**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 12月 1日

**目录**

[一、项目简介 3](#_Toc30149613)

[1.1项目介绍 3](#_Toc30149614)

[1.2开发工具 3](#_Toc30149615)

[1.3开发环境 3](#_Toc30149616)

[二、Web日志分析系统概述 3](#_Toc30149617)

[三、准备数据 4](#_Toc30149618)

[3.1 批量导入数据至HDFS 4](#_Toc30149619)

[四、日志文件分析 4](#_Toc30149620)

[五、算法模型：Hadoop并行算法 5](#_Toc30149621)

[5.1算法模型图 5](#_Toc30149622)

[5.2并行算法的设计 5](#_Toc30149623)

[六、架构设计：日志KPI系统架构 6](#_Toc30149624)

[6.1日志KPI系统架构图 6](#_Toc30149625)

[6.2 日志KPI系统架构数据流动图 6](#_Toc30149626)

[七、程序开发 7](#_Toc30149627)

[7.1 PVMapper.java 7](#_Toc30149628)

[7.2 PVReducer.java 9](#_Toc30149629)

[7.3 PVRun.java 9](#_Toc30149630)

[7.4 IPMapper.java 11](#_Toc30149631)

[7.5 IPReducer.java 12](#_Toc30149632)

[7.6 IPRun.java 13](#_Toc30149633)

[7.7 分析输出结果 15](#_Toc30149634)

[7.8 原理分析图 18](#_Toc30149635)

[八、总结与展望 18](#_Toc30149636)

[8.1 项目总结 18](#_Toc30149637)

[8.2 展望 18](#_Toc30149638)

# 一、项目简介

## 1.1项目介绍

本项目主要运用Java语言，利用Hadoop以及VMware Workstation Pro在eclipse环境下，对年气温数据进行简单筛选处理，得到年最高气温的简单案例的实现，能够有效的实现对气温数据的简单筛选分析。

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与分析。

此外，本项目主要是对于大量数据按照时间进行数据清洗，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的简单应用。

## 1.2开发工具

Java语言 、Hadoop、VMware Workstation Pro

## 1.3开发环境

Eclipse开发环境

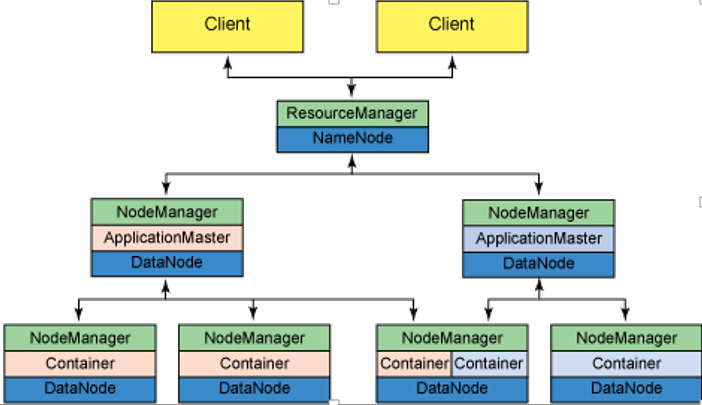
# 二、准备数据

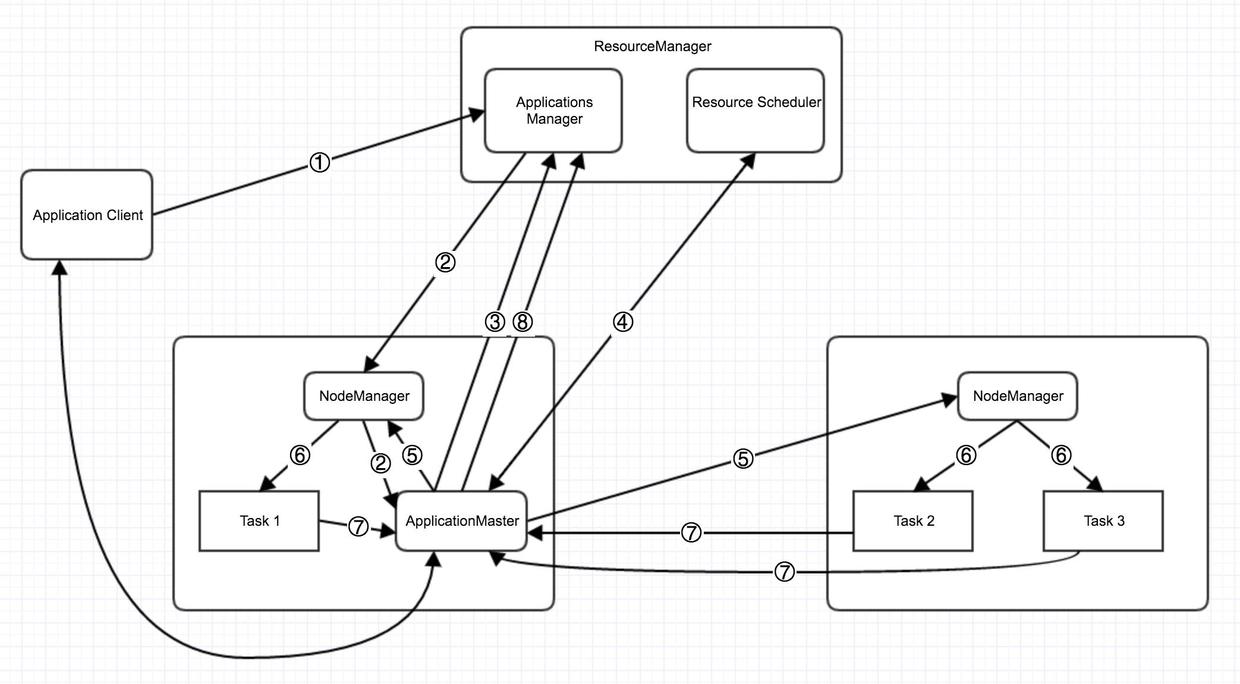
## 2.1 批量导入数据至HDFS

data.txt

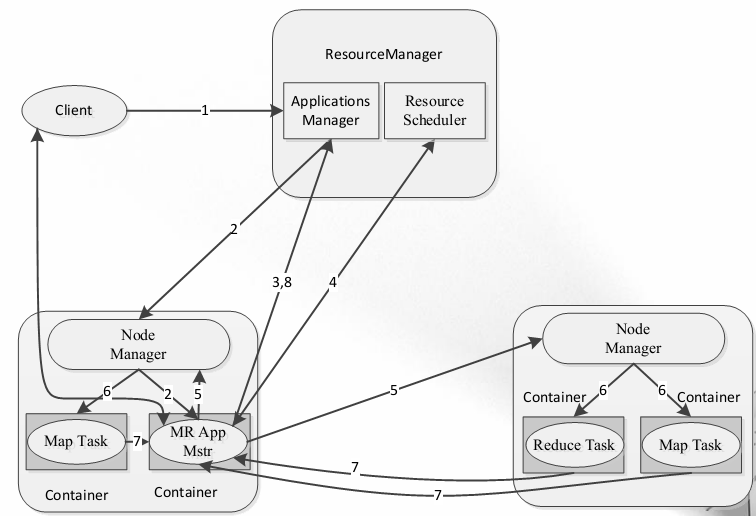
# 三、算法模型：MR并行算法

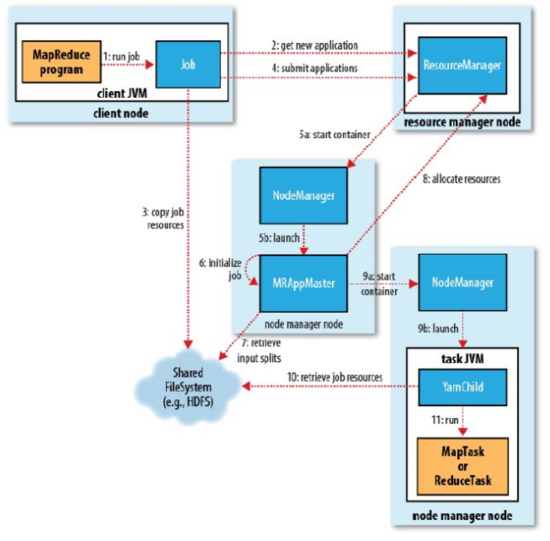
## 3.1算法框架图



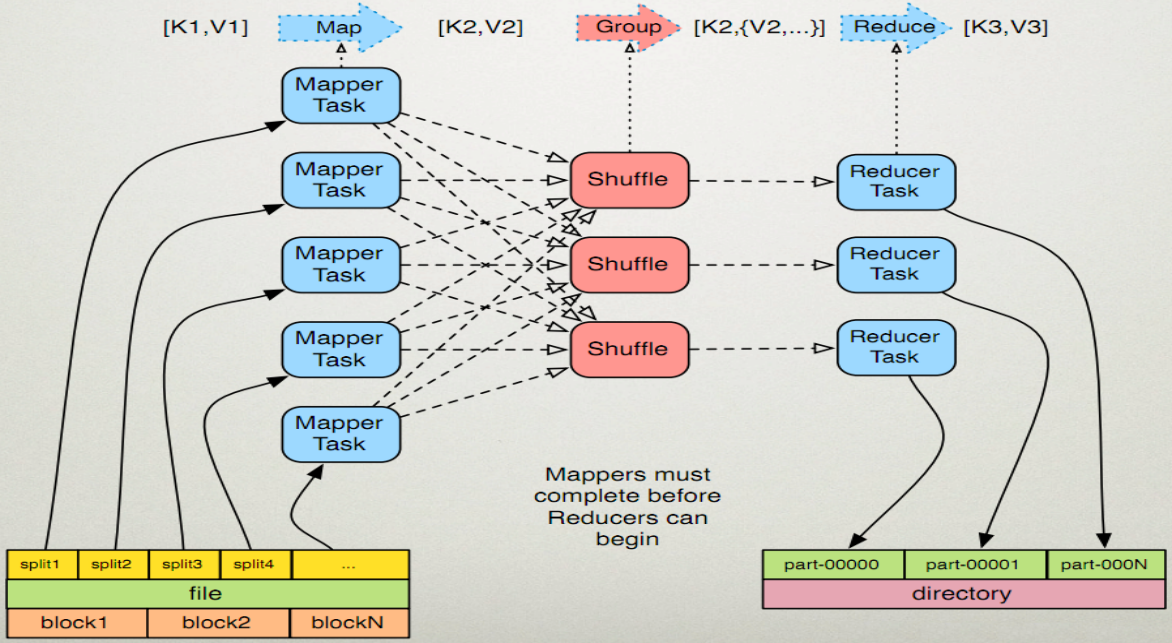


## **3.2算法工作流程图**





## 3.3 洗牌过程



# 四、程序开发

开发流程：

对温度数据的简单清洗及解析🡪Map函数实现🡪Reduce函数实现🡪启动程序实现

## 4.1 Weather.java

package com.neuedu.nuist1801.weather;

import java.io.DataInput;

import java.io.DataOutput;

import java.io.IOException;

import org.apache.hadoop.io.WritableComparable;

/\*\*

\* 1.实体类：传递数据，实现hadoop的序列化同时实现默认排序

\*

\*/

public class Weather implements WritableComparable<Weather> {

private int year;

private int hot;

public Weather() {

}

public Weather(int year, int hot) {

super();

this.year = year;

this.hot = hot;

}

@Override

public String toString() {

return this.year + "\t" + this.hot;

}

@Override

public void write(DataOutput out) throws IOException {

//序列化

out.writeInt(this.year);

out.writeInt(this.hot);

}

@Override

public void readFields(DataInput in) throws IOException {

//反序列化

this.year = in.readInt();

this.hot = in.readInt();

}

@Override

public int compareTo(Weather other) {

//默认排序规则，也可以用于对象比较：1是大于，0是等于，-1是小于

//不同年

//return this.year-other.year;

if (this.year != other.year) {

return Integer.compare(this.year, other.year);

}

//相同年

return Integer.compare(this.hot, other.hot);

}

public int getYear() {

return year;

}

public void setYear(int year) {

this.year = year;

}

public int getHot() {

return hot;

}

public void setHot(int hot) {

this.hot = hot;

}

}

## 4.2 WeatherMapper.java

package com.neuedu.nuist1801.weather;

import java.io.IOException;

import org.apache.commons.lang.StringUtils;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

/\*\*

\* 2.自定义Mapper类：用于将一行数据，转换为Weather对象

\* @author Aria

\*/

public class WeatherMapper extends Mapper<LongWritable, Text, Weather, Text> {

@Override

protected void map(LongWritable key, Text value, Mapper<LongWritable, Text, Weather, Text>.Context context)

throws IOException, InterruptedException {

//1949-10-01 14:21:02 34℃

//1949-10-01 14:21:02 4℃

String line = value.toString();

if (StringUtils.isBlank(line)) {

return;

}

//拆分字符串，并转换为Weather对象

String[] items = line.split("\t");

int year = Integer.parseInt(items[0].substring(0, 4)); //含前不含后

int hot = Integer.parseInt(items[1].substring(0, items[1].length()-1));

Weather w = new Weather(year, hot);

//输出：<Weather对象，整行详细数据字符串>

context.write(w, value);

}

}

## 4.3 WeatherPartitioner.java

package com.neuedu.nuist1801.weather;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Partitioner;

/\*\*

\* 3.自定义分区类：1个分区处理同一年份的数据

\* @author Aria

\*

\*/

public class WeatherPartitioner extends Partitioner<Weather, Text> {

@Override

public int getPartition(Weather key, Text value, int numPartitions) {

//1949-->numPartitions=3-->1949 - 1940 = 9 --> 9 % 3 = 0

//1950-->numPartitions=3-->1950 - 1940 = 10 --> 10 % 3 = 1

//1949-->numPartitions=3-->1949 - 1940 = 11 --> 11 % 3 = 2

return (key.getYear() - 1940) % numPartitions;

}

}

## 4.4 WeatherSort.java

package com.neuedu.nuist1801.weather;

import org.apache.hadoop.io.WritableComparable;

import org.apache.hadoop.io.WritableComparator;

/\*\*

\* 4.自定义排序：继承自WritableComparator,构造器中需要注意

\* @author Aria

\*/

public class WeatherSort extends WritableComparator {

public WeatherSort() {

super(Weather.class, true);

}

@Override

public int compare(WritableComparable a, WritableComparable b) {

Weather w1 = (Weather) a;

Weather w2 = (Weather) b;

//不同年份

if (w1.getYear() != w2.getYear()) {

return Integer.compare(w1.getYear(), w2.getYear());

}

//相同年份，温度从高到低

return - Integer.compare(w1.getHot(), w2.getHot());

}

}

## 4.5 WeatherGrouping.java

package com.neuedu.nuist1801.weather;

import org.apache.hadoop.io.WritableComparable;

import org.apache.hadoop.io.WritableComparator;

/\*\*

\* 5.自定义分组：继承自WritableComparator,构造器中需要注意

\* @author Aria

\*/

public class WeatherGrouping extends WritableComparator {

public WeatherGrouping() {

super(Weather.class, true);

}

@Override

public int compare(WritableComparable a, WritableComparable b) {

Weather w1 = (Weather) a;

Weather w2 = (Weather) b;

//根据年份比较，升序

return Integer.compare(w1.getYear(), w2.getYear());

}

}

## 4.6 WeatherReducer.java

**package** com.neuedu.nuist1801.weather;

**import** java.io.IOException;

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

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

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

/\*\*

\* 6.自定义reducer：完成每个最高温度的统计

\* **@author** Aria

\*

\*/

**public** **class** WeatherReducer **extends** Reducer<Weather, Text, Weather, NullWritable> {

**private** **static** **final** **int** ***TOPK*** = 1;

@Override

**protected** **void** reduce(Weather key, Iterable<Text> values, Reducer<Weather, Text, Weather, NullWritable>.Context context)

**throws** IOException, InterruptedException {

**int** i = 0;

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

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

**if**(++i >= ***TOPK***) {

**break**;

}

}

}

}

## 4.7 Run.java

**package** com.neuedu.nuist1801.weather;

**import** org.apache.hadoop.conf.Configuration;

**import** org.apache.hadoop.fs.FileSystem;

**import** org.apache.hadoop.fs.Path;

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

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

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

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

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

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

**import** org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

/\*\*

\* 6.运行类

\* **@author** Aria

\*/

**public** **class** Run {

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

//配置对象

Configuration conf = **new** Configuration();

**try** {

//HDFS

FileSystem hdfs = FileSystem.*get*(conf);

//输入与输出目标路径

//定义输入目录：意味着读取该目录下所有文件

// String input = "/books";

// 定义输入文件路径，仅读取一个文件

String input = "/weather\_data";

//输出路径是不允许存在的，否则异常

String output = "/weather\_result";

//判断输出目录是否存在，存在则删除

**if** (hdfs.exists(**new** Path(output))) {

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

}

//创建Job任务

Job job = Job.*getInstance*(conf, "weather");

//设置输入：选择最长的包名

job.setInputFormatClass(TextInputFormat.**class**);

//选择最长的包名

FileInputFormat.*setInputPaths*(job, input);

//设置mapper

job.setMapperClass(WeatherMapper.**class**);

job.setMapOutputKeyClass(Weather.**class**);

job.setMapOutputValueClass(Text.**class**);

//设置分区

job.setPartitionerClass(WeatherPartitioner.**class**);

job.setNumReduceTasks(3);

//设置排序

job.setSortComparatorClass(WeatherSort.**class**);

//设置分组

job.setGroupingComparatorClass(WeatherGrouping.**class**);

//设置reducer

job.setReducerClass(WeatherReducer.**class**);

//设置输出

job.setOutputKeyClass(Weather.**class**);

job.setOutputValueClass(NullWritable.**class**);

//选择最长的包名

job.setOutputFormatClass(TextOutputFormat.**class**);

//选择最长的包名

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

//运行

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

//查看结果

**if**(flag) {

System.***out***.println("温度统计结束... ...");

}

} **catch** (Exception e) {

e.printStackTrace();

}

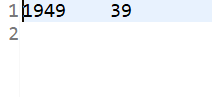
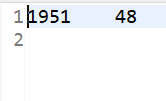
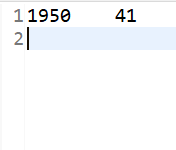
}

}

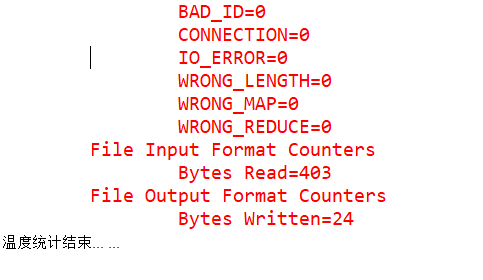
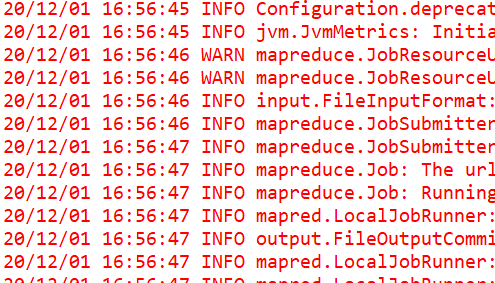
## 4.8 分析输出结果

**1)将年最高温按年份分区输出（仅输出年最高温度的第一个值）：**

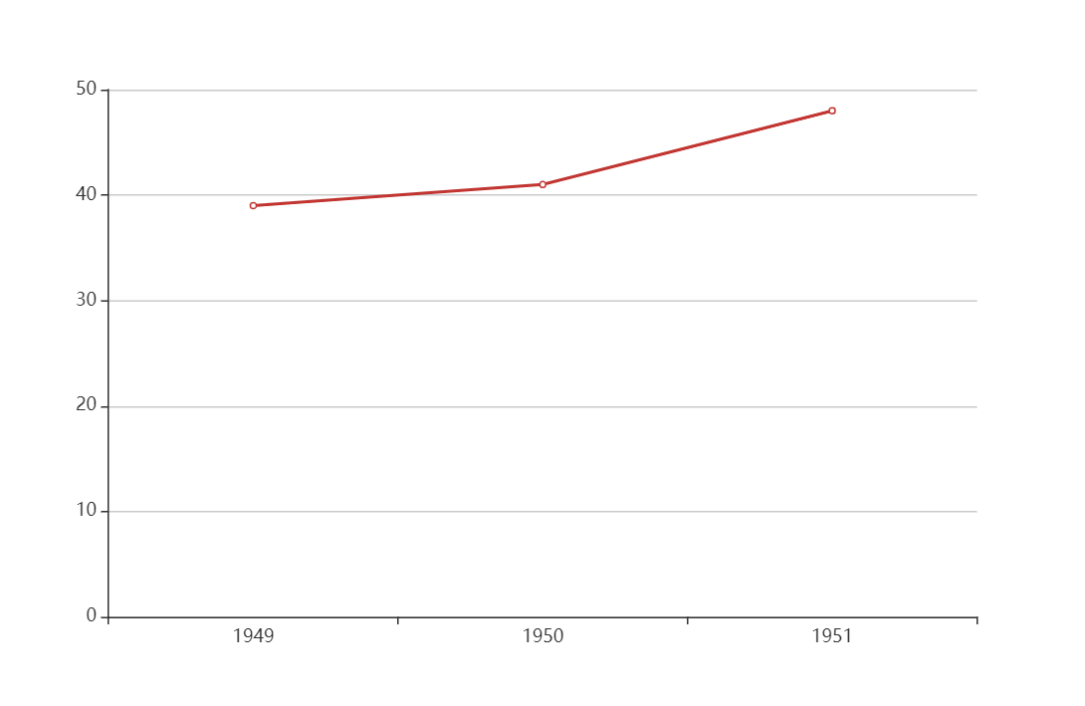
**weather\_result 截图**



**Run 运行日志部分截图**

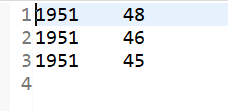
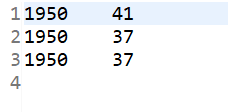
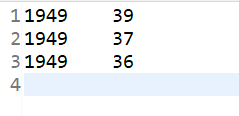


**weather\_result部分结果分析图**

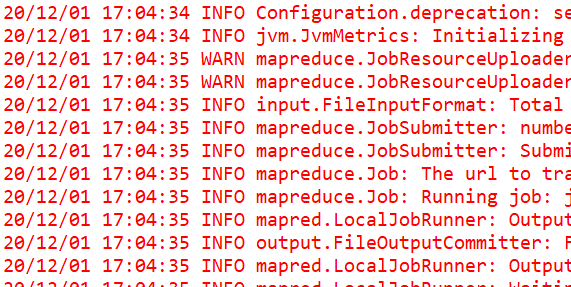
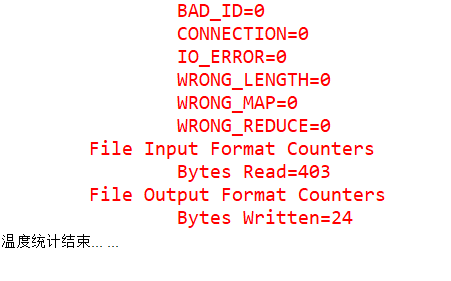
****

**2)将年前三高温按年份分区输出（输出三个值）：**

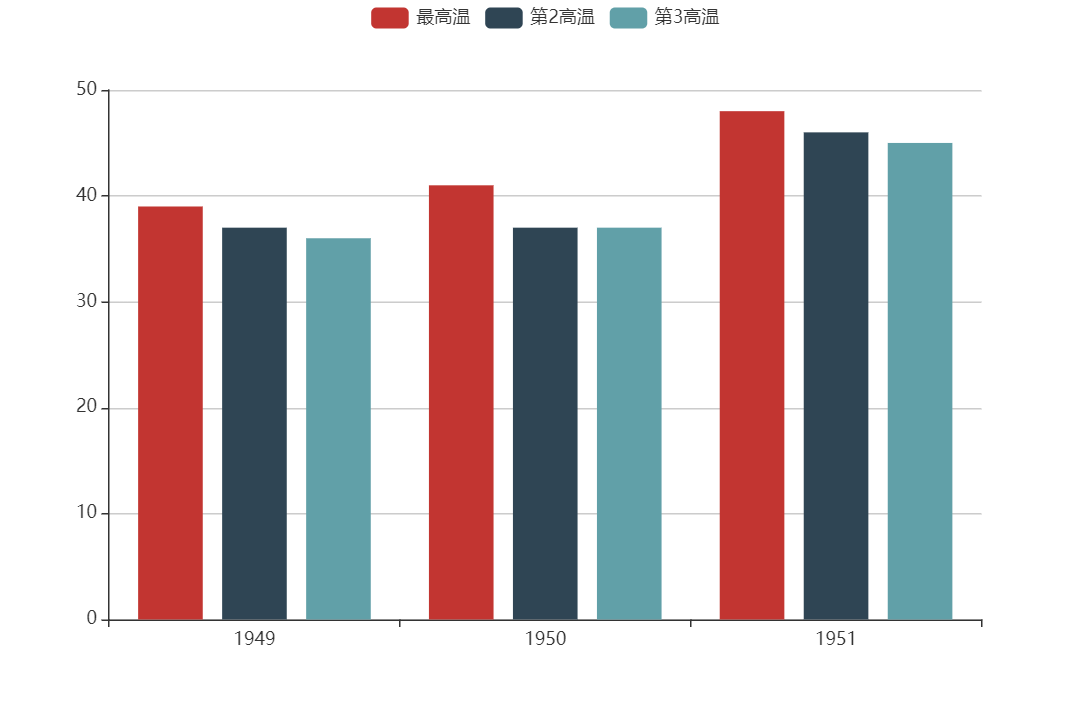
**weather­\_result 截图**



**Run 运行日志截图**



**weather\_result部分结果分析图**

****

通过Run的运行结果，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得到年最高气温数据，能够有效的实现对数据的简单筛选分析。

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，方便对于不同年份天气状况的分析，简单作图后可以看出年最高温的走向、趋势，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。本次实践主要运用了大数据并行计算即MapReducer的知识，让我对MapReducer有了一定的了解。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，意识到并开始培养编写代码的同时编写注释的良好编程习惯。