实验名称

计算HDFS上的学生成绩写入HBase

实验目的

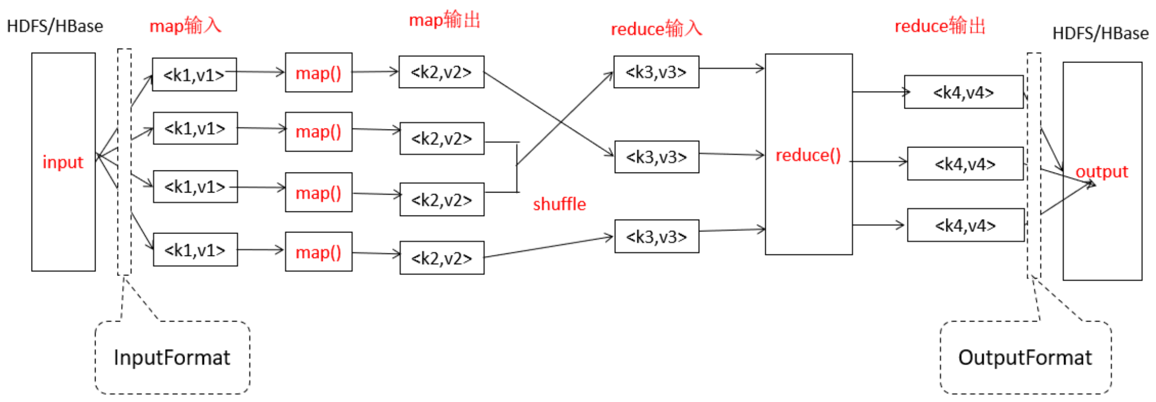
熟练掌握MapReduce的编程模型
熟练开发基于HDFS和HBase的MapReduce程序

实验背景

对于海量的离线数据的处理，我们一般采用的是MapReduce来进行计算。而数据一般存储在HDFS或HBase表中，如何来完成HDFS与HBase表中数据的计算呢？本节实验将HDFS的数据通过MapReduce编程计算后写入HBase。

实验原理

一、MapReduce编程模型



从MapReduce自身的命名特点可以看出，MapReduce由两个阶段组成：Map和Reduce。用户只需要编写map()和reduce()两个函数，即可完成分布式程序的设计。

实验环境

ubuntu 22.10
hadoop 3.1.3
jdk 1.8
hbase 2.2.2

建议课时

2课时

实验步骤

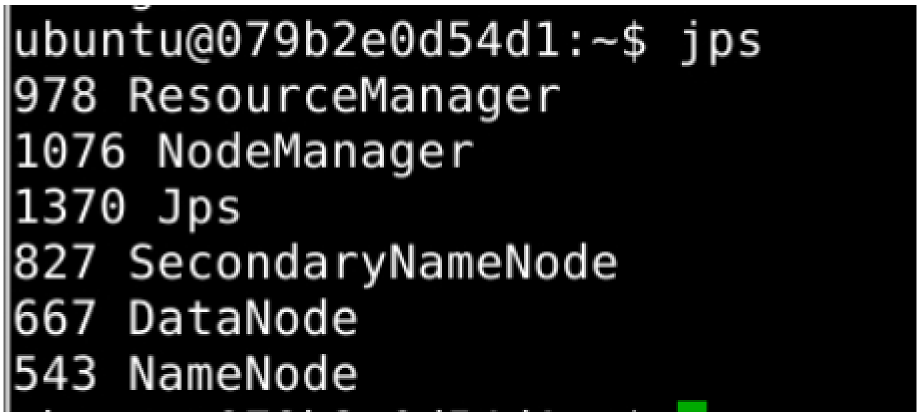
一、环境准备

本实验在idea进行开发。

首先启动Hadoop环境：

```
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
```

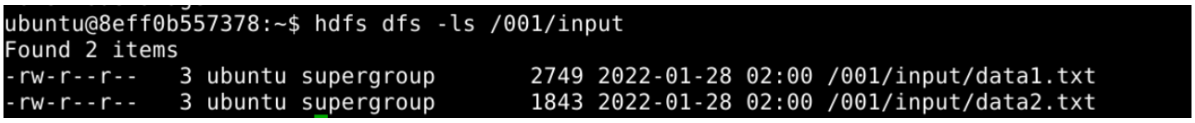
数据源：使用实验一数据源

在HDFS上创建目录/{学号}/input，并将需要计算的文件上传至该目录下（实验一已经上传的，该步骤可省略）。

```
hdfs dfs -mkdir -p /001/input
hdfs dfs -put ~/data1.txt /001/input
hdfs dfs -put ~/data2.txt /001/input
```

查看文件是否上传成功

```
hdfs dfs -ls /001/input
```



```
ubuntu@8eff0b557378:~$ hdfs dfs -ls /001/input
Found 2 items
-rw-r--r--   3 ubuntu supergroup       2749 2022-01-28 02:00 /001/input/data1.txt
-rw-r--r--   3 ubuntu supergroup       1843 2022-01-28 02:00 /001/input/data2.txt
```

出现如上图所示，则表示文件上传成功

启动HBase

另打开一个终端

启动hbase环境

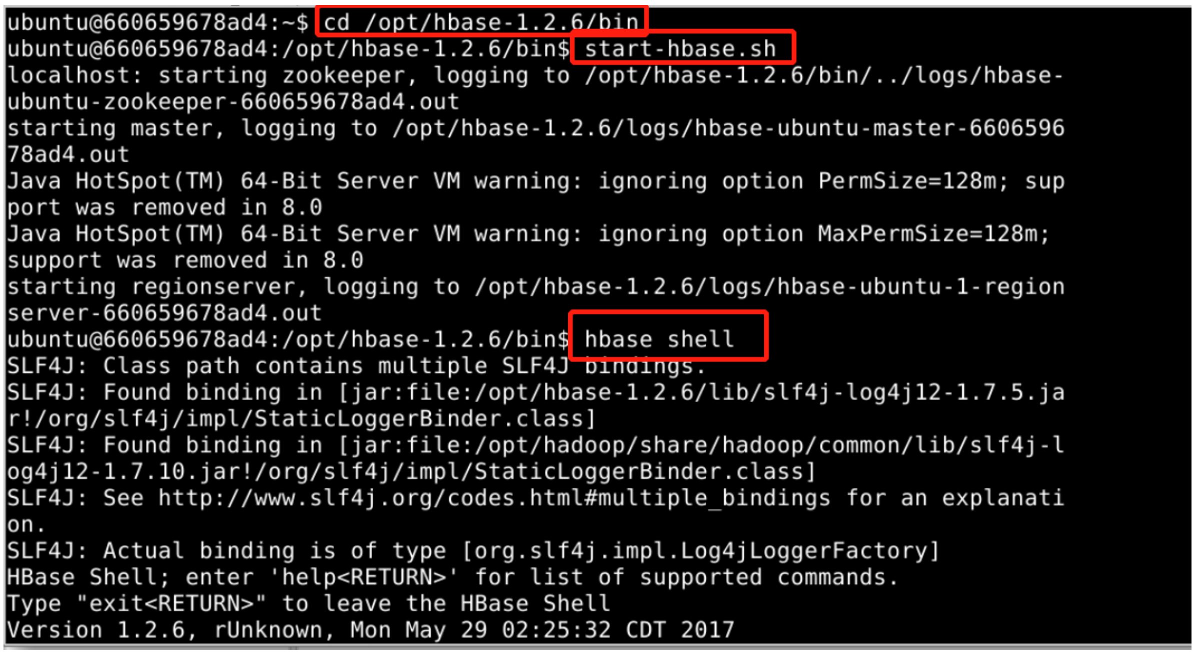
```
cd /opt/hbase-1.2.6/bin
start-hbase.sh
```

启动hbase shell

```
hbase shell
```

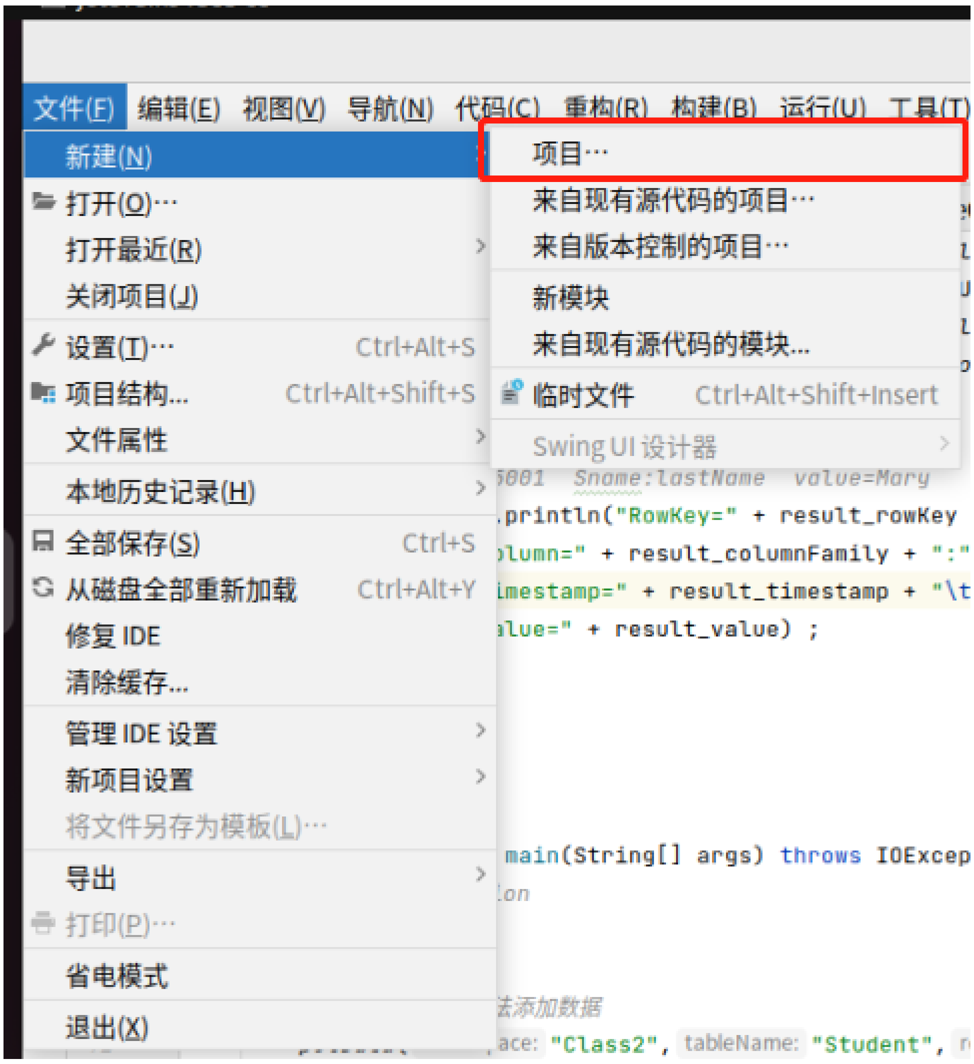
创建result表

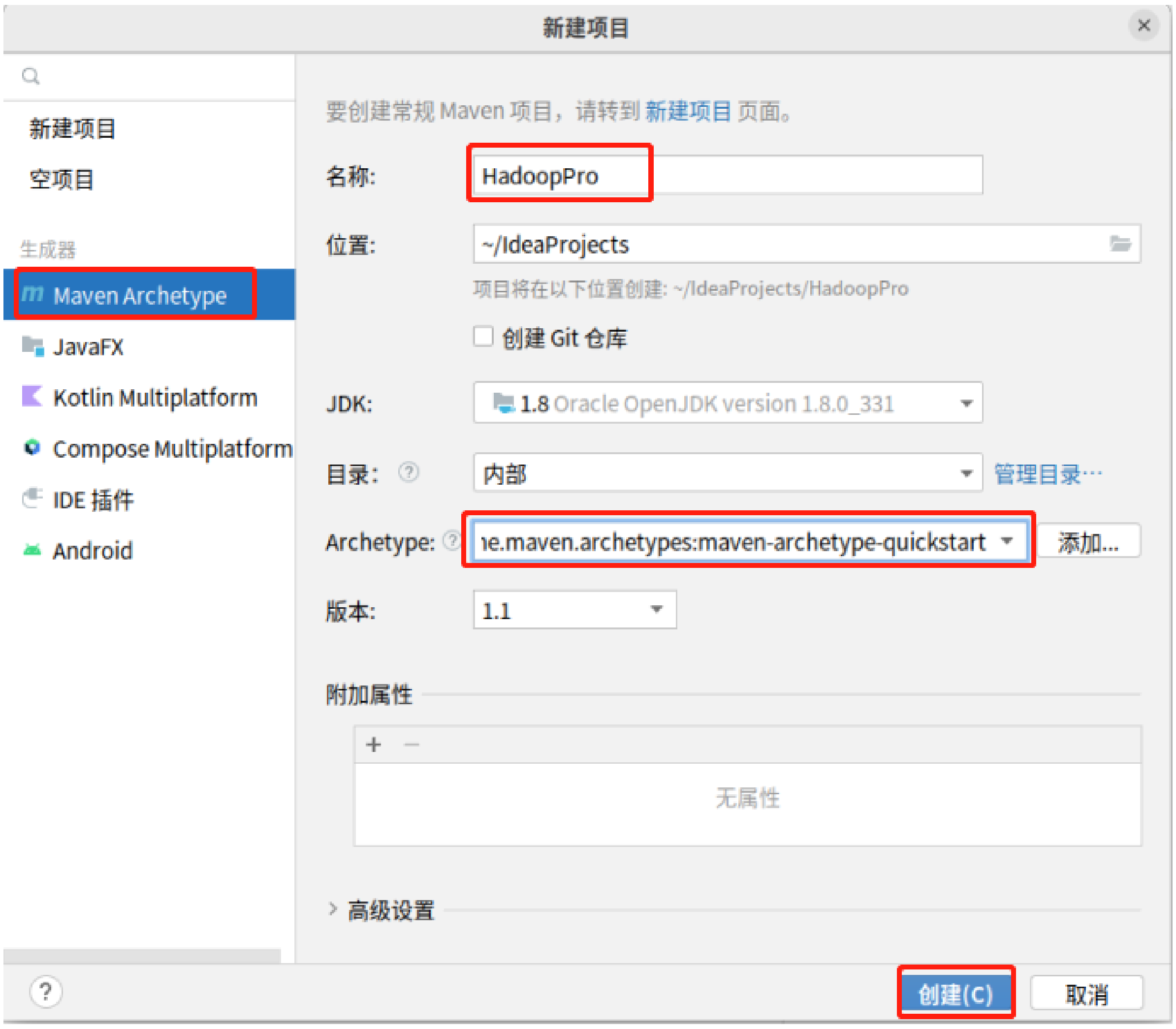
```
create 'result','content'
```


```
ubuntu@660659678ad4:~$ cd /opt/hbase-1.2.6/bin
ubuntu@660659678ad4:/opt/hbase-1.2.6/bin$ start-hbase.sh
localhost: starting zookeeper, logging to /opt/hbase-1.2.6/bin/../logs/hbase-
ubuntu-zookeeper-660659678ad4.out
starting master, logging to /opt/hbase-1.2.6/logs/hbase-ubuntu-master-6606596
78ad4.out
Java HotSpot(TM) 64-Bit Server VM warning: ignoring option PermSize=128m; sup
port was removed in 8.0
Java HotSpot(TM) 64-Bit Server VM warning: ignoring option MaxPermSize=128m;
support was removed in 8.0
starting regionserver, logging to /opt/hbase-1.2.6/logs/hbase-ubuntu-1-region
server-660659678ad4.out
ubuntu@660659678ad4:/opt/hbase-1.2.6/bin$ hbase shell
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/hbase-1.2.6/lib/slf4j-log4j12-1.7.5.ja
r!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/hadoop/share/hadoop/common/lib/slf4j-l
og4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanati
on.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
HBase Shell; enter 'help<RETURN>' for list of supported commands.
Type "exit<RETURN>" to leave the HBase Shell
Version 1.2.6, rUnknown, Mon May 29 02:25:32 CDT 2017
```

二、代码编写

1、创建maven project（实验一已创建的，该步骤省略）





要创建常规 Maven 项目，请转到 新建项目 页面。

名称: HadoopPro

位置: ~/IdeaProjects

项目将在以下位置创建: ~/IdeaProjects/HadoopPro

创建 Git 仓库

JDK: 1.8 Oracle OpenJDK version 1.8.0_331

目录: 内部

Archetype: ıe.maven.archetypes:maven-archetype-quickstart

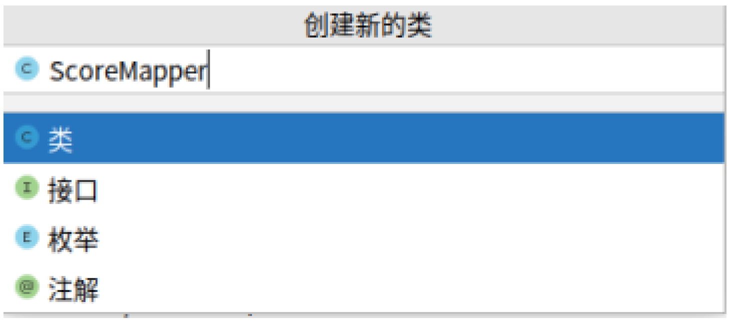
版本: 1.1

创建成功后，打开pom.xml，添加hadoop,hbase相关依赖

```
<dependencies>
        <!-- Hadoop 依赖-->
        <dependency>
            <groupId>org.apache.hadoop</groupId>
            <artifactId>hadoop-client</artifactId>
            <version>3.1.3</version>
        </dependency>
        <dependency>
            <groupId>org.apache.hadoop</groupId>
            <artifactId>hadoop-common</artifactId>
            <version>3.1.3</version>
        </dependency>
        <dependency>
            <groupId>org.apache.hadoop</groupId>
            <artifactId>hadoop-hdfs</artifactId>
            <version>3.1.3</version>
        </dependency>
        <dependency>
            <groupId>org.apache.hadoop</groupId>
            <artifactId>hadoop-mapreduce-client-core</artifactId>
            <version>3.1.3</version>
        </dependency>
        <!--Hbase 依赖-->
        <dependency>
            <groupId>org.apache.hbase</groupId>
            <artifactId>hbase-client</artifactId>
            <version>2.2.2</version>
        </dependency>
        <dependency>
            <groupId>org.apache.hbase</groupId>
            <artifactId>hbase-server</artifactId>
            <version>2.2.2</version>
        </dependency>
        <dependency>
            <groupId>org.apache.hbase</groupId>
            <artifactId>hbase-mapreduce</artifactId>
            <version>2.2.2</version>
        </dependency>
    </dependencies>
```

2、编写map()

创建类ScoreMapper


创建新的类

ScoreMapper

ScoreMapper完整代码如下:

```
import java.io.IOException;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class ScoreMapper extends Mapper<LongWritable,Text,Text,IntWritable>{

    @Override
    protected void map(LongWritable key, Text value, Mapper<LongWritable, Text,
Text, IntWritable>.Context context)
            throws IOException, InterruptedException {

        //根据分割符分割数据，返回字符串数组
        String[] info = value.toString().split("\t");

        //输出数据，Key值为班级，value值为分数
        context.write(new Text(info[5]), new
IntWritable(Integer.valueOf(info[4])));
    }

}
```

3、编写reduce()

用同样的方法创建ScoreReducer，继承Reducer类，重写reduce方法，需要注意的是，因为我们要把最终结果写入HBase表中，所以此时我们继承的是TableReducer，其中ImmutableBytesWritable是rowkey的类型，完整代码如下：

```
import java.io.IOException;

import org.apache.hadoop.hbase.client.Mutation;
import org.apache.hadoop.hbase.client.Put;
import org.apache.hadoop.hbase.io.ImmutableBytesWritable;
import org.apache.hadoop.hbase.mapreduce.TableReducer;
import org.apache.hadoop.hbase.util.Bytes;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class ScoreReducer extends
TableReducer<Text,IntWritable,ImmutableBytesWritable>{

    @Override
    protected void reduce(Text k3, Iterable<IntWritable> v3,
                          Reducer<Text, IntWritable, ImmutableBytesWritable,
Mutation>.Context context)
            throws IOException, InterruptedException {
        //计算学生个数
        int count = 0;
        //计算总成绩
        int sum = 0;

        //循环遍历获取value值，并求出成绩总和，与学生个数
        for(IntWritable v:v3) {
            count += v.get();
            sum ++;
        }

        //获取平均值
        int avg = count/sum;

        //创建put对象
        Put put = new Put(Bytes.toBytes(k3.toString()));
        //添加列与列值
        put.addColumn(Bytes.toBytes("content"), Bytes.toBytes("avg"),
Bytes.toBytes(String.valueOf(avg)));
        put.addColumn(Bytes.toBytes("content"), Bytes.toBytes("count"),
Bytes.toBytes(String.valueOf(count)));
        put.addColumn(Bytes.toBytes("content"), Bytes.toBytes("sum"),
Bytes.toBytes(String.valueOf(sum)));

        //输出数据
        context.write(new
ImmutableBytesWritable(Bytes.toBytes(k3.toString())),put);
    }



}
```

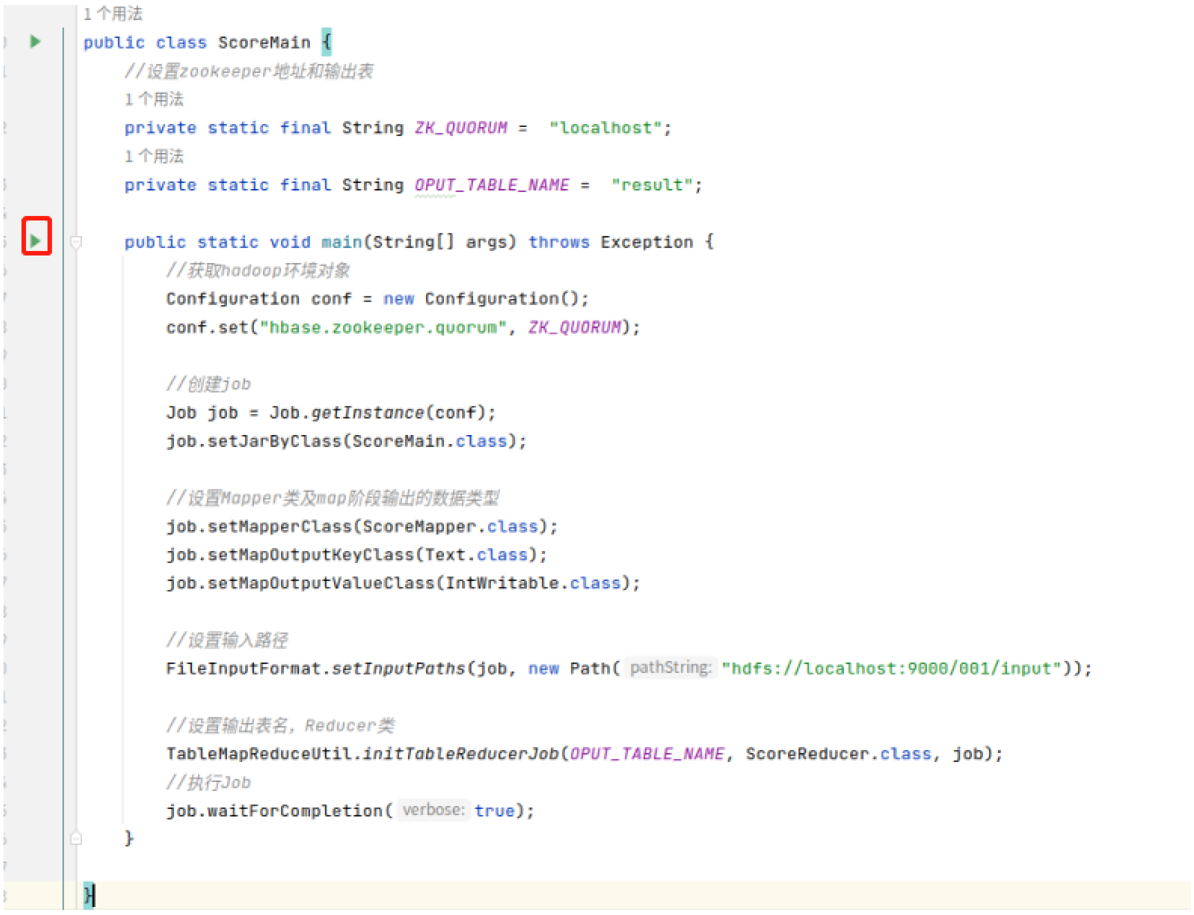
4、编写Driver

用同样的方法创建ScoreMain类，在main()中编写mapreduce相关的设置代码，

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.hbase.mapreduce.TableMapReduceUtil;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;


public class ScoreMain {
    //设置zookeeper地址和输出表
    private static final String ZK_QUORUM =  "localhost";
    private static final String OPUT_TABLE_NAME =  "result";

    public static void main(String[] args) throws Exception {
        //获取hadoop环境对象
        Configuration conf = new Configuration();
        conf.set("hbase.zookeeper.quorum", ZK_QUORUM);

        //创建job
        Job job = Job.getInstance(conf);
        job.setJarByClass(ScoreMain.class);

        //设置Mapper类及map阶段输出的数据类型
        job.setMapperClass(ScoreMapper.class);
        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(IntWritable.class);

        //设置输入路径
        FileInputFormat.setInputPaths(job, new
Path("hdfs://localhost:9000/001/input"));

        //设置输出表名，Reducer类
        TableMapReduceUtil.initTableReducerJob(OPUT_TABLE_NAME,
ScoreReducer.class, job);
        //执行Job
        job.waitForCompletion(true);
    }

}
```

5、运行程序并查看结果

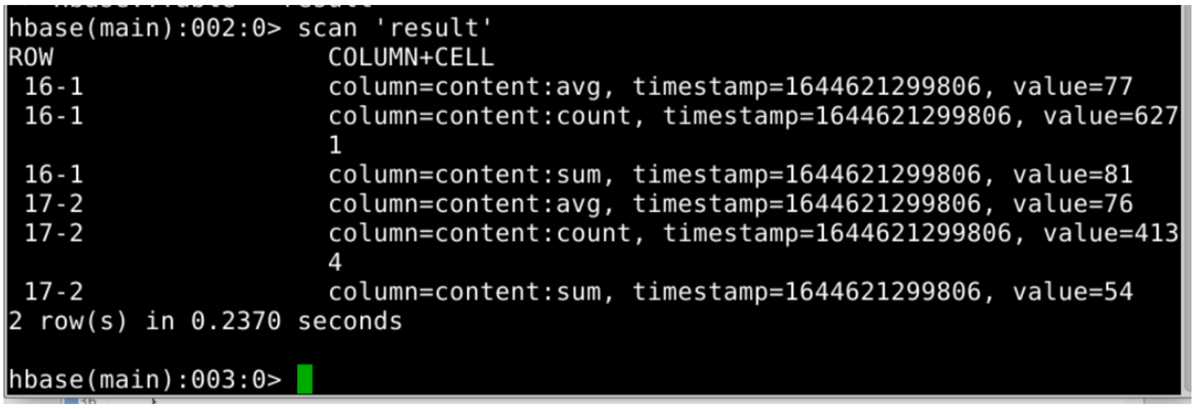
运行main()



```
1 个用法
public class ScoreMain {
    //设置zookeeper地址和输出表
    1 个用法
    private static final String ZK_QUORUM =  "localhost";
    1 个用法
    private static final String OPUT_TABLE_NAME =  "result";

    public static void main(String[] args) throws Exception {
        //获取hadoop环境对象
        Configuration conf = new Configuration();
        conf.set("hbase.zookeeper.quorum", ZK_QUORUM);

        //创建job
        Job job = Job.getInstance(conf);
        job.setJarByClass(ScoreMain.class);

        //设置Mapper类及map阶段输出的数据类型
        job.setMapperClass(ScoreMapper.class);
        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(IntWritable.class);

        //设置输入路径
        FileInputFormat.setInputPaths(job, new Path( pathString: "hdfs://localhost:9000/001/input"));

        //设置输出表名，Reducer类
        TableMapReduceUtil.initTableReducerJob(OPUT_TABLE_NAME, ScoreReducer.class, job);
        //执行Job
        job.waitForCompletion( verbose: true);
    }

}
```

在hbase shell界面查看结果

```
scan 'result'
```



```
hbase(main):002:0> scan 'result'
ROW                    COLUMN+CELL
 16-1                  column=content:avg, timestamp=1644621299806, value=77
 16-1                  column=content:count, timestamp=1644621299806, value=627
                       1
 16-1                  column=content:sum, timestamp=1644621299806, value=81
 17-2                  column=content:avg, timestamp=1644621299806, value=76
 17-2                  column=content:count, timestamp=1644621299806, value=413
                       4
 17-2                  column=content:sum, timestamp=1644621299806, value=54
2 row(s) in 0.2370 seconds

hbase(main):003:0>
```

实验总结

该实验主要是练习基于HDFS和HBase的MapReduce的编写，其中，源数据存储在HDFS上，故在编写MyMapper方法时，需要继承的是Mapper类，即重写的map()是Mapper类中的方法，我们最终将结果写入HBase表，即处理完成的reduce()结果最终由outputFormat类写入hbase中，故在编写MyReducer类时，需要继承的是TableReducer类，即重写的reduce()是TableReducer类中的方法，同理，若数据来源于hbase，最终结果写入HDFS，则需要重写的TableMapper中的map()方法和Reducer类的reduce()，即需要继承的两个类分别是TableMapper和Reducer。