深圳職業技術學院
SHENZHEN POLYTECHNIC

深圳职业技术学院 2022 届毕业生

毕 业 设 计 （论 文）

题　　目： 基于 HADOOP 的租房数据清洗与分析

姓　　名： 林泽平

学　　院： 人工智能学院

专　　业： 大数据技术与应用

班　　级： 19 大数据 2

指导教师： 邓雪峰

深圳职业技术学院

2022 年 6 月

摘 要

互联网的快速发展给我们生活带来便捷的同时，也给互联网技术带来了更多的挑战。本文主要讲的是大数据在实际应用上的示例，由于大数据技术应用场景的特殊性，传统的计算平台不适用于大数据开发，所以需要一套适合大数据开发的平台——Hadoop 及其生态圈。所以在进行大数据处理开发前需要先搭建一套适用于大数据的平台，包括计算组件与存储组件等，用于数据的计算或存储。平台搭建完成后进行适当的测试后就可进行大数据开发，根据需求完成对应的处理，如数据清洗、数据分析等。大数据技术在大数据应用场景上计算快速，数据来源广泛，通过从大量不同类型数据中挖掘出对未来趋势与模式预测分析有价值的数据，通过大数据分析发现新规律与新知识。本项目中采用 Hadoop 大数据平台，对用户提供的租房数据，使用 Spark 和 MapReduce 技术进行数据清洗与分析，分析数据中体现出来的特点，挖掘隐藏在数据背后的价值，获取用户所需的信息。

关键字：HADOOP；数据清洗与分析；大数据技术；计算；存储

Abstract

The rapid development of Internet brings convenience to our life, but also brings more challenges to Internet technology. This paper mainly focuses on the practical application of big data. Due to the particularity of the application scenarios of big data technology, traditional computing platforms are not suitable for big data development, so a suitable platform for big data development -- Hadoop and its ecosystem are needed. Therefore, before developing big data processing, it is necessary to build a set of platform suitable for big data, including computing components and storage components for data calculation or storage. After the completion of the platform construction and proper testing, big data development can be carried out, and corresponding processing can be completed according to the requirements, such as data cleaning, data analysis, etc. Big data technology has fast calculation and wide data sources in big data application scenarios. Valuable data for future trend and pattern prediction and analysis can be mined from a large number of different types of data, and new laws and new knowledge can be found through big data analysis. In this project, the Hadoop big data platform is adopted to clean and analyze the rental data provided by users, and Spark and MapReduce technologies are used to analyze the characteristics reflected in the data, dig out the value hidden behind the data, and obtain the information

required by users.

Keywords: HADOOP; Data cleaning and analysis; Big data technology; Calculation; storage

目 录

第一章 选题背景

1.1、 课题概述

本项目使用 Hadoop 生态圈为大数据技术发展提供一套稳定高效，适合大数据开发应用场景的基础平台，根据项目需求搭建对应大数据组件，再进行相应的调试确认组件搭建完毕并可以正常工作。

完成平台搭建后分析项目需求，利用搭建完毕的大数据平台对数据进行对应计算或存储操作，编写相关程序进行数据清洗与分析等操作，将结果保存至存储组件上，可以使用 sql 语言执行相关查询展示数据。

1.2、 技术背景

计算机和信息技术的迅猛发展和各行业大规模的普及应用，行业应用系统的规模迅速扩大，其所产生的数据呈指数型的增长，动辄达到数百 TB 级甚至数十至数百 PB 级规模的大数据已经远远超出了传统的计算技术和信息系统的处理能力，从而促进了大数据技术的产生及快速发展。

Hadoop 是一个由 Apache 基金会所开发的分布式系统基础架构，Hadoop 是目前应用最为广泛的分布式大数据处理框架，其具备可靠、高效、可伸缩等特点。Hadoop 的核心组件是 HDFS、MapReduce。随着处理任务不同，各种组件相继出现，丰富 Hadoop 生态圈。

1.3、 涉及技术

本项目选择 Hadoop 生态圈里部分组件，进行平台搭建。涉及 Hadoop、Spark、Hive、Zookeeper、MySQL 以及 Flume 等组件；在进行数据清洗程序时，使用 Java 语言与 Scala 语言编写程序，开发环境涉及 Xshell，Xftp，IntelliJ IDEA 以及 VMware Workstation 等软件。

第二章 用户需求分析

2.1、 用户需求

用户的需求为手里有一份租房数据信息，想挖掘数据的价值，于是找到了我，想实现快速可靠的数据处理与数据存储，并且能快速挖掘出数据的价值。综合用户的需求，决定使用大数据技术，具体分两步进行：第一步：系统平台搭建；第二步：数据清洗与分析。

2.2、 系统平台的搭建

在开始搭建大数据平台之前，需要先准备三台虚拟机，三台虚拟机都需要完成配置网络服务，创建操作集群用户，关闭防火墙等前置工作。为了使集群具有更好的健壮性，再配置集群时间同步服务，避免因集群时间不同步而导致出现错误等问题。

虚拟机的基础配置完成后就可以进行大数据系统平台的搭建，本项目涉及搭建组件如下：

- Hadoop-HA
- Hive
- Spark
- HBase
- MySQL
- Sqoop

- Kafka
- Zookeeper
- Flume

完成以上组件搭建同时，为了便于集群的群启与群关功能，可以编写 shell 脚本使得对集群的操作更加便捷，例如：

（1）启动与关闭 Hadoop 集群；

（2）启动与关闭 Zookeeper 集群；

（3）启动与关闭 Kafka 集群；

平台搭建完成与脚本编写完成后仍需进行必要的测试，若测试结果与预期一致，则可以进行下一项内容。否则仍需调试，直至测试通过。

2.3、 数据清洗与分析

本项目数据来源于深圳租房网站，房屋位置都在深圳室内，由于数据是通过数据爬虫获取，不可避免会出现一些缺失值、异常值与数据重复等情况。为了使数据更加规范，我们需要对数据进行数据清洗与分析操作：

程序 1：去除重复值以及空行

程序 2：填补缺失值

程序 3：数据字段规范化处理

进行数据清洗时可以编写 MapReduce 程序或者 Spark 程序并将程序打包上传至大数据平台，执行程序对数据进行相关计算，得到清洗

完的数据，可将数据上传保存到 HDFS 或数据库中。

将处理完成后的数据导入 Hive 数据库，使用 HQL 语言分析数据特点，如若遇到特殊需求，可以用户自定义函数简化查询操作。根据需要编写 HQL 语句，得出对应结果并以表格形式展示数据，分析数据具备什么特点，以此得到结论。

第三章 详细设计

3.1、 系统环境

（1）平台搭建部分组件版本

3-1 组件版本

名称	版本
Hadoop	hadoop-2.8.3
JDK	jdk-1.8
HBase	hbase-1.3.1
Spark	spark-2.4.7
Hive	hive-1.1.0
Flume	flume-1.9.0
Kafka	kafka-2.11-0.11
Zookeeper	zookeeper-3.5.7
Sqoop	sqoop-1.4.7
Scala	Scala-2.11.12

（2）数据清洗部分开发环境版本

3-2 开发环境

名称	版本
IntelliJ IDEA	IntelliJ IDEA 2019.1

VMware Workstation	VMware Workstation 16 Pro
XShell	Xshell5
XFtp	XFtp5
Maven	Maven 3.3.9

3.2、 平台搭建

修改三台虚拟机主机名分别为：hadoop100、hadoop101、hadoop102，并创建 hadoop 用于集群的安装和使用，提前安装 jdk 并配置环境变量。同时配置自定义脚本的路径为“~/bin”，用于保存自定义群起群关脚本。

（1） 集群时间同步

使用 ntp 工具实现集群时间同步，先安装 ntp 工具后，配置/etc/ntp.conf 文件，安装实际情况进行配置，使用 crontab 命令设置定时任务，每小时执行一次时间同步，每次其他机器都去与 hadoop100 的时间同步。

（2） Zookeeper 搭建

Zookeeper 是一个为分布式应用提供一致性服务的软件，简称集群润滑剂，提供的功能包括：配置维护、域名服务、分布式同步、组服务等。

在搭建 Hadoop-HA、Kafka 以及 HBase 之前需要先搭建 Zookeeper，由于 Zookeeper 的特性一般部署在奇数台机器上，所以至少在三台虚拟机上部署。

- 核心配置：

```
dataDir=/opt/module/zookeeper-3.5.7/zkData


server.1=hadoop100:2888:3888
server.2=hadoop101:2888:3888
server.3=hadoop102:2888:3888
```

由于 Zookeeper 集群需要到每台集群上开启与关闭，会大大增加开启与关闭集群的时间，所以编写一个 Shell 脚本用于 Zookeeper 的开启、关闭以及查看集群状态。

- ZK 脚本：

```
#!/bin/bash
case $1 in
"start"){
    for i in hadoop100 hadoop101 hadoop102
    do
        echo ---------- zookeeper $i 启动 ----------
        ssh $i "source /etc/profile;/opt/module/zookeeper-3.5.7/bin/zkServer.sh start"
    done
}
;;
"stop"){
```

```
        for i in hadoop100 hadoop101    hadoop102
        do
            echo ---------- zookeeper $i  停止  ----------
            ssh                         $i                          "source
/etc/profile;/opt/module/zookeeper-3.5.7/bin/zkServer.sh stop"
        done
}
;;
"status"){
        for i in hadoop100 hadoop101 hadoop102
        do
            echo ---------- zookeeper $i  状态  ----------
            ssh                         $i                          "source
/etc/profile;/opt/module/zookeeper-3.5.7/bin/zkServer.sh status"
        done
}
;;
esac
```

（3） Hadoop-HA 搭建

在全分布模式下，一个 NameNode 可能会因为压力过大导致崩溃，由于 NameNode 只有一个，一旦崩溃集群也会随之崩溃，容错较低，为了避免此问题，本项目选择 Hadoop-HA，部署两个 NameNode 以及

ResourceManager，分别部署在 hadoop100 与 hadoop101，依靠 Zookeeper 进行监控 NameNode 的工作状态，避免存在两个 NameNode 同时工作的脑裂现象。同时配置历史服务器，用于执行程序出差时方便查看错误日志以定位错误。

- 核心配置：

```
<!--两个 NameNode，分别是 nn1，nn2 -->
<property>
<name>dfs.ha.namenodes.mycluster</name>
<value>nn1,nn2</value>
</property>
<!—hadoop100 的 RPC 通信地址 -->
<property>
<name>dfs.namenode.rpc-address.mycluster.nn1</name>
<value>hadoop100:8020</value>
</property>
<!—hadoop101 的 RPC 通信地址 -->
<property>
<name>dfs.namenode.rpc-address.mycluster.nn2</name>
<value>hadoop101:8020</value>
</property>
<!-- hadoop100 的 http 通信地址 -->
<property>
```

```
<name>dfs.namenode.http-address.mycluster.nn1</name>
<value>hadoop100:50070</value>
</property>
<!-- hadoop101 的 http 通信地址 -->
<property>
<name>dfs.namenode.http-address.mycluster.nn2</name>
<value>hadoop101:50070</value>
</property>
```

（4） HBase 搭建

HBase 是一个分布式的、面向列的开源数据库，该技术来源于 Fay Chang 所撰写的 Google 论文“Bigtable：一个结构化数据的分布式存储系统”

- 核心配置：

```
<!-- 配置 Zookeeper -->
<property>
    <name>hbase.zookeeper.quorum</name>
    <value>hadoop100,hadoop101,hadoop102</value>
</property>
<property>
    <name>hbase.zookeeper.property.dataDir</name>
    <value>/opt/module/zookeeper-3.5.7/zkData</value>
</property>
```

（5） Kafka 搭建

Kafka 是一种高吞吐量的分布式发布订阅消息系统，它可以处理消费者在网站中的所有动作流数据。

- 核心配置：

```
## broker.id 每台机器都不同，即 broker.id 唯一
broker.id=0
log.dirs=/opt/module/kafka/data
zookeeper.connect=hadoop100:2181,hadoop101:2181,hadoop102:2181
```

与 Zookeeper 类似，Kafka 的启动与关闭都需要在每台机器上执行，同样编写一个群启群关脚本。

- kafka 脚本：

```
#!/bin/bash
case $1 in
"start"){
    for i in hadoop100 hadoop101 hadoop102
    do
        echo "================ [ $i start kafka] ================"
        ssh $i "source /etc/profile;/opt/module/kafka/bin/kafka-server-start.sh -daemon /opt/module/kafka/config/server.properties"
    done
```

```
}
;;
"stop"){
    for i in hadoop100 hadoop101 hadoop102
    do
        echo "================ [ $i stop kafka] ================"
        ssh $i "/opt/module/kafka/bin/kafka-server-stop.sh"
    done
}
;;
esac
```

（6） Hive 搭建

Hive 是基于 Hadoop 的一个数据仓库工具，用来进行数据提取、转化、加载，这是一种可以存储、查询和分析存储在 Hadoop 中的大规模数据的机制。hive 数据仓库工具能将结构化的数据文件映射为一张数据库表，并提供 SQL 查询功能，能将 SQL 语句转变成 MapReduce 任务来执行

Hive 需要把元数据存储在其他关系型数据库上，通常我们都会选择 MySQL 数据库，所以在搭建 Hive 之前需要搭建 MySQL，然后再进行 Hive 的配置。

- 核心配置：

```
</configuration>
<property>
<name>javax.jdo.option.ConnectionURL</name>
<value>jdbc:mysql://hadoop100:3306/hive?createDatabaseIfNotExist=tru
e</value>
</property>
<property>
<name>javax.jdo.option.ConnectionDriverName</name>
<value>com.mysql.jdbc.Driver</value>
</property>
<property>
<name>javax.jdo.option.ConnectionUserName</name>
<value>root</value>
</property>
<property>
<name>javax.jdo.option.ConnectionPassword</name>
<value>123456</value>
</property>
<property>
<name>hive.metastore.schema.verification</name>
<value>false</value>
</property>
```

```
</configuration>
```

（7） Spark

Spark 启用了内存分布数据集，除了能够提供交互式查询外，它还可以优化迭代工作负载。计算速度快与 MapReduce，约为 MapReduce 的 100 倍左右。Spark 是在 Scala 语言中实现的，它将 Scala 用作其应用程序框架。与 Hadoop 不同，Spark 和 scala 能够紧密集成，其中的 Scala 可以像操作本地集合对象一样轻松地操作分布式数据集。

Spark 部署模式较多，此处选择本地模式安装与 yarn 模式，后续数据清洗时编写的 Spark 程序也可以使用本地模式进行调试。

- 核心配置

```
export JAVA_HOME=/opt/module/java-1.8
SPARK_MASTER_HOST=hadoop100
SPARK_MASTER_PORT=7077
```

（8） Flume 搭建

Flume 是 Cloudera 提供的一个高可用的，高可靠的，分布式的海量日志采集、聚合和传输的系统，Flume 支持在日志系统中定制各类数据发送方，用于收集数据；同时，Flume 提供对数据进行简单处理，并写到各种数据接受方（可定制）的能力。

Flume 的搭建也较为简单，解压压缩包后删除 lib 包下的 guava-11.0.2.jar 文件，避免与 Hadoop 的 jar 包版本不一致出现不兼容问题。

（9） Sqoop 搭建

Sqoop 是一款开源的工具，主要用于在 Hadoop(Hive)与传统的数据库间进行数据的传递，可以将一个关系型数据库，例如 ：MySQL ,Oracle ,Postgres 等中的数据导进到 Hadoop 的 HDFS 中，也可以将 HDFS 的数据导进到关系型数据库中。

- 核心配置：

```
export HADOOP_COMMON_HOME=/opt/module/hadoop-2.8.3
export HADOOP_MAPRED_HOME=/opt/module/hadoop-2.8.3
export HBASE_HOME=/opt/module/hbase-1.3.1
export HIVE_HOME=/opt/module/hive-1.1.0
export ZOOCFGDIR=/opt/module/zookeeper-3.5.7
export ZOOKEEPER_HOME=/opt/module/zookeeper-3.5.7
```

（10）在平台搭建完成后，需要查看进程情况，在集群中切换机器查看进程效率低下，可以通过自定义脚本来完成查询集群所有进程。

- 查看进程脚本：

```
#!/bin/bash
for i in hadoop100 hadoop101 hadoop102
do
    echo ---------- [ $i ] ----------
    ssh $i "source /etc/profile;jps"
done
```

3.3、 数据清洗

由于数据存在一些空行、缺失值、异常值与数据重复等情况，需要对数据进行数据清洗。选择的数据为租房数据，数据属性如下：

3-3 数据属性

租房方式	rent_way
名称	name
区域	region
地点	district
面积	area
朝向	orientation
房屋类型	house_type
价格	price
楼层	floor
代理人	agent

通过观察租房数据可知，该租房数据一共与 10 个字段，涵盖了租房的一些信息，其中有部分数据重复，需要对数据执行去重处理；字段租房方式、名称两项都有缺失值，对于缺失值可以选择去除或者替换处理，以下为数据处理方式：

（1） 使用 Java 语言编写 MapReduce 程序，将数据作为 Map 阶段输出的 Key，同时判断是否存在空行，若存在则将空行去除；在 shuffle 阶段按 Key 排序后传输值 Reduce 阶段，在 Reduce 阶段将 Key 去重后，使用计数器统计租房方式的数量，并将计数结果打印至控制台上，最后将程序运行结果保存至 HDFS 上。

- Map 核心代码：

```
    Text k = new Text();
    @Override
    protected void map(LongWritable key, Text value, Context context)
            throws IOException, InterruptedException {
        String data = value.toString();
        if ("".equals(data)) {
            return ;
        }
        k.set(data);
        context.write(k, NullWritable.get());
    }
```

- Reduce 核心代码：

```
    @Override
    protected void reduce(Text key, Iterable<NullWritable> values,
Context context)
            throws IOException, InterruptedException {
        String[] datas = key.toString().split(",");
        if ("整租".equals(datas[0])) {
            context.getCounter("RentWay","整租").increment(1);
        } else if ("合租".equals(datas[0])) {
            context.getCounter("RentWay","合租").increment(1);
        }
```

```
            context.write(NullWritable.get(), key);
        }
```

（2） 使用 Java 语言编写 MapReduce 程序，根据第一步在控制台打印的计数器结果，选择次数最多的租房方式用于填充空值，并将名称为空的字段填充为“独栋出租”，最后将结果保存至 HDFS 上。

- Map 核心代码：

```
    @Override
    protected void reduce(Text key, Iterable<NullWritable> values, Context context)
                throws IOException, InterruptedException {
        String[] datas = key.toString().split(",");
        if ("整租".equals(datas[0])) {
            context.getCounter("RentWay","整租").increment(1);
        } else if ("合租".equals(datas[0])) {
            context.getCounter("RentWay","合租").increment(1);
        }
        context.write(NullWritable.get(), key);
    }
```

（3） 使用 Scala 语言编写 Spark 程序，将面积的单位“㎡”与价格的单位“元/月”去除，再将楼层中的层数分离，增加一个“floor_num”字段表示层数，程序执行结果保存至

HDFS 上。

- 核心代码：

```
val dataFrame = spark.sparkContext.textFile("./data/result1")
    .map(line => {
        val values = line.split(",")
        val area = values(4).replace("㎡","")
        val price = values(7).replace("元/月", "")
        val floors = values(8).split("/")
        val floor = floors(0)
        val floor_num = floors(1).replace("层", "")
        Data(values(0), values(1), values(2), values(3), area, values(5),
values(6),
            price, floor, floor_num,values(9))
    }).toDF()
dataFrame.show(10)
dataFrame.write.format("csv")
    .option("header",false)
    .option("sep","\t")
    .save("./data/result2")
```

3.4、 数据分析

进行数据清洗后，数据较为规范化，将数据保存后，在 Hive 中

按表 3-3 新建外部表 room，将结果数据导入到 Hive 中，编写 HQL 语句进行数据分析与查询。

- 建表语句：

```
create external table if not exists room (
rent_way String,
name String,
region String,
district String,
area Double,
orientation String,
house_type String,
price Double,
floor String,
floor_num Int,
agent String)
row format delimited fields terminated by '\t';
```

HQL 语句与 SQL 语句类似，使用 HQL 语句可以提高开发效率，且 Hive 的底层计算框架默认是 MapReduce，非常适用于大数据应用场景，对数据进行一些聚合统计、查询计算等操作，获取我们需要的结果。

第四章 系统运行

4.1、 启动集群

在使用集群时需要先启动集群，查看控制台打印的日志消息是否有报错，若集群正常开启，将数据上传到预定位置。

- 查看集群进程：

```
[hadoop@hadoop100 ~]$ jpsall
--------------- [ hadoop100 ] -------------------
3136 ApplicationHistoryServer
3398 Jps
1751 QuorumPeerMain
2616 ResourceManager
2073 DataNode
1964 NameNode
2271 JournalNode
2463 DFSZKFailoverController
2735 NodeManager
--------------- [ hadoop101 ] -------------------
1504 QuorumPeerMain
2480 Jps
1909 DFSZKFailoverController
1686 DataNode
2247 ResourceManager
1784 JournalNode
2026 NodeManager
1611 NameNode
--------------- [ hadoop102 ] -------------------
1504 QuorumPeerMain
2049 Jps
1603 DataNode
1701 JournalNode
1835 NodeManager
[hadoop@hadoop100 ~]$
```

集群正常开启之后，就可以在集群上执行自己打包的程序，调用对应的计算组件，进行数据清洗操作。

4.2、 数据清洗

将数据清洗程序打包并上传到集群上，调用相关命令执行程序，运行程序后控制台结果如下：

（1） 去除空行，重复值。由控制台可知“整租”的次数最多，所以将使用“整租”来填充空值（详细代码见附件程序 1）。

- 运行结果

```
                Combine input records=0
                Combine output records=0
                Reduce input groups=2612
                Reduce shuffle bytes=288275
                Reduce input records=2683
                Reduce output records=2612
                Spilled Records=5366
                Shuffled Maps =1
                Failed Shuffles=0
                Merged Map outputs=1
                GC time elapsed (ms)=147
                CPU time spent (ms)=1720
                Physical memory (bytes) snapshot=336478208
                Virtual memory (bytes) snapshot=4161343488
                Total committed heap usage (bytes)=219676672
        RentWay
                合租=561
                整租=1999
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=285586
        File Output Format Counters
                Bytes Written=275530
[hadoop@hadoop100 ~]$
```

（2） 根据上一步得出的结果进一步清洗，填充字段“rent_way”和“name”的空缺值（详细代码见附件程序 2）。

- 运行结果

```
[hadoop@hadoop100 ~]$
[hadoop@hadoop100 ~]$ hadoop jar ~/jars/mr-1.0-SNAPSHOT.jar com.szpt.mr.fill.Driver hdfs://mycluster/Linzp/output/room_result_01
hdfs://mycluster/Linzp/output/room_result_02
22/04/08 03:30:55 INFO input.FileInputFormat: Total input files to process : 1
22/04/08 03:30:55 INFO mapreduce.JobSubmitter: number of splits:1
22/04/08 03:30:55 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1649401647657_0002
22/04/08 03:30:56 INFO impl.YarnClientImpl: Submitted application application_1649401647657_0002
22/04/08 03:30:56 INFO mapreduce.Job: The url to track the job: http://hadoop100:8088/proxy/application_1649401647657_0002/
22/04/08 03:30:56 INFO mapreduce.Job: Running job: job_1649401647657_0002
22/04/08 03:31:03 INFO mapreduce.Job: Job job_1649401647657_0002 running in uber mode : false
22/04/08 03:31:03 INFO mapreduce.Job:  map 0% reduce 0%
22/04/08 03:31:09 INFO mapreduce.Job:  map 100% reduce 0%
22/04/08 03:31:09 INFO mapreduce.Job: Job job_1649401647657_0002 completed successfully
22/04/08 03:31:09 INFO mapreduce.Job: Counters: 30
        File System Counters
                FILE: Number of bytes read=0
                FILE: Number of bytes written=160040
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=275652
                HDFS: Number of bytes written=276466
                HDFS: Number of read operations=5
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=2
        Job Counters
                Launched map tasks=1
                Data-local map tasks=1
                Total time spent by all maps in occupied slots (ms)=3763
                Total time spent by all reduces in occupied slots (ms)=0
```

（3） 对上一步的结果再进一步清洗，去除字段“area”和“price”的单位，再将“floor”拆分成两个字段，“floor”以及“floor_num”（详细代码见附件程序 3）。

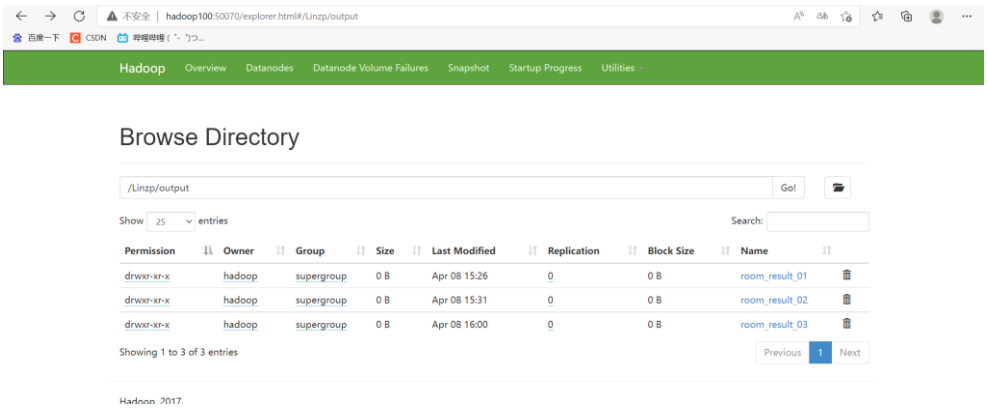
- 运行结果

```
)
22/04/08 04:00:40 INFO BlockManagerInfo: Added broadcast_0_piece0 in memory on 192.168.66.101:41800 (size: 22.9 KB, free: 413.9 M
B)
22/04/08 04:00:42 INFO TaskSetManager: Finished task 0.0 in stage 0.0 (TID 0) in 6157 ms on 192.168.66.101 (executor 1) (1/1)
22/04/08 04:00:42 INFO DAGScheduler: ResultStage 0 (show at FillNull.scala:28) finished in 6.345 s
22/04/08 04:00:42 INFO TaskSchedulerImpl: Removed TaskSet 0.0, whose tasks have all completed, from pool
22/04/08 04:00:42 INFO DAGScheduler: Job 0 finished: show at FillNull.scala:28, took 6.439829 s
+--------+--------+------+--------+------+-----------+----------+-----+------+---------+--------+
|rent_way|    name|region|district|  area|orientation|house_type|price|  floor|floor_num|   agent|
+--------+--------+------+--------+------+-----------+----------+-----+------+---------+--------+
|    整租|独栋招租|光明区|    公明|111.25|         南| 5室3厅2卫| 5800|高楼层|        6|  郭维月|
|    整租|独栋招租|南山区|    前海| 28.20|         东| 1室0厅1卫| 6500|低楼层|       27|    周紫|
|    整租|独栋招租|宝安区|宝安中心|148.00|         东| 3室2厅2卫|12000|低楼层|       99|  潘凯嘉|
|    整租|独栋招租|宝安区|宝安中心|148.48|         东| 4室2厅2卫|19500|高楼层|       57|    周紫|
|    整租|独栋招租|宝安区|宝安中心| 52.00|         东| 1室1厅1卫| 8690|低楼层|       40|    周紫|
|    整租|独栋招租|宝安区|    沙井| 42.19|         南| 1室1厅1卫| 3000|低楼层|       30|  叶正茂|
|    整租|独栋招租|宝安区|    石岩| 75.90|         西| 3室2厅1卫| 5300|高楼层|       45|    邓斐|
|    整租|独栋招租|宝安区|    石岩| 76.00|         南| 3室2厅1卫| 5500|中楼层|       45|    邓斐|
|    整租|独栋招租|宝安区|    石岩| 90.64|         南| 3室2厅2卫| 5050|中楼层|       45|马扎来叶|
|    整租|独栋招租|福田区|    景田|127.00|         南| 3室2厅2卫|22000|中楼层|       43|  韦相行|
+--------+--------+------+--------+------+-----------+----------+-----+------+---------+--------+
only showing top 10 rows

22/04/08 04:00:42 INFO BlockManagerInfo: Removed broadcast_1_piece0 on 192.168.66.101:41800 in memory (size: 7.2 KB, free: 413.9
MB)
22/04/08 04:00:42 INFO BlockManagerInfo: Removed broadcast_1_piece0 on hadoop100:36532 in memory (size: 7.2 KB, free: 413.9 MB)
22/04/08 04:00:42 INFO FileOutputCommitter: File Output Committer Algorithm version is 1
22/04/08 04:00:42 INFO SQLHadoopMapReduceCommitProtocol: Using output committer class org.apache.hadoop.mapreduce.lib.output.File
OutputCommitter
22/04/08 04:00:42 INFO SparkContext: Starting job: save at FillNull.scala:32
```

在编写程序时可以在开发环境中进行适当调试，确保在集群上执行时避免出现错误，提高集群的健壮性与稳定性，同时对代码进行优化，提高代码的健壮性与容错性，提高程序执行效率，规范化代码，便于查看与理解，在程序出现错误时也可以方便查找错误。

执行完数据清洗的三个程序后，运行结果都保存到 HDFS 如下图所示：



4.3、 数据分析

使用 Hive 进行数据分析与处理，编写适当的 HQL 语句，按照需求查询数据，并对数据进行分析。

（1）按区域查询平均面积与平均价格。

- 查询结果：

```
region  avg_area        avg_price
光明区  79.94   4223.08
南山区  60.49   9131.38
坪山区  80.44   3282.57
大鹏新区        67.0    2500.0
宝安区  78.9    5620.42
盐田区  86.17   6200.0
福田区  60.8    7633.84
罗湖区  57.5    5363.47
龙华区  91.37   7968.71
龙岗区  181.79  3995.7
Time taken: 25.099 seconds, Fetched: 10 row(s)
```

（2）按照房屋类型查询平均面积与平均价格

- 查询结果：

```
house_type      avg_area        avg_price
1室0厅0卫       86.13   8902.95
1室0厅1卫       34.58   3640.59
1室1厅0卫       108.81  9200.0
1室1厅1卫       43.41   4442.08
1室1厅2卫       60.0    8875.0
1室2厅1卫       55.39   7855.56
1室2厅2卫       65.93   7699.33
2室0厅1卫       48.71   8650.0
2室1厅0卫       69.71   7500.0
2室1厅1卫       61.87   5256.88
2室1厅2卫       63.68   9688.76
2室2厅1卫       73.05   5688.25
2室2厅2卫       86.28   7938.23
2室2厅3卫       39.37   4926.67
3室0厅1卫       63.75   6100.0
3室0厅3卫       30.0    3000.0
3室1厅1卫       58.52   4757.69
3室1厅2卫       82.28   6617.71
3室2厅1卫       488.99  6158.0
3室2厅2卫       103.65  8251.22
3室2厅3卫       143.93  28262.5
4室0厅0卫       11.0    2100.0
4室0厅1卫       67.0    5250.0
4室0厅2卫       15.0    2200.0
4室0厅4卫       30.0    3666.67
4室1厅1卫       14.0    2103.52
4室1厅2卫       14.01   2360.56
```

（3）按照小区名称查询每平方米的价格多少

- HQL 代码：

```
select
    name, round(avg(avg_price)) avg_price
from (select name,price/area avg_price from room)t1 group by name;
```

- 查询结果：

麟恒中心广场一期	72.0
麟恒中心广场二期	63.0
黄埔雅苑一期	121.0
黄埔雅苑三期	148.0
黄埔雅苑二期	113.0
黄埔雅苑四期	147.0
黄金大厦	77.0
鼎太风华一期	90.0
鼎太风华七期	240.0
鼎太风华二期	229.0
龙丽园	79.0
龙光玖云著	52.0
龙光玖誉府	58.0
龙光玖钻	131.0
龙光玖龙台	62.0
龙兴大厦	64.0
龙华花半里花园	65.0
龙华金茂府	173.0
龙城华府	52.0
龙威花园一期	117.0

（4）查询房屋平均楼层高度

- 查询结果

```
avg_floor
23.0
Time taken: 23.022 seconds, Fetched: 1 row(s)
```

- 分析：

数据清洗完成后对租房数据进行分析，对比数据各个方面的属性，将按照需求编写的 HQL 语句展示的结果进行理解与分析，互相对比。

通过上述几个查询结果分析来看，深圳租房总体价格较高，房屋面积大小适中，内部价格因房屋地理位置印象而差距较大，平均楼层较高，出行应都具有电梯，较为方便，采光问题也较好解决。

若是希望节省金钱，且对地理位置要求不高，可以选择一些房屋价格较为便宜的地区或者是一些较为便宜的小区，如坪山区、大鹏新区等价格便宜的区域；或者选择面积小一些，房屋类型较小，但位置较好的房屋，例如：光明区、龙岗区等。

如若为了有较好的生活质量，则可以选择一些不仅出行发便、交通便捷的地区，而且房屋类型好、面积大的房子，可供的选择较多，如福田区、南山区。或者考虑其他因素，也可以从此大方向出发扩散寻找合适的房屋。

第五章 关键技术

数据清洗完整代码：

（1） 程序 1 实现代码如下：

Mapper 类：对数据进行判断，如果数据中包含空行，则建空行剔除

具体如下图 5-1-1 所示：

5-1-1 程序 1 中 Mapper 类

```
package com.szpt.mr.room;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
import java.io.IOException;
public class MyMapper extends Mapper<LongWritable, Text, Text, NullWritable> {
    Text k = new Text();
    @Override
    protected void map(LongWritable key, Text value, Context context)
            throws IOException, InterruptedException {
        String data = value.toString();
        if ("".equals(data)) {
            return ;
```

```
            }
            k.set(data);
            context.write(k, NullWritable.get());
        }
}
```

Reducer 类：判断租房数据中第一个字段——租房方式是什么，并使用计数器统计各自分别出现的次数，最终在终端打印出统计结果。具体如下图 5-1-2 所示：

5-1-2 程序 1 中 Reducer 类

```
package com.szpt.mr.room;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;
import java.io.IOException;
public class MyReducer extends Reducer<Text, NullWritable, NullWritable, Text> {
    @Override
    protected void reduce(Text key, Iterable<NullWritable> values, Context context)
            throws IOException, InterruptedException {
        String[] datas = key.toString().split(",");
```

```
            if ("整租".equals(datas[0])) {
                context.getCounter("RentWay","整租").increment(1);
            } else if ("合租".equals(datas[0])) {
                context.getCounter("RentWay","合租").increment(1);
            }
            context.write(NullWritable.get(), key);
        }
}
```

Driver 类：程序 1 的驱动类，即程序入口，实现 Tool 接口，调用自定义的 Mapper 和 Reducer 类，执行程序得出结果。具体如图 5-1-3 所示：

5-1-3 程序 1 中 Driver 类

```
package com.szpt.mr.room;
import org.apache.hadoop.conf.Configuration;
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.output.FileOutputFormat;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;
```

```
public class Driver implements Tool {
    private Configuration configuration;
    public static void main(String[] args) {
        Configuration conf = new Configuration();
        try {
            ToolRunner.run(conf,new Driver(),args);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    public int run(String[] strings) throws Exception {
        Job job = Job.getInstance(this.configuration);
        job.setJarByClass(Driver.class);
        job.setMapperClass(MyMapper.class);
        job.setReducerClass(MyReducer.class);
        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(NullWritable.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(NullWritable.class);
        FileInputFormat.setInputPaths(job, new Path(strings[0]));
        FileOutputFormat.setOutputPath(job, new Path(strings[1]));
        boolean flag = job.waitForCompletion(true);
```

```
        return flag ? 0 : 1;
    }
    public void setConf(Configuration configuration) {
        this.configuration = configuration;
    }
    public Configuration getConf() {
        return this.configuration;
    }
}
```

（2） 程序 2 实现代码如下：

Mapper 类：自定义 fill()方法将第一个字段——租房方式中的空缺值填充并返回填充完的字符串，在 map()方法中调用 fill()方法，实现空缺值的填充。具体如图 5-2-1 所示：

5-2-1 程序 2 中 Mapper 类

```
package com.szpt.mr.fill;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
import java.io.IOException;
public class MyMapper extends Mapper<LongWritable, Text, Text,
```

```
NullWritable> {
    Text k = new Text();
    @Override
    protected void map(LongWritable key, Text value, Context context)
            throws IOException, InterruptedException {
        k.set(fill(value));
        context.write(k, NullWritable.get());
    }
    private String fill(Text value) {
        StringBuffer buff = new StringBuffer();
        String[] values = value.toString().split(",");
        if ("".equals(values[0])) {
            values[0] = "整租";
        }
        if ("".equals(values[1])) {
            values[1] = "独栋招租";
        }
        for (int i = 0; i < values.length - 1; i++) {
            buff.append(values[i] + ",");
        }
        buff.append(values[values.length - 1]);
        return buff.toString();
```

```
    }
}
```

Driver 类：由于填充空缺值在 Map 阶段就可以完成，无需再浪费资源去开启 Reduce 阶段，所以不执行 Reduce 阶段，只执行 Map 阶段，在驱动类中将 ReduceTask 动态设置为 0，关闭 Reduce；再调用自定义的 Mapper 类执行程序，得到结果。具体如图 5-2-2 所示：

5-2-2 程序 2 中 Driver 类

```
package com.szpt.mr.fill;
import org.apache.hadoop.conf.Configuration;
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.output.FileOutputFormat;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;
public class Driver implements Tool {
    private Configuration configuration;
    public static void main(String[] args) {
```

```
        Configuration conf = new Configuration();
        try {
            ToolRunner.run(conf,new Driver(),args);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    public int run(String[] strings) throws Exception {
        Job job = Job.getInstance(this.configuration);
        job.setJarByClass(Driver.class);
        job.setNumReduceTasks(0);
        job.setMapperClass(MyMapper.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(NullWritable.class);
        FileInputFormat.setInputPaths(job, new Path(strings[0]));
        FileOutputFormat.setOutputPath(job, new Path(strings[1]));
        boolean flag = job.waitForCompletion(true);
        return flag ? 0 : 1;
    }
    public void setConf(Configuration configuration) {
        this.configuration = configuration;
    }
```

```
    public Configuration getConf() {
        return this.configuration;
    }
}
```

（3） 程序 3 实现代码如下：

使用 Scala 语言编写程序，对数据进行最后一步的清洗。对需要处理的字段执行相应操作，再定义样例类 Data 对处理完成的数据进行封装，然后将数据从 RDD 转换为 DataFrame，最后将数据保存到指定位置。具体如图 5-3-1 所示：

5-3-1 程序 3 中 Spark 程序

```
package com.szpt.spark.room
import org.apache.spark.sql.SparkSession
object FillNull {
  def main(args: Array[String]): Unit = {
    val spark = SparkSession.builder().appName("fill_null").getOrCreate()
    import spark.implicits._
    val dataFrame = spark.sparkContext.textFile(args(0))
      .map(line => {
        val values = line.split(",")
        val area = values(4).replace("㎡","")
        val price = values(7).replace("元/月", "")
```

```
            val floors = values(8).split("/")
            val floor = floors(0)
            val floor_num = floors(1).replace("层", "")
            Data(values(0), values(1), values(2), values(3), area, values(5),
values(6),
                price, floor, floor_num,values(9))
        }).toDF()
    dataFrame.write.format("csv")
        .option("header",false)
        . option("sep","\t")
        . save(args(1))
    spark. stop()
  }
}
case class Data(val rent_way:String, val name:String, val region:String,
val district:String, val area:String, val orientation:String, val
house_type:String, val price:String, val floor:String, val floor_num:String,
val agent:String)
```

第六章 项目收获

在这次毕业设计中，主要是使用 Java 和 Scala 语言实现数据清洗任务，同时在搭建平台与编写脚本时，在发现问题与解决问题的过程中，使得我们在原来的基础上更加熟悉、掌握这些技术与语言。

Hadoop 等组件的搭建与使用，组件 API 的使用都是学习中的重点，在这次的毕业设计中都或多或少的涉及到，涵盖了平时课堂上的内容，在不断的使用所学知识中，牢牢的记住相关的知识点；出现问题时，往往需要花费时间去定位错误，想要快速的单位错误，只有在项目中练习，通过查看错误日志等方式找到错误，研究出现错误的原因以及解决方法，或者查阅相关资料，找出最优的解决方案。

在完成项目的过程中查漏补缺，认识到自身存在的缺陷并加以纠正，脚踏实地，一步一步完善，巩固知识点，使得自身技术不断的提高，在项目中能够更加细心完成对应工作。

第七章 致谢

感谢我的导师邓雪峰老师，他幽默风趣，严谨仔细的作风一直是我工作、学习中的榜样，他循循善诱的教导使我思想受到无尽的启动，萌发更多的想法，付诸实践与努力，收获不断的进步与 优异的成果。

同时也感谢同学对我的帮助和指点。在没有他人的帮助下，可能无法如此快速切顺利的完成论文，在与他人分享与讨论的过程中，不断的对比，不断的学习下，使得论文不断的完善。

在我的论文即将完成之际，心情久久无法平静，从开始选题、进入课题后直到论文的顺利完成，有多少可敬的师长、同学、朋友给了我无言的帮助，在这里请接受我诚挚的谢意！