**南京信息工程大学**

**Spark技术应用课程设计报告**

学院名称： 数学与统计学院

专业： 信息与计算科学（嵌入式培养）

班级： 18（1）班

学号：

姓名：

2020年 1月 15日

**目录**

[一、项目简介 3](#_Toc61614738)

[1.1项目介绍 3](#_Toc61614739)

[1.2开发工具 3](#_Toc61614740)

[1.3开发环境 3](#_Toc61614741)

[二、准备数据 3](#_Toc61614742)

[2.1 批量导入数据至HDFS 3](#_Toc61614743)

[三、算法模型：MR并行算法 4](#_Toc61614744)

[3.1算法框架图 4](#_Toc61614745)

[3.2算法工作流程图 5](#_Toc61614746)

[3.3 洗牌过程 6](#_Toc61614747)

[四、Hbase概述 6](#_Toc61614748)

[4.1 hbase表特点 6](#_Toc61614749)

[4.2 hbase应用场景 7](#_Toc61614750)

[4.3 hbase数据模型 7](#_Toc61614751)

[4.3 hbase体系结构 8](#_Toc61614752)

[五、程序开发 9](#_Toc61614753)

[5.1 TempMaxOfYear.java-数据加载及清洗 9](#_Toc61614754)

[5.2 TempMaxOfYear.java-统计每年的最高气温 12](#_Toc61614755)

[5.3 TempMinOfYear.java-统计每年的最低气温 14](#_Toc61614756)

[5.4 TempAvgOfYear.java-统计每年的平均气温 17](#_Toc61614757)

[5.5 RainDayOfYear.java-统计每年的下雨天数 19](#_Toc61614758)

[5.6 ForecastAfter.java-预测明天气温 22](#_Toc61614759)

[5.7 HbaseUtil.java- hbase工具类 27](#_Toc61614760)

[5.8 Starter.java-整体功能：菜单与系统 29](#_Toc61614761)

[5.9 运行结果 35](#_Toc61614762)

[五、总结与展望 40](#_Toc61614763)

# 一、项目简介

## 1.1项目介绍

本项目主要运用Java语言，利用Hadoop、Hbase以及VMware Workstation Pro在eclipse环境下，对较大量的气温数据进行简单筛选处理并分析，得到年最高、最低、平均气温、下雨天数等数据分析结果的简单案例的实现，并能够以数据分析所得到的结果对未来天气进行简单预测，能够有效的实现对气温数据的简单分析及预测。

利用该项目设计成果能够高效率的完成对大量天气温度等数据的筛选、清洗，方便对于不同年份天气状况的分析，便于实际问题中对大量气温数据进行处理、分析及预测。

此外，本项目主要是对于大量数据按照时间进行数据清洗与分析，简单修改后可以处理其他温度数据，以及不同时刻的简单数据。是一个应用面非常广泛，适用情况非常多的实用案例。

## 1.2开发工具

Java语言 、Hadoop、Hbase、VMware Workstation Pro

## 1.3开发环境

Eclipse开发环境

# 二、准备数据

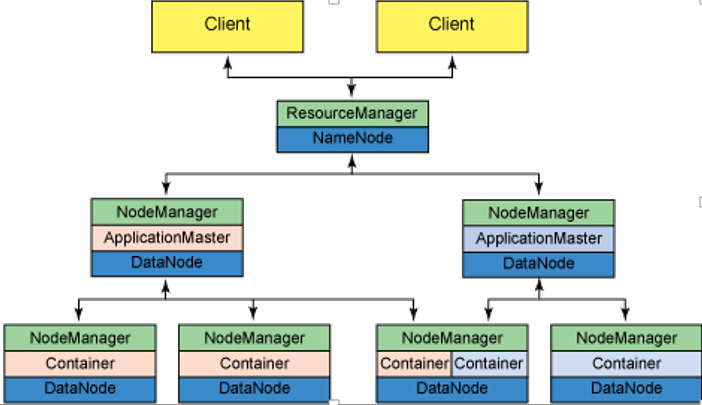
## 2.1 批量导入数据至HDFS

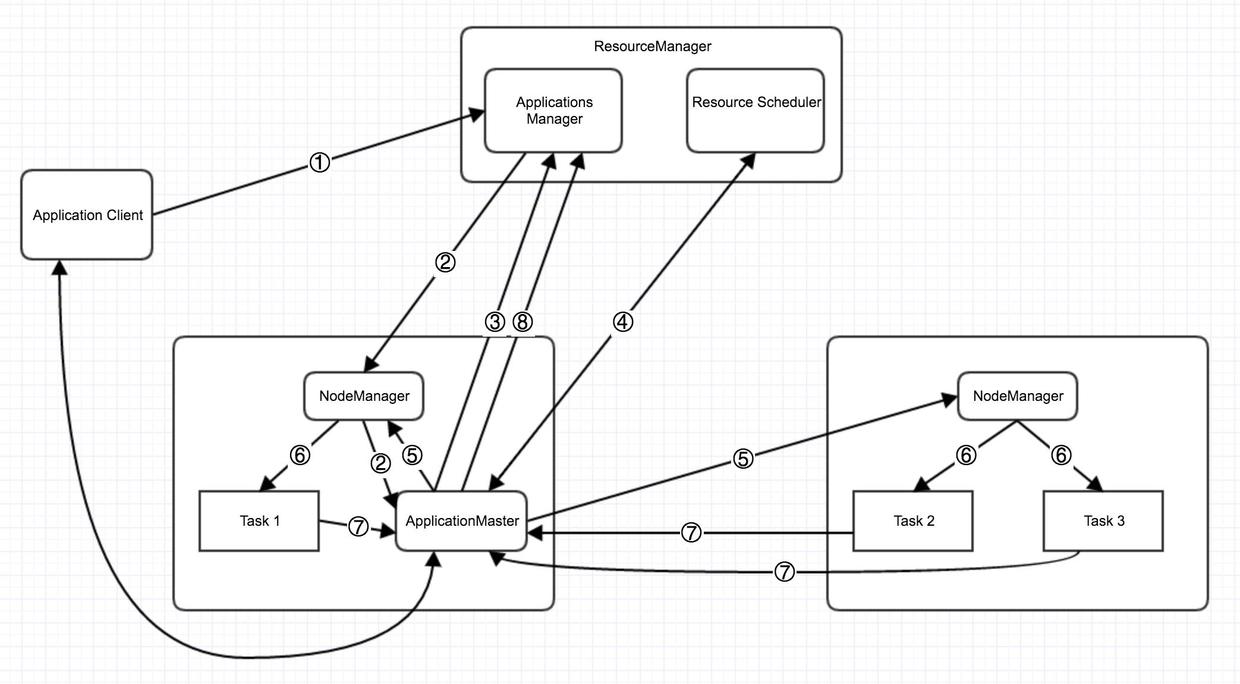
data.txt

参考**5.1BulkLoad.java-数据导入及清洗**

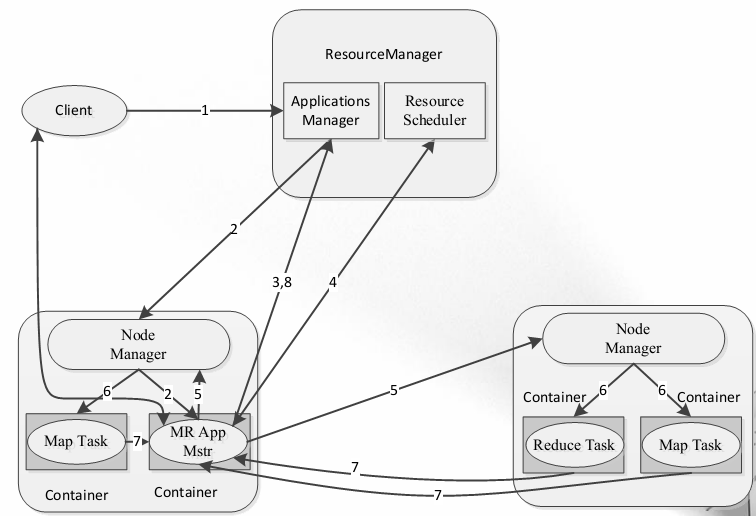
# 三、算法模型：MR并行算法

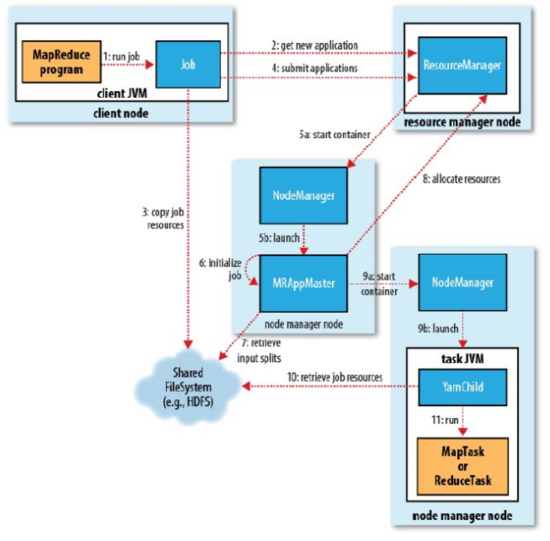
## 3.1算法框架图



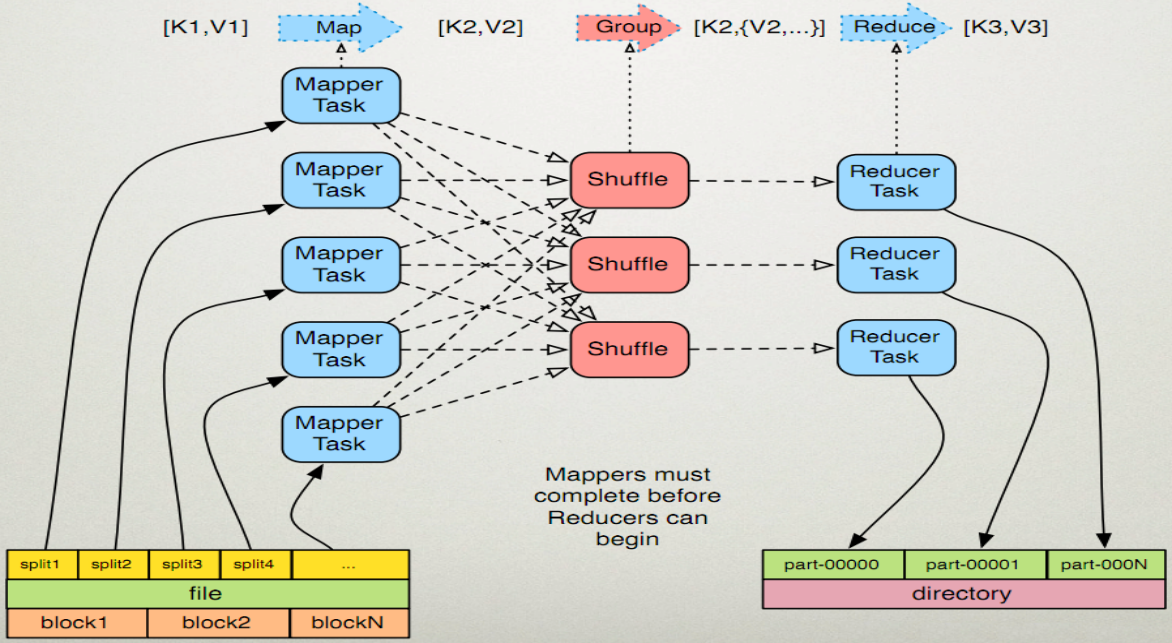


## **3.2算法工作流程图**

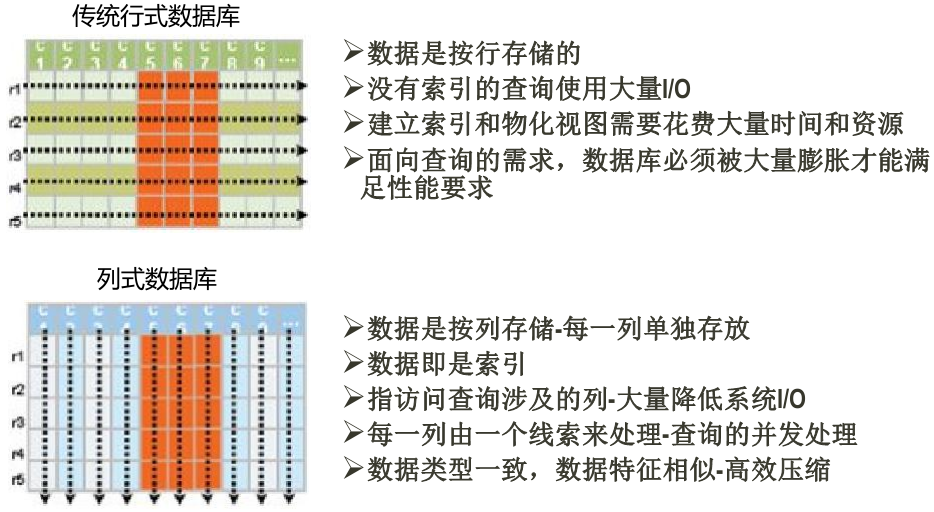




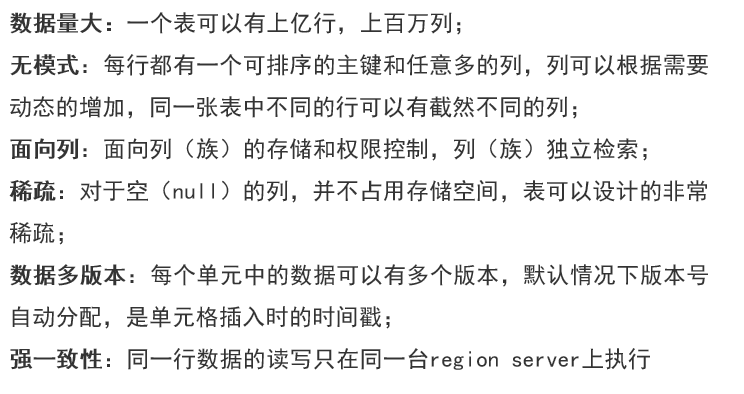
## 3.3 洗牌过程



# 四、Hbase概述



## 4.1 hbase表特点



## 4.2 hbase应用场景

对象存储🡪新闻、图片、网页、足迹

时序数据🡪以前使用，目前已经有时序数据库产品，时序数据库产品是未来主流

推荐画像🡪用户画像

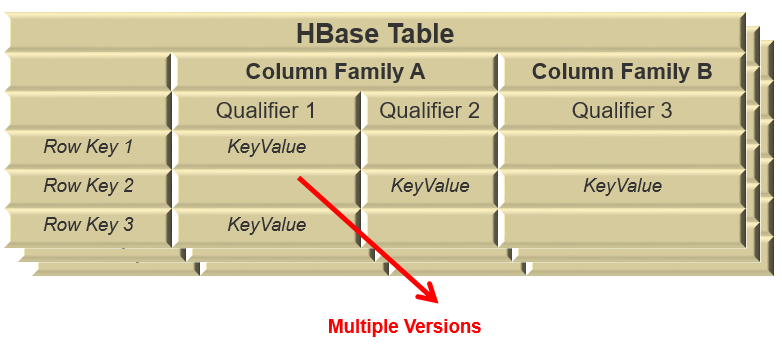
时空数据🡪交通大时空：轨迹

消息/订单🡪底层存储，涉及kafka

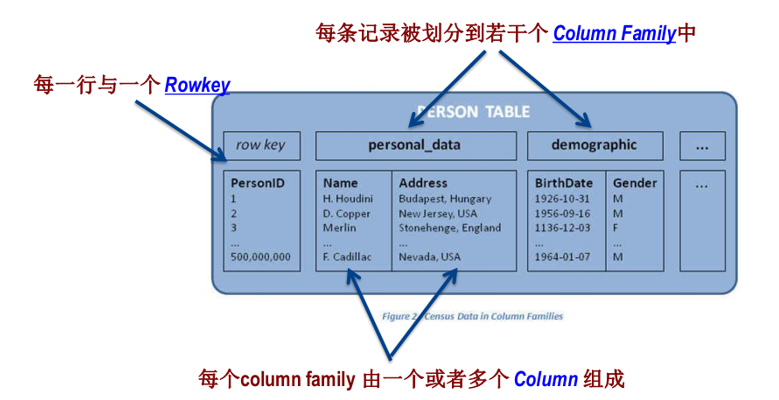
高并发操作🡪PB

商品推荐🡪hadoop并行计算（协调过滤算法:UserCF、ItemCF）结果存储至hbase

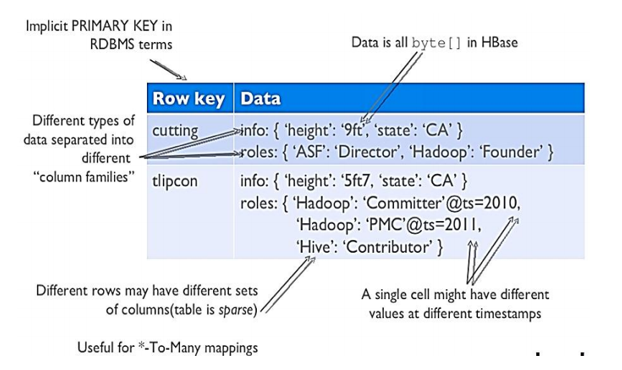
## 4.3 hbase数据模型



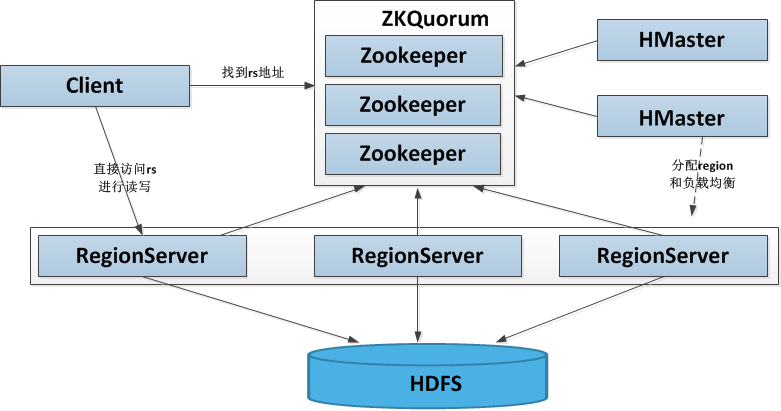
转换为逻辑视图，仅仅是方便理解，不能当作是物理存储



Key/Value系统



## 4.3 hbase体系结构



# 五、程序开发

## 5.1 TempMaxOfYear.java-数据加载及清洗

package com.neuedu.nuist18\_weathersystem;

import java.io.IOException;

import java.net.URI;

import org.apache.commons.lang3.StringUtils;

import org.apache.hadoop.fs.FileSystem;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.hbase.TableName;

import org.apache.hadoop.hbase.client.Admin;

import org.apache.hadoop.hbase.client.Connection;

import org.apache.hadoop.hbase.client.Put;

import org.apache.hadoop.hbase.client.RegionLocator;

import org.apache.hadoop.hbase.client.Table;

import org.apache.hadoop.hbase.io.ImmutableBytesWritable;

import org.apache.hadoop.hbase.mapreduce.HFileOutputFormat2;

import org.apache.hadoop.hbase.tool.LoadIncrementalHFiles;

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

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.Mapper;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

/\*\*

\* 数据加载与清洗

\*

\*/

public class BulkLoad {

static class DataBulkMapper extends Mapper<LongWritable, Text, ImmutableBytesWritable, Put> {

@Override

protected void map(LongWritable key, Text value,

Mapper<LongWritable, Text, ImmutableBytesWritable, Put>.Context context)

throws IOException, InterruptedException {

String line = value.toString();

// 跳过空内容、第一行标题栏

if (StringUtils.isBlank(line) || line.startsWith("Estacao")) {

return;

}

// 只导入83377巴西利亚的数据

if (line.indexOf("83377") < 0) {

return;

}

// Estacao;Data;Hora;Precipitacao;TempBulboSeco;TempBulboUmido;TempMaxima;TempMinima;UmidadeRelativa;PressaoAtmEstacao;PressaoAtmMar;DirecaoVento;VelocidadeVento;Insolacao;Nebulosidade;Evaporacao

// Piche;Temp Comp Media;Umidade Relativa Media;Velocidade do Vento Media;

// 82024;01/01/1961;0000;;;;32.3;;;;;;;4.4;;;26.56;82.5;3;

// 82024;01/01/1961;1200;;26;23.9;;22.9;83;994.2;;5;5;;8;;;;;

// 82024;01/01/1961;1800;;32.3;27;;;65;991.6;;5;3;;9;;;;;

// 固定强制拆分成19列

String[] items = line.split(";", 19);

String code = items[0];

String date = items[1];

String hora = items[2];

float precipitacao = Float.valueOf(StringUtils.isBlank(items[3])? "0" : items[3]);

float tempMax = Float.valueOf(StringUtils.isBlank(items[6])? "0" : items[6]);

float tempMin = Float.valueOf(StringUtils.isBlank(items[7])? "0" : items[7]);

float tempAvg = Float.valueOf(StringUtils.isBlank(items[16])? "0" : items[16]);

;

// 构建Put

byte[] row = Bytes.toBytes(code + "\_" + date + "\_" + hora);

byte[] family = Bytes.toBytes("info");

byte[] c1 = Bytes.toBytes("precipitacao");

byte[] c2 = Bytes.toBytes("tempMax");

byte[] c3 = Bytes.toBytes("tempMin");

byte[] c4 = Bytes.toBytes("tempAvg");

Put put = new Put(row);

put.addColumn(family, c1, Bytes.toBytes(precipitacao));

put.addColumn(family, c2, Bytes.toBytes(tempMax));

put.addColumn(family, c3, Bytes.toBytes(tempMin));

put.addColumn(family, c4, Bytes.toBytes(tempAvg));

// 构建ImmutableBytesWritable

ImmutableBytesWritable rw = new ImmutableBytesWritable(row);

// 写入数据ImmutableBytesWritable

context.write(rw, put);

}

}

public static void main(String[] args) {

// 创建表

HbaseUtil hbaseUtil = new HbaseUtil();

Connection conn = hbaseUtil.getConnection();

Admin admin = hbaseUtil.getAdmin(conn);

String tableName = "weathers";

hbaseUtil.createTable(admin, tableName, true, "info");

try {

String input = "hdfs://master.hadoop.nuist:9000/brazil/\*.csv";

String output = "hdfs://master.hadoop.nuist:9000/brazil\_bulk";

FileSystem hdfs = FileSystem.get(URI.create(output), HbaseUtil.conf);

hdfs.delete(new Path(output), true);

hdfs.close();

// 1:生成hfile

Job job = Job.getInstance(hbaseUtil.conf);

job.setInputFormatClass(TextInputFormat.class);

FileInputFormat.setInputPaths(job, input);

job.setMapperClass(DataBulkMapper.class);//注意修改

job.setMapOutputKeyClass(ImmutableBytesWritable.class);

job.setMapOutputValueClass(Put.class);

// job.setReducerClass(PutSortReducer.class);//默认

job.setOutputFormatClass(HFileOutputFormat2.class);

FileOutputFormat.setOutputPath(job, new Path(output));

Table table = hbaseUtil.geTable(conn, tableName);

RegionLocator regionLocator = conn.getRegionLocator(TableName.valueOf(tableName));

HFileOutputFormat2.configureIncrementalLoad(job, table, regionLocator);

boolean flag = job.waitForCompletion(true);

// 2：移动hfile文件至合适的hregionserver

LoadIncrementalHFiles loader = new LoadIncrementalHFiles(HbaseUtil.conf);

loader.doBulkLoad(new Path(output), admin, table, regionLocator);

System.out.println("数据导入成功~~~");

// 读取数据

} catch (Exception e) {

e.printStackTrace();

}

}

}

## 5.2 TempMaxOfYear.java-统计每年的最高气温

package com.neuedu.nuist18\_weathersystem;

import java.io.IOException;

import javax.validation.constraints.Max;

import org.apache.hadoop.hbase.Cell;

import org.apache.hadoop.hbase.CellUtil;

import org.apache.hadoop.hbase.client.Admin;

import org.apache.hadoop.hbase.client.Connection;

import org.apache.hadoop.hbase.client.Mutation;

import org.apache.hadoop.hbase.client.Put;

import org.apache.hadoop.hbase.client.Result;

import org.apache.hadoop.hbase.client.Scan;

import org.apache.hadoop.hbase.io.ImmutableBytesWritable;

import org.apache.hadoop.hbase.mapreduce.TableMapReduceUtil;

import org.apache.hadoop.hbase.mapreduce.TableMapper;

import org.apache.hadoop.hbase.mapreduce.TableOutputFormat;

import org.apache.hadoop.hbase.mapreduce.TableReducer;

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

import org.apache.hadoop.io.FloatWritable;

import org.apache.hadoop.io.NullWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.Mapper;

import org.apache.hadoop.mapreduce.Reducer;

/\*\*

\* 统计每年的最高气温：从weathers表中读取数据 ，统计出每年最高温度，并写入表result中

\*

\*/

public class TempMaxOfYear {

static class TempMaxMapper extends TableMapper<Text, FloatWritable> {

@Override

protected void map(ImmutableBytesWritable key, Result value,

Mapper<ImmutableBytesWritable, Result, Text, FloatWritable>.Context context)

throws IOException, InterruptedException {

String row = Bytes.toString(value.getRow());

System.out.println(row);

String[] items = row.split("\_");

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

String cname = Bytes.toString(CellUtil.cloneQualifier(cell));

if ("tempMax".equals(cname)) {

float max = Bytes.toFloat(CellUtil.cloneValue(cell));

context.write(new Text(items[1].substring(6)), new FloatWritable(max));

}

}

}

}

static class TempMaxReducer extends TableReducer<Text, FloatWritable, NullWritable> {

@Override

protected void reduce(Text key, Iterable<FloatWritable> values,

Reducer<Text, FloatWritable, NullWritable, Mutation>.Context context)

throws IOException, InterruptedException {

// <1961,list<1,2,3,4,5,6,7,8,9>>

float max = 0;

for (FloatWritable v : values) {

max = Math.max(max, v.get());

}

// 构建Put构建

byte[] row = Bytes.toBytes(key.toString());

byte[] family = Bytes.toBytes("info");

byte[] c1 = Bytes.toBytes("max");

Put put = new Put(row);

put.addColumn(family, c1, Bytes.toBytes(max));

// 输出

context.write(NullWritable.get(), put);

}

}

public static void main(String[] args) {

// 创建表

HbaseUtil hbaseUtil = new HbaseUtil();

Connection conn = hbaseUtil.getConnection();

Admin admin = hbaseUtil.getAdmin(conn);

String tableName = "results";

hbaseUtil.createTable(admin, tableName, false, "info");

try {

HbaseUtil.conf.set(TableOutputFormat.OUTPUT\_TABLE, tableName);

Job job = Job.getInstance(HbaseUtil.conf);

//设置读取源表weathers中的列

Scan scan = new Scan();

byte[] family = Bytes.toBytes("info");

byte[] c1 = Bytes.toBytes("tempMax");

scan.addColumn(family, c1);

//设置mapper和redcuer

TableMapReduceUtil.initTableMapperJob("weathers", scan, TempMaxMapper.class, Text.class, FloatWritable.class, job);

TableMapReduceUtil.initTableReducerJob(tableName, TempMaxReducer.class, job);

//执行

boolean flag = job.waitForCompletion(true);

if (flag) {

System.out.println("每年最高温度统计完成~~~");

}

} catch (Exception e) {

e.printStackTrace();

}

}

}

## 5.3 TempMinOfYear.java-统计每年的最低气温

package com.neuedu.nuist18\_weathersystem;

import java.io.IOException;

import org.apache.hadoop.hbase.Cell;

import org.apache.hadoop.hbase.CellUtil;

import org.apache.hadoop.hbase.client.Admin;

import org.apache.hadoop.hbase.client.Connection;

import org.apache.hadoop.hbase.client.Mutation;

import org.apache.hadoop.hbase.client.Put;

import org.apache.hadoop.hbase.client.Result;

import org.apache.hadoop.hbase.client.Scan;

import org.apache.hadoop.hbase.io.ImmutableBytesWritable;

import org.apache.hadoop.hbase.mapreduce.TableMapReduceUtil;

import org.apache.hadoop.hbase.mapreduce.TableMapper;

import org.apache.hadoop.hbase.mapreduce.TableOutputFormat;

import org.apache.hadoop.hbase.mapreduce.TableReducer;

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

import org.apache.hadoop.io.FloatWritable;

import org.apache.hadoop.io.NullWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.Mapper;

import org.apache.hadoop.mapreduce.Reducer;

/\*\*

\* 统计每年的最低气温：从weathers表中读取数据 ，统计出每年最低温度，并写入表result中

\*

\*/

public class TempMinOfYear {

static class TempMinMapper extends TableMapper<Text, FloatWritable> {

@Override

protected void map(ImmutableBytesWritable key, Result value,

Mapper<ImmutableBytesWritable, Result, Text, FloatWritable>.Context context)

throws IOException, InterruptedException {

String row = Bytes.toString(value.getRow());

System.out.println(row);

String[] items = row.split("\_");

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

String cname = Bytes.toString(CellUtil.cloneQualifier(cell));

if ("tempMin".equals(cname)) {

float max = Bytes.toFloat(CellUtil.cloneValue(cell));

context.write(new Text(items[1].substring(6)), new FloatWritable(max));

}

}

}

}

static class TempMinReducer extends TableReducer<Text, FloatWritable, NullWritable> {

@Override

protected void reduce(Text key, Iterable<FloatWritable> values,

Reducer<Text, FloatWritable, NullWritable, Mutation>.Context context)

throws IOException, InterruptedException {

// <1961,list<1,2,3,4,5,6,7,8,9>>

float max = 1;

for (FloatWritable v : values) {

if (v.get()<=0) {

continue;

}

max = Math.min(max, v.get());

}

// 构建Put构建

byte[] row = Bytes.toBytes(key.toString());

byte[] family = Bytes.toBytes("info");

byte[] c1 = Bytes.toBytes("min");

Put put = new Put(row);

put.addColumn(family, c1, Bytes.toBytes(max));

// 输出

context.write(NullWritable.get(), put);

}

}

public static void main(String[] args) {

// 创建表

HbaseUtil hbaseUtil = new HbaseUtil();

Connection conn = hbaseUtil.getConnection();

Admin admin = hbaseUtil.getAdmin(conn);

String tableName = "results";

hbaseUtil.createTable(admin, tableName, false, "info");

try {

HbaseUtil.conf.set(TableOutputFormat.OUTPUT\_TABLE, tableName);

Job job = Job.getInstance(HbaseUtil.conf);

//设置读取源表weathers中的列

Scan scan = new Scan();

byte[] family = Bytes.toBytes("info");

byte[] c1 = Bytes.toBytes("tempMin");

scan.addColumn(family, c1);

//设置mapper和redcuer

TableMapReduceUtil.initTableMapperJob("weathers", scan, TempMinMapper.class, Text.class, FloatWritable.class, job);

TableMapReduceUtil.initTableReducerJob(tableName, TempMinReducer.class, job);

//执行

boolean flag = job.waitForCompletion(true);

if (flag) {

System.out.println("每年最低温度统计完成~~~");

}

} catch (Exception e) {

e.printStackTrace();

}

}

}

## 5.4 TempAvgOfYear.java-统计每年的平均气温

**package** com.neuedu.nuist18\_weathersystem;

**import** java.io.IOException;

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

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

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

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

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

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

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

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

**import** org.apache.hadoop.hbase.io.ImmutableBytesWritable;

**import** org.apache.hadoop.hbase.mapreduce.TableMapReduceUtil;

**import** org.apache.hadoop.hbase.mapreduce.TableMapper;

**import** org.apache.hadoop.hbase.mapreduce.TableOutputFormat;

**import** org.apache.hadoop.hbase.mapreduce.TableReducer;

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

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

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

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

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

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

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

/\*\*

\* 统计每年的平均气温：从weathers表中读取数据 ，统计出每年平均温度，并写入表result中

\*

\*/

**public** **class** TempAvgOfYear {

**static** **class** TempAvgMapper **extends** TableMapper<Text, FloatWritable> {

@Override

**protected** **void** map(ImmutableBytesWritable key, Result value,

Mapper<ImmutableBytesWritable, Result, Text, FloatWritable>.Context context)

**throws** IOException, InterruptedException {

String row = Bytes.*toString*(value.getRow());

System.***out***.println(row);

String[] items = row.split("\_");

**for** (Cell cell : value.rawCells()) {

String cname = Bytes.*toString*(CellUtil.*cloneQualifier*(cell));

**if** ("tempAvg".equals(cname)) {

**float** max = Bytes.*toFloat*(CellUtil.*cloneValue*(cell));

context.write(**new** Text(items[1].substring(6)), **new** FloatWritable(max));

}

}

}

}

**static** **class** TempAvgReducer **extends** TableReducer<Text, FloatWritable, NullWritable> {

@Override

**protected** **void** reduce(Text key, Iterable<FloatWritable> values,

Reducer<Text, FloatWritable, NullWritable, Mutation>.Context context)

**throws** IOException, InterruptedException {

// <1961,list<1,2,3,4,5,6,7,8,9>>

**float** sum = 0;

**int** day = 0;

**for** (FloatWritable v : values) {

**if** (v.get()>0) {

sum += v.get();

day++;

}

}

// 构建Put构建

**byte**[] row = Bytes.*toBytes*(key.toString());

**byte**[] family = Bytes.*toBytes*("info");

**byte**[] c1 = Bytes.*toBytes*("avg");

Put put = **new** Put(row);

put.addColumn(family, c1, Bytes.*toBytes*(sum/day));

// 输出

context.write(NullWritable.*get*(), put);

}

}

**public** **static** **void** main(String[] args) {

// 创建表

HbaseUtil hbaseUtil = **new** HbaseUtil();

Connection conn = hbaseUtil.getConnection();

Admin admin = hbaseUtil.getAdmin(conn);

String tableName = "results";

hbaseUtil.createTable(admin, tableName, **false**, "info");

**try** {

HbaseUtil.*conf*.set(TableOutputFormat.***OUTPUT\_TABLE***, tableName);

Job job = Job.*getInstance*(HbaseUtil.*conf*);

//设置读取源表weathers中的列

Scan scan = **new** Scan();

**byte**[] family = Bytes.*toBytes*("info");

**byte**[] c1 = Bytes.*toBytes*("tempAvg");

scan.addColumn(family, c1);

//设置mapper和redcuer

TableMapReduceUtil.*initTableMapperJob*("weathers", scan, TempAvgMapper.**class**, Text.**class**, FloatWritable.**class**, job);

TableMapReduceUtil.*initTableReducerJob*(tableName, TempAvgReducer.**class**, job);

//执行

**boolean** flag = job.waitForCompletion(**true**);

**if** (flag) {

System.***out***.println("每年平均温度统计完成~~~");

}

} **catch** (Exception e) {

e.printStackTrace();

}

}

}

## 5.5 RainDayOfYear.java-统计每年的下雨天数

**package** com.neuedu.nuist18\_weathersystem;

**import** java.io.IOException;

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

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

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

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

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

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

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

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

**import** org.apache.hadoop.hbase.io.ImmutableBytesWritable;

**import** org.apache.hadoop.hbase.mapreduce.TableMapReduceUtil;

**import** org.apache.hadoop.hbase.mapreduce.TableMapper;

**import** org.apache.hadoop.hbase.mapreduce.TableOutputFormat;

**import** org.apache.hadoop.hbase.mapreduce.TableReducer;

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

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

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

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

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

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

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

/\*\*

\* 统计每年的下雨天数：从weathers表中读取数据 ，统计出每年下雨天数，并写入表result中

\*

\*/

**public** **class** RainDayOfYear {

**static** **class** TempMinMapper **extends** TableMapper<Text, FloatWritable> {

@Override

**protected** **void** map(ImmutableBytesWritable key, Result value,

Mapper<ImmutableBytesWritable, Result, Text, FloatWritable>.Context context)

**throws** IOException, InterruptedException {

String row = Bytes.*toString*(value.getRow());

System.***out***.println(row);

String[] items = row.split("\_");

**for** (Cell cell : value.rawCells()) {

String cname = Bytes.*toString*(CellUtil.*cloneQualifier*(cell));

**if** ("precipitacao".equals(cname)) {

**float** max = Bytes.*toFloat*(CellUtil.*cloneValue*(cell));

context.write(**new** Text(items[1].substring(6)), **new** FloatWritable(max));

}

}

}

}

**static** **class** TempMinReducer **extends** TableReducer<Text, FloatWritable, NullWritable> {

@Override

**protected** **void** reduce(Text key, Iterable<FloatWritable> values,

Reducer<Text, FloatWritable, NullWritable, Mutation>.Context context)

**throws** IOException, InterruptedException {

// <1961,list<0,0,1,2,2,2,0,3,3>>

**float** day = 0;

**int** count =1;

**for** (FloatWritable v : values) {

**if** (v.get()>0) {

day++;

}

//？如何判断？

count++;

//每2天降雨量是1天，需要重置

**if** (count>3) {

count = 1;

}

}

// 构建Put构建

**byte**[] row = Bytes.*toBytes*(key.toString());

**byte**[] family = Bytes.*toBytes*("info");

**byte**[] c1 = Bytes.*toBytes*("rainday");

Put put = **new** Put(row);

put.addColumn(family, c1, Bytes.*toBytes*(day));

// 输出

context.write(NullWritable.*get*(), put);

}

}

**public** **static** **void** main(String[] args) {

// 创建表

HbaseUtil hbaseUtil = **new** HbaseUtil();

Connection conn = hbaseUtil.getConnection();

Admin admin = hbaseUtil.getAdmin(conn);

String tableName = "results";

hbaseUtil.createTable(admin, tableName, **false**, "info");

**try** {

HbaseUtil.*conf*.set(TableOutputFormat.***OUTPUT\_TABLE***, tableName);

Job job = Job.*getInstance*(HbaseUtil.*conf*);

//设置读取源表weathers中的列

Scan scan = **new** Scan();

**byte**[] family = Bytes.*toBytes*("info");

**byte**[] c1 = Bytes.*toBytes*("precipitacao");

scan.addColumn(family, c1);

//设置mapper和redcuer

TableMapReduceUtil.*initTableMapperJob*("weathers", scan, TempMinMapper.**class**, Text.**class**, FloatWritable.**class**, job);

TableMapReduceUtil.*initTableReducerJob*(tableName, TempMinReducer.**class**, job);

//执行

**boolean** flag = job.waitForCompletion(**true**);

**if** (flag) {

System.***out***.println("每年下雨天数统计完成~~~");

}

} **catch** (Exception e) {

e.printStackTrace();

}

}

}

## 5.6 ForecastAfter.java-预测明天气温

package com.neuedu.nuist18\_weathersystem;

import java.io.DataInput;

import java.io.DataOutput;

import java.io.IOException;

import org.apache.hadoop.hbase.Cell;

import org.apache.hadoop.hbase.CellUtil;

import org.apache.hadoop.hbase.client.Admin;

import org.apache.hadoop.hbase.client.Connection;

import org.apache.hadoop.hbase.client.Mutation;

import org.apache.hadoop.hbase.client.Put;

import org.apache.hadoop.hbase.client.Result;

import org.apache.hadoop.hbase.client.Scan;

import org.apache.hadoop.hbase.io.ImmutableBytesWritable;

import org.apache.hadoop.hbase.mapreduce.TableMapReduceUtil;

import org.apache.hadoop.hbase.mapreduce.TableMapper;

import org.apache.hadoop.hbase.mapreduce.TableOutputFormat;

import org.apache.hadoop.hbase.mapreduce.TableReducer;

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

import org.apache.hadoop.io.FloatWritable;

import org.apache.hadoop.io.NullWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.io.WritableComparable;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.Mapper;

import org.apache.hadoop.mapreduce.Reducer;

/\*\*

\* 预测

\*

\*/

public class ForecastAfter {

static class Weather implements WritableComparable<Weather> {

private String date;

private float temp;

private float max;

private float min;

public Weather() {

}

public Weather(String date, float temp, float max, float min) {

super();

this.date = date;

this.temp = temp;

this.max = max;

this.min = min;

}

@Override

public int compareTo(Weather other) {

if (null == other) {

return 1;

}

return this.date.compareTo(other.date);

}

@Override

public void write(DataOutput out) throws IOException {

out.writeUTF(this.date);

out.writeFloat(this.temp);

out.writeFloat(this.max);

out.writeFloat(this.min);

}

@Override

public void readFields(DataInput in) throws IOException {

this.date = in.readUTF();

this.temp = in.readFloat();

this.max = in.readFloat();

this.min = in.readFloat();

}

public String getDate() {

return date;

}

public void setDate(String date) {

this.date = date;

}

public float getTemp() {

return temp;

}

public void setTemp(float temp) {

this.temp = temp;

}

public float getMax() {

return max;

}

public void setMax(float max) {

this.max = max;

}

public float getMin() {

return min;

}

public void setMin(float min) {

this.min = min;

}

}

static class TempForecastMapper extends TableMapper<Text, Weather> {

@Override

protected void map(ImmutableBytesWritable key, Result value,

Mapper<ImmutableBytesWritable, Result, Text, Weather>.Context context)

throws IOException, InterruptedException {

String row = Bytes.toString(value.getRow());

String[] items = row.split("\_");

String after\_date = context.getConfiguration().get("date");

//31/12/2019

//31/12

if (!after\_date.equals(items[1].substring(0,5))) {

return;

}

Weather w = new Weather();

w.setDate(after\_date);

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

String cname = Bytes.toString(CellUtil.cloneQualifier(cell));

if ("tempMax".equals(cname)) {

float max = Bytes.toFloat(CellUtil.cloneValue(cell));

System.out.println("max" + max);

w.setMax(max);

}else if ("tempMin".equals(cname)) {

float min = Bytes.toFloat(CellUtil.cloneValue(cell));

System.out.println("min" + min);

w.setMin(min);

}else if ("tempAvg".equals(cname)) {

float avg = Bytes.toFloat(CellUtil.cloneValue(cell));

System.out.println("avg" + avg);

w.setTemp(avg);

}

}

System.out.println("line end...");

context.write(new Text(after\_date), w);

}

}

static class TempForecastReducer extends TableReducer<Text, Weather, NullWritable> {

@Override

protected void reduce(Text key, Iterable<Weather> values,

Reducer<Text, Weather, NullWritable, Mutation>.Context context)

throws IOException, InterruptedException {

// <31/12,list<w,w,w>>

float min = 0;

float max = 0;

float avg = 0;

for (Weather w : values) {

if (w.getMax()>0) {

max = w.getMax();

}

if (w.getMin()>0) {

min = w.getMin();

}

if (w.getTemp()>0) {

avg = w.getTemp();

}

}

// 构建Put构建

byte[] row = Bytes.toBytes(key.toString());

byte[] family = Bytes.toBytes("info");

byte[] c1 = Bytes.toBytes("min");

byte[] c2 = Bytes.toBytes("max");

byte[] c3 = Bytes.toBytes("avg");

Put put = new Put(row);

put.addColumn(family, c1, Bytes.toBytes(min));

put.addColumn(family, c2, Bytes.toBytes(max));

put.addColumn(family, c3, Bytes.toBytes(avg));

// 输出

context.write(NullWritable.get(), put);

}

}

public static void forecast(String date) {

// 创建表

HbaseUtil hbaseUtil = new HbaseUtil();

Connection conn = hbaseUtil.getConnection();

Admin admin = hbaseUtil.getAdmin(conn);

String tableName = "forecast";

hbaseUtil.createTable(admin, tableName, false, "info");

//预测日期:31/12

HbaseUtil.conf.set("date", date);

try {

HbaseUtil.conf.set(TableOutputFormat.OUTPUT\_TABLE, tableName);

Job job = Job.getInstance(HbaseUtil.conf);

//设置读取源表weathers中的列

Scan scan = new Scan();

byte[] family = Bytes.toBytes("info");

byte[] c1 = Bytes.toBytes("tempMin");

byte[] c2 = Bytes.toBytes("tempMax");

byte[] c3 = Bytes.toBytes("tempAvg");

scan.addColumn(family, c1);

scan.addColumn(family, c2);

scan.addColumn(family, c3);

//设置mapper和redcuer

TableMapReduceUtil.initTableMapperJob("weathers", scan, TempForecastMapper.class, Text.class, Weather.class, job);

TableMapReduceUtil.initTableReducerJob(tableName, TempForecastReducer.class, job);

//执行

boolean flag = job.waitForCompletion(true);

if (flag) {

System.out.println("预测统计完成~~~");

}

} catch (Exception e) {

e.printStackTrace();

}

}

}

## 5.7 HbaseUtil.java- hbase工具类

package com.neuedu.nuist18\_weathersystem;

import java.io.IOException;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.hbase.HBaseConfiguration;

import org.apache.hadoop.hbase.TableName;

import org.apache.hadoop.hbase.client.Admin;

import org.apache.hadoop.hbase.client.ColumnFamilyDescriptorBuilder;

import org.apache.hadoop.hbase.client.Connection;

import org.apache.hadoop.hbase.client.ConnectionFactory;

import org.apache.hadoop.hbase.client.Table;

import org.apache.hadoop.hbase.client.TableDescriptorBuilder;

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

/\*\*

\* hbase工具类：创建表，删除表…… 项目配置文件：hbase-site.xml、core-site.xml、hdfs-site.xml

\*/

public class HbaseUtil {

public static Configuration conf;// 单例模式

protected Connection conn;

protected Admin admin;

protected Table table;

/\*\* 创建Connection对象 \*/

public Connection getConnection() {

try {

if (null == conf) {

// 使用提供的配置文件自动配置

conf = HBaseConfiguration.create();

}

this.conn = ConnectionFactory.createConnection(conf);

} catch (IOException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

return this.conn;

}

/\*\* 创建Admin对象 \*/

public Admin getAdmin(Connection conn) {

try {

this.admin = conn.getAdmin();

} catch (IOException e) {

e.printStackTrace();

}

return this.admin;

}

/\*\* 创建Table对象 \*/

public Table geTable(Connection conn, String tableName) {

try {

TableName tn = TableName.valueOf(tableName);

this.table = conn.getTable(tn);

} catch (IOException e) {

e.printStackTrace();

}

return table;

}

/\*\* 创建表 \*/

public boolean createTable(Admin admin, String tableName, boolean isDroped, String familyName) {

try {

TableName tn = TableName.valueOf(tableName);

if (admin.tableExists(tn)) {

if (isDroped) {

admin.disableTable(tn);// 禁用表

admin.deleteTable(tn);// 删除表

} else {

// 表存在但不删除

return false;

}

}

TableDescriptorBuilder tdb = TableDescriptorBuilder.newBuilder(tn);

byte[] family = Bytes.toBytes(familyName);

ColumnFamilyDescriptorBuilder cdb = ColumnFamilyDescriptorBuilder.newBuilder(family);

tdb.setColumnFamily(cdb.build());

admin.createTable(tdb.build());

} catch (IOException e) {

e.printStackTrace();

return false;

}

return true;

}

/\*\* 资源释放 \*/

public void close() {

try {

if (null != this.table) {

this.table.close();

}

if (null != this.admin) {

this.admin.close();

}

if (null != this.conn) {

this.conn.close();

}

} catch (IOException e) {

e.printStackTrace();

}

}

}

## 5.8 Starter.java-整体功能：菜单与系统

**package** com.neuedu.nuist18\_weathersystem;

**import** java.util.Scanner;

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

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

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

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

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

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

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

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

**import** org.apache.hadoop.hbase.filter.BinaryComparator;

**import** org.apache.hadoop.hbase.filter.Filter;

**import** org.apache.hadoop.hbase.filter.RowFilter;

**import** org.apache.hadoop.hbase.filter.SubstringComparator;

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

**public** **class** Starter {

**private** **static** Scanner *input* = **new** Scanner(System.***in***);

**private** **static** HbaseUtil *hbaseUtil* = **new** HbaseUtil();

**public** **static** **void** showMenu() {

System.***out***.println("\*\*\*\*\*\*\*\*\*\*\*天气查询系统\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

System.***out***.println("1、查询某一天的天气数据");

System.***out***.println("2、查询每年的最高气温");

System.***out***.println("3、查询每年的最低气温");

System.***out***.println("4、查询每年的平均气温");

System.***out***.println("5、查询每年的下雨的天数");

System.***out***.println("6、预测明天的天气");

System.***out***.println("0、退出");

System.***out***.print("请选择【0-6】进行操作：");

}

**public** **static** **void** main(String[] args) {

System.***out***.println("系统初始中……");

// BulkLoad.main(args);// 导入数据

// TempMaxOfYear.main(args);// 统计每年的最高气温

// TempMinOfYear.main(args);// 统计每年的最低气温

// TempAvgOfYear.main(args);// 统计每年的平均气温

// RainDayOfYear.main(args);// 统计每年的下雨的天数

**boolean** flag = **false**;

**do** {

*showMenu*();

**int** choice = *input*.nextInt();

**switch** (choice) {

**case** 1:

*method1*();

**break**;

**case** 2:

*method2*();

**break**;

**case** 3:

*method3*();

**break**;

**case** 4:

*method4*();

**break**;

**case** 5:

*method5*();

**break**;

**case** 6:

*method6*();

**break**;

**case** 0:

flag = **true**;

**break**;

**default**:

System.***out***.println("输入错误~~");

**break**;

}

} **while** (!flag);

System.***out***.println("感谢您的使用，再见~~~");

}

**private** **static** **void** method6() {

**try** {

//预测

String date="16/01";

ForecastAfter.*forecast*(date);

//显示

Connection conn = *hbaseUtil*.getConnection();

Table table = *hbaseUtil*.geTable(conn, "forecast");

Scan scan = **new** Scan();

Filter filter = **new** RowFilter(CompareOperator.***EQUAL***, **new** BinaryComparator(Bytes.*toBytes*(date)));

scan.setFilter(filter);

ResultScanner rs = table.getScanner(scan);

System.***out***.println(date + "预测的天气信息如下 ：");

**for** (Result r : rs) {

**for** (Cell cell : r.rawCells()) {

String cname = Bytes.*toString*(CellUtil.*cloneQualifier*(cell));

Float value = Bytes.*toFloat*(CellUtil.*cloneValue*(cell));

System.***out***.println(cname+ ":"+value);

}

}

} **catch** (Exception e) {

e.printStackTrace();

}

}

**private** **static** **void** method5() {

**try** {

Connection conn = *hbaseUtil*.getConnection();

Table table = *hbaseUtil*.geTable(conn, "results");

Scan scan = **new** Scan();

ResultScanner rs = table.getScanner(scan);

System.***out***.println("每年的下雨的天数，信息如下 ：");

**for** (Result r : rs) {

**for** (Cell cell : r.rawCells()) {

String cname = Bytes.*toString*(CellUtil.*cloneQualifier*(cell));

**if** ("rainday".equals(cname)) {

System.***out***.println(Bytes.*toString*(r.getRow())+"年下雨的天数"+Bytes.*toFloat*(CellUtil.*cloneValue*(cell)));

}

}

}

} **catch** (Exception e) {

e.printStackTrace();

}

}

**private** **static** **void** method4() {

**try** {

Connection conn = *hbaseUtil*.getConnection();

Table table = *hbaseUtil*.geTable(conn, "results");

Scan scan = **new** Scan();

ResultScanner rs = table.getScanner(scan);

System.***out***.println("每年的平均气温，信息如下 ：");

**for** (Result r : rs) {

**for** (Cell cell : r.rawCells()) {

String cname = Bytes.*toString*(CellUtil.*cloneQualifier*(cell));

**if** ("avg".equals(cname)) {

System.***out***.println(Bytes.*toString*(r.getRow())+"年平均气温"+Bytes.*toFloat*(CellUtil.*cloneValue*(cell)));

}

}

}

} **catch** (Exception e) {

e.printStackTrace();

}

}

**private** **static** **void** method3() {

**try** {

Connection conn = *hbaseUtil*.getConnection();

Table table = *hbaseUtil*.geTable(conn, "results");

Scan scan = **new** Scan();

ResultScanner rs = table.getScanner(scan);

System.***out***.println("每年的最低气温，信息如下 ：");

**for** (Result r : rs) {

**for** (Cell cell : r.rawCells()) {

String cname = Bytes.*toString*(CellUtil.*cloneQualifier*(cell));

**if** ("min".equals(cname)) {

System.***out***.println(Bytes.*toString*(r.getRow())+"年最低气温"+Bytes.*toFloat*(CellUtil.*cloneValue*(cell)));

}

}

}

} **catch** (Exception e) {

e.printStackTrace();

}

}

**private** **static** **void** method2() {

**try** {

Connection conn = *hbaseUtil*.getConnection();

Table table = *hbaseUtil*.geTable(conn, "results");

Scan scan = **new** Scan();

ResultScanner rs = table.getScanner(scan);

System.***out***.println("每年的最高气温，信息如下 ：");

**for** (Result r : rs) {

**for** (Cell cell : r.rawCells()) {

String cname = Bytes.*toString*(CellUtil.*cloneQualifier*(cell));

**if** ("max".equals(cname)) {

System.***out***.println(Bytes.*toString*(r.getRow())+"年最高气温"+Bytes.*toFloat*(CellUtil.*cloneValue*(cell)));

}

}

}

} **catch** (Exception e) {

e.printStackTrace();

}

}

**private** **static** **void** method1() {

**try** {

System.***out***.println("请输入查询的日期(dd/mm/yyyy)：");

String date = *input*.next();

Connection conn = *hbaseUtil*.getConnection();

Table table = *hbaseUtil*.geTable(conn, "weathers");

Scan scan = **new** Scan();

Filter filter = **new** RowFilter(CompareOperator.***EQUAL***, **new** SubstringComparator(date));

scan.setFilter(filter);

ResultScanner rs = table.getScanner(scan);

System.***out***.println(date+",详细信息如下 ：");

**for** (Result r : rs) {

**for** (Cell cell : r.rawCells()) {

String cname = Bytes.*toString*(CellUtil.*cloneQualifier*(cell));

Float value = Bytes.*toFloat*(CellUtil.*cloneValue*(cell));

System.***out***.println(cname+ ":"+value);

}

}

} **catch** (Exception e) {

e.printStackTrace();

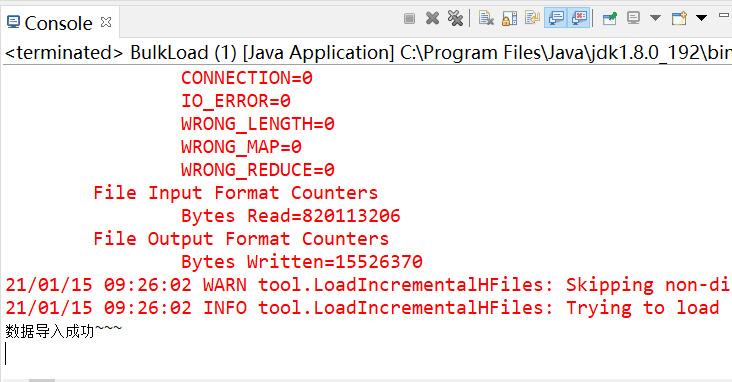
}

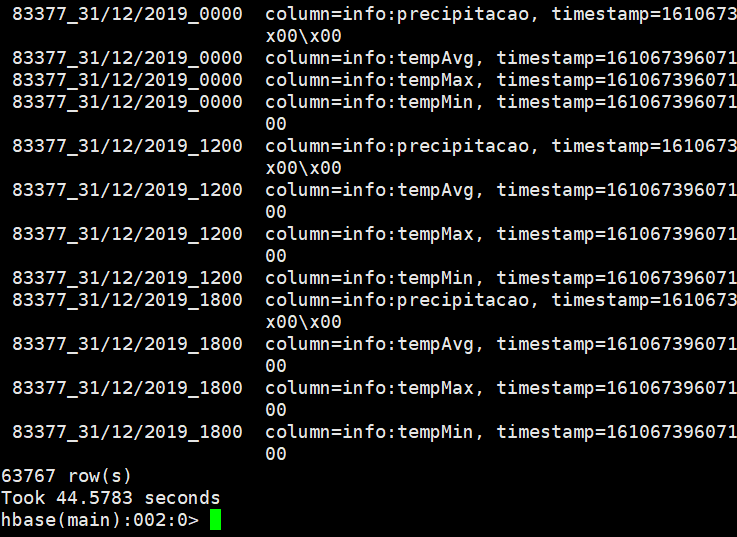
}

}

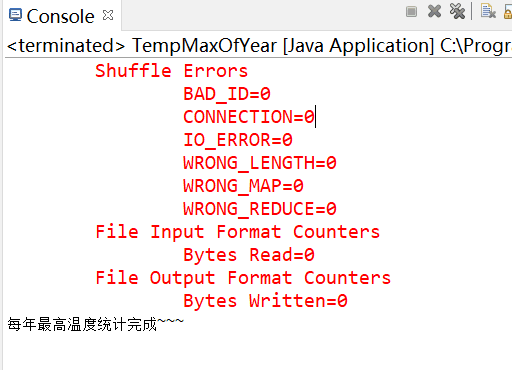
## 5.9 运行结果

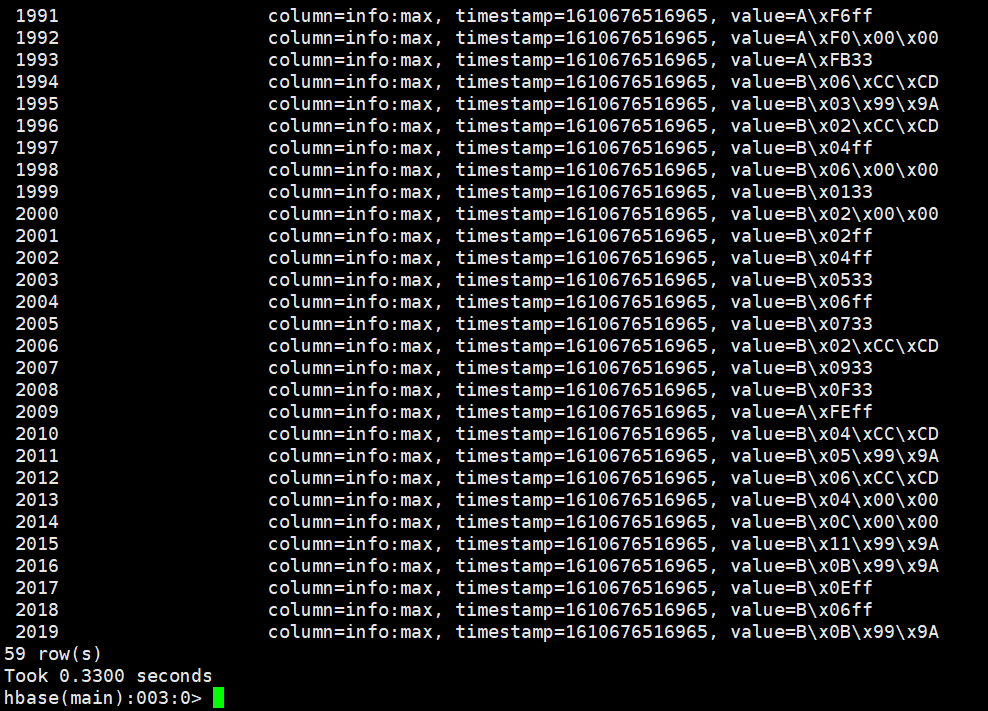
**1)数据导入**



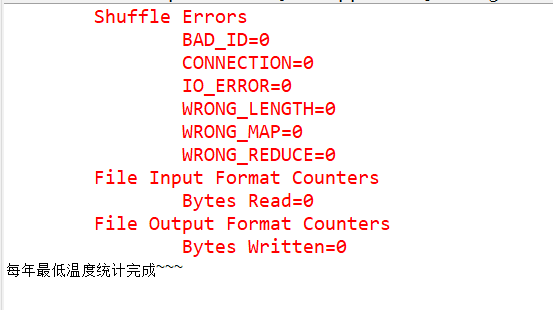


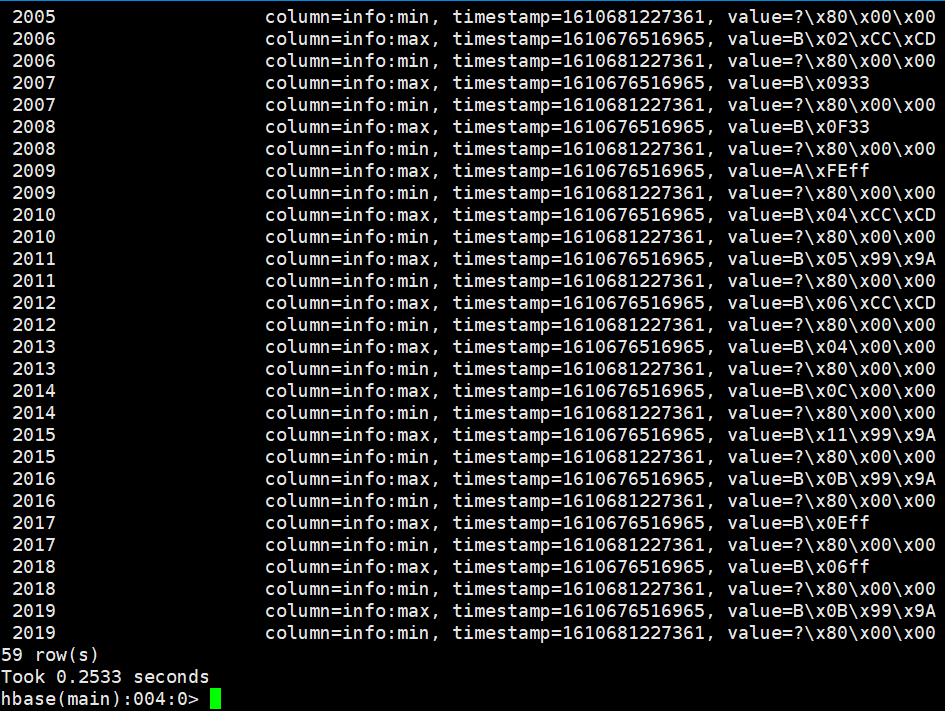
**2)年最高温**



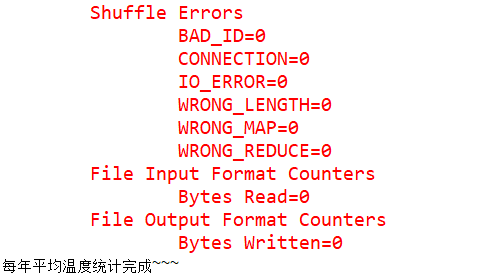


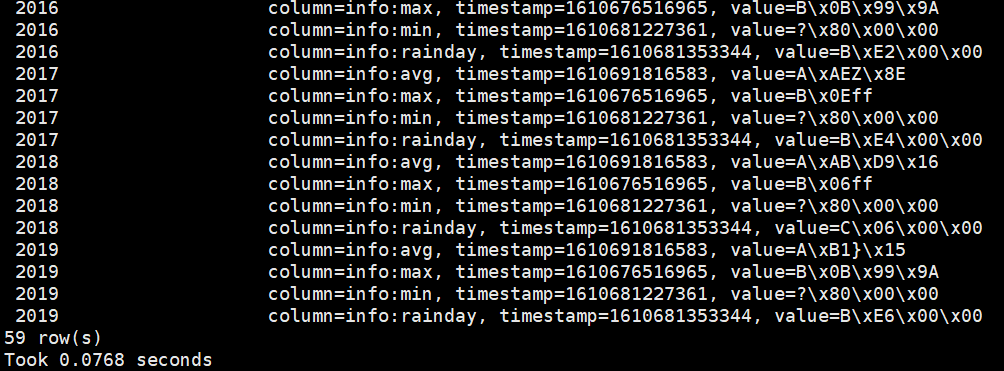
**3)年最低温**



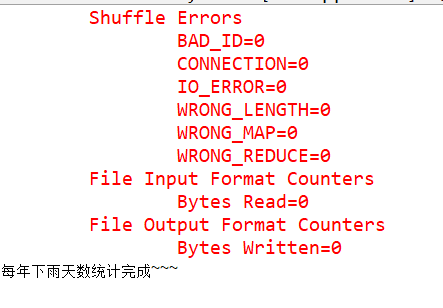


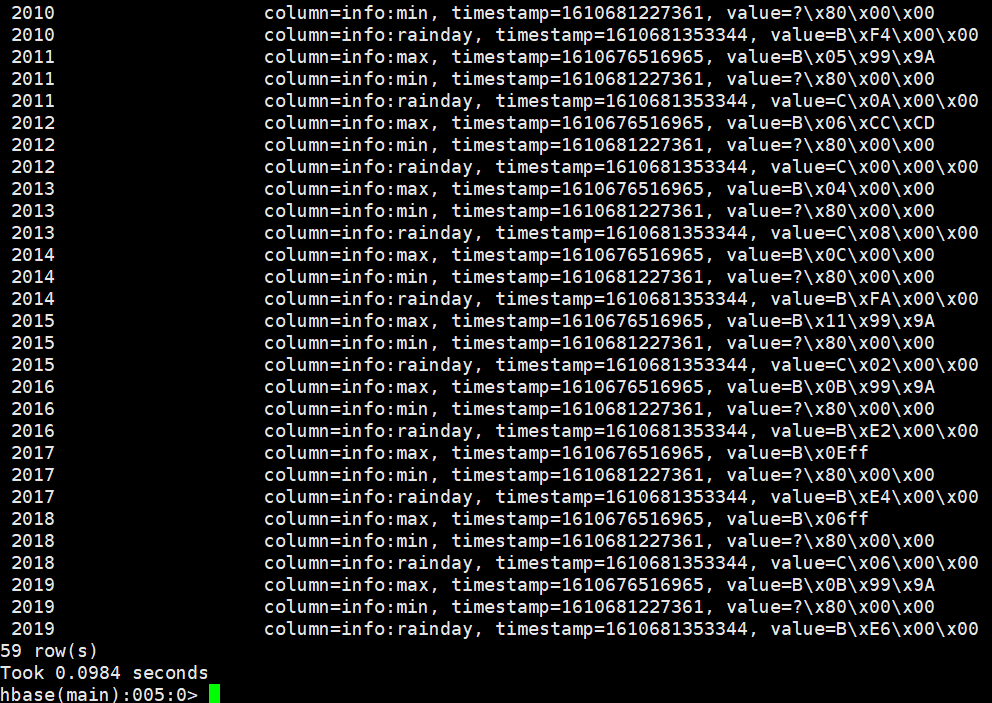
**4)年均温**

****

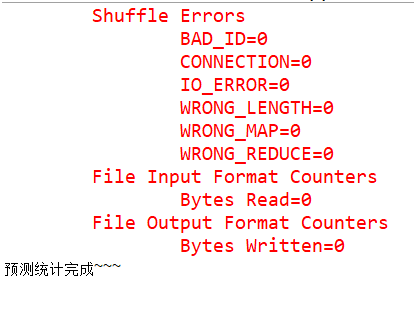
****

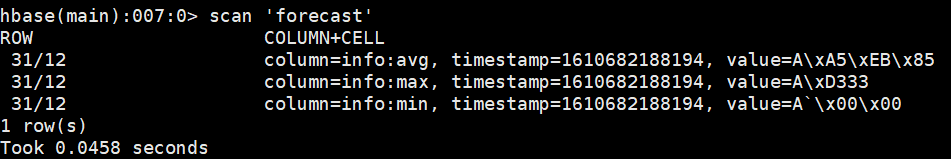
**5)年下雨天数**



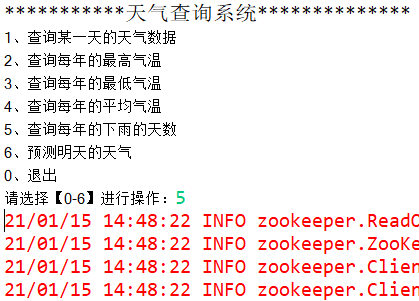
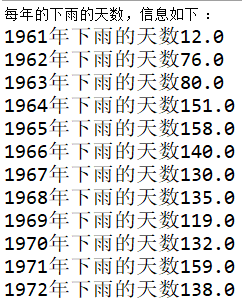
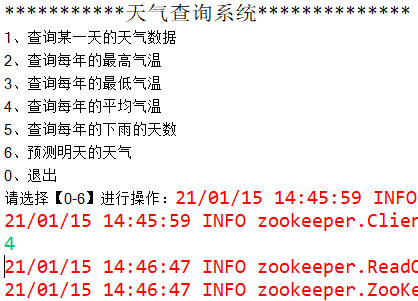
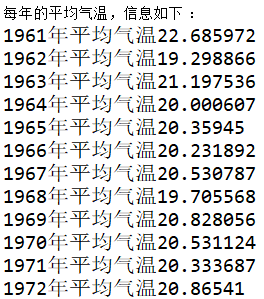
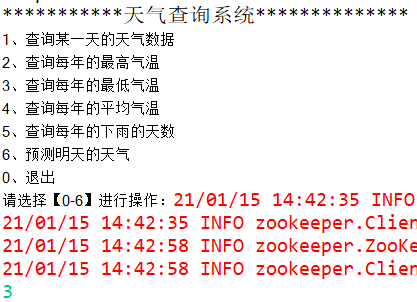
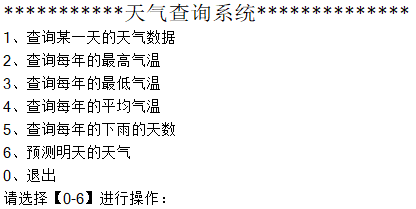
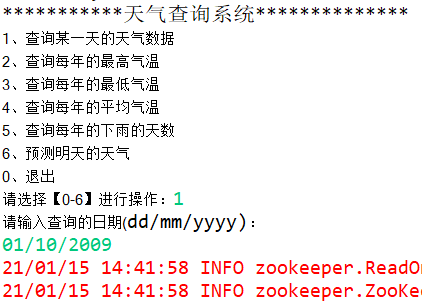


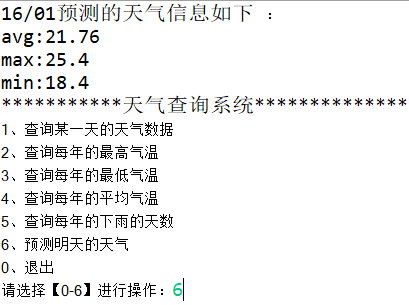
**6)预测次日气温**

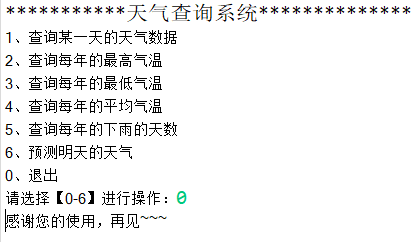




**7)** **整体功能：菜单与系统**



****

****

运行结果显示，我们可以对年气温数据进行简单筛选处理，从大量数据中快速便捷地对气温数据进行分析，能够有效的实现对大量数据进行清洗及分析处理。

利用该项目设计成果能够高效率的完成对大量数据的筛选、清洗、分析及预测，方便对于不同年份天气状况的分析，便于实际研究、数据处理中对大量气温数据进行处理与分析。

# 五、总结与展望

通过本次实践，我学会了如何运用java语言，利用Hadoop、Hbase以及VMware Workstation Pro在eclipse环境下进行简单数据处理：包括简单数据清洗及简单数据筛选。这次项目实践更让我的逻辑思维能力得到了锻炼，学会了调试程序的一些技巧。