第5章

拉勾招聘岗位数据分析

本章将编写一个较为完整的拉勾招聘智能数据分析案例。前面已学习了Hadoop相关的基础知识，本章将开启Hadoop数据分析的实战应用。通过该案例的设计与实现，读者可以掌握数据分析的完整开发流程。

【学习目标】

- 掌握拉勾招聘智能数据分析案例的系统架构设计。
- 学习拉勾招聘智能数据分析案例的开发流程规划与实施。
- 理解并掌握数据清洗与预处理的过程。
- 能够进行数据分析，并运用可视化技术展示分析结果。
- 掌握拉勾招聘智能数据分析案例的打包、部署及运行过程。

5.1 项目背景

进入21世纪，随着数字化、网络化、智能化技术的不断普及渗透，基于互联网的岗位招聘求职已经成为连接求职者和用人企业的重要桥梁，随着时间的推移，招聘网站必然会积累海量的求职与招聘信息数据。这些信息中不仅蕴含着以往人才市场的岗位需求状况、大学热门专业及求职者的主流愿景等各种与就业相关的信息，而且还隐藏着人才需求变化的规律。通过大数据技术分析研究这些招聘求职数据不仅能够反映以往就业形势的变化，而且可以预测未来的就业趋势，对政府人力资源管理部门、招聘企业、高等院校、求职者四方都有重要的意义。

5.2 分析任务

本案例从拉勾招聘网站获取岗位招聘数据，并对其进行数据挖掘，主要围绕以下几个维度进行数据分析。

- 招聘最高薪资前10的城市。
- 不同工作经验招聘的需求情况。
- 不同岗位的平均薪资水平统计。
- 不同学历和不同工作经验的薪资对比分析。
- 发布招聘岗位数量最多的前10家公司统计。
- 2023年招聘岗位数量在不同时间节点的变化分析。

通过以上分析,为求职者提供最适宜自身的招聘岗位或求职时间,为招聘企业提供所需岗位的最佳招聘时间和人才特点。最终达到提升企业招聘效率以及求职的成功率的目的。

5.3 系统设计

5.3.1 系统架构设计

该系统采用分层架构设计,主要包括数据资源层、数据处理层和业务应用层,如图 5-1 所示。

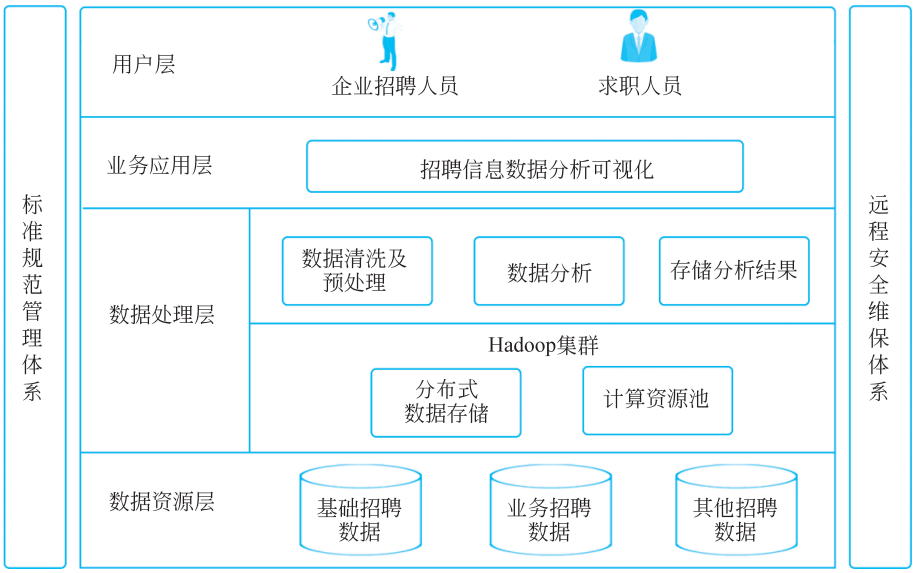

图 5-1 拉勾招聘系统架构设计

1. 数据资源层

数据资源层是整个系统的基础,主要是从拉勾招聘网站获取招聘信息,并将获取的数据保存为 CSV 文件,为后续的数据处理和分析提供基础。

2. 数据处理层

数据处理层主要负责对数据源进行清洗、预处理和分析操作。

数据清洗与预处理采用 MapReduce 框架对 HDFS 中的原始数据进行处理,包括去除重复数据、处理缺失值等操作。

数据分析通过 MapReduce 框架对不同维度的招聘信息进行离线分析,如职位分布、薪资水平、需求趋势等。分析结果存储在 MySQL 中,为业务应用层提供数据支持。

3. 业务应用层

业务应用层采用 Spring Boot 和 MyBatis 技术栈构建 Web 项目,通过引入 ECharts 库,将数据分析结果用柱状图、折线图、饼图等不同图标展示,用户可以直观、明了地查看数据分析结果。

5.3.2　开发流程设计

本系统的开发流程设计主要包括数据采集、数据存储、清洗及预处理、数据分析、分析结果可视化等关键步骤，如图 5-2 所示。

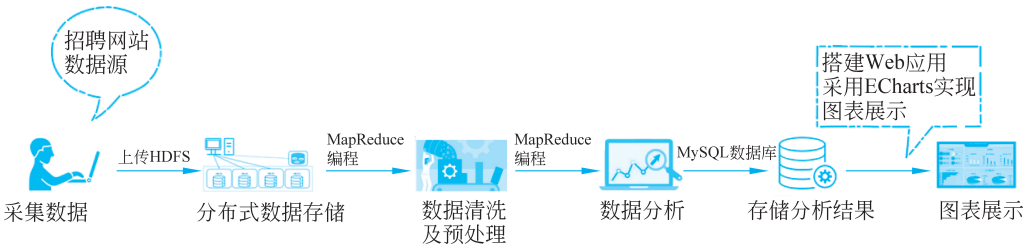

图 5-2　拉勾招聘岗位数据分析流程图

1. 数据采集

采集拉勾招聘网站的招聘数据，数据源主要包括职位名称、公司名称、职位类型、薪资范围、学历要求、工作经验、职位描述、城市、发布日期等信息。

2. 数据上传至 HDFS

将采集到的招聘数据源上传到 HDFS 进行存储。

3. 数据清洗与预处理

采用 MapReduce 实现数据清洗，处理空值、重复记录以及异常数据（如职位描述信息不完整的记录）。删除或修正不符合规范的数据，确保数据的准确性和一致性。清洗及预处理后的数据保存到 HDFS，以便后续的数据分析。

4. MapReduce 分析数据

采用 MapReduce 对清洗及预处理后的招聘数据进行热门职位、行业薪资水平、学历与薪资的关系等不同维度的分析，分析后的结果写入 MySQL 数据库。

5. Web 项目开发及可视化展示

采用 Spring Boot 和 MyBatis 等企业级框架开发 Web 应用程序，引入 ECharts 库，将岗位招聘数据的分析结果通过折线图、柱状图、热力图等图表展示出来。

6. 项目部署运行

数据处理及分析模块被打包并部署到 Hadoop 集群中，在集群环境下提交并执行作业完成数据处理任务。可视化展示模块则打包部署到 Hadoop 集群主节点的 Tomcat 服务器的 webapps 目录下运行。

5.3.3　系统开发实验环境

本案例所使用的软件环境及版本信息如下。

- 操作系统：Windows 10、Windows 11、macOS、Ubuntu、CentOS 等。
- 开发工具：IntelliJ IDEA。
- 虚拟机：VMware Workstation 16 Pro。
- 大数据开发平台：Hadoop（3.1.2 及以上版本）。
- Web 服务器：Tomcat。

- J2EE 企业级框架：Spring Boot(2.0.4 及以上版本)、MyBatis。
- 关系数据库：MySQL(8.0)。

5.4 数据集介绍

该数据集来源于拉勾招聘网站，作为中国领先的互联网招聘平台，拉勾招聘汇集了大量企业的招聘信息和求职者的简历。数据集包含了公司名称、职位名称、薪资待遇、工作地点、要求的技能和经验等多个字段。该案例的数据源字段见表 5-1。

表 5-1 拉勾岗位招聘数据表字段

字 段	字 段 说 明
category	技术种类
bigcategory	工作种类
positionId	职位 ID
companyId	公司 ID
companyFullName	公司全名
companySize	公司规模
financeStage	公司发展阶段
positionType	工作类别
createTime	创造时间(时间格式：YYYYMMDD HH:mm)
city	城市
district	区域
salary	薪资(单位：k)
workYear	工作年数(单位：年)
jobNature	工作性质
education	教育程度
create_at	发布时间(时间格式：YYYYMMDD HH:mm)
salary_lowest	最低薪资(单位：k)
salary_highest	最高薪资(单位：k)
salary_avg	平均薪资(单位：k)

5.5 数据源存储

本案例的数据源文件为 job.csv，该文件位于本书配套资源第 5 章源码的 data 目录下。将该文件上传至 HDFS 存储。

(1) 启动 Hadoop 集群，命令如下：

```
[root@node01 ~]#start-all.sh
[root@node01 ~]#start-yarn.sh
```

（2）Hadoop 集群服务启动成功后，将 job.csv 文件上传至 HDFS，命令如下：

```
[root@node01 ~]#hdfs dfs -mkdir /recruitmentdata
[root@node01 data]#hdfs dfs -put /opt/data/job.csv /recruitmentdata
[root@node01 data]#hdfs dfs -ls /recruitmentdata
Found 1 items
-rw-r--r--   3 root supergroup     6836691 2024-09-15 16:50 /recruitmentdata/
job.csv
```

5.6　数据清洗及预处理


岗位招聘数据清洗及预处理

数据清洗阶段，通过比对和去重删除重复的招聘信息，确保数据记录的唯一性。同时，针对数据中的缺失值，根据数据特征和业务逻辑进行合理的填充或删除，以保证数据的完整性。数据预处理阶段，将薪资字段分离为最高和最低薪资两个字段。

构建 Maven 项目，采用 MapReduce 实现拉勾岗位招聘数据清洗、预处理及分析操作，实现过程如下。

1. 创建 maven 项目

使用 IntelliJ IDEA 创建一个名为 JobRecruitmentExample 的 Maven 项目，实现项目规范化管理。之后，在 JobRecruitmentExample 中添加 JobRecruitmentClean 子模块，实现拉勾岗位招聘数据分析的数据清洗及预处理操作。创建后的项目结构如图 5-3 所示。

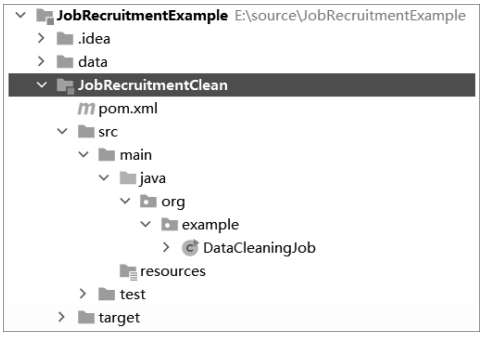

图 5-3　JobRecruitmentClean 数据清洗及预处理模块项目结构

2. 添加项目依赖

拉勾岗位招聘数据分析的父工程主要包含了数据清洗及预处理、数据分析和可视化三个子模块。由于使用不同的工具和版本可能会影响程序的正常运行，为确保系统的稳定性，通常在父工程中声明子模块共享的版本信息。

修改 JobRecruitmentExample 父工程中的 pom.xml 文件，添加本案例三个模块共用的依赖包的版本属性，如代码清单 5-1 所示。

代码清单 5-1　修改 pom.xml 文件中的属性

```
<properties>
    <maven.compiler.source>8</maven.compiler.source>
    <maven.compiler.target>8</maven.compiler.target>
    <hadoop.version>3.2.3</hadoop.version>
    <mysql.version>8.0.20</mysql.version>
</properties>
```

在 JobRecruitmentExample/pom.xml 文件中添加 Hadoop 核心依赖包，如代码清单 5-2 所示。

代码清单 5-2　修改 pom.xml 文件添加 Hadoop 依赖包

```
<dependencies>
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>${mysql.version}</version>
    </dependency>
    <dependency>
        <groupId>org.apache.hadoop</groupId>
        <artifactId>hadoop-common</artifactId>
        <version>${hadoop.version}</version>
    </dependency>
    <dependency>
        <groupId>org.apache.hadoop</groupId>
        <artifactId>hadoop-hdfs</artifactId>
        <version>${hadoop.version}</version>
    </dependency>
    <dependency>
        <groupId>org.apache.hadoop</groupId>
        <artifactId>hadoop-client</artifactId>
        <version>${hadoop.version}</version>
    </dependency>
    </dependencies>
```

3. 数据清洗及预处理

设计思路：数据清洗及预处理分为两个作业完成，本案例采用链式作业执行，第一个作业完成数据清洗；第二个作业实现数据预处理，将数据源中的薪资字段(salary)值拆分为最低薪资和最高薪资两个字段。

(1) 数据清洗主要处理数据源中包含双引号和逗号的特征值，将双引号内的逗号替换为指定的符号“##”，并通过去除双引号和替换逗号，确保数据能够正确解析和处理。

在 JobRecruitmentClean 子模块的 org/example 包下创建 DataCleaningMapper 类实现数据清洗，该类的关键代码如代码清单 5-3 所示。

代码清单 5-3　DataCleaningMapper 类的关键代码

```
public static class DataCleaningMapper extends Mapper < LongWritable, Text,
LongWritable, Text> {
    //定义一个正则表达式模式,用于匹配双引号内的内容
    private static final Pattern pattern = Pattern.compile("\"(.*?)\"");
    protected void map(LongWritable key, Text value, Context context) throws
IOException, InterruptedException {
        String line = value.toString();
        Matcher matcher = pattern.matcher(line);
        StringBuffer cleanedLine = new StringBuffer();
        while (matcher.find()) {
            String group = matcher.group(1);
```

```
            //将逗号替换为##
            String cleanedGroup = group.replace(",", "##");
            //将处理后的内容重新替换回原位置
            matcher.appendReplacement(cleanedLine, "\"" + cleanedGroup + "\"");
        }
        matcher.appendTail(cleanedLine);
        //删除数据源中的双引号
        String resultLine = cleanedLine.toString().replaceAll("\"", "");
        context.write(key, new Text(resultLine));
    }
```

（2）数据预处理主要实现薪资字段的拆分。从数据源中提取薪资字段，将薪资字段中的列值拆分为最低薪资和最高薪资，并去除非数字字符。拆分后的最低薪资和最高薪资追加到原始数据源的最后两列。在 JobRecruitmentClean 子模块的 org/example 包下创建 DataPreprocessingMapper 类实现数据预处理，关键代码如代码清单 5-4 所示。

代码清单 5-4　数据预处理 Mapper 类

```
public static class DataPreprocessingMapper extends Mapper<LongWritable, Text,
NullWritable, Text> {
@Override
    protected void map(LongWritable key, Text value, Context context) throws
IOException, InterruptedException {
        String line = value.toString();
        String[] fields = line.split(",");
        //提取薪资特征列
        String salary = fields[14];
        String[] salaryRange = salary.split("-");
        String minSalary = salaryRange[0].replaceAll("[^\\d]", "");
        String maxSalary = salaryRange.length > 1 ? salaryRange[1].replaceAll("
[^\\d]", "") : minSalary;
        //拆分后的最低薪资和最高薪资追加到数据行尾部
        String finalLine = line + "," + minSalary + "," + maxSalary;
        context.write(NullWritable.get(), new Text(finalLine));
    }
}
```

（3）编写 main 方法配置并启动数据清洗及预处理的 MapReduce 作业。该方法包含两个作业，首先是数据清洗作业，通过 DataCleaningMapper 类实现数据清洗操作，指定输入和输出路径。其次是数据预处理作业，通过 DataPreprocessingMapper 类进行薪资拆分等预处理操作，设置输入和输出路径并执行作业。main 方法的关键代码如代码清单 5-5 所示。

代码清单 5-5　main 方法的关键代码

```
public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    //数据清洗
    Job job1 = Job.getInstance(conf, "数据清洗");
```

```
        job1.setJarByClass(DataCleaningJob.class);
        job1.setMapperClass(DataCleaningMapper.class);
        job1.setNumReduceTasks(0);
        job1.setOutputKeyClass(LongWritable.class);
        job1.setOutputValueClass(Text.class);
        FileInputFormat.addInputPath(job1, new Path(args[0]));
        FileOutputFormat.setOutputPath(job1, new Path(args[1]));
        boolean job1Success = job1.waitForCompletion(true);
        if (!job1Success) {
            System.exit(1);
        }
        //数据预处理
        Job job2 = Job.getInstance(conf, "薪资拆分");
        job2.setJarByClass(DataCleaningJob.class);
        job2.setMapperClass(DataPreprocessingMapper.class);
        job2.setNumReduceTasks(0);
        job2.setOutputKeyClass(NullWritable.class);
        job2.setOutputValueClass(Text.class);
        FileInputFormat.addInputPath(job2, new Path(args[1]));
        FileOutputFormat.setOutputPath(job2, new Path(args[2]));
        System.exit(job2.waitForCompletion(true) ? 0 : 1);
    }
```

(4) 打包部署运行。

采用 Maven 编译、打包 JobRecruitmentClean 模块,生成名为 JobRecruitmentClean-1.0-SNAPSHOT.jar 的文件。将 JobRecruitmentClean-1.0-SNAPSHOT.jar 文件上传至主节点 node01 的/opt/jar 目录中。

上传完成后,执行以下命令实现拉勾岗位招聘数据清洗及预处理的作业:

```
[root@node01 data]# hadoop jar /opt/jar/JobRecruitmentClean-1.0-SNAPSHOT.jar
org.example.DataCleaningJob /recruitmentdata/job.csv /recruitmentdata/temp_
output /recruitmentdata/final_output
```

作业执行完成后,数据清洗及预处理的结果存入 HDFS 的/recruitmentdata/final_output 目录中。

招聘薪资待遇最高的前10个城市

5.7 数据分析

在 JobRecruitmentExample 中添加 JobRecruitmentProcess 子模块,采用 MapReduce 实现对拉勾岗位招聘数据的不同维度分析,分析后的结果写入 MySQL 数据库。

1. 编写工具类

在 JobRecruitmentProcess 子模块的 org.example.util 包下新建 DatabaseUtils 工具类,用于实现将数据分析后的结果写入数据库。该类的关键代码如代码清单 5-6 所示。

代码清单 5-6 DatabaseUtils 类的关键代码

```
public class DatabaseUtils {
```

```
    private static final String DRIVER = "com.mysql.cj.jdbc.Driver";
    private static final String JDBCURL = "jdbc:mysql://192.168.198.101:3306/
recruitdb?useUnicode=true&characterEncoding=utf8&useSSL=false&serverTimezone=
UTC";
    private static final String USERNAME = "root";
    private static final String PASSWORD = "123456";
    static {
        try {
            //加载 MySQL 驱动
            Class.forName(DRIVER);
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
    }
    //获取数据库连接
    public static Connection getConnection() throws SQLException {
        return DriverManager.getConnection(JDBCURL, USERNAME, PASSWORD);
    }
    //执行批量插入操作的通用方法
     public static void bulkInsertToDatabase(String hdfsPath, String sql,
Configuration conf, ObjectParser parser) {
        try (Connection connection = getConnection()) {
            //获取 HDFS 文件系统
            FileSystem fs = FileSystem.get(conf);
            RemoteIterator<LocatedFileStatus> fileStatusListIterator = fs.
listFiles(new Path(hdfsPath), false);
            while (fileStatusListIterator.hasNext()) {
                LocatedFileStatus fileStatus = fileStatusListIterator.next();
                try (BufferedReader br = new BufferedReader(new InputStreamReader
(fs.open(fileStatus.getPath())));
                     PreparedStatement statement = connection.prepareStatement(sql)) {
                     String line;
                     while ((line = br.readLine()) != null) {
                         Object[] parameters = parser.parse(line);
                         for (int i = 0; i < parameters.length; i++) {
                             statement.setObject(i + 1, parameters[i]);
                         }
                         statement.addBatch();
                     }
                     //执行批量插入
                     statement.executeBatch();
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    //检查表是否存在,不存在则创建
    public static void checkAndCreateTable(String tableName, String createTableSQL) {
        try (Connection connection = getConnection();
```

```
                Statement stmt = connection.createStatement()) {
                DatabaseMetaData metaData = connection.getMetaData();
                ResultSet resultSet = metaData.getTables(null, null, tableName, null);
                if (!resultSet.next()) {
                    //表不存在,创建表
                    stmt.execute(createTableSQL);
                }
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
        //解析行数据的接口
        public interface ObjectParser {
            Object[] parse(String line);
        }
    }
```

2. 招聘薪资待遇最高的前 10 个城市

设计思路：在 Map 阶段获取数据源中的最高薪资以及城市信息，组成键值对输出，在 Reduce 阶段采用 TreeMap 实现按薪资进行排序并输出平均薪资前 10 的城市。之后，在 reduce 任务完成后，在 cleanup 中汇总输出排序结果。最后，将分析后的结果写入 MySQL 数据库。

在 JobRecruitmentExample/JobRecruitmentProcess/java/org.example.recruit 包下创建一个名为 Top10CitiesByAverageSalary 的类，实现招聘最高薪资待遇前 10 的城市。

(1) 在 Top10CitiesByAverageSalary 类中编写静态内部类 SalaryMapper，该类的关键代码如代码清单 5-7 所示。

代码清单 5-7　SalaryMapper 类的关键代码

```
public static class SalaryMapper extends Mapper<LongWritable, Text, Text,
DoubleWritable> {
    @Override
    protected void map(LongWritable key, Text value, Context context) throws
IOException, InterruptedException {
        String[] fields = value.toString().split(",");
        if (fields.length < 20) return;   //Ensure data has enough fields
        String city = fields[12];
        double minSalary = Double.parseDouble(fields[fields.length - 2]);
        double maxSalary = Double.parseDouble(fields[fields.length - 1]);
        double avgSalary = (minSalary + maxSalary) / 2;
        context.write(new Text(city), new DoubleWritable(avgSalary));
    }
};
```

(2) 在 Top10CitiesByAverageSalary 类中编写静态内部类 AverageSalaryReducer，关键代码如代码清单 5-8 所示。