

目 录

一、项目背景.....	4
1.1 项目背景.....	4
1.2 技术背景.....	4
1.3 项目意义.....	5
二、HADOOP 集群部署.....	5
2.1 安装与配置环境.....	5
2.2 配置核心组件.....	5
2.3 分发配置文件与启动集群.....	6
2.4 WINDOWS 环境配置与访问.....	6
三、HDFS 本地部署.....	7
3.1 HDFS 概述.....	7
3.2 HDFS 结构.....	7
3.3 HDFS 开发环境配置.....	8
四、MAPREDUCE 计算模型.....	8
4.1 MAPREDUCE 并行编程概述.....	8
4.2 MAPREDUCE 架构.....	8
4.3 MAP 与 REDUCE 端的任务处理.....	9
4.4 SHUFFLE 与优化.....	9
五、业务功能分析与实现.....	10
5.1 数据清洗与导入.....	10
5.1.1 功能分析.....	10
5.1.2 具体代码.....	11
5.1.3 运行效果.....	12
5.2 查询指定日期天气.....	12
5.2.1 功能分析.....	12
5.2.2 具体代码.....	13
5.2.3 运行效果.....	14
5.3 统计每年天气.....	14
5.3.1 功能分析.....	14
5.3.2 具体代码.....	14
5.3.3 运行效果.....	16
5.4 预测指定日期天气.....	16
5.4.1 功能分析.....	16

5.4.2 具体代码	17
5.4.3 运行效果	18
5.5 预测未来 15 天天气	19
5.5.1 功能分析	19
5.5.2 具体代码	20
5.5.3 运行效果	21
5.6 每年天气可视化	21
5.6.1 功能分析	21
5.6.2 具体代码	22
5.6.3 运行效果	22
5.7 查询指定年份每月天气数据	23
5.7.1 功能分析	23
5.7.2 具体代码	24
5.7.3 运行效果	25
5.8 每年降雨量可视化	26
5.8.1 功能分析	26
5.8.2 具体代码	26
5.8.3 运行效果	27
六、数据进一步分析	27
6.1 基于 SARIMA 模型未来天气预测	27
6.1.1 SARIMA 模型	27
6.1.2 模型具体代码	27
6.1.3 模型结果与可视化	29
6.2 月平均温度与降雨量相关性分析	30
6.2.1 散点图可视化	30
6.2.2 具体代码	30
6.2.3 皮尔逊相关性分析	31
七、总结	32

一、 项目背景

1.1 项目背景

随着全球气候变化日益严峻，天气数据的准确性和及时性在各行各业的应用中变得愈加重要。气象数据不仅关乎自然环境的变化，还直接影响人类的日常生活、生产活动以及社会经济的运行。从极端天气事件的预测到长期气候趋势的分析，天气数据的获取和处理已成为现代社会应对气候变化、提高社会韧性的重要手段。

随着信息技术的不断进步，大数据技术逐渐成为处理这些庞大气象数据的核心工具。传统的天气预报和气候分析手段已无法应对日益增长的数据量和复杂性，分布式计算平台和大数据存储系统提供了一个强有力的技术支持，可以高效地存储、处理和分析来自不同气象站、卫星、传感器等多渠道的海量数据。

为了让用户能够更加便捷地获取和理解天气数据，数据可视化技术成为了一个重要的研究方向。通过可视化的方式，天气数据不仅可以被直观地展示，还可以通过交互式图表和仪表盘帮助用户更好地理解气候变化的趋势和影响。气象数据的可视化不仅提升了数据的可操作性，也让用户能够在复杂的信息中发现潜在的模式和规律，进而做出更加科学、精准的决策。

因此，构建一个集天气数据查询、分析、预测和可视化为一体的系统，能够帮助各行业快速获取所需的气象信息，并进行深度分析和决策支持。

1.2 技术背景

随着气象观测手段的日益多样化和数据源的不断扩展，气象数据的规模和复杂度迅速增加，给传统的计算和存储系统带来了前所未有的挑战。过去，气象数据的处理主要依赖单机计算和关系型数据库，但随着数据量的激增，这些传统方法已无法有效地支持气象数据的存储、处理和分析需求。因此，大数据技术的出现，尤其是分布式计算框架如 Hadoop 的广泛应用，极大地推动了气象数据处理能力的发展。

Hadoop 生态系统作为一个开源的分布式存储和计算框架，能够处理 PB 级别的海量数据，特别适合存储和处理分布在多个计算节点上的大规模数据。Hadoop 的核心组件紧密结合，通过分布式架构提供了高效、可靠的解决方案。Hadoop Distributed File System (HDFS)，作为 Hadoop 的分布式文件系统，能够将数据划分成多个数据块，并将这些数据块分布存储在不同的节点上。这样不仅提高了存储的可靠性和效率，还能通过数据复制机制确保数据在硬件故障时的高可用性，满足了气象数据庞大且高度分散的特点。

同时，Hadoop 中的 MapReduce 计算模型使得大规模数据的并行处理成为可能。MapReduce 通过将计算任务划分为多个“Map”和“Reduce”阶段，能够充分利用多台计算机的并行处理能力。每个 Map 任务会对输入数据进行处理并生成中间结果，而 Reduce 任务则会对中间结果进行汇总和最终处理。这种方式特别适用于气象数据的批量处理，诸如气温、降水量等历史数据的清洗、转换和分析都可以通过 MapReduce 高效完成。气象数据往往具有强烈的时序性和地域性，通过 MapReduce 的并行化处理，可以加速数据分析的速度，提供更及时的天气预测。

1.3 项目意义

该天气数据处理与可视化系统的意义深远，涵盖了业务价值、技术创新以及广阔的应用前景。首先，从业务价值的角度来看，系统的核心目标是提升决策效率。通过对天气数据的精准分析和预测，系统能够为农业、交通、能源等行业提供及时、可靠的信息，帮助它们应对不断变化的天气条件，从而优化生产、运输和资源分配计划。此外，通过对历史数据的深入分析，预测未来天气趋势，为政府、企业乃至社会大众提供科学的决策支持，有助于减少对天气变化的盲目应对，提高应急反应的效率和准确性。

在技术创新方面，项目通过引入先进的大数据平台和数据可视化技术，推动了气象数据处理方式的革新。大数据技术使得系统能够处理海量天气数据并进行高效分析，为天气预测提供了更为精准和实时的支持。尤其是在数据量庞大且实时变化的背景下，分布式计算能够显著提高数据处理的效率和精度。此外，数据可视化技术通过将复杂的天气数据图形化、交互化的展示，使得用户可以更直观地理解和分析天气变化趋势。

最后，项目在应用前景方面具有广阔的潜力。随着相关技术的不断发展，未来的天气预测系统将更加智能化和自动化，能够提供更高精度、更具实时性的气象服务。机器学习算法的引入将使系统能够更准确地分析天气模式，自动学习和优化预测模型，从而实现更加精准的气象预报。同时，气象数据的应用将超越传统的气象部门，越来越多的行业开始依赖天气数据来指导决策。气象数据不仅是气象学家的工具，更是各行各业优化运营、提升效率和应对风险的重要资源，促进了社会各界在气象数据共享与合作方面的进一步发展。

二、 Hadoop 集群部署

2.1 安装与配置环境

（1）下载并解压 Hadoop

部署 Hadoop 集群的第一步是下载 Hadoop 的二进制包。从 Hadoop 的官方网站选择合适的版本进行下载。下载完成后，将其解压到目标目录。这一过程需要确保所有节点上都安装了相同版本的 Hadoop。

（2）配置环境变量

安装 Hadoop 后，需要在集群的每个节点上配置环境变量。首先，通过设置 `HADOOP_HOME` 环境变量指定 Hadoop 的安装路径，`JAVA_HOME` 指定 Java 的安装路径。可以通过编辑配置文件来设置这些环境变量。这一步确保所有 Hadoop 进程能够正确找到相关依赖。

2.2 配置核心组件

（1）配置 `hadoop-env.sh`

`hadoop-env.sh` 是一个非常重要的配置文件，用于配置 Java 环境和其他系统参数。在此文件中，最重要的是设置 `JAVA_HOME`，指向正确的 Java 安装路径。还可以调整内存、堆栈大小等与性能相关的参数，确保 Hadoop 在集群中的运行高效稳定。

（2）配置核心 XML 文件

Hadoop 集群的核心配置文件包括 `core-site.xml`、`hdfs-site.xml`、`mapred-site.xml` 和 `yarn-site.xml`，它们是集群的基本运行方式。

core-site.xml：这个文件用于配置 Hadoop 的核心文件系统设置。最关键的配置项是 `fs.defaultFS`，它指定了 HDFS 的 URI 格式，这个配置确保了所有节点能够正确访问 HDFS。

hdfs-site.xml：该文件配置了 HDFS 的相关参数，最重要的是数据的存储位置和副本数。常见的配置项包括：设置数据块的副本数，指定 NameNode 存储 HDFS 元数据的目录，指定 DataNode 存储数据块的目录。

mapred-site.xml：该配置文件用于配置 MapReduce 框架，特别是关于资源调度和任务管理的参数。关键配置项有：指定 MapReduce 的执行框架，设置 JobTracker 的地址。

yarn-site.xml：用于配置 YARN（资源管理器）。YARN 负责管理集群的计算资源和调度任务。配置中重要的参数包括：指定 ResourceManager 的地址和指定节点管理器的辅助服务，如日志服务。

2.3 分发配置文件与启动集群

（1）分发配置文件

完成所有配置后，需要将配置文件从 NameNode 节点分发到集群中的每个节点。这可以通过 `scp` 或其他文件传输工具实现。确保所有节点都使用相同的配置文件，以保证集群的一致性和稳定性。

（2）格式化 NameNode

在 Hadoop 集群的首次启动之前，必须对 NameNode 进行格式化。格式化操作会初始化 HDFS 元数据并清除可能存在的旧数据。

（3）查看进程与验证

启动集群后，可以使用 `jps` 命令来查看 Hadoop 进程是否成功启动。成功启动的进程包括 NameNode、DataNode、

2.4 Windows 环境配置与访问

（1）配置 Windows 系统的 hosts 文件

对于在 Windows 上部署 Hadoop 集群的情况，通常需要修改 `hosts` 文件，以确保集群中的各个节点能够通过主机名相互访问。`hosts` 文件需要手动添加集群各个节点的 IP 和主机名映射关系。

（2）使用浏览器访问 Hadoop

Hadoop 集群启动后，可以通过 Web 浏览器访问集群的管理界面。在浏览器中输入 NameNode 的 IP 地址和端口可以访问 HDFS 的 Web UI，查看集群的存储状态和数据分布。输入 ResourceManager 的 IP 地址和端口可以查看 YARN 的资源管理状态。

三、 HDFS 本地部署

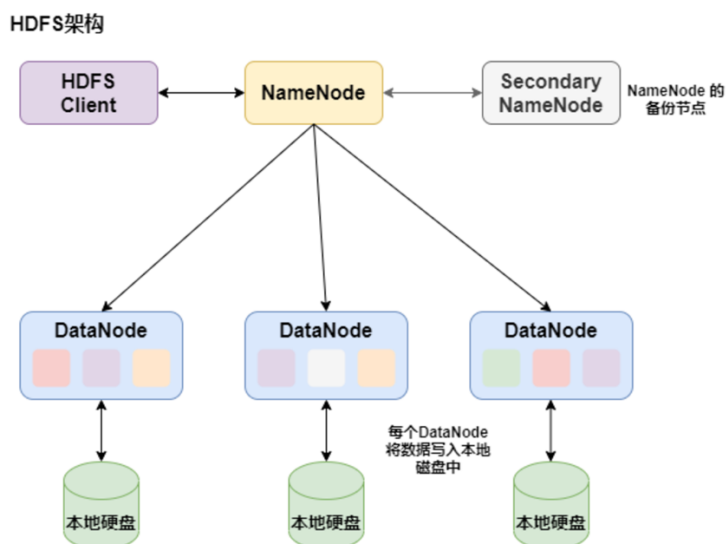
3.1 HDFS 概述

HDFS (Hadoop Distributed File System) 是 Hadoop 生态系统的核心组件之一，专门设计用来存储大规模数据。随着大数据时代的到来，传统的文件系统已经难以应对数据量的爆炸式增长，HDFS 通过分布式存储和高容错机制，提供了高效且可扩展的数据存储方案。HDFS 不仅适用于大数据存储，还为大规模数据处理提供了基础支持。

HDFS 有几个关键特性，使其能够满足大数据存储的需求：HDFS 将每个文件切割成多个数据块，并在集群中多个节点上存储数据块的副本。这样即使某个 DataNode 发生故障，数据仍然可以从其他副本中恢复，确保数据的安全；同时，适合批量处理大文件，它能够高效地进行顺序读取，最大限度地提升数据存储与读取的吞吐量。

3.2 HDFS 结构

HDFS 的体系结构主要由三个核心组件构成：**NameNode**、**DataNode** 和**客户端**。它的架构设计是为了保证系统的高可用性、容错性和高效的数据存储。



(1) **NameNode**: HDFS 的元数据服务器，负责存储整个文件系统的目录结构和文件的元数据。NameNode 不存储实际的数据块内容，而是记录哪些数据块存储在哪些 DataNode 上。它的主要任务是管理文件的命名空间和数据块的位置。NameNode 通常是整个 HDFS 集群的瓶颈，因此需要非常高的可靠性。

(2) **DataNode**: HDFS 的工作节点，负责存储文件的实际数据块。每个 DataNode 负责管理本地的磁盘存储，周期性地向 NameNode 发送心跳信号，报告其存活状态。DataNode 也会定期检查存储的数据块的副本情况，确保副本数满足要求。

(3) **客户端**: HDFS 客户端向 HDFS 系统发送请求，进行数据的上传、读取、删除等操作。客户端通过与 NameNode 交互来获取文件的存储位置，然后直接与 DataNode 进行数据交互。

3.3 HDFS 开发环境配置

(1) 本地环境部署

首先需要安装 JDK，并设置环境变量。然后，使用 Maven 作为项目构建工具，它可以帮助我们管理依赖包，如 Hadoop 的依赖包。接着，配置 Hadoop 环境，包括 HDFS 和 YARN 的设置。

(2) 开发工具的安装与配置

常用的开发工具包括 Eclipse、IntelliJ IDEA 等。安装并配置这些工具后，可以创建 Java 项目，并添加 Hadoop 相关的依赖项。对于使用 Maven 的项目，可以在 pom.xml 中添加 Hadoop 依赖。

(3) 创建项目与添加依赖

在 IDE 中创建一个 Java 项目，并将所需的 Hadoop 依赖包添加到项目中。这些依赖包包括 hadoop-common、hadoop-hdfs 等，可以通过 Maven 自动管理。

四、 MapReduce 计算模型

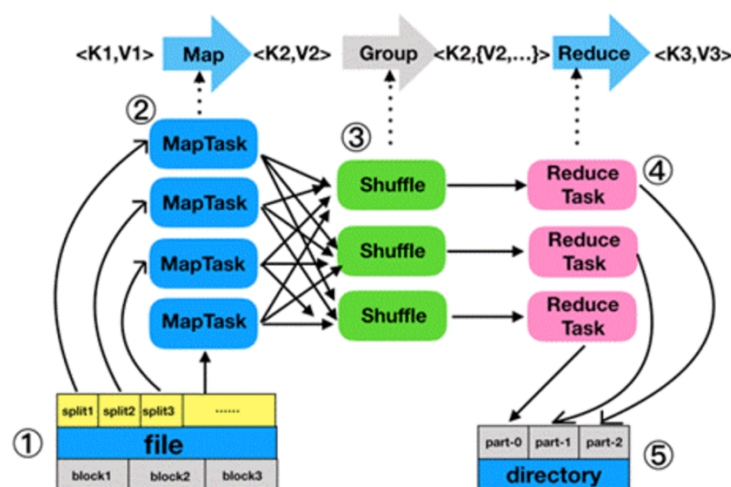
4.1 MapReduce 并行编程概述

MapReduce 是 Hadoop 生态系统中处理大规模数据的核心计算模型，它采用并行计算的思路将任务分解成“Map”和“Reduce”两个阶段，分别进行数据的映射和归约操作，广泛应用于大数据处理和分布式计算场景。

在 Hadoop 中，MapReduce 通过将大规模的数据拆分为多个小任务，并行地在集群中分发执行，从而实现高效的计算。并行计算的思想是通过将计算任务分解为多个子任务，利用集群的多个节点同时执行，从而加快整体计算过程，尤其在处理海量数据时可以显著提高效率。

4.2 MapReduce 架构

MapReduce 的执行过程可以分为三个主要步骤：Map 阶段、Shuffle 阶段和 Reduce 阶段。



(1) Map 阶段：在这个阶段，输入数据被分割成若干块，由多个 Map 任务并行处理。每个 Map 任务根据输入数据生成一系列键值对，这些键值对将作为 Shuffle 阶段的输入。

(2) Shuffle 阶段：Shuffle 是 MapReduce 的一个重要过程，它用于对 Map 阶段生成的键值对进行排序、分区和合并。Shuffle 阶段包括数据的排序、分组和分区等步骤，这些操作确保每个 Reduce 任务接收到相同键的所有值。Shuffle 过程的效率直接影响 MapReduce 作业的整体性能。

(3) Reduce 阶段：在 Reduce 阶段，系统会将 Shuffle 阶段输出的结果（已经按键分组）传递给各个 Reducer 任务。每个 Reducer 任务将对相同键的数据进行处理并产生最终结果。

4.3 Map 与 Reduce 端的任务处理

在 MapReduce 的执行过程中，Map 端和 Reduce 端各自承担着不同的计算任务。

(1) Map 端任务

Map 任务的主要任务是将输入数据转换为键值对，并将这些键值对传递给 Shuffle 阶段。每个 Map 任务会根据输入数据的内容生成相应的键值对，在此过程中，开发者通常会定义自定义的 Map 函数，来实现数据的转换与处理。

(2) Reduce 端任务

Reduce 任务接收 Map 任务输出的数据，并对相同的键进行聚合处理。Reduce 的输出可以是汇总后的结果，也可以是其他形式的数据。Reduce 阶段的性能在很大程度上取决于 Shuffle 的效率，因此优化 Shuffle 过程是提升整体性能的关键。

4.4 Shuffle 与优化

Shuffle 过程是 MapReduce 中的核心，它决定了 Map 和 Reduce 之间的数据传输和分配。在 Shuffle 过程中，Map 输出的键值对会根据键的值进行分组和排序，以便每个 Reducer 只接收属于同一个键的数据。为了提高 Shuffle 的效率，可以使用 Combiner 和 Partitioner 来优化数据传输和分配。

(1) Combiner

Combiner 是一个优化机制，它允许在 Map 阶段进行本地聚合，以减少网络传输的数据量。Combiner 实际上是一个“局部的 Reduce”操作，在 Map 任务本地完成数据的汇总，然后再传输到 Reduce 端，降低了 Reduce 端的计算负担。

(2) Partitioner

Partitioner 决定了 Map 输出的键值对应该分配到哪个 Reducer。在默认情况下，Hadoop 会使用哈希算法将键值对均匀地分配给不同的 Reducer。Partitioner 可以自定义，以实现数据的特定分配需求。

五、 业务功能分析与实现

5.1 数据清洗与导入

5.1.1 功能分析

编写 Hadoop MapReduce 程序，用于对巴西天气数据进行清洗和处理。该程序由 Mapper 和 Reducer 两个主要部分组成，对数据进行处理，并输出清洗后的天气数据。

（1）导入相关依赖

Hadoop 相关类用于输入输出格式、作业配置、路径操作等，StringUtils 提供了用于处理字符串的常用方法；同时自定义 WeatherWritable 类，包含天气数据。

（2）Mapper 数据映射

Mapper 类中输入键值对是 LongWritable 和 Text，输出键值对是 Text 和 WeatherWritable。

map 函数处理每一行输入数据。每行数据是用分号分隔的字段，程序对每行数据进行以下处理：

- 跳过空行和包含“Estacao”的行（这些可能是标题或者无关的数据）。
- 按照分号分割数据行成 19 个字段，如果字段数不等于 19，则跳过该行。
- 只处理以“83377”开头的行，可能是指定某种特定的天气站或地区。
- 提取特定字段（如天气站代码、日期、降水量、温度等）并进行必要的转换。如果某些字段为空或无效，将其设置为默认值。
- 创建一个 WeatherWritable 对象，作为 Mapper 的输出。

（3）Reducer 数据规约

Reducer 类输入的键是 Text，值是 WeatherWritable，输出是 WeatherWritable 和 NullWritable。

reduce 函数将 Map 阶段输出的键值对进行归约处理：

- 按照 code（天气站代码）和 date（日期）分组。
- 遍历同一 code 和 date 的所有 WeatherWritable 值，将它们的降水量、最大温度、最小温度和平均温度进行累加。
- 如果累加后的最大温度、最小温度或平均温度小于等于 0，表示数据无效，跳过该记录。
- 否则，将结果写入输出 context.write，以 WeatherWritable 作为输出数据，NullWritable 作为值。

（4）配置和执行 MapReduce 作业

使用 run 方法配置并执行 MapReduce 作业，并根据输出查看完成结果。

5.1.2 具体代码

```
1 package brazil;
2
3 import org.apache.commons.lang3.StringUtils;
4 import org.apache.hadoop.fs.Path;
5 import org.apache.hadoop.io.LongWritable;
6 import org.apache.hadoop.io.NullWritable;
7 import org.apache.hadoop.io.Text;
8 import org.apache.hadoop.mapreduce.Job;
9 import org.apache.hadoop.mapreduce.Mapper;
10 import org.apache.hadoop.mapreduce.Reducer;
11 import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
12 import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
13 import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
14 import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
15 import java.io.IOException;
16
17 public class Step1 { 1个用法
18     public static class Step1Mapper extends Mapper<LongWritable, Text, Text, WeatherWritable> { 1个用法
19         @Override
20         @Override
21         protected void map(LongWritable key, Text value, Mapper<LongWritable, Text,
22             Text, WeatherWritable>.Context context) throws IOException, InterruptedException {
23             String line = value.toString();
24             if (StringUtils.isBlank(line)) {
25                 return;
26             }
27             if (line.contains("Estacao")) {
28                 return;
29             }
30             String[] fields = line.split(";", limit: 19);
31             if (fields.length != 19) {
32                 return;
33             }
34             if (!line.startsWith("83377")) {
35                 return;
36             }
37             String code = fields[0];
38             String date = fields[1];
39             double precipitation = Double.parseDouble(StringUtils.isBlank(fields[3])?"0":fields[3]);
40             double maxTemperature = Double.parseDouble(StringUtils.isBlank(fields[6])?"0":fields[6]);
41             double minTemperature = Double.parseDouble(StringUtils.isBlank(fields[7])?"0":fields[7]);
42             double avgTemperature = Double.parseDouble(StringUtils.isBlank(fields[16])?"0":fields[16]);
43             WeatherWritable w = new WeatherWritable(code, date, precipitation, maxTemperature, minTemperature, avgTemperature);
44             context.write(new Text(StringUtils.code + "_" + date), w);
45         }
46     }
47     public static class Step1Reducer extends Reducer<Text, WeatherWritable, WeatherWritable, NullWritable> { 1个用法
48         @Override
49         @Override
50         protected void reduce(Text key, Iterable<WeatherWritable> values, Reducer<Text, WeatherWritable,
51             WeatherWritable, NullWritable>.Context context) throws IOException, InterruptedException {
52             String code = key.toString().split("_")[0];
53             String date = key.toString().split("_")[1];
54             WeatherWritable w = new WeatherWritable(code, date);
55             for (WeatherWritable v : values) {
56                 w.setPrecipitation(w.getPrecipitation() + v.getPrecipitation());
57                 w.setMaxTemperature(w.getMaxTemperature() + v.getMaxTemperature());
58                 w.setMinTemperature(w.getMinTemperature() + v.getMinTemperature());
59                 w.setAvgTemperature(w.getAvgTemperature() + v.getAvgTemperature());
60             }
61             if (w.getMaxTemperature() <= 0 || w.getMinTemperature() <= 0 || w.getAvgTemperature() <= 0) {
62                 return;
63             }
64             context.write(w, NullWritable.get());
65         }
66     }
67     public static void run(String input, String output) { 1个用法
68         try {
69             //String input = "/brazil_weather";
70             Path inputPath = new Path(input);
71             //String output = "/step1_output";
72             Path outputPath = new Path(output);
73             myweather2.HadoopUtils.delete(outputPath);
74             Job job = Job.getInstance(HadoopUtils.getConf(), jobName: "数据清洗与导入");
75             job.setInputFormatClass(TextInputFormat.class);
76             FileInputFormat.setInputPaths(job, inputPath);
77             job.setMapperClass(Step1Mapper.class);
78             job.setMapOutputKeyClass(Text.class);
79             job.setMapOutputValueClass(WeatherWritable.class);
80             job.setReducerClass(Step1Reducer.class);
81             job.setOutputKeyClass(WeatherWritable.class);
82
83             job.setOutputValueClass(NullWritable.class);
84             job.setOutputFormatClass(TextOutputFormat.class);
85             FileOutputFormat.setOutputPath(job, outputPath);
86             boolean flag = job.waitForCompletion(verbose: true);
87             if(flag){
88                 System.out.println("数据清洗与导入，运行结束");
89             }else{
90                 System.out.println("mapreduce任务执行错误");
91             }
92         } catch (Exception e) {
93             e.printStackTrace();
94         }
95     }
96 }
```

5.1.3 运行效果

数据清洗与导入，运行结束

```
=====
                        欢迎使用天气数据分析与预测可视化系统
=====

1.数据清洗与导入
2.查询指定日期的天气
3.统计汇总每年的天气
4.预测指定日期的天气
5.预测未来15天的天气
```

File contents						
83377	01/01/1963	0.0	29.0	16.7	21.74	
83377	01/01/1964	3.2	26.0	18.0	20.84	
83377	01/01/1965	21.2	24.7	16.6	19.66	
83377	01/01/1966	20.0	27.8	17.5	20.7	
83377	01/01/1967	0.1	27.6	15.4	21.2	
83377	01/01/1968	0.0	28.6	17.8	22.28	
83377	01/01/1970	45.2	26.3	16.3	19.8	
83377	01/01/1971	0.0	30.3	18.5	24.02	

5.2 查询指定日期天气

5.2.1 功能分析

(1) Mapper 数据映射

Mapper 类：它接受输入的 LongWritable 和 Text，并输出 WeatherWritable 和 NullWritable。

map 方法：在该方法中，首先将每行文本转换为字符串，然后按制表符（\t）分割成字段数组。

-如果字段数量不为 6（代表预期的数据格式），则跳过当前行。

-从字段数组中提取天气站代码、日期、降水量、最大温度、最小温度和平均温度。

-从配置文件中获取一个指定日期，如果当前行的日期和指定日期不匹配，则跳过该行。

-将这些信息封装成一个 WeatherWritable 对象，并将它与 NullWritable 一起输出。

(2) Reducer 数据规约

Reducer 类：其输入是 WeatherWritable 作为键，NullWritable 作为值。输出也是 WeatherWritable 作为键，NullWritable 作为值。

reduce 方法：该方法会收到所有映射过程中 WeatherWritable 类型的键（即天气记录）以及它的所有相关值。

(3) 配置和执行 MapReduce 作业

- 通过 `HadoopUtils.getConf().set("date", date)` 将查询日期传递给 MapReduce 作业的配置。Mapper 中会使用这个配置来过滤符合日期的记录。
- 输入路径和输出路径由参数指定。input 是输入文件路径，output 是结果文件存储路径。
- 设置作业的输入格式、Mapper 和 Reducer 类、输出格式等。TextInputFormat 用于读取文本输入文件，TextOutputFormat 用于输出文本格式的结果。
- 通过 `job.waitForCompletion` 提交作业并等待作业完成。

5.2.2 具体代码

```
1 package brazil;
2
3 import org.apache.commons.lang3.StringUtils;
4 import org.apache.hadoop.fs.Path;
5 import org.apache.hadoop.io.LongWritable;
6 import org.apache.hadoop.io.NullWritable;
7 import org.apache.hadoop.io.Text;
8 import org.apache.hadoop.mapreduce.Job;
9 import org.apache.hadoop.mapreduce.Mapper;
10 import org.apache.hadoop.mapreduce.Reducer;
11 import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
12 import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
13 import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
14 import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
15 import java.io.IOException;
16
17 public class Step2 { 1 个用法
18     public static class Step2Mapper extends Mapper<LongWritable, Text, WeatherWritable, NullWritable> { 1 个用法
19         @Override
20         protected void map(LongWritable key, Text value, Mapper<LongWritable, Text,
21             WeatherWritable, NullWritable>.Context context) throws IOException, InterruptedException {
22             String line = value.toString();
23             if (StringUtils.isBlank(line)) {
24                 return;
25             }
26             String[] fields = line.split(" ");
27             if (fields.length != 6) {
28                 return;
29             }
30             String code = fields[0];
31             String date = fields[1];
32             String searchDate = context.getConfiguration().get("date");
33             if (!date.equals(searchDate)) {
34                 return;
35             }
36             double precipitation = Double.parseDouble(fields[2]);
37             double maxTemperature = Double.parseDouble(fields[3]);
38             double minTemperature = Double.parseDouble(fields[4]);
39             double avgTemperature = Double.parseDouble(fields[5]);
40             WeatherWritable w = new WeatherWritable(code, date, precipitation, maxTemperature, minTemperature, avgTemperature);
41             context.write(w, NullWritable.get());
42         }
43     }
44
45     private static class Step2Reducer extends Reducer<WeatherWritable, NullWritable, WeatherWritable, NullWritable> { 1 个用法
46         @Override
47         protected void reduce(WeatherWritable key, Iterable<NullWritable> values, Reducer<WeatherWritable, NullWritable,
48             WeatherWritable, NullWritable>.Context context) throws IOException, InterruptedException {
49             context.write(key, NullWritable.get());
50         }
51     }
52
53     public static void run(String input, String output, String date) { 1 个用法
54         try {
55             HadoopUtils.getConf().set("date", date);
56             //String input = "/brazil_weather";
57             Path inputPath = new Path(input);
58             //String output = "/step1_output";
59             Path outputPath = new Path(output);
60             HadoopUtils.delete(outputPath);
61             Job job = Job.getInstance(HadoopUtils.getConf());
62             job.setInputFormatClass(TextInputFormat.class);
63             FileInputFormat.setInputPaths(job, inputPath);
64             job.setMapperClass(Step2Mapper.class);
65             job.setMapOutputKeyClass(WeatherWritable.class);
66             job.setMapOutputValueClass(NullWritable.class);
67             job.setReducerClass(Step2Reducer.class);
68             job.setOutputKeyClass(WeatherWritable.class);
69             job.setOutputValueClass(NullWritable.class);
70             FileOutputFormat.setOutputPath(job, outputPath);
71             boolean flag = job.waitForCompletion(true);
72             if (flag) {
73                 System.out.println("查询指定日期的天气，运行结束");
74             } else {
75                 System.out.println("mapreduce任务执行错误");
76             }
77         } catch (Exception e) {
78             e.printStackTrace();
79         }
80     }
81 }
```

5.2.3 运行效果

0. 退出

请输入您的选择: 2

请输入要查询的日期(dd/MM/yyyy):

02/02/2002

bytes in 1000ms

查询指定日期的天气, 运行结束

83377 02/02/2002 10.0 28.1 19.1 22.92

=====

欢迎使用天气数据分析与预测可视化系统

=====

1. 数据清洗与导入

2. 查询指定日期的天气

3. 统计汇总每年的天气

4. 预测指定日期的天气

5.3 统计每年天气

5.3.1 功能分析

(1) Mapper 数据映射

-输入数据: 每行数据代表一个特定天气站在某一天的天气记录, 包括天气站代码、日期、降水量、最大温度、最小温度、平均温度。

-数据清洗: 如果行数据为空或格式不对, 该行数据被跳过。

-提取年份: 从日期字段中提取年份。

-创建 WeatherWritable 对象: 将每个字段(天气站代码、年份、降水量、温度等)封装进 WeatherWritable 对象。

-输出: 将 code + "_" + year 作为 Map 阶段的输出键(键是天气站代码与年份的组合), WeatherWritable 作为值。

(2) Reducer: 数据归约

-输入数据: 每个 key 是 code + "_" + year (天气站代码和年份), values 是所有该天气站和年份的天气记录(多个 WeatherWritable 对象)。

-统计过程: 通过遍历所有相关记录, 对每个天气站和年份的降水量、最大温度、最小温度、平均温度进行累加并对所有记录的平均温度求平均。

-结果输出: 计算完所有数据后, w.setAvgTemperature 用于计算平均温度。

(3) 配置和执行 MapReduce 作业

-作业配置: 在 run 方法中, 配置了 MapReduce 作业的所有参数, Mapper 输出的键值类型为 Text 和 WeatherWritable, Reducer 的输出为 WeatherWritable 和 NullWritable。

-执行作业: 使用 job.waitForCompletion(true) 启动 MapReduce 作业并等待其完成。

5.3.2 具体代码

```

1 package brazil;
2
3 import org.apache.commons.lang3.StringUtils;
4 import org.apache.hadoop.fs.Path;
5 import org.apache.hadoop.io.LongWritable;
6 import org.apache.hadoop.io.NullWritable;
7 import org.apache.hadoop.io.Text;
8 import org.apache.hadoop.mapreduce.Job;
9 import org.apache.hadoop.mapreduce.Mapper;
10 import org.apache.hadoop.mapreduce.Reducer;
11 import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
12 import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
13 import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
14 import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
15 import java.io.IOException;
16
17 public class Step3 { 1个用法
18     public static class Step3Mapper extends Mapper<LongWritable, Text, Text, WeatherWritable> { 1个用法
19         @Override
20         protected void map(LongWritable key, Text value, Mapper<LongWritable, Text,
21             Text, WeatherWritable>.Context context) throws IOException, InterruptedException {
22             String line = value.toString();
23             if (StringUtils.isBlank(line)) {
24                 return;
25             }
26             String[] fields = line.split( regex: "\\t");
27             if (fields.length != 6) {
28                 return;
29             }
30             String code = fields[0];
31             String date = fields[1];
32             String year = date.substring( beginIndex: date.lastIndexOf( str: "/")+1);
33             double precipitation = Double.parseDouble(fields[2]);
34             double maxTemperature = Double.parseDouble(fields[3]);
35             double minTemperature = Double.parseDouble(fields[4]);
36             double avgTemperature = Double.parseDouble(fields[5]);
37             WeatherWritable w = new WeatherWritable(code, year, precipitation, maxTemperature, minTemperature, avgTemperature);
38             context.write(new Text( string: code + "_" + year), w);
39         }
40     }
41
42     private static class Step3Reducer extends Reducer<Text, WeatherWritable, WeatherWritable, NullWritable> { 1个用法
43         @Override
44         protected void reduce(Text key, Iterable<WeatherWritable> values, Reducer<Text, WeatherWritable,
45             WeatherWritable, NullWritable>.Context context) throws IOException, InterruptedException {
46             String code = key.toString().split( regex: "_")[0];
47             String year = key.toString().split( regex: "_")[1];
48             WeatherWritable w = new WeatherWritable(code, year);
49             w.setMinTemperature(Double.MAX_VALUE);
50             int count = 0;
51             for(WeatherWritable v : values){
52                 w.setPrecipitation(w.getPrecipitation() + v.getPrecipitation());
53                 w.setMaxTemperature(Math.max(w.getMaxTemperature(), v.getMaxTemperature()));
54                 w.setMinTemperature(Math.min(w.getMinTemperature(), v.getMinTemperature()));
55                 w.setAvgTemperature(w.getAvgTemperature() + v.getAvgTemperature());
56                 count++;
57             }
58             w.setAvgTemperature(w.getAvgTemperature() / count);
59             context.write(w, NullWritable.get());
60         }
61     }
62
63     public static void run(String input, String output) { 1个用法
64         try {
65             //String input = "/brazil_weather";
66             Path inputPath = new Path(input);
67             //String output = "/step1_output";
68             Path outputPath = new Path(output);
69             HadoopUtils.delete(outputPath);
70             Job job = Job.getInstance(HadoopUtils.getConf(), jobName: "统计汇总每年的天气");
71             job.setInputFormatClass(TextInputFormat.class);
72             FileInputFormat.setInputPaths(job, inputPath);
73             job.setMapperClass(Step3Mapper.class);
74             job.setMapOutputKeyClass(Text.class);
75             job.setMapOutputValueClass(WeatherWritable.class);
76             job.setReducerClass(Step3Reducer.class);
77             job.setOutputKeyClass(WeatherWritable.class);
78             job.setOutputValueClass(NullWritable.class);
79             job.setOutputFormatClass(TextOutputFormat.class);
80
81             job.setMapOutputValueClass(WeatherWritable.class);
82             job.setReducerClass(Step3Reducer.class);
83             job.setOutputKeyClass(WeatherWritable.class);
84             job.setOutputValueClass(NullWritable.class);
85             job.setOutputFormatClass(TextOutputFormat.class);
86             FileOutputFormat.setOutputPath(job, outputPath);
87             boolean flag = job.waitForCompletion( verbose: true);
88             if(flag){
89                 System.out.println("统计汇总每年的天气，运行结束");
90             }else{
91                 System.out.println("mapreduce任务执行错误");
92             }
93         } catch (Exception e) {
94             e.printStackTrace();
95         }
96     }
97 }

```

5.3.3 运行效果

统计汇总每年的天气，运行结束

```
83377 1961 32.4 33.4 10.8 22.685970149253734
83377 1962 827.3000000000001 33.0 5.3 19.31308943089431
83377 1963 1285.5000000000005 34.5 9.0 21.197534246575344
83377 1964 1960.1 30.7 6.0 20.00060109289618
83377 1965 2004.3999999999999 33.0 9.2 20.359452054794506
83377 1966 1687.0999999999992 32.0 9.0 20.2318904109589
83377 1967 1260.9999999999995 32.5 9.5 20.530794520547936
83377 1968 1668.5999999999995 33.3 7.8 19.705573770491803
83377 1969 1448.6 33.2 7.1 20.85223463687151
83377 1970 1733.6 31.6 10.1 20.53112328767125
83377 1971 1833.8999999999999 31.2 9.4 20.333698630136983
83377 1972 1468.9999999999993 32.8 10.3 20.865409836065584
83377 1973 1434.6000000000001 32.5 7.4 21.21742465753424
83377 1974 1291.0000000000002 32.0 8.6 20.526483516483523
83377 1975 1353.8999999999994 32.8 1.6 20.439616438356172
83377 1976 1503.1 31.4 9.8 20.401803278688533
83377 1977 1221.1 32.0 3.2 20.9252602739726
83377 1978 1580.6999999999998 32.5 5.0 20.27797260273971
83377 1979 1455.3000000000006 32.4 7.2 20.51975903614458
83377 1980 1662.5 33.4 11.4 20.62754098360654
83377 1981 1475.2999999999995 33.0 6.6 20.299879154078525
83377 1982 1476.7 31.6 10.8 20.462630136986295
83377 1983 1284.2 31.2 8.6 20.659871382636656
83377 1984 1118.3000000000002 31.8 9.1 20.855846994535508
83377 1985 1525.6999999999996 31.3 3.3 20.342197802197802
-----
```

5.4 预测指定日期天气

5.4.1 功能分析

(1) Mapper 数据映射

- 输入：每行数据包含天气站代码、日期、降水量、最高温度、最低温度和平均温度，这些数据以制表符分隔。
- 数据清洗：程序首先检查数据行是否为空，并验证数据行是否符合预期格式。如果不符合要求，则跳过该行。
- 提取日期：从日期字段中提取出“月/日”部分（substring(0, 5)），并与配置文件中的目标日期（predictDate）进行匹配。如果不匹配，则跳过该行。
- 构建 WeatherWritable 对象：将处理后的数据（天气站代码、日期、降水量、温度等）封装进 WeatherWritable 对象，这个对象是自定义的 Writable 类。
- 输出：Text(code + "_" + date)作为键，WeatherWritable 作为值，写入上下游传递。

(2) Reducer：数据归约

- 输入：key 是天气站代码和日期的组合（例如 code_date），values 是该天气站和日期的所有天气数据（WeatherWritable 对象）。
- 统计过程：通过遍历所有与该 key 匹配的 WeatherWritable 对象，计算出该日期和天气站的统计信息：降水量，最高温度，最低温度，平均温度
- 结果输出：计算完成后，使用 context.write(w, NullWritable.get())将结果写入输出，NullWritable.get()表示没有实际的值输出。

(3) 配置和执行 MapReduce 作业

- 配置：该方法首先设置配置项，将目标日期存入 Hadoop 作业配置中，供 Mapper 使用（HadoopUtils.getConf().set("date", date)）。

- 输入输出路径：设置输入路径 inputPath 和输出路径 outputPath，并确保输出路径不存在之前的数据（通过 HadoopUtils.delete(outputPath)）。
- 作业配置：通过 Job.getInstance() 配置 MapReduce 作业，指定输入格式、Mapper 类、Reducer 类、输出格式等。
- 执行作业：通过 job.waitForCompletion(true) 启动 MapReduce 作业并等待其完成。如果作业执行成功，输出“预测指定日期的天气，运行结束”；如果失败，则输出错误信息。

5.4.2 具体代码

```
1 package brazil;
2
3 import org.apache.commons.lang3.StringUtils;
4 import org.apache.hadoop.fs.Path;
5 import org.apache.hadoop.io.LongWritable;
6 import org.apache.hadoop.io.NullWritable;
7 import org.apache.hadoop.io.Text;
8 import org.apache.hadoop.mapreduce.Job;
9 import org.apache.hadoop.mapreduce.Mapper;
10 import org.apache.hadoop.mapreduce.Reducer;
11 import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
12 import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
13 import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
14 import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
15 import java.io.IOException;
16
17 public class $Step4 { 1个用法
18     public static class Step4Mapper extends Mapper<LongWritable, Text, Text, WeatherWritable> { 1个用法
19         @Override
20         protected void map(LongWritable key, Text value, Mapper<LongWritable, Text,
21             Text, WeatherWritable>.Context context) throws IOException, InterruptedException {
22             String line = value.toString();
23             if (StringUtils.isBlank(line)) {
24                 return;
25             }
26             String[] fields = line.split( regex: "\\t");
27             if (fields.length != 6) {
28                 return;
29             }
30             String code = fields[0];
31             String date = fields[1].substring(0,5);
32             String predictDate = context.getConfiguration().get("date");
33             if(!date.equals(predictDate)){
34                 return;
35             }
36             double precipitation = Double.parseDouble(fields[2]);
37             double maxTemperature = Double.parseDouble(fields[3]);
38             double minTemperature = Double.parseDouble(fields[4]);
39             double avgTemperature = Double.parseDouble(fields[5]);
40             WeatherWritable w = new WeatherWritable(code,date,precipitation,maxTemperature,minTemperature,avgTemperature);
41             context.write(new Text( string: code + "_" + date),w);
```

```

42     }
43 }
44
45 private static class Step4Reducer extends Reducer<Text, WeatherWritable, WeatherWritable, NullWritable> { 1个用法
46     @Override
47     protected void reduce(Text key, Iterable<WeatherWritable> values, Reducer<Text, WeatherWritable,
48         WeatherWritable, NullWritable>.Context context) throws IOException, InterruptedException {
49         String code = key.toString().split( regex: "_" )[0];
50         String date = key.toString().split( regex: "_" )[1];
51         WeatherWritable w = new WeatherWritable(code, date);
52         int count = 0;
53         for(WeatherWritable v : values){
54             w.setPrecipitation(w.getPrecipitation() + v.getPrecipitation());
55             w.setMaxTemperature(w.getMaxTemperature() + v.getMaxTemperature());
56             w.setMinTemperature(w.getMinTemperature() + v.getMinTemperature());
57             w.setAvgTemperature(w.getAvgTemperature() + v.getAvgTemperature());
58             count++;
59         }
60         w.setPrecipitation(w.getPrecipitation() / count);
61         w.setMaxTemperature(w.getMaxTemperature() / count);
62         w.setMinTemperature(w.getMinTemperature() / count);
63         w.setAvgTemperature(w.getAvgTemperature() / count);
64         context.write(w, NullWritable.get());
65     }
66 }
67
68 public static void run(String input, String output, String date) { 1个用法
69     try {
70         HadoopUtils.getConf().set("date", date);
71         //String input = "/brazil_weather";
72         Path inputPath = new Path(input);
73         //String output = "/step1_output";
74         Path outputPath = new Path(output);
75         HadoopUtils.delete(outputPath);
76         Job job = Job.getInstance(HadoopUtils.getConf(), jobName: "预测指定日期的天气");
77         job.setInputFormatClass(TextInputFormat.class);
78         FileInputFormat.setInputPaths(job, inputPath);
79         job.setMapperClass(Step4Mapper.class);
80         job.setMapOutputKeyClass(Text.class);
81         job.setMapOutputValueClass(WeatherWritable.class);
82         job.setReducerClass(Step4Reducer.class);
83         job.setOutputKeyClass(WeatherWritable.class);
84         job.setOutputValueClass(NullWritable.class);
85         job.setOutputFormatClass(TextOutputFormat.class);
86         FileOutputFormat.setOutputPath(job, outputPath);
87         boolean flag = job.waitForCompletion( verbose: true );
88         if(flag){
89             System.out.println("预测指定日期的天气，运行结束");
90         }else{
91             System.out.println("mapreduce任务执行错误");
92         }
93     } catch (Exception e) {
94         e.printStackTrace();
95     }
96 }
97 }
98

```

5. 4. 3 运行效果

0. 退出

请输入您的选择：4

请输入要查询的日期(dd/MM):

11/03

预测指定日期的天气，运行结束

83377 11/03 6.971929824561401 27.0421052631579 17.89122807017544 21.66631578947369

=====

欢迎使用天气数据分析与预测可视化系统

=====

1. 数据清洗与导入

2. 查询指定日期的天气

3. 统计汇总每年的天气

5.5 预测未来 15 天天气

5.5.1 功能分析

(1) Mapper 数据映射

- 输入数据：每行数据包含天气站代码、日期、降水量、最大温度、最小温度和平均温度，这些字段通过制表符分隔。
- 数据清洗：检查空行或不符合格式的行（字段数不为 6），若不符合格式则跳过。
- 日期提取：从日期字段中提取出 MM/dd 部分（只取月和日），并与当前日期之后的 15 天进行匹配。
- 日期匹配：使用 Calendar 类和 SimpleDateFormat 类计算未来 15 天的日期。如果当前行的数据日期匹配未来 15 天中的任意一天（predict_date），则设置 isFuture15 = true。若日期不匹配未来 15 天中的任何一天，跳过该行数据。
- 构建 WeatherWritable 对象：若日期匹配，构建一个 WeatherWritable 对象，并将天气站代码、日期、降水量、温度等数据封装到该对象中。
- 输出：将 code + ";" + date 作为键（Text 类型），WeatherWritable 对象作为值输出。

(2) Reducer：数据归约

- 输入数据：key 为 code + ";" + date（天气站代码和日期），values 为该天气站和日期的所有 WeatherWritable 对象（可能有多个相同日期的记录）。
- 数据统计：通过遍历所有 values，累加降水量、最大温度、最小温度和平均温度。count 用于记录该日期的数据条数，计算平均温度时使用。
- 计算完毕后，将总降水量、最大温度、最小温度和平均温度除以记录数，得到这些值的平均值。
- 输出结果：context.write 将处理后的数据写入输出。

(3) 配置和执行 MapReduce 作业

- 配置作业：在 run 方法中，首先设置输入路径和输出路径，并确保输出路径之前没有存在的数据（通过 HadoopUtils.delete(outputPath) 删除旧的数据）。
- 作业配置：设置输入格式为 TextInputFormat，即按行读取文本文件，设置 Mapper 和 Reducer 类，输出结果为文本文件。
- 执行作业：使用 job.waitForCompletion(true) 启动作业并等待其完成。

5.5.2 具体代码

```
1 package brazil;
2
3 import myweather2.HadoopUtils;
4 import org.apache.commons.lang3.StringUtils;
5 import org.apache.hadoop.fs.Path;
6 import org.apache.hadoop.io.LongWritable;
7 import org.apache.hadoop.io.NullWritable;
8 import org.apache.hadoop.io.Text;
9 import org.apache.hadoop.mapreduce.Job;
10 import org.apache.hadoop.mapreduce.Mapper;
11 import org.apache.hadoop.mapreduce.Reducer;
12 import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
13 import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
14 import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
15 import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
16
17 import java.io.IOException;
18 import java.text.SimpleDateFormat;
19 import java.util.Calendar;
20 import java.util.Date;
21
22
23 public class Step5 { 1 个用法
24     private static class Step4Mapper extends Mapper<LongWritable, Text, Text, WeatherWritable> { 1 个用法
25         @Override
26         protected void map(LongWritable key, Text value, Mapper<LongWritable, Text,
27             Text, WeatherWritable>.Context context) throws IOException, InterruptedException {
28             String line = value.toString();
29             if (StringUtils.isBlank(line)){
30                 return;
31             }
32             String[] fields = line.split( regex: "\\t");
33             if (fields.length !=6){
34                 return;
35             }
36             String code = fields[0];
37             String date = fields[1].substring(0,5);
38             Calendar cal = Calendar.getInstance();
39             Date today = new Date();
40             cal.setTime(today);
41             boolean isFuture15 = false;
42
43             SimpleDateFormat sdf = new SimpleDateFormat( pattern: "dd/MM");
44             for (int i=1;i<=15;i++){
45                 String predict_date = sdf.format(cal.getTime());
46                 System.out.println(date+"\t"+predict_date);
47                 if (date.equals(predict_date)){
48                     isFuture15 = true;
49                 }
50                 cal.add(Calendar.DATE, amount: 1);
51             }
52             if (!isFuture15){
53                 return;
54             }
55             double precipitation = Double.parseDouble(fields[2]);
56             double maxTemperature = Double.parseDouble(fields[3]);
57             double minTemperature = Double.parseDouble(fields[4]);
58             double avgTemperature = Double.parseDouble(fields[5]);
59             WeatherWritable w = new WeatherWritable(code,date,precipitation,maxTemperature,minTemperature,avgTemperature);
60
61             context.write(new Text( string: code+"*"+date),w);
62         }
63     }
64     private static class Step4Reducer extends Reducer<Text,WeatherWritable,WeatherWritable, NullWritable> { 1 个用法
65         @Override
66         protected void reduce(Text key, Iterable<WeatherWritable> values, Reducer<Text, WeatherWritable,
67             WeatherWritable, NullWritable>.Context context) throws IOException, InterruptedException {
68             String code = key.toString().split( regex: "*")[0];
69             String date = key.toString().split( regex: "*")[1];
70             WeatherWritable w= new WeatherWritable(code,date);
71             int count =0;
72             for (WeatherWritable v :values){
73                 w.setPrecipitation(w.getPrecipitation()+v.getPrecipitation());
74                 w.setMaxTemperature(w.getMaxTemperature()+v.getMaxTemperature());
75                 w.setMinTemperature(w.getMinTemperature()+v.getMinTemperature());
76                 w.setAvgTemperature(w.getAvgTemperature() + v.getAvgTemperature());
77                 count++;
78             }
79         }
80     }
81 }
```

```

82         context.write(w, NullWritable.get());
83     }
84 }
85
86
87 public static void run(String input, String output) {
88     try {
89         // String input = "/brazil_weather";
90         Path inputPath = new Path(input);
91         // String output = "/step1_output";
92         Path outputPath = new Path(output);
93         HadoopUtils.delete(outputPath);
94         Job job = Job.getInstance(HadoopUtils.getConf(), jobName: "预测未来15天天气");
95
96         job.setInputFormatClass(TextInputFormat.class);
97         FileInputFormat.setInputPaths(job, inputPath);
98
99         job.setMapperClass(Step4Mapper.class);
100        job.setMapOutputKeyClass(Text.class);
101        job.setMapOutputValueClass(WeatherWritable.class);
102
103        job.setReducerClass(Step4Reducer.class);
104        job.setOutputKeyClass(WeatherWritable.class);
105        job.setOutputValueClass(NullWritable.class);
106
107        job.setOutputFormatClass(TextOutputFormat.class);
108        FileOutputFormat.setOutputPath(job, outputPath);
109
110        boolean flag = job.waitForCompletion(verbose: true);
111        if (flag) {
112            System.out.println("预测未来15天天气，运行结束~");
113        } else {
114            System.out.println("任务执行错误~");
115        }
116    } catch (Exception e) {
117        e.printStackTrace();
118    }
}

```

5.5.3 运行效果

预测未来15天天气，运行结束~

83377	03/12	7.427586206896552	26.60344827586206	17.818965517241374	21.426206896551722
83377	04/12	7.53392857142857	26.882142857142856	17.857142857142858	21.535357142857134
83377	05/12	5.7789473684210515	26.8	17.98070175438596	21.55824561403508
83377	06/12	8.094736842105263	26.69473684210526	17.87368421052632	21.440701754385962
83377	07/12	7.318965517241378	26.82241379310345	17.910344827586208	21.50793103448276
83377	08/12	9.256896551724138	26.727586206896557	18.041379310344833	21.547241379310343
83377	09/12	9.943859649122809	26.429824561403503	17.956140350877195	21.31929824561404
83377	10/12	5.244827586206896	26.639655172413793	18.06551724137931	21.428275862068965
83377	11/12	9.296491228070174	26.347368421052636	18.107017543859648	21.474736842105262
83377	12/12	7.712280701754386	26.33684210526316	17.970175438596495	21.448771929824566
83377	13/12	6.30000000000000025	26.473684210526308	17.989473684210523	21.471929824561414
83377	14/12	8.625862068965514	26.431034482758612	17.939655172413797	21.275172413793104
83377	15/12	7.60000000000000005	26.410526315789472	17.761403508771934	21.20491228070175
83377	16/12	11.59824561403509	26.0140350877193	17.842105263157897	21.065263157894734
83377	17/12	11.324137931034484	26.294827586206893	17.632758620689653	21.151379310344822

欢迎使用天气数据分析与预测可视化系统

1.数据清洗与导入

5.6 每年天气可视化

5.6.1 功能分析

(1) 读取数据

gerWeathers(new Path(PATHS.get("step3_output"))): 这个方法调用的是从指定路径读取天气数据。

(2) 创建数组

-y1: 存储每条记录的最大温度数据。

-y2: 存储每条记录的平均温度数据。

- y3: 存储每条记录的最小温度数据。
- x: 存储每条记录的日期数据（假设是每年的数据）。

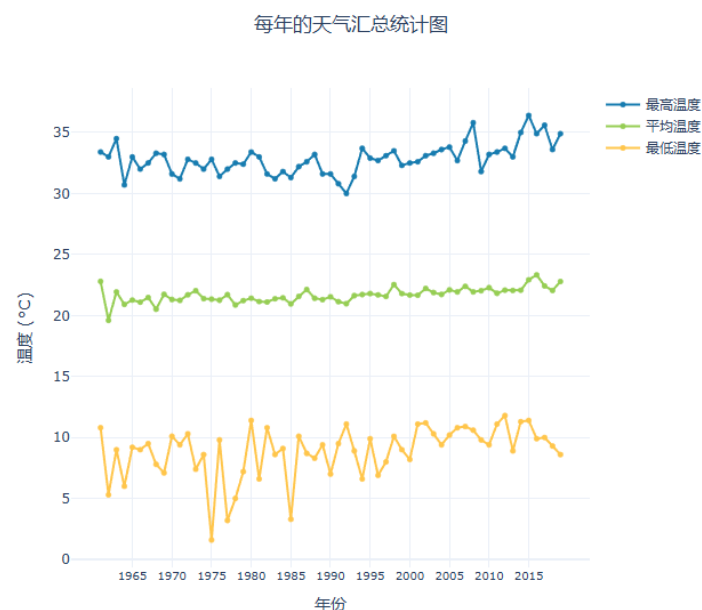
(3) 创建图表

- setTitle(“每年的天气汇总统计图”)：设置图表的标题。
- setLegend(): 设置图例。
- addXAxis(“年份”，x)：添加 X 轴，X 轴的标签是“年份”，X 轴数据来自数组 x。
- addYAxis(): 设置 Y 轴，虽然没有直接设置 Y 轴的标签和数据，但会根据数据自动调整。
- addSeries(): 添加三条数据系列，分别表示“最高温度”、“平均温度”和“最低温度”，它们的数值分别来自数组 y1、y2 和 y3。

5.6.2 具体代码

```
private static void runStep6() throws IOException { 1个用法
    List<WeatherWritable> weathers = gerWeathers(new Path(PATHS.get("step3_output")));
    Number[] y1 = new Number[weathers.size()];
    Number[] y2 = new Number[weathers.size()];
    Number[] y3 = new Number[weathers.size()];
    String[] x = new String[weathers.size()];
    for (int i = 0; i < weathers.size(); i++) {
        y1[i] = weathers.get(i).getMaxTemperature();
        y2[i] = weathers.get(i).getAvgTemperature();
        y3[i] = weathers.get(i).getMinTemperature();
        x[i] = weathers.get(i).getDate();
    }
    Line line = new Line()
        .setTitle("每年的天气汇总统计图")
        .setLegend()
        .addXAxis( name: "年份", x)
        .addYAxis()
        .addSeries( name: "最高温度", y1)
        .addSeries( name: "平均温度", y2)
        .addSeries( name: "最低温度", y3);
    Engine engine = new Engine();
    String file = "step3.html";
    engine.render(file, line);
    Runtime.getRuntime().exec( command: "explorer " + file);
}
```

5.6.3 运行效果



5.7 查询指定年份每月天气数据

5.7.1 功能分析

(1) Mapper 数据映射

- 目标年份：在 setup 方法中，从 context 中获取目标年份，并存储到 targetYear 变量中。这确保了 Mapper 在运行时知道需要过滤的数据年份。
- 数据格式检查：对于每一行输入数据，使用 line.split(“\t”) 将数据按制表符(\t) 分割成数组。If (fields.length != 6) 确保数据的格式正确，至少包含 6 个字段。
- 年份过滤：通过 date.substring(date.lastIndexOf(“/”) + 1) 提取日期中的年份，并与目标年份进行匹配。如果当前行的年份不等于目标年份，则跳过这条记录。
- 月份提取：通过 date.substring(0, 7) 提取年月部分（即 yyyy-MM），用于按月进行分组。
- 数据输出：对于符合条件的数据，创建一个 WeatherWritable 对象，并通过 context.write(new Text(code + “_” + month), w) 输出。这里的 Text(code + “_” + month) 作为 Map 阶段的键，WeatherWritable 作为值。

(2) Reducer：数据归约

- 输入数据：key 是 Text 类型，格式为 “天气站_年月”（例如 “Station1_2022-03”）。Values 是该键下的所有 WeatherWritable 对象，这些对象包含了该月份的温度和降水量等数据。
- 降水量、温度的聚合：遍历 values，累加降水量、最大温度、最小温度、和温度总和，求得该月的汇总数据。
- 计算平均温度：w.setAvgTemperature(w.getAvgTemperature() / count) 计算每月的平均温度。
- 最小温度初始化：使用 Double.MAX_VALUE 来初始化最小温度，以便后续取最小值。
- 格式化输出：使用 DecimalFormat 对输出的数字进行格式化。

(3) 配置和执行 MapReduce 作业

- job.getConfiguration().set(“targetYear”, targetYear)：将目标年份传递给 Mapper 类。
- job.setMapperClass(Step3Mapper.class)：设置 Mapper 类，负责从输入数据中提取指定年份的数据并按月份分组。
- job.setReducerClass(Step3Reducer.class)：设置 Reducer 类，负责聚合每月数据并格式化输出。
- 作业执行：job.waitForCompletion(true) 启动 MapReduce 作业，并等待执行完成。

5.7.2 具体代码

```
1 package brazil;
2
3 import org.apache.commons.lang3.StringUtils;
4 import org.apache.hadoop.fs.Path;
5 import org.apache.hadoop.io.LongWritable;
6 import org.apache.hadoop.io.NullWritable;
7 import org.apache.hadoop.io.Text;
8 import org.apache.hadoop.mapreduce.Job;
9 import org.apache.hadoop.mapreduce.Mapper;
10 import org.apache.hadoop.mapreduce.Reducer;
11 import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
12 import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
13 import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
14 import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
15 import java.io.IOException;
16 import java.text.DecimalFormat;
17
18 // 匿名类用法
19 public class Step7 {
20     public static class Step3Mapper extends Mapper<LongWritable, Text, Text, WeatherWritable> { 1个用法
21         private String targetYear; 2个用法
22
23         @Override
24         protected void setup(Context context) throws IOException, InterruptedException {
25             // 获取用户输入的年份
26             targetYear = context.getConfiguration().get("targetYear");
27         }
28
29         @Override
30         protected void map(LongWritable key, Text value, Mapper<LongWritable, Text, Text, WeatherWritable>.Context context) throws IOException, InterruptedException {
31             String line = value.toString();
32             if (StringUtils.isBlank(line)) {
33                 return;
34             }
35             String[] fields = line.split( regex: "\\t");
36             if (fields.length != 6) {
37                 return;
38             }
39             String code = fields[0];
40             String date = fields[1];
41             String year = date.substring( beginIndex: date.lastIndexOf("-") + 1);
42
43             // 如果该记录不属于目标年份，则跳过
44             if (!year.equals(targetYear)) {
45                 return;
46             }
47
48             String month = date.substring(0, 7);
49             double precipitation = Double.parseDouble(fields[2]);
50             double maxTemperature = Double.parseDouble(fields[3]);
51             double minTemperature = Double.parseDouble(fields[4]);
52             double avgTemperature = Double.parseDouble(fields[5]);
53
54             // 将天气数据封装成 WeatherWritable 对象
55             WeatherWritable w = new WeatherWritable(code, year, precipitation, maxTemperature, minTemperature, avgTemperature);
56             context.write(new Text( string: code + "_" + month), w);
57         }
58     }
59
60     private static class Step3Reducer extends Reducer<Text, WeatherWritable, Text, NullWritable> { 1个用法
61         @Override
62         protected void reduce(Text key, Iterable<WeatherWritable> values, Reducer<Text, WeatherWritable, Text, NullWritable>.Context context) throws IOException, InterruptedException {
63             String code = key.toString().split( regex: "_")[0];
64             String month = key.toString().split( regex: "_")[1];
65             WeatherWritable w = new WeatherWritable(code, month);
66             w.setMinTemperature(Double.MAX_VALUE);
67             int count = 0;
68
69             for (WeatherWritable v : values) {
70                 w.setPrecipitation(w.getPrecipitation() + v.getPrecipitation());
71                 w.setMaxTemperature(Math.max(w.getMaxTemperature(), v.getMaxTemperature()));
72                 w.setMinTemperature(Math.min(w.getMinTemperature(), v.getMinTemperature()));
73                 w.setAvgTemperature(w.getAvgTemperature() + v.getAvgTemperature());
74                 count++;
75             }
76
77             // 计算该月的平均温度
78             w.setAvgTemperature(w.getAvgTemperature() / count);
79         }
80     }
81 }
```

```

77         w.setAvgTemperature(w.getAvgTemperature() / count);
78
79         // 格式化输出（例如：保留两位小数）
80         DecimalFormat df = new DecimalFormat(pattern: "0.00");
81         String formattedOutput = String.format("%s %s %s %s %s %s",
82             w.getYear(), // 输出年份
83             month.substring(beginIndex: 5), // 输出月份（仅获取"MM"部分）
84             df.format(w.getMaxTemperature()), // 输出最高温度
85             df.format(w.getMinTemperature()), // 输出最低温度
86             df.format(w.getAvgTemperature()), // 输出平均温度
87             df.format(w.getPrecipitation())); // 输出降水量
88
89         // 输出格式化的字符串
90         context.write(new Text(formattedOutput), NullWritable.get());
91     }
92 }
93
94 public static void run(String input, String output, String targetYear) { 0个用法
95     try {
96         // 设置用户输入的年份
97         Job job = Job.getInstance(HadoopUtils.getConf(), jobName: "统计汇总指定年份每月天气");
98         job.getConfiguration().set("targetYear", targetYear); // 将年份传递给Mapper
99
100         Path inputPath = new Path(input);
101         Path outputPath = new Path(output);
102         HadoopUtils.delete(outputPath);
103
104         job.setInputFormatClass(TextInputFormat.class);
105         FileInputFormat.setInputPaths(job, inputPath);
106
107         job.setMapperClass(Step3Mapper.class);
108         job.setMapOutputKeyClass(Text.class);
109         job.setMapOutputValueClass(WeatherWritable.class);
110
111         job.setReducerClass(Step3Reducer.class);
112         job.setOutputKeyClass(Text.class); // 输出为文本
113         job.setOutputValueClass(NullWritable.class);
114
115         Path inputPath = new Path(input);
116         Path outputPath = new Path(output);
117         HadoopUtils.delete(outputPath);
118
119         job.setInputFormatClass(TextInputFormat.class);
120         FileInputFormat.setInputPaths(job, inputPath);
121
122         job.setMapperClass(Step3Mapper.class);
123         job.setMapOutputKeyClass(Text.class);
124         job.setMapOutputValueClass(WeatherWritable.class);
125
126         job.setReducerClass(Step3Reducer.class);
127         job.setOutputKeyClass(Text.class); // 输出为文本
128         job.setOutputValueClass(NullWritable.class);
129
130         job.setOutputFormatClass(TextOutputFormat.class);
131         FileOutputFormat.setOutputPath(job, outputPath);
132
133         boolean flag = job.waitForCompletion(verbose: true);
134         if (flag) {
135             System.out.println("统计指定年份每月的天气，运行结束");
136         } else {
137             System.out.println("MapReduce任务执行错误");
138         }
139     } catch (Exception e) {
140         e.printStackTrace();
141     }
142 }
143 }

```

5. 7. 3 运行效果

请输入查询年份（格式：yyyy）：**2002**

2002 年的每月天气统计：

年份	月份	最高温度	最低温度	平均温度	降水量
2002	1	30.1	17.2	22.43870967741936	208.3
2002	2	29.7	16.1	22.30535714285714	233.0
2002	3	29.9	16.8	22.94032258064516	133.8
2002	4	28.7	14.9	22.211666666666662	90.1
2002	5	29.5	12.3	21.298387096774192	18.200000000000003
2002	6	26.6	11.8	19.665000000000003	0.0
2002	7	29.0	11.6	20.56451612903226	16.5
2002	8	30.5	14.2	21.78548387096774	17.8
2002	9	31.1	11.2	22.395	67.8
2002	10	32.8	18.1	24.93870967741936	57.5
2002	11	30.7	15.7	23.008333333333333	175.1
2002	12	33.1	15.7	22.985483870967744	371.4

5.8 每年降雨量可视化

5.8.1 功能分析

(1) 读取数据

- getWeathers: 从路径 step3_output 中读取天气数据，返回一个列表。
WeatherWritable 是自定义的类，可能包含每条天气记录的字段，如日期、降水量、温度等。
- weathers.size(): 获取天气数据列表的大小，代表天气记录的数量。
- precipitationData: 用来存储每条记录的降水量数据。
- years: 用来存储每条记录对应的年份。

(2) 创建数组

- 循环遍历 weathers 列表，每次迭代获取一条 WeatherWritable 对象 w。
- w.getPrecipitation(): 从 WeatherWritable 对象中获取降水量，并存入 precipitationData 数组中。
- w.getDate().substring(0, 4): 从日期中提取年份部分，存入 years 数组中。

(3) 创建柱状图

- BarChart: 假设 BarChart 是一个自定义的类，用于绘制柱状图。
- setTitle("每年降水量汇总"): 设置图表的标题。
- setLegend(): 启用图例，显示柱状图系列的名称。
- addXAxis("年份", years): 设置 X 轴为年份，years 数组提供年份数据。
- addYAxis("降水量 (mm)": 设置 Y 轴为降水量，单位为毫米。
- addSeries("降水量", precipitationData): 添加一个系列，名称为“降水量”，数据来自 precipitationData 数组。

5.8.2 具体代码

```
private static void runStep7() throws IOException { 0个用法
    List<WeatherWritable> weathers = getWeathers(new Path(PATHS.get("step3_output")));

    Number[] precipitationData = new Number[weathers.size()];
    String[] years = new String[weathers.size()];

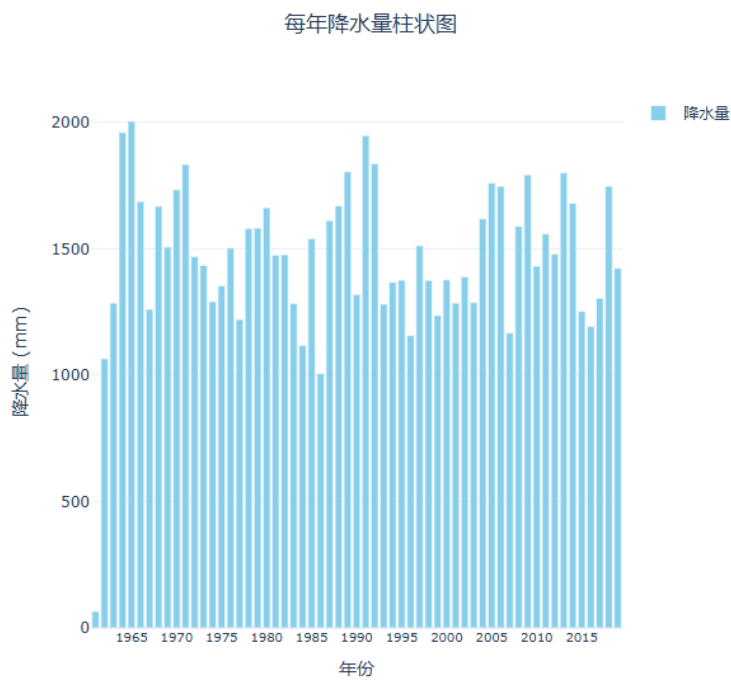
    for(int i = 0; i < weathers.size(); i++){
        WeatherWritable w = weathers.get(i);
        precipitationData[i] = w.getPrecipitation();
        years[i] = w.getDate().substring(0, 4);
    }

    BarChart barChart = new BarChart()
        .setTitle("每年降水量汇总")
        .setLegend()
        .addXAxis("年份", years)
        .addYAxis("降水量 (mm)")
        .addSeries("降水量", precipitationData);

    Engine engine = new Engine();
    String file = "Step3_BarChart.html";
    engine.render(file, barChart);

    try {
        Runtime.getRuntime().exec("explorer " + file);
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

5.8.3 运行效果



六、 数据进一步分析

6.1 基于 SARIMA 模型未来天气预测

6.1.1 SARIMA 模型

SARIMA 模型是对 ARIMA（自回归积分滑动平均模型）的扩展，主要用于解决季节性时间序列数据的问题。ARIMA 模型适用于非季节性数据，而 SARIMA 在此基础上增加了季节性部分，能够处理季节性周期性的变化。

SARIMA 模型的数学公式通常表示为以下形式的差分方程：

$$y_t = \mu + \sum_{i=1}^p \phi_i y_{t-i} + \sum_{i=1}^q \theta_i \epsilon_{t-i} + \sum_{i=1}^P \Phi_i y_{t-is} + \sum_{i=1}^Q \Theta_i \epsilon_{t-is} + \epsilon_t$$

y_t ：时间序列在时刻的值。

μ ：常数项（趋势项）。

ϕ_i ：自回归系数。

θ_i ：滑动平均系数。

ϵ_t ：随机误差项。

s ：季节性周期（12 个月）。

Φ_i ：季节性自回归系数。

Θ_i ：季节性滑动平均系数。

6.1.2 模型具体代码

```
1 import pandas as pd
2 import matplotlib.pyplot as plt
3 from statsmodels.tsa.statespace.sarimax import SARIMAX
4 from pandas.plotting import register_matplotlib_converters
5
6 register_matplotlib_converters()
7
8 # 读取数据
9 file_path = '/Users/wei/Desktop/monthly_weather_precipitation.csv' # 你的CSV文件路径
10 df = pd.read_csv(file_path)
11
12 # 合并年份和月份为日期
13 df['Date'] = pd.to_datetime(df[['Year', 'Month']].assign(DAY=1))
14
15 # 设置日期为索引
16 df.set_index(keys='Date', inplace=True)
17
18 # 提取需要预测的列
19 temperature = df['AvgTemp']
20 precipitation = df['DailyPrecipitation']
21
22 # 可视化温度和降水量的时间序列
23 plt.figure(figsize=(10, 6))
24 plt.subplot(*args: 2, 1, 1)
25 plt.plot(*args: temperature, Label='Average Temperature')
26 plt.title('Monthly Average Temperature Over Time')
27 plt.xlabel('Date')
28 plt.ylabel('Temperature (°C)')
29
30 plt.subplot(*args: 2, 1, 2)
31 plt.plot(*args: precipitation, Label='Monthly Precipitation', color='orange')
32 plt.title('Monthly Precipitation Over Time')
33 plt.xlabel('Date')
34
35 plt.ylabel('Precipitation (mm)')
36
37 plt.tight_layout()
38 plt.show()
39
40 # SARIMA模型预测函数
41 2个用法
42 def fit_sarima_model(series, order=(1, 1, 1), seasonal_order=(1, 1, 1, 12), steps=60):
43     model = SARIMAX(series, order=order, seasonal_order=seasonal_order)
44     model_fit = model.fit(dispatch=False)
45     forecast = model_fit.forecast(steps=steps)
46     return forecast
47
48 # 预测未来60个月的平均温度和降水量
49 temperature_forecast = fit_sarima_model(temperature, steps=24)
50 precipitation_forecast = fit_sarima_model(precipitation, steps=24)
51
52 # 生成未来60个月的时间范围, 从2020年1月开始
53 forecast_dates = pd.date_range(start='2020-01-01', periods=24, freq='M')
54
55 # 创建一个包含预测结果的 DataFrame
56 forecast_df = pd.DataFrame({
57     'Month': forecast_dates.strftime('%Y-%m'),
58     'Forecasted Temperature (°C)': temperature_forecast,
59     'Forecasted Precipitation (mm)': precipitation_forecast
60 })
```

```

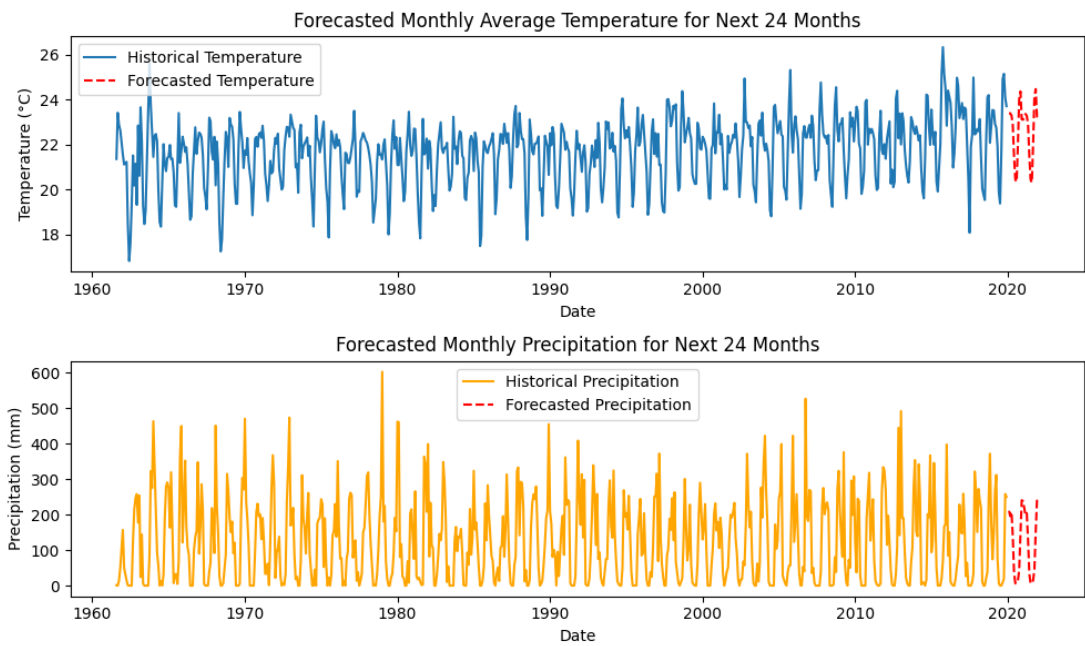
60 # 输出预测结果
61 print("未来24个月的预测结果:")
62 print(forecast_df)
63
64 # 可视化预测结果
65 plt.figure(figsize=(10, 6))
66
67 # 预测温度
68 plt.subplot(2, 1, 1)
69 plt.plot(*args: temperature, label='Historical Temperature')
70 plt.plot(*args: forecast_dates, temperature_forecast, label='Forecasted Temperature', linestyle='--', color='red')
71 plt.title('Forecasted Monthly Average Temperature for Next 24 Months')
72 plt.xlabel('Date')
73 plt.ylabel('Temperature (°C)')
74 plt.legend()
75
76 # 预测降水量
77 plt.subplot(2, 1, 2)
78 plt.plot(*args: precipitation, label='Historical Precipitation', color='orange')
79 plt.plot(*args: forecast_dates, precipitation_forecast, label='Forecasted Precipitation', linestyle='--', color='r')
80 plt.title('Forecasted Monthly Precipitation for Next 24 Months')
81 plt.xlabel('Date')
82 plt.ylabel('Precipitation (mm)')
83 plt.legend()
84
85 plt.tight_layout()
86 plt.show()

```

6.1.3 模型结果与可视化

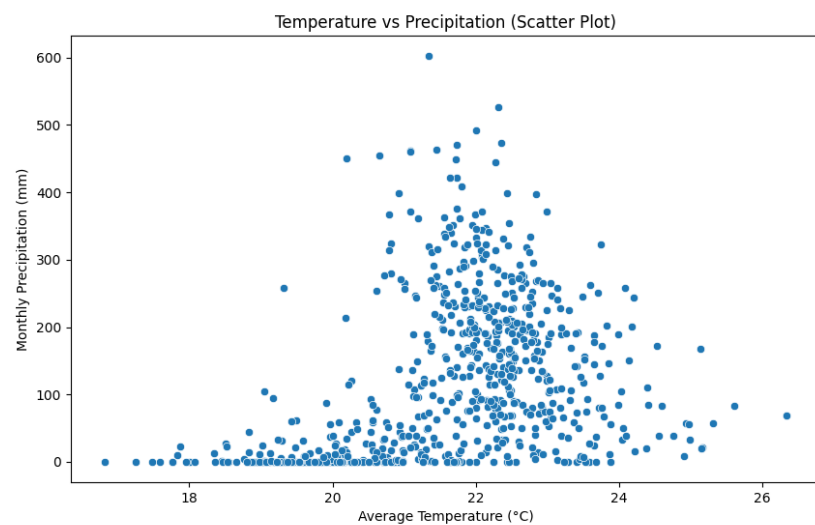
未来24个月的预测结果:

	Month	Forecasted Temperature (°C)	Forecasted Precipitation (mm)
699	2020-01	23.387627	210.754627
700	2020-02	23.394277	196.014232
701	2020-03	23.292274	207.531791
702	2020-04	22.883861	139.858320
703	2020-05	21.434312	35.492473
704	2020-06	20.446200	6.327262
705	2020-07	20.338273	6.297467
706	2020-08	22.029448	14.511199
707	2020-09	24.017338	43.568677
708	2020-10	24.366408	144.910959
709	2020-11	23.121412	240.481697
710	2020-12	23.106939	238.198632
711	2021-01	23.223155	217.970239
712	2021-02	23.362639	198.671631
713	2021-03	23.274675	203.952527
714	2021-04	22.902997	129.238161
715	2021-05	21.507957	33.000910
716	2021-06	20.442339	6.313438
717	2021-07	20.305754	6.708834
718	2021-08	22.045068	15.430903
719	2021-09	24.127562	45.748280
720	2021-10	24.467834	152.675539
721	2021-11	23.236898	239.425326
722	2021-12	23.194907	237.471320



6.2 月平均温度与降雨量相关性分析

6.2.1 散点图可视化



这个散点图展示了平均温度 (°C) 与月降水量 (mm) 之间的关系，数据呈现出大致的钟形分布，表明在中间温度范围 (约 20-22°C) 时降水量较高，而在较低或较高温度时降水量相对较少。温度范围主要集中在 18°C 到 26°C 之间，而大多数降水量分布在 200 mm 以内，极端降水量 (如 600 mm 以上) 为少数异常值，可能反映了极端天气或特殊条件。

6.2.2 具体代码

```

1  import pandas as pd
2  from scipy.stats import pearsonr
3  import matplotlib.pyplot as plt
4  import seaborn as sns
5
6  # 读取数据
7  file_path = '/Users/wei/Desktop/monthly_weather_precipitation.csv' # 你的CSV文件路径
8  df = pd.read_csv(file_path)
9
10 # 选择相关的列
11 temperature = df['AvgTemp']
12 precipitation = df['DailyPrecipitation']
13
14 # 计算皮尔逊相关系数
15 correlation, _ = pearsonr(temperature, precipitation)
16
17 # 输出相关系数
18 print(f"皮尔逊相关系数: {correlation:.2f}")
19
20 # 可视化部分: 绘制散点图
21 plt.figure(figsize=(10, 6))
22 sns.scatterplot(x=temperature, y=precipitation)
23 plt.title('Temperature vs Precipitation (Scatter Plot)')
24 plt.xlabel('Average Temperature (°C)')
25 plt.ylabel('Monthly Precipitation (mm)')
26 plt.show()
27
28 # 可视化部分: 绘制温度和降水量的相关性热图
29 # 计算相关系数矩阵
30 corr_matrix = df[['AvgTemp', 'DailyPrecipitation']].corr()
31
32 # 使用seaborn绘制热图
33 plt.figure(figsize=(6, 4))
34 sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f', vmin=-1, vmax=1)
35 plt.title('Correlation Heatmap of Temperature and Precipitation')
36 plt.show()
37

```

6.2.3 皮尔逊相关性分析

使用皮尔逊相关性分析进一步探讨降水量和平均温度之间的关系，皮尔逊相关系数是衡量两个变量之间线性关系强度和方向的统计量。皮尔逊相关系数是衡量两个变量之间线性关系强度和方向的标准化的统计量，取值范围在-1 到 +1 之间。值越接近 1 或 -1，表示线性关系越强。

$$r = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^n (x_i - \bar{x})^2 \sum_{i=1}^n (y_i - \bar{y})^2}}$$

r : 皮尔逊相关系数

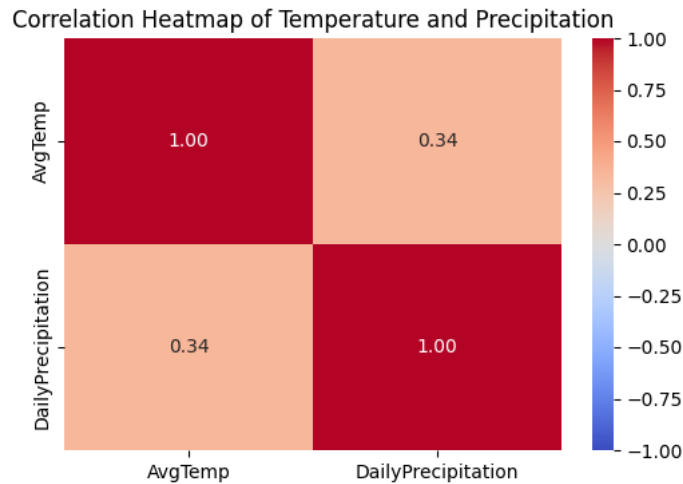
x_i : 第 i 个样本点的 x 值

y_i : 第 i 个样本点的 y 值

$\bar{x}$: x 的均值

$\bar{y}$: y 的均值

n : 数据点的总数



根据计算可得，降水量和平均温度之间皮尔逊相关系数为 0.34，表示平均温度和降水量之间有轻微的正相关关系。即随着温度的增加，降水量倾向于增加，但这种关系并不强烈。在一定范围内，温度的升高可能会稍微增加降水量，但对与所有温度并不适用，这个结果与散点图中结果相似，即在 16° C – 23° C 的范围内，平均温度的提升会带来降水量的提升，但对于 23° C – 26° C 的情况，这个结果并不适用。

七、 总结

项目基于 Hadoop 生态系统，搭建了搭建了包含多个节点的 Hadoop 分布式计算集群，成功部署了 HDFS（Hadoop 分布式文件系统）和 YARN 资源管理框架，确保了系统的高效存储、资源调度与任务管理能力。通过合理配置和参数调整，实现了节点间的高效通信与协作。利用 HDFS 实现了大规模数据的分布式存储，解决了单机系统存储空间受限的问题，同时通过副本机制增强了数据的安全性和容错能力。同时，设计并实现了一系列 MapReduce 任务，用于执行包括数据清洗、统计分析、排序与聚合等在内的多种处理操作。这些任务成功验证了 Hadoop 在处理结构化和非结构化数据方面的强大能力。

通过参与本项目，深入掌握了 Hadoop 生态系统的原理和应用技能，提升了对分布式计算核心技术的理解，为进一步技术创新奠定了扎实的基础。