

实验名称

topN-每个用户订单总额的top3

实验目的

保存每个用户所有订单中消费总额最多的三个订单top3

实验背景

MapReduce中的key可自动完成排序，但在有些时候我们需要对value值进行排序，这时我们就可以使用WritableComparable接口来实现

实验原理

MapReduce中WritableComparable该接口实现了Writable和Comparable接口，而Writable接口中定义了readFields(DataInput in)方法和write(DataOutput out)方法，分别用来实现序列化和反序列化，而Comparable接口中定义了compareTo方法，该方法用来重写shuffle过程中对key的排序，因此如果想要让自定义的类作为map方法中的输出的key，必须让自定义方法实现WritableComparable接口，如果仅仅让自定义类作为map端输出的value，则可以仅仅实现Writable接口不实现Comparable接口。

Writable接口是用来序列化和反序列化用的，而Comparable接口是为了保证自定义类能够在shuffle过程中对key排序

实验环境

ubuntu 16.04

hadoop 2.7.3

jdk 1.8

maven 3.6.1

实验课时

2课时

实验步骤

一、环境准备

本实验在eclipse进行开发。

首先启动Hadoop环境：

```
hadoop namenode -format
start-all.sh
jps
```

当看下以下进程，则Hadoop启动成功

```
ubuntu@079b2e0d54d1:~$ jps
978 ResourceManager
1076 NodeManager
1370 Jps
827 SecondaryNameNode
667 DataNode
543 NameNode
```

配置maven 环境

另打开一个窗口，输入命令mvn -v检查是否已有Maven环境

```
mvn -v
```

```
ubuntu@660659678ad4:~$ mvn -v
Apache Maven 3.6.1 (d66c9c0b3152b2e69ee9bac180bb8fcc8e6af555; 2019-04-04T19:00:29Z)
Maven home: /opt/apache-maven-3.6.1
Java version: 1.8.0_231, vendor: Oracle Corporation, runtime: /opt/jdk8/jre
Default locale: en_US, platform encoding: UTF-8
OS name: "linux", version: "4.4.0-194-generic", arch: "amd64", family: "unix"
ubuntu@660659678ad4:~$
```

可看到maven环境已经配置成功，接下来就可以编写mapreduce代码了

二、数据准备

数据格式：

订单号	用户ID	商品名称	单价	购买数量
order001	u001	xiaomi6	1999.9	2
order001	u001	coffee	99	2
order001	u001	anmuxi	250	2
order001	u001	shuangxi	200	4
order001	u001	computerb	400	2
order002	u002	xiaomi	199	3
order002	u002	pear	15	10
order002	u002	apple	4.5	20
order002	u002	orange	10	40
order003	u001	xiaomi6	1999.9	2
order003	u001	coffee	99	2
order003	u001	anmuxi	250	2
order003	u001	shuangxi	200	4
order003	u001	computerb	400	2

打开命令行窗口，并切换路径至~

```
cd ~
```

使用vi命令创建文件order.csv,输入以下内容

```
order001,u001,xiaomi6,1999.9,2
order001,u001,coffee,99.0,2
order001,u001,anmuxi,250.0,2
order001,u001,shuangxi,200.0,4
order001,u001,computerbag,400.0,2
```

```
order002,u002,xiaomi,199.0,3
order002,u002,pear,15.0,10
order002,u002,apple,4.5,20
order002,u002,orange,10.0,40
order003,u001,xiaomi6,1999.9,2
order003,u001,coffee,99.0,2
order003,u001,anmuxi,250.0,2
order003,u001,shuangxi,200.0,4
order003,u001,computerbag,400.0,2
```

数据上传HDFS

在HDFS上创建目录/{学号}/shiyang4/input，并将需要计算的文件上传至该目录下，此处学号用001代替

```
hdfs dfs -mkdir -p /001/shiyang4/input
hdfs dfs -put ~/order.csv /001/shiyang4/input
```

三、代码编写

1、编写订单总额的top3的代码

在/src/main/java目录下创建包/com.hadoop.bean



在com.hadoop.bean包下创建实体类Order，实现WritableComparable接口



并实现接口的方法：

```

package com.hadoop.bean;

import org.apache.hadoop.io.WritableComparable;

import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;

public class Order implements WritableComparable {

    public int compareTo(Object o) {

        return 0;

    }

    public void write(DataOutput dataOutput) throws IOException {

    }

    public void readFields(DataInput dataInput) throws IOException {

    }

}

```

完整代码如下:

```

package com.hadoop.bean;

import org.apache.hadoop.io.WritableComparable;

import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;

public class Order implements WritableComparable<Order> {

    private String orderId;    //订单id
    private String userId;    //用户id
    private String product;    //产品
    private float price;    //价格
    private int number;    //数量
    private float sum;    //总价格

    //空参构造方法
    public Order() {
    }

    //全参构造方法
    public Order(String orderId, String userId, String product, float price, int
number, float sum) {
        this.orderId = orderId;
        this.userId = userId;
    }
}

```

```

        this.product = product;
        this.price = price;
        this.number = number;
        this.sum = sum;
    }

    //读取数据反序列化操作
    public void readFields(DataInput in) throws IOException {
        // TODO Auto-generated method stub
        this.orderId = in.readUTF();
        this.userId = in.readUTF();
        this.product = in.readUTF();
        this.price = in.readFloat();
        this.number = in.readInt();
        this.sum = in.readFloat();
    }

    //写数据时序列化操作
    public void write(DataOutput out) throws IOException {
        // TODO Auto-generated method stub
        out.writeUTF(this.orderId);
        out.writeUTF(this.userId);
        out.writeUTF(this.product);
        out.writeFloat(this.price);
        out.writeInt(this.number);
        out.writeFloat(this.sum);
    }

    //排序的对比方法
    public int compareTo(Order o) {
        return (int) (o.sum - this.sum);
    }

    //属性的getter与setter方法
    public String getOrderId() {
        return orderId;
    }

    public void setOrderId(String orderId) {
        this.orderId = orderId;
    }

    public String getUserId() {
        return userId;
    }

    public void setUserId(String userId) {
        this.userId = userId;
    }

    public String getProduct() {
        return product;
    }

```

```

    public void setProduct(String product) {
        this.product = product;
    }

    public float getPrice() {
        return price;
    }

    public void setPrice(float price) {
        this.price = price;
    }

    public int getNumber() {
        return number;
    }

    public void setNumber(int number) {
        this.number = number;
    }

    public float getSum() {
        return sum;
    }

    public void setSum(float sum) {
        this.sum = sum;
    }

    @Override
    public String toString() {
        return "Order [orderId=" + orderId + ", userId=" + userId + ", product="
+ product + ", price=" + price
        + ", number=" + number + ", sum=" + sum + "]";
    }
}

```

接下来在/src/main/java目录下创建OrderMapper类，并重写map()方法，步骤请参考之前的实验完整代码如下：

```

import java.io.IOException;

import com.hadoop.bean.Order;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class OrderMapper extends Mapper<LongWritable, Text, Text, Order> {
    @Override
    protected void map(LongWritable key, Text value,
        Mapper<LongWritable, Text, Text, Order>.Context context)
        throws IOException, InterruptedException {

        //根据“,”分割数据
    }
}

```

```

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

if (lines.length > 1) {
    //获取单价
    float price = Float.parseFloat(lines[3]);
    //获取数量
    int number = Integer.parseInt(lines[4]);
    //计算总价
    float sum = price * number;
    //创建Order实例
    Order order = new Order(lines[0], lines[1], lines[2], price, number,
sum);

    //使用userID作为Key值，Order对象作为Value值输出到reduce端
    context.write(new Text(lines[1]), order);
}
}
}

```

接下来创建OrderReducer类，并重写reduce()方法，步骤请参考之前的实验

完整代码如下：

```

import java.io.IOException;
import java.util.ArrayList;
import java.util.Collections;

import com.hadoop.bean.Order;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class OrderReducer extends Reducer<Text, Order, Order, NullWritable>{

    @Override
    protected void reduce(Text arg0, Iterable<Order> value,
                          Reducer<Text, Order, Order, NullWritable>.Context
context)
        throws IOException, InterruptedException {

        ArrayList<Order> list = new ArrayList<Order>();

        //循环遍历订单列表，并将数据添加到list列表中。
        for(Order order:value) {
            Order o = new
Order(order.getOrderid(),order.getUserid(),order.getProduct(),
            order.getPrice(),order.getNumber(),order.getSum());
            list.add(o);
        }
        //根据总金额进行排序
        Collections.sort(list);

        //输出Top3
        for(int i=0;i<3;i++) {
            context.write(list.get(i), NullWritable.get());
        }
    }
}

```

```
}  
}
```

最后完成main()的编写

完整代码如下:

```
import com.hadoop.bean.Order;  
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;  
  
public class OrderMain {  
    public static void main(String[] args) throws Exception {  
        // 设置hadoop运行配置  
        Configuration conf = new Configuration();  
  
        //创建任务, 并设置任务的驱动类  
        Job job = Job.getInstance(conf);  
        job.setJarByClass(OrderMain.class);  
  
        //配置mapper类, 并设置Map端输出格式  
        job.setMapperClass(OrderMapper.class);  
        job.setMapOutputKeyClass(Text.class);  
        job.setMapOutputValueClass(Order.class);  
  
        //配置reducer类, 并设置reduce端输出格式  
        job.setReducerClass(OrderReducer.class);  
        job.setOutputKeyClass(Order.class);  
        job.setOutputValueClass(NullWritable.class);  
  
        //设置数据读取路径  
        FileInputFormat.setInputPaths(job, new  
Path("hdfs://localhost:9000/001/shiyan4/input"));  
        //设置数据输出路径  
        FileOutputFormat.setOutputPath(job, new  
Path("hdfs://localhost:9000/001/shiyan4/output"));  
        job.waitForCompletion(true);  
    }  
}
```

四、执行程序并查看代码

执行程序:


```

8 import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
9
10 1个用法
11 public class OrderMain {
12     public static void main(String[] args) throws Exception {
13         // 设置hadoop运行配置
14         Configuration conf = new Configuration();
15
16         // 创建任务，并设置任务的驱动类
17         Job job = Job.getInstance(conf);
18         job.setJarByClass(OrderMain.class);
19
20         // 配置mapper类，并设置Map端输出格式
21         job.setMapperClass(OrderMapper.class);
22         job.setMapOutputKeyClass(Text.class);
23         job.setMapOutputValueClass(Order.class);
24
25         // 配置reducer类，并设置reduce端输出格式
26         job.setReducerClass(OrderReducer.class);
27         job.setOutputKeyClass(Order.class);
28         job.setOutputValueClass(NullWritable.class);
29     }
30 }

```

打开命令行窗口，使用hadoop shell查看结果

```

hdfs dfs -ls /001/shiyan4/output
hdfs dfs -cat /001/shiyan4/output/part-r-00000

```

```

ubuntu@809f6fceb3d:~$ hdfs dfs -ls /output
Found 2 items
-rw-r--r--  3 ubuntu supergroup          0 2022-02-18 09:10 /output/ SUCCESS
-rw-r--r--  3 ubuntu supergroup        534 2022-02-18 09:09 /output/part-r-00000
ubuntu@809f6fceb3d:~$ hdfs dfs -cat /output/part-r-00000
Order [orderId=order001, userId=u001, product=xiaomi6, price=1999.9, number=2, sum=3999.8]
Order [orderId=order003, userId=u001, product=xiaomi6, price=1999.9, number=2, sum=3999.8]
Order [orderId=order003, userId=u001, product=shuangxi, price=200.0, number=4, sum=800.0]
Order [orderId=order002, userId=u002, product=xiaomi, price=199.0, number=3, sum=597.0]
Order [orderId=order002, userId=u002, product=orange, price=10.0, number=40, sum=400.0]
Order [orderId=order002, userId=u002, product=pear, price=15.0, number=10, sum=150.0]
ubuntu@809f6fceb3d:~$

```

到此实验完成

实验总结

MapReduce中WritableComparable该接口实现了Writable和Comparable接口，分别用来实现序列化、反序列化和排序，这样我们也可以根据计算的结果进行排序来获取最值