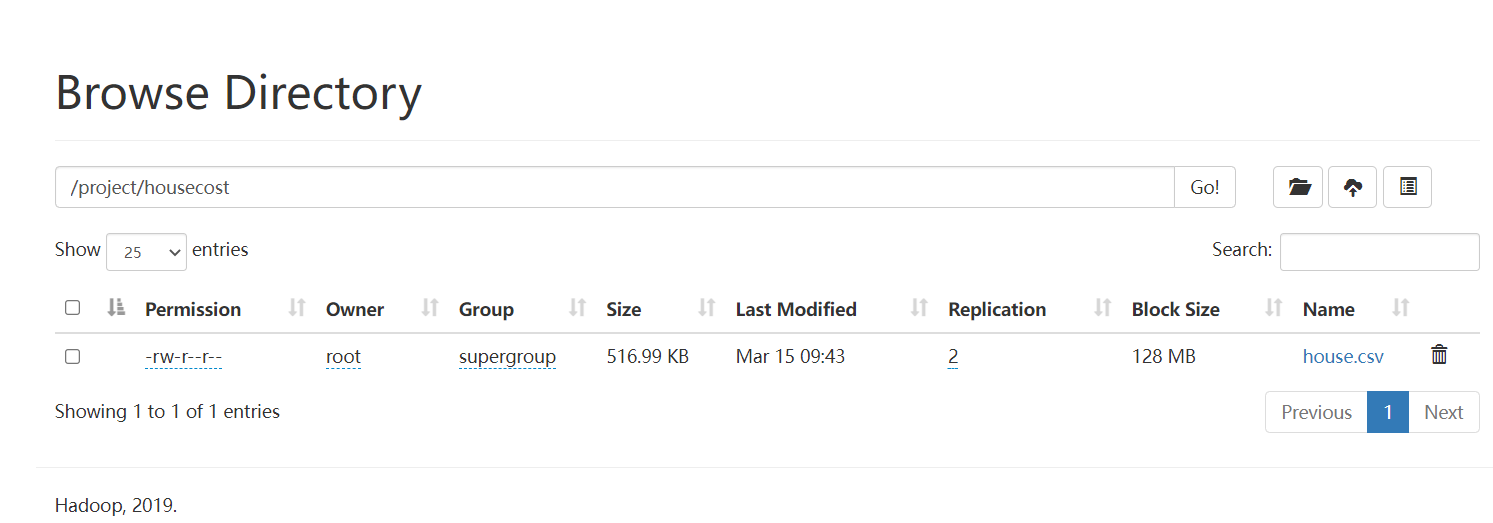
1. 将房价csv数据上传到hdfs上
2. 先建项目目录hdfs dfs -mkdir -p /project/housecost
3. 上传至项目目录hdfs dfs -put /data/file/house.csv /project/housecost



1. 编写计算房屋成本的实体类

package com.house.entity;

import org.apache.hadoop.io.WritableComparable;

import java.io.DataInput;

import java.io.DataOutput;

import java.io.IOException;

import java.math.BigDecimal;

public class HouseDetail implements WritableComparable<HouseDetail> {

/\*\*

\* 住宅面积

\*/

private Integer residenceSpace;

/\*\*

\* 住宅面积的单价

\*/

private BigDecimal unitPriceOfResidenceSpace;

/\*\*

\* 建筑面积

\*/

private Integer buildingSpace;

/\*\*

\* 建筑面积的单价

\*/

private BigDecimal unitPriceOfBuildingSpace;

/\*\*

\* 汇率

\*/

private BigDecimal exchangeRate;

public HouseDetail() {

}

public Integer getResidenceSpace() {

return residenceSpace;

}

public void setResidenceSpace(Integer residenceSpace) {

this.residenceSpace = residenceSpace;

}

public BigDecimal getUnitPriceOfResidenceSpace() {

return unitPriceOfResidenceSpace;

}

public void setUnitPriceOfResidenceSpace(BigDecimal unitPriceOfResidenceSpace) {

this.unitPriceOfResidenceSpace = unitPriceOfResidenceSpace;

}

public Integer getBuildingSpace() {

return buildingSpace;

}

public void setBuildingSpace(Integer buildingSpace) {

this.buildingSpace = buildingSpace;

}

public BigDecimal getUnitPriceOfBuildingSpace() {

return unitPriceOfBuildingSpace;

}

public void setUnitPriceOfBuildingSpace(BigDecimal unitPriceOfBuildingSpace) {

this.unitPriceOfBuildingSpace = unitPriceOfBuildingSpace;

}

public BigDecimal getExchangeRate() {

return exchangeRate;

}

public void setExchangeRate(BigDecimal exchangeRate) {

this.exchangeRate = exchangeRate;

}

@Override

public String toString() {

return residenceSpace + "," + unitPriceOfResidenceSpace + "," + buildingSpace + "," + unitPriceOfBuildingSpace + "," + exchangeRate;

}

@Override

public void write(DataOutput dataOutput) throws IOException {

dataOutput.writeInt(residenceSpace);

dataOutput.writeDouble(unitPriceOfResidenceSpace.doubleValue());

dataOutput.writeInt(buildingSpace);

dataOutput.writeDouble(unitPriceOfBuildingSpace.doubleValue());

dataOutput.writeDouble(exchangeRate.doubleValue());

}

@Override

public void readFields(DataInput dataInput) throws IOException {

this.residenceSpace = dataInput.readInt();

this.unitPriceOfResidenceSpace = BigDecimal.valueOf(dataInput.readDouble());

this.buildingSpace = dataInput.readInt();

this.unitPriceOfBuildingSpace = BigDecimal.valueOf(dataInput.readDouble());

this.exchangeRate = BigDecimal.valueOf(dataInput.readDouble());

}

@Override

public int compareTo(HouseDetail o) {

return o.getResidenceSpace().compareTo(o.getBuildingSpace());

}

}

1. 编写mapper类

Key是由住宅面积的单价、住宅面积、建筑面积的单价、建筑空间、汇率封装的实体 /\*\*

\* Mapper阶段

\*/

private static class MyMapper extends Mapper<LongWritable, Text, HouseDetail, NullWritable> {

HouseDetail houseDetail = new HouseDetail();

@Override

protected void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {

if (key.get() == 0) {

return;

}

String[] split = value.toString().split(",");

houseDetail.setResidenceSpace(Integer.valueOf(split[3]));

houseDetail.setUnitPriceOfResidenceSpace(BigDecimal.valueOf(Double.parseDouble(split[18])));

houseDetail.setBuildingSpace(Integer.valueOf(split[4]));

houseDetail.setUnitPriceOfBuildingSpace(BigDecimal.valueOf(Double.parseDouble(split[19])));

houseDetail.setExchangeRate(BigDecimal.valueOf(Double.parseDouble(split[17])));

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

}

}

1. 编写reducer类

/\*\*

\* Reducer阶段

\*/

private static class MyReducer extends Reducer<HouseDetail, NullWritable, HouseDetail, DoubleWritable> {

@Override

protected void reduce(HouseDetail k2, Iterable<NullWritable> v2s, Context context) throws IOException, InterruptedException {

//住宅面积

int residenceSpace = k2.getResidenceSpace();

//住宅面积的单价

BigDecimal unitPriceOfResidenceSpace = k2.getUnitPriceOfResidenceSpace();

//建筑面积

int buildingSpace = k2.getBuildingSpace();

//建筑面积的单价

BigDecimal unitPriceOfBuildingSpace = k2.getUnitPriceOfBuildingSpace();

//汇率

BigDecimal exchangeRate = k2.getExchangeRate();

//总成本的计算规则是。

//(住宅面积的单价\*住宅面积+建筑面积的单价\*建筑

//空间）\*汇率=总成本。

double cost = ((unitPriceOfResidenceSpace.multiply(BigDecimal.valueOf(residenceSpace)))

.add(unitPriceOfBuildingSpace.multiply(BigDecimal.valueOf(buildingSpace)))).multiply(exchangeRate)

.setScale(2, RoundingMode.UP).doubleValue();

context.write(k2,new DoubleWritable(cost));

}

}

1. 编写driver类

public static void main(String[] args) {

try {

if (args.length < 2){

System.exit(0);

}

//指定Job需要配置的参数

Configuration conf = new Configuration();

conf.set("fs.default.name", "hdfs://192.168.135.131:9000");

conf.set("hadoop.job.user", "hadoop");

conf.set("mapreduce.framework.name", "yarn");

//获取Job实例

Job job = Job.getInstance(conf);

FileSystem fileSystem = FileSystem.get(conf);

fileSystem.delete(new Path(args[1]), true);

FileInputFormat.setInputPaths(job, new Path(args[0]));

FileOutputFormat.setOutputPath(job, new Path(args[1]));

job.setJarByClass(CompanyEveryHouseCost.class);

job.setMapperClass(MyMapper.class);

job.setMapOutputKeyClass(HouseDetail.class);

job.setMapOutputValueClass(NullWritable.class);

job.setReducerClass(MyReducer.class);

job.setNumReduceTasks(2);

job.setOutputKeyClass(HouseDetail.class);

job.setOutputValueClass(DoubleWritable.class);

//提交Job

job.waitForCompletion(true);

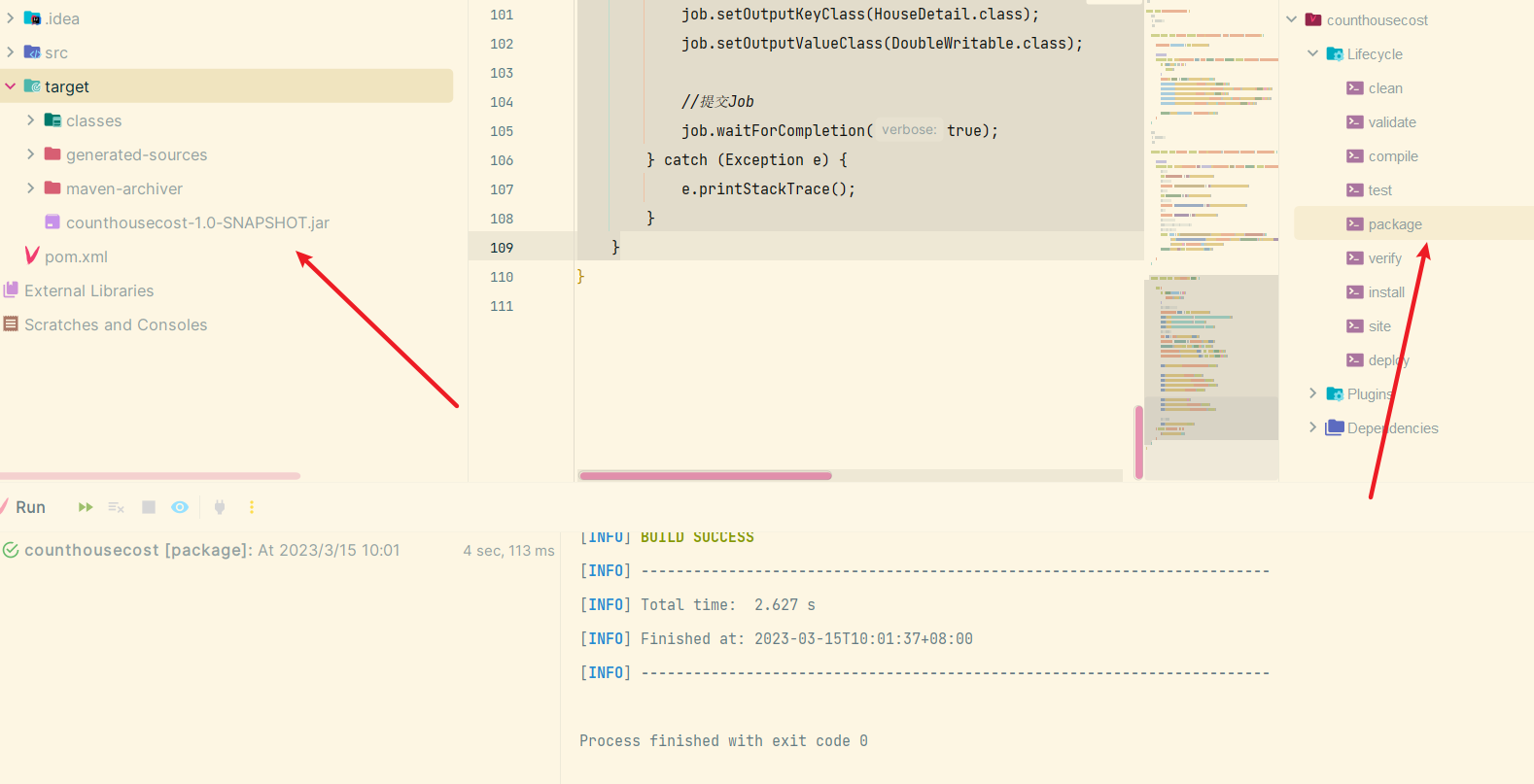
} catch (Exception e) {

e.printStackTrace();

}

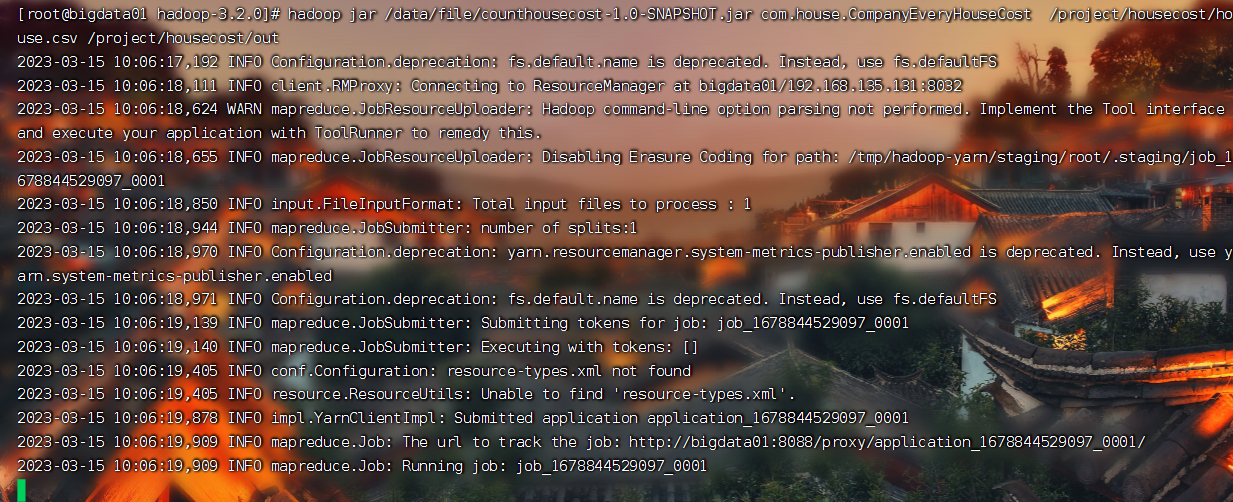
}

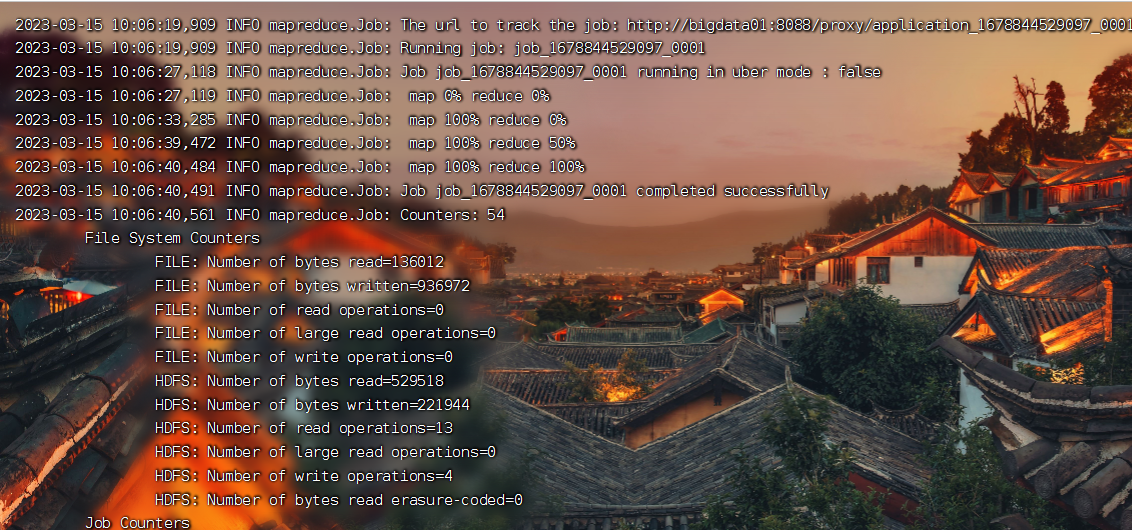
1. 项目打包提交到hadoop上面运行



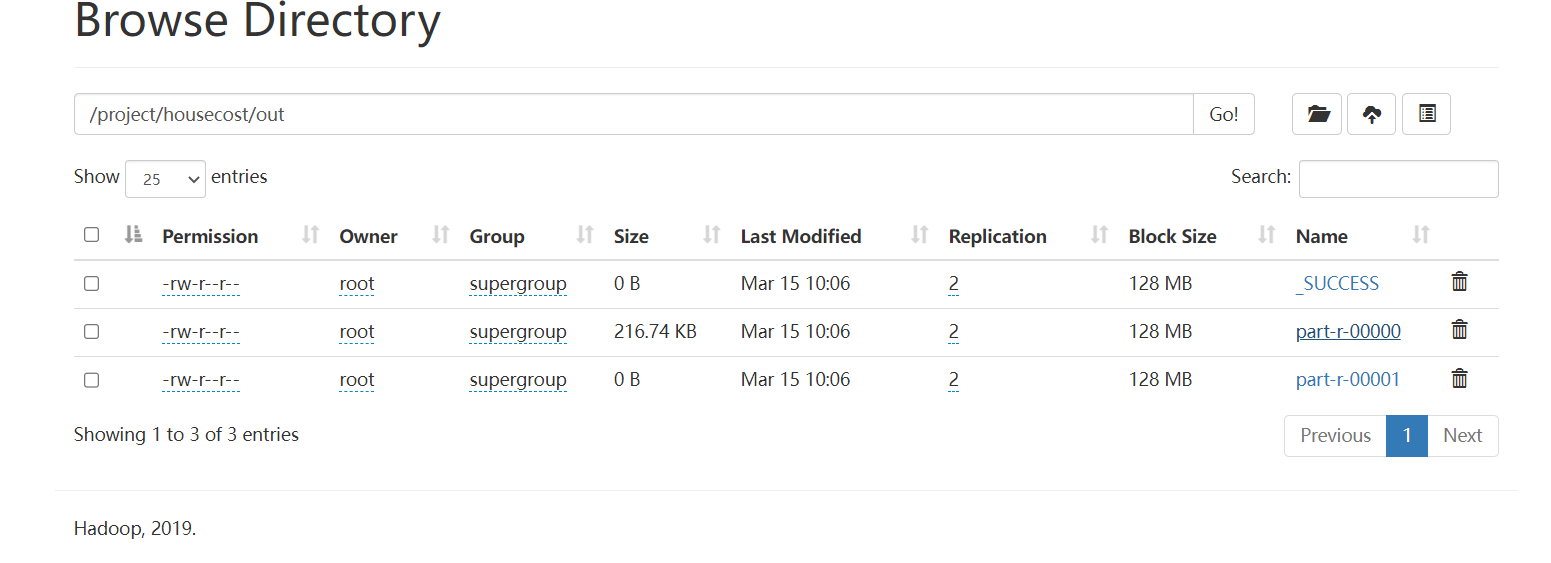
执行命令

hadoop jar /data/file/counthousecost-1.0-SNAPSHOT.jar com.house.CompanyEveryHouseCost /project/housecost/house.csv /project/housecost/out





1. 查看房价最终的计算结果



由于设置的reducer并行度是2，所以结果会有两个文件，内容如图所示



结果的第二列是按(住宅面积的单价\*住宅面积+建筑面积的单价\*建筑空间）\*汇率=总成本这个公式计算出来的每个房子的总成本