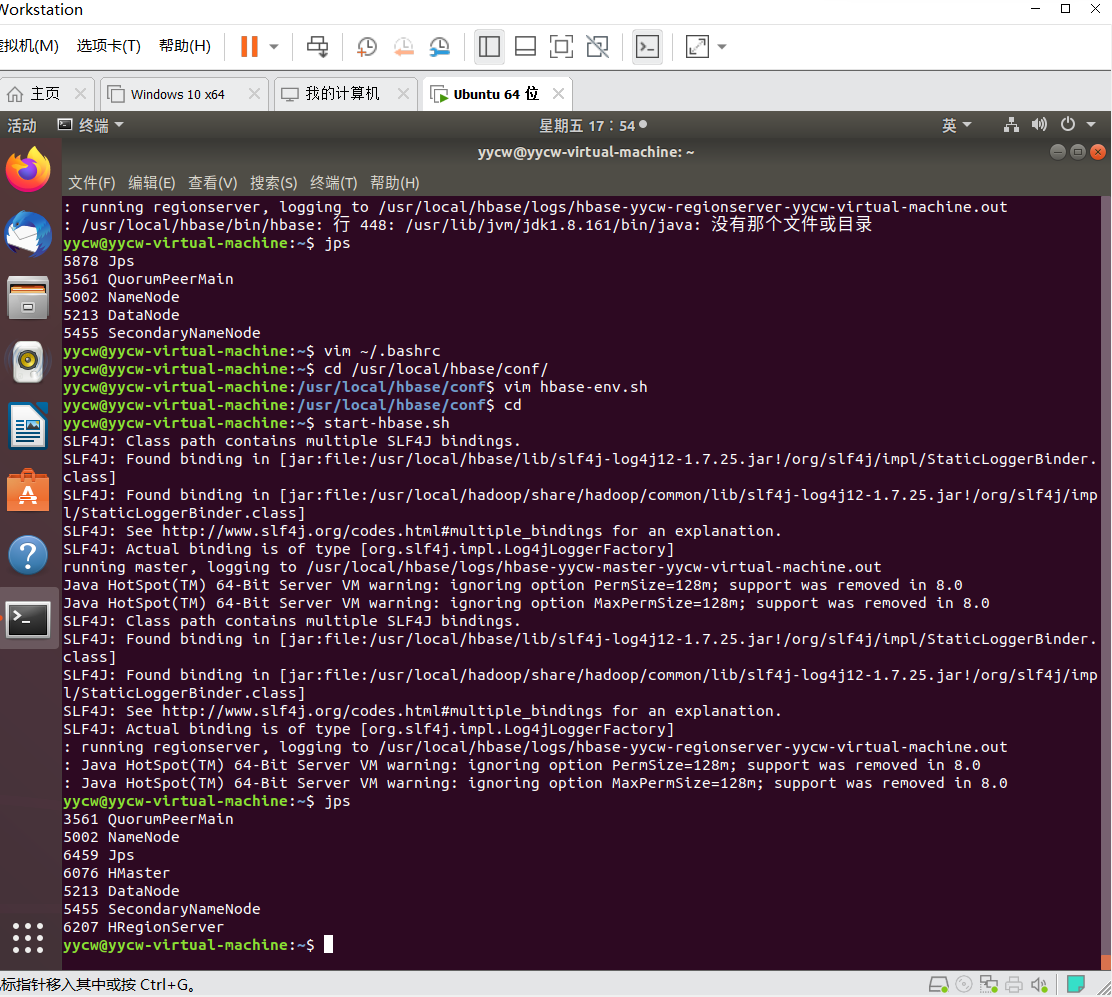
****

**2）安装hbase**

****

**3）伪分布式环境配置**

环境配置应该与之前的保持一致

****

**2.2 Hbase的shell命令实践**

**1）进入hbase shell**

进入hbase shell时出现了报错和警告：

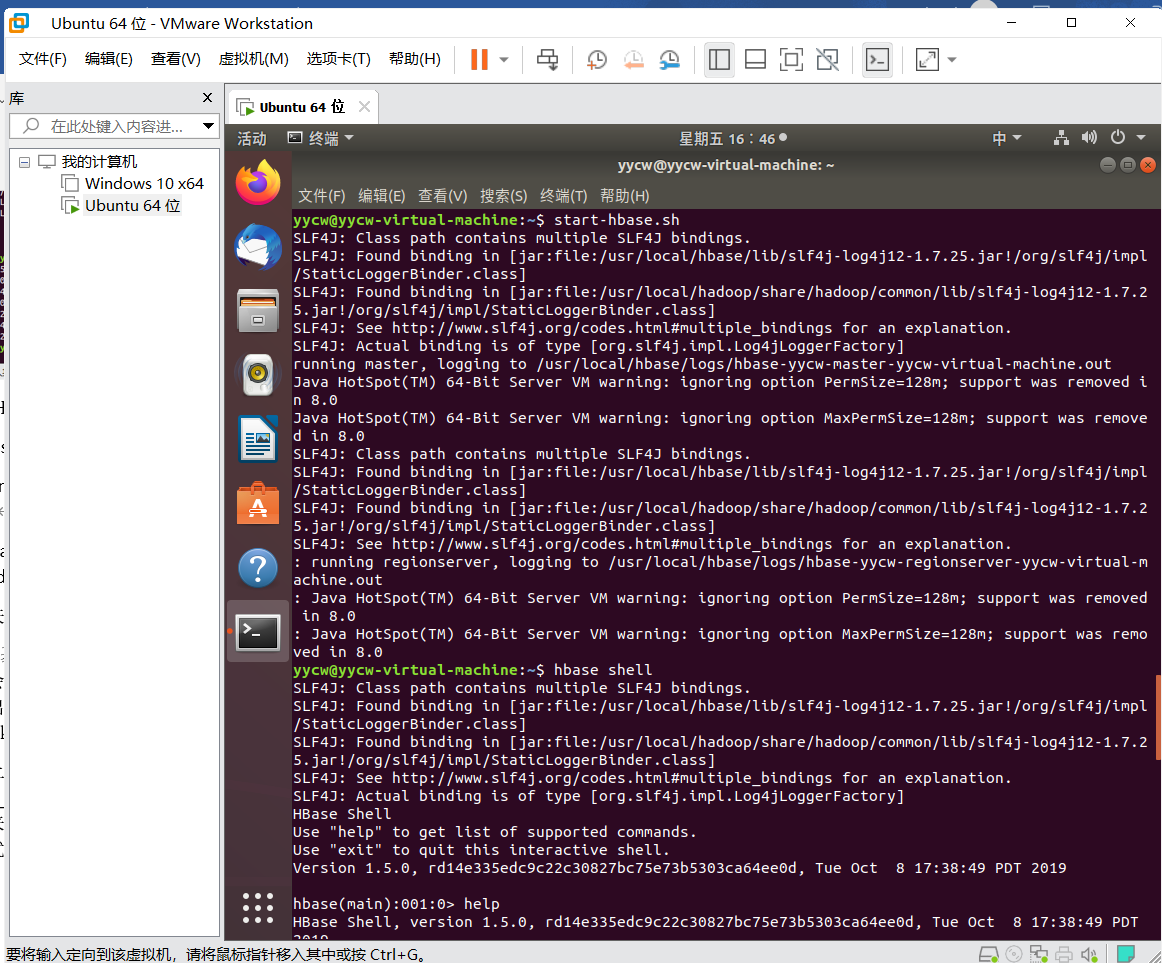
ERROR [main] zookeeper.RecoverableZooKeeper: ZooKeeper exists failed after 4 attempts

WARN[main] zookeeper.ZKUtil: hconnection-0x177515d10x0, quorum=localhost:2181, baseZNode=/hbase Unable to set watcher on znode (/hbase/hbaseid)

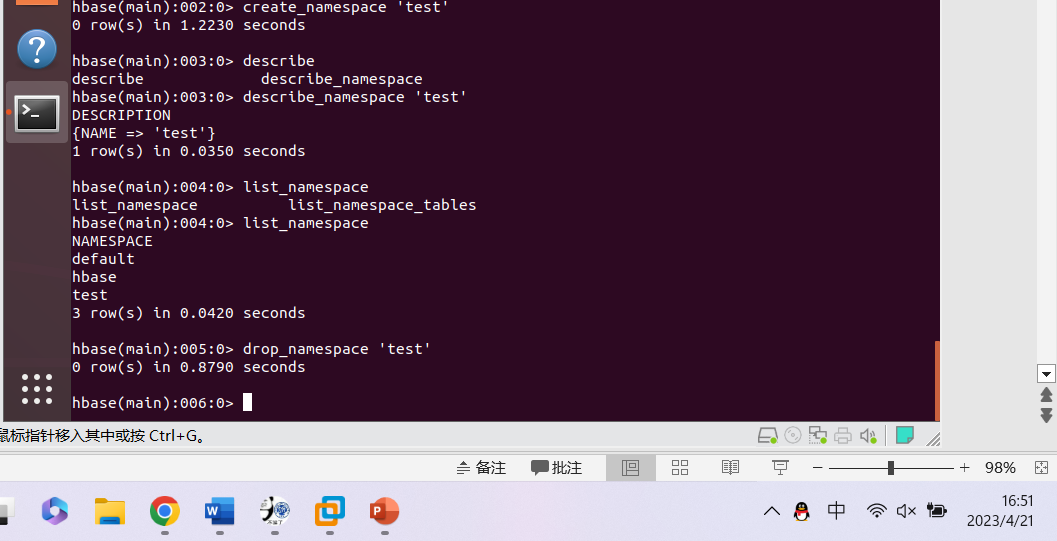
查阅相关资料时我发现

HBASE是基于zookeeper的，而zookeeper是基于Hadoop的，启动了Hadoop后我们总会启动zookeeper再启动HBASE，而hbase内是自带zookeeper的，这就可能会出现情况 ，在配置时是否使用了hbase自带的zookeeper或者是自己安装的zookeeper，导致端口2181是否被抢占。

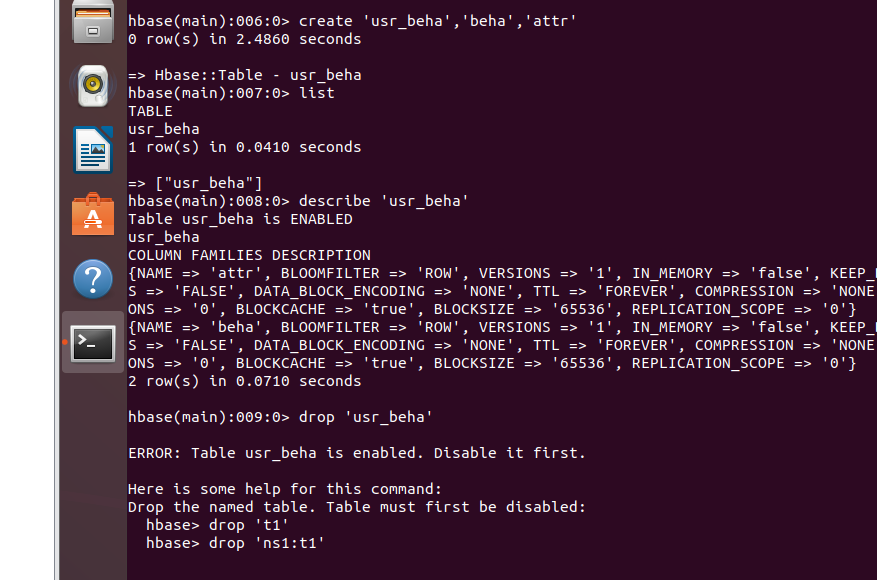
根据上面这个思路，我只需要将hbase-env.sh中的 export HBASE\_MANAGES\_ZK 属性设置为false，当我用vim去修改时我发现，这个属性本来设置的就是false，突然，我想明白了，并不是2181端口被抢占，而是我压根就没启动它，所以我只需要在hbase启动前启动zookeeper即可。



**2）创建、查看、删除命名空间**

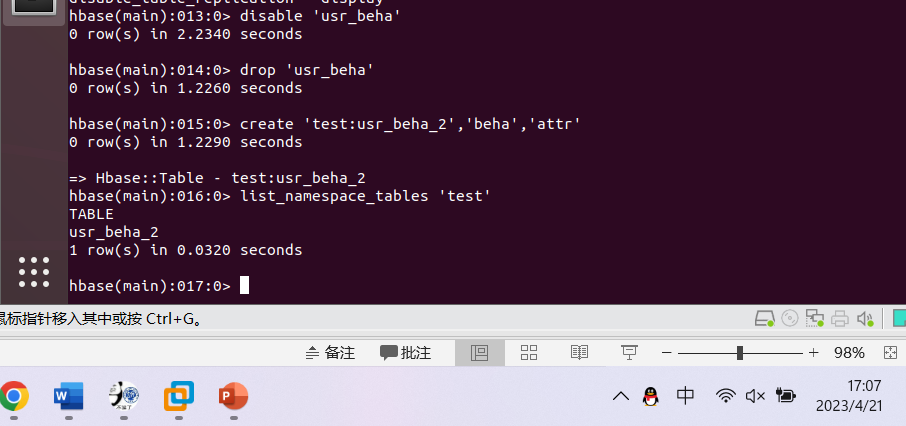
****

**3）创建、查看、删除表以及使表有效无效**

****

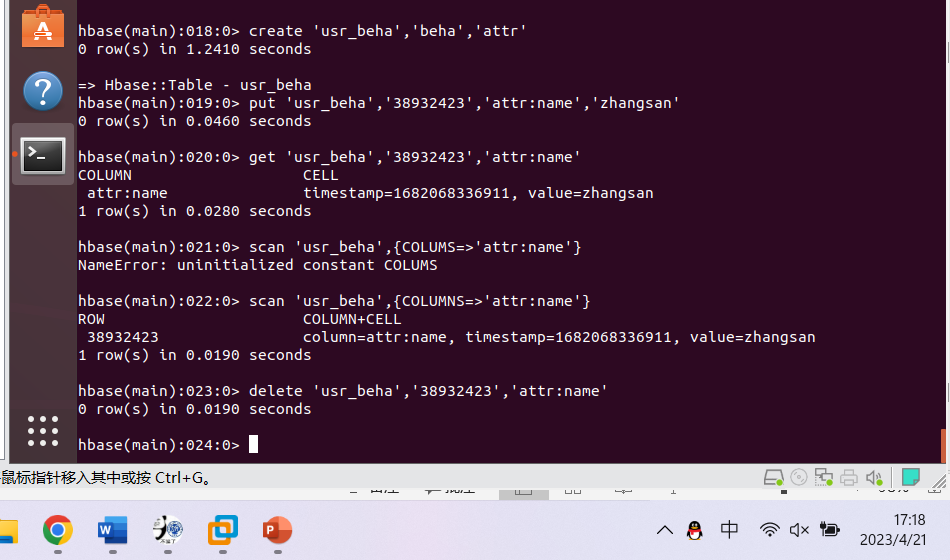
当我进行删除表时，出现报错ERROR: Table usr\_beha is enabled. Disable it first.

这说明了有效表不能被删除，应该先disable ‘usr\_beha’使其无效在删除。

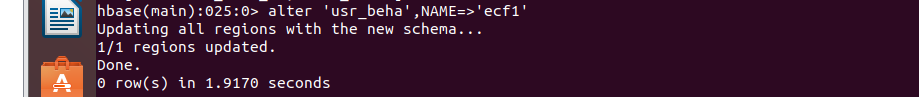
  
**4）添加、获取、删除单元格中的数据**

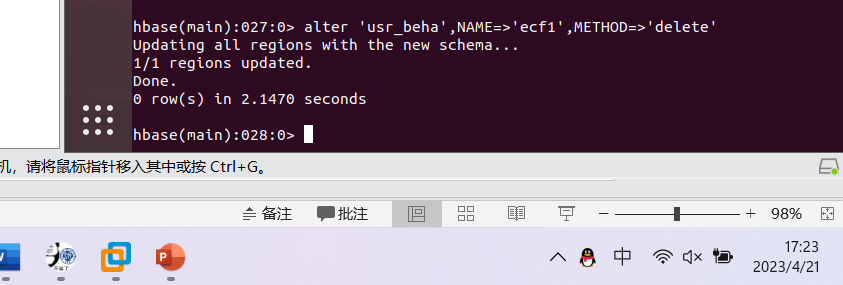
使用scan命令查看整个表的信息：

**scan ‘usr\_beha’,{COLUMNS=>’attr:name’}**



**5)修改表的结构**

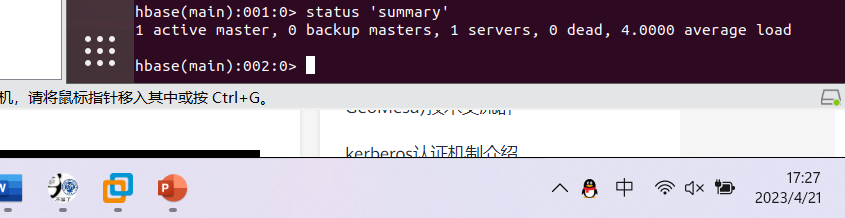




**6）退出hbase shell**

****

**7）查看hbase集群的状态**

****

**2.3 Hbase的Java API接口操作实践**

**1）pom.xml**

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>org.example</groupId>

<artifactId>HBASE</artifactId>

<version>1.0-SNAPSHOT</version>

<properties>

<maven.compiler.source>8</maven.compiler.source>

<maven.compiler.target>8</maven.compiler.target>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

</properties>

<dependencies>

<dependency>

<groupId>org.apache.hbase</groupId>

<artifactId>hbase-client</artifactId>

<version>1.5.0</version>

</dependency>

<dependency>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-clean-plugin</artifactId>

<version>2.5</version>

</dependency>

<dependency>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.1</version>

</dependency>

<dependency>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-deploy-plugin</artifactId>

<version>2.7</version>

</dependency>

<dependency>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-install-plugin</artifactId>

<version>2.4</version>

</dependency>

<dependency>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-jar-plugin</artifactId>

<version>2.4</version>

</dependency>

<dependency>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-resources-plugin</artifactId>

<version>2.6</version>

</dependency>

<dependency>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-site-plugin</artifactId>

<version>3.3</version>

</dependency>

<dependency>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-surefire-plugin</artifactId>

<version>2.12.4</version>

</dependency>

<dependency>

<groupId>log4j</groupId>

<artifactId>log4j</artifactId>

<version>1.2.17</version>

</dependency>

</dependencies>

</project>

**2)HbaseApp**

package org.example;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.hbase.HBaseConfiguration;

import org.apache.hadoop.hbase.\*;

import org.apache.hadoop.hbase.client.\*;

import org.apache.hadoop.hbase.util.Bytes;

import org.apache.log4j.Logger;

import java.io.IOException;

import java.util.ArrayList;

import java.util.List;

public class HbaseApp {

public static Admin admin; //admin对象

public static Connection connection;//连接HBase的对象

public static void main(String[] args){

init(); //建立连接

try{

createTable("usr\_beha",new String[] {"beha","attr"}); //建表

User user1 = new User("38932423","zhangsan","356");

insertData("usr\_beha",user1);//插入数据

User user2 = new User("38234278","lisi","237");

insertData("usr\_beha",user2);

List<User> list = getAllData("usr\_beha"); //查询数据

for(User user : list){

System.out.println(user.toString());//打印输出

}

User user3 = getDataByRowKey("usr\_beha","38932423");//根据行键查询

System.out.println(user3.toString());

//获取某单元格的数据

String username = getDatafromCell("usr\_beha","38932423","attr","username");

System.out.println("username: "+username);

//删除某单元格的数据

deleteDataofCell("usr\_beha","38932423","attr","username");

//查看删除之后的结果

User user4 = getDataByRowKey("usr\_beha","38932423");

System.out.println(user4.toString());

}catch(IOException e){

e.printStackTrace();

}

close();//关闭连接

}

//初始化，建立与HBase的连接，并基于所建立的连接获取admin对象

public static void init(){

Configuration configuration = HBaseConfiguration.create();

//设置hbase.rootdir目录，这个目录可用来持久化HBase的目录

configuration.set("hbase.rootdir","hdfs://localhost:9000/hbase");

try{

connection = ConnectionFactory.createConnection(configuration);

//获取admin对象

admin = connection.getAdmin();

}catch (IOException e){

e.printStackTrace();

}

}

//根据输入的列族名称的集合创建表

public static void createTable(String strTableName,String[] strColFamilies) throws IOException{

TableName tableName = TableName.valueOf(strTableName);

//首先通过admin对象判断同名的表是否已经存在

if(admin.tableExists(tableName)){

System.out.println(strTableName+"exists!");

}else{

//创建一个封装表的描述信息的对象

HTableDescriptor hTableDescriptor = new HTableDescriptor(tableName);

//创建封装表列族描述信息的对象，并将它们添加到HtableDescriptor对象中

for(String col : strColFamilies){

HColumnDescriptor hColumnDescriptor = new HColumnDescriptor(col);

hTableDescriptor.addFamily(hColumnDescriptor);

}

//基于HtableDescriptor对象，通过admin对象的create Table()方法来创建表

admin.createTable(hTableDescriptor);

}

}

//写入数据，数据封装在User对象中

public static void insertData(String tableName,User user) throws IOException{

//根据表名获取表的实例

Table table = connection.getTable(TableName.valueOf(tableName));

Put put = new Put(user.getId().getBytes());

put.addColumn("attr".getBytes(),"username".getBytes(),user.getUsername().getBytes());

put.addColumn("beha".getBytes(),"watch".getBytes(),user.getWatch().getBytes());

table.put(put);

}

public static User getDataByRowKey(String tableName,String rowKey) throws IOException{

Table table = connection.getTable(TableName.valueOf(tableName));

Get get = new Get(rowKey.getBytes());

User user = new User();

user.setId(rowKey);

Result result = table.get(get);

for(Cell cell : result.rawCells()){

String colName = Bytes.toString(cell.getQualifierArray(),cell.getQualifierOffset(),cell.getQualifierLength());

String value = Bytes.toString(cell.getValueArray(),cell.getValueOffset(),cell.getValueLength());

if(colName.equals("username")){

user.setUsername(value);

}

if(colName.equals("watch")){

user.setWatch(value);

}

}

return user;

}

public static String getDatafromCell(String tableNAme,String rowKey,String colFamily,String col) throws IOException{

Table table = connection.getTable(TableName.valueOf(tableNAme));

Get get = new Get(rowKey.getBytes());

get.addColumn(colFamily.getBytes(),col.getBytes());

if(table.exists(get)){

Result result = table.get(get);

String strResult = new String(result.getValue(colFamily.getBytes(),col.getBytes()));

return strResult;

}else{

System.out.print("The cell is empty!");

return null;

}

}

public static List<User> getAllData(String tableName){

Table table = null;

List<User> list = new ArrayList<User>();

try{

table = connection.getTable(TableName.valueOf(tableName));

ResultScanner results = table.getScanner(new Scan());

User user = null;

for(Result result : results){

String id = new String(result.getRow());

user = new User();

for(Cell cell : result.rawCells()){

String row = Bytes.toString(cell.getRowArray(),cell.getRowOffset(),cell.getRowLength());

String family = Bytes.toString(cell.getFamilyArray(),cell.getFamilyOffset(),cell.getFamilyLength());

String colName = Bytes.toString(cell.getQualifierArray(),cell.getFamilyOffset(),cell.getFamilyLength());

String value = Bytes.toString(cell.getValueArray(),cell.getValueOffset(),cell.getValueLength());

user.setId(row);

if(colName.equals("username")){

user.setUsername(value);

}

if(colName.equals("watch")){

user.setWatch(value);

}

}

list.add(user);

}

}catch (IOException e){

e.printStackTrace();

}

return list;

}

public static void deleteDataofCell(String tableName,String rowKey,String colFamily,String col) throws IOException{

Table table = connection.getTable(TableName.valueOf(tableName));

Delete delete = new Delete(rowKey.getBytes());

delete.addColumns(colFamily.getBytes(),col.getBytes());

table.delete(delete);

}

public static void deleteTable(String tableName){

try{

TableName tablename = TableName.valueOf(tableName);

admin.disableTable(tablename);

admin.deleteTable(tablename);

}catch (IOException e){

e.printStackTrace();

}

}

public static void close(){

try{

if(admin != null){

admin.close();

}

if(connection != null){

connection.close();

}

}catch (IOException e){

e.printStackTrace();

}

}

}

**3)User**

package org.example;

public class User {

private String id = null;

private String username = null;

private String watch = null;

public User(String id,String username,String watch){

this.id = id;

this.username = username;

this.watch = watch;

}

public User(){}

public String getId(){

return id;

}

public void setId(String id){

this.id = id;

}

public String getUsername(){

return username;

}

public void setUsername(String username){

this.username = username;

}

public String getWatch(){

return watch;

}

public void setWatch(String watch){

this.watch = watch;

}

public String toString(){

return "User\_info:" + "id = '" + id + '\''

+ ",username = '" + username + '\''+

",watch = '" + watch + '\'';

}

}

**4)运行结果**

