Beijing University of Posts and Telecommunications

# 华为云鲲鹏大数据基础实验体系 3 HBase 应用实践

| | |
|---|---|
| 编写负责人： | 鄂海红、宋美娜 |
| 参编人员： | 程祥、张忠宝、朱一凡、 |
| | 魏文定、刘钟允、汤子辰、罗子霄、王浩田 等 |

北京邮电大学《大数据技术基础》课程教研室

# 华为云鲲鹏大数据基础实验体系 3：HBase 应用实践

## 1.1.1. 实验描述

在《基于华为云大数据实践 1：构建大数据实验环境》、《基于华为云大数据实践 2：搭建Hadoop集群并实践HDFS》2个实验全部完成后，搭建好的集群环境上，继续安装HBase、Zookeeper，实践 HBase 基本使用。

## 1.1.2. 实验目的

掌握 HBase、ZooKeeper 的安装与使用，批量将 HBase 表上的数据导入到 HDFS 中，学习本实验能快速掌握 HBase 数据库在分布式计算中的应用，理解 Java API 读取 HBase 数据等相关内容。

北京邮电大学《大数据技术基础》课程教研室

## 1.1.3. 实验步骤

### 1.1.3.1. 集群各节点的软件规划

本实验手册示例命令中，节点名称是 name-number-000{编号}，学生需要修改主机名为对应的姓名缩写+学号。

| 机器名称 | 进程名称 |
|---|---|
| name-number-0001 | QuorumPeerMain、NameNode、ResourceManager、Hmaster |
| name-number-0002 | QuorumPeerMain、DataNode、NodeManager、JournalNode、HRegionServer |
| name-number-0003 | QuorumPeerMain、DataNode、NodeManager、JournalNode、HRegionServer |
| name-number-0004 | QuorumPeerMain、DataNode、NodeManager、JournalNode、HRegionServer |

开始本次实验前请确保已完成第一章和第二章的实验，安装好 Hadoop 并配置好环境变量。

### 1.1.3.2. 下载安装并配置 zookeeper

在用户目录下下载 zookeeper 压缩包并解压

```
wget https://archive.apache.org/dist/zookeeper/zookeeper-3.4.6/zookeeper-3.4.6.tar.gz
mv zookeeper-3.4.6.tar.gz /usr/local
cd /usr/local
tar -zxvf zookeeper-3.4.6.tar.gz
```

建立软链接，便于后期版本更换。

```
ln -s zookeeper-3.4.6 zookeeper
```

打开配置文件。

```
vim /etc/profile
```

添加 ZooKeeper 到环境变量。

```
export ZOOKEEPER_HOME=/usr/local/zookeeper
export PATH=$ZOOKEEPER_HOME/bin:$PATH
```

使环境变量生效。

```
source /etc/profile
```

进入 ZooKeeper 所在目录。

```
cd /usr/local/zookeeper/conf
```

拷贝配置文件。

```
cp zoo_sample.cfg zoo.cfg
```

修改配置文件。

```
vim zoo.cfg
```

修改数据目录。

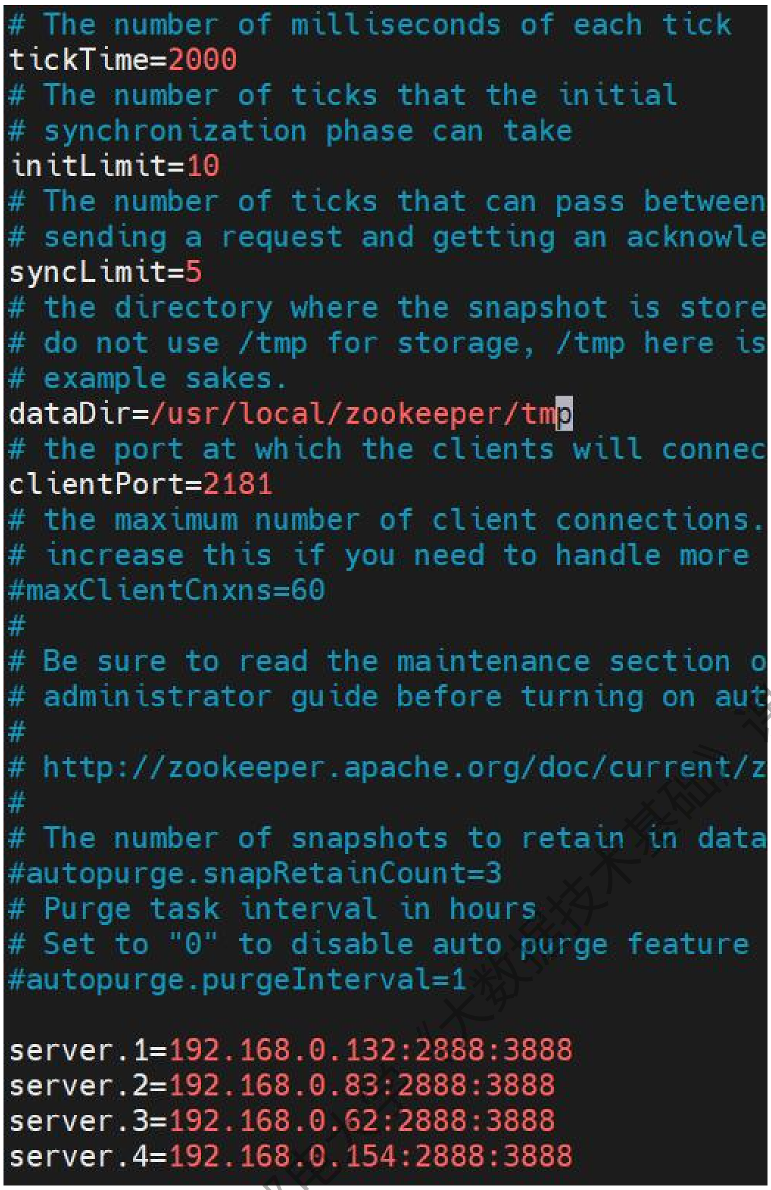
```
dataDir=/usr/local/zookeeper/tmp
```

在最后添加如下代码，server.1-4 是部署 ZooKeeper 的节点，1，2，3，4 分别是各服务器/usr/local/zookeeper/tmp/myid 文件的内容。这里 192.168.0.xxx 对应的是运行 QuorumPeerMain 的服务器的内网 IP，需要改成自己集群的。

```
server.1=192.168.0.132:2888:3888
server.2=192.168.0.83:2888:3888
server.3=192.168.0.62:2888:3888
server.4=192.168.0.154:2888:3888
```

修改后的 zoo.cfg 如下：

北京邮电大学《大数据技术基础》课程教研室



```
# The number of milliseconds of each tick
tickTime=2000
# The number of ticks that the initial
# synchronization phase can take
initLimit=10
# The number of ticks that can pass between
# sending a request and getting an acknowle
syncLimit=5
# the directory where the snapshot is store
# do not use /tmp for storage, /tmp here is
# example sakes.
dataDir=/usr/local/zookeeper/tmp
# the port at which the clients will connec
clientPort=2181
# the maximum number of client connections.
# increase this if you need to handle more
#maxClientCnxns=60
#
# Be sure to read the maintenance section o
# administrator guide before turning on aut
#
# http://zookeeper.apache.org/doc/current/z
#
# The number of snapshots to retain in data
#autopurge.snapRetainCount=3
# Purge task interval in hours
# Set to "0" to disable auto purge feature
#autopurge.purgeInterval=1

server.1=192.168.0.132:2888:3888
server.2=192.168.0.83:2888:3888
server.3=192.168.0.62:2888:3888
server.4=192.168.0.154:2888:3888
```

北京邮电大学《大数据技术基础》课程教研室

创建 tmp 目录作数据目录。

```
mkdir /usr/local/zookeeper/tmp
```

在 tmp 目录中创建一个空文件 myid，并向该文件写入 ID。

```
touch /usr/local/zookeeper/tmp/myid
echo 1 > /usr/local/zookeeper/tmp/myid
```

将配置好的 ZooKeeper 拷贝到其它节点。（也可以将 zookeeper 压缩包拷贝到其他节点，在进行相同的配置，这样等待时间较短）

```
scp -r /usr/local/zookeeper-3.4.6 root@name-number-0002:/usr/local
scp -r /usr/local/zookeeper-3.4.6 root@name-number-0003:/usr/local
scp -r /usr/local/zookeeper-3.4.6 root@name-number-0004:/usr/local
```

登录 name-number-0002、name-number-0003、name-number-0004，创建软链接并修改 myid 内容。

name-number-0002：

```
cd /usr/local
ln -s zookeeper-3.4.6 zookeeper
echo 2 > /usr/local/zookeeper/tmp/myid
```

name-number-0003：

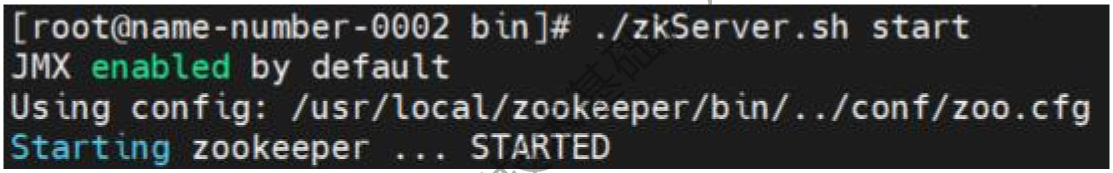
```
cd /usr/local
ln -s zookeeper-3.4.6 zookeeper
echo 3 > /usr/local/zookeeper/tmp/myid
```

name-number-0004：

```
cd /usr/local
ln -s zookeeper-3.4.6 zookeeper
echo 4 > /usr/local/zookeeper/tmp/myid
```

分别在 name-number-0002，name-number-0003，name-number-0004 上启动 ZooKeeper。

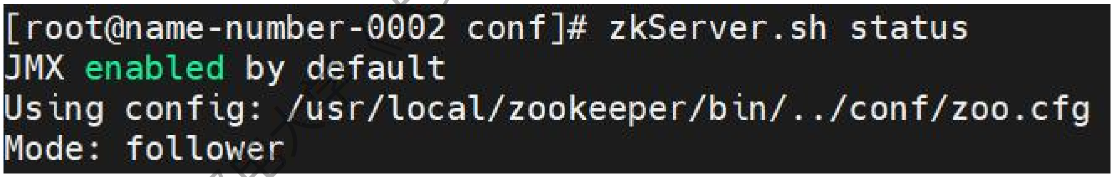
```
cd /usr/local/zookeeper/bin
./zkServer.sh start
```



```
[root@name-number-0002 bin]# ./zkServer.sh start
JMX enabled by default
Using config: /usr/local/zookeeper/bin/../conf/zoo.cfg
Starting zookeeper ... STARTED
```

查看 ZooKeeper 状态，注意，Mode 应为 leader 或 follower。

```
./zkServer.sh status
```



```
[root@name-number-0002 conf]# zkServer.sh status
JMX enabled by default
Using config: /usr/local/zookeeper/bin/../conf/zoo.cfg
Mode: follower
```

### 1.1.3.3. 下载并安装 HBase

下载 HBase，下载地址：

```
https://archive.apache.org/dist/hbase/2.0.2/hbase-2.0.2-bin.tar.gz
```

将 hbase-2.0.2.tar.gz 放置于 name-number-0001 节点的“/usr/local”目录，并解压。

```
mv hbase-2.0.2.tar.gz /usr/local
cd /usr/local
tar -zxvf hbase-2.0.2.tar.gz
```

建立软链接，便于后期版本更换。

```
ln -s hbase-2.0.2 hbase
```

编辑“/etc/profile”文件。

北京邮电大学大数据技术基础课程教研室

```
vim /etc/profile
```

在文件底部添加环境变量，如下所示。

```
export HBASE_HOME=/usr/local/hbase
export PATH=$HBASE_HOME/bin:$HBASE_HOME/sbin:$PATH
```

使环境变量生效。

```
source /etc/profile
```

修改 HBase 配置文件

HBase 所有的配置文件都在“HBASE_HOME/conf”目录下，修改以下配置文件前，切换到“HBASE_HOME/conf”目录。

```
cd $HBASE_HOME/conf
```

修改 hbase-env.sh 文件。

```
vim hbase-env.sh
```

修改环境变量 JAVA_HOME 为绝对路径，注意 JAVA_HOME 和 HBASE_LIBRARY_PATH 要与自己实际安装配置的一致，HBASE_MANAGES_ZK 设为 false。

```
export JAVA_HOME=/usr/local/jdk8u252-b09
export HBASE_MANAGES_ZK=false
export HBASE_LIBRARY_PATH=/usr/local/hadoop/lib/native
```

修改 hbase-site.xml 文件。

```
vim hbase-site.xml
```

添加或修改 configuration 标签范围内的部分参数。

```
<configuration>
    <property>
        <name>hbase.rootdir</name>
        <value>hdfs://name-number-0001:8020/HBase</value>
    </property>
    <property>
        <name>hbase.tmp.dir</name>
        <value>/usr/local/hbase/tmp</value>
    </property>
    <property>
        <name>hbase.cluster.distributed</name>
        <value>true</value>
    </property>
    <property>
```

北京邮电大学《大数据技术基础》课程教研室

```
        <name>hbase.unsafe.stream.capability.enforce</name>
        <value>false</value>
    </property>
    <property>
        <name>hbase.zookeeper.quorum</name>
        
<value>name-number-0002:2181,name-number-0003:2181,name-number-0004:2181</value>
    </property>
    <property>
        <name>hbase.unsafe.stream.capability.enforce</name>
        <value>false</value>
    </property>
</configuration>
```

修改 regionservers

编辑 regionservers 文件。

```
vim regionservers
```

将 regionservers 文件内容替换为 agent 节点 IP（可用主机名代替，记得改名）。

```
name-number-0002
name-number-0003
name-number-0004
```

拷贝 hdfs-site.xml

拷贝 hadoop 目录下的的 hdfs-site.xml 文件到"hbase/conf/"目录，可选择软链接或拷贝。

```
cp /usr/local/hadoop/etc/hadoop/hdfs-site.xml /usr/local/hbase/conf/hdfs-site.xml
```

拷贝 hbase-2.0.2 到 name-number-0002、name-number-0003、name-number-0004 节点的"/usr/local"目录。（也可以将压缩包拷贝到其他节点，再进行相同的配置，这样等待时间较短）

```
for i in {1..3};do scp -r /usr/local/hbase-2.0.2 root@name-number-000${i}:/usr/local/ ;done
```

分别登录到 name-number-0002、name-number-0003、name-number-0004 节点，为 hbase-2.0.2 建立软链接。

```
cd /usr/local
ln -s hbase-2.0.2 hbase
```

北京邮电大学《大数据技术基础》课程教研室

依次启动 ZooKeeper 和 Hadoop。

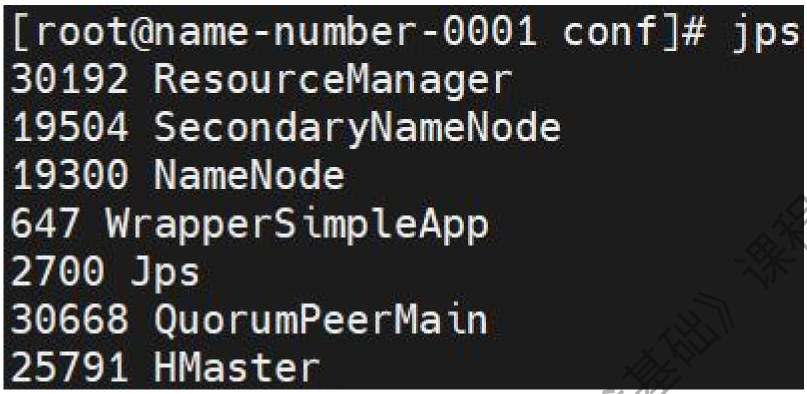
在 name-number-0001 节点上启动 HBase 集群。

```
/usr/local/hbase/bin/start-hbase.sh
```

观察进程是否都正常启动。

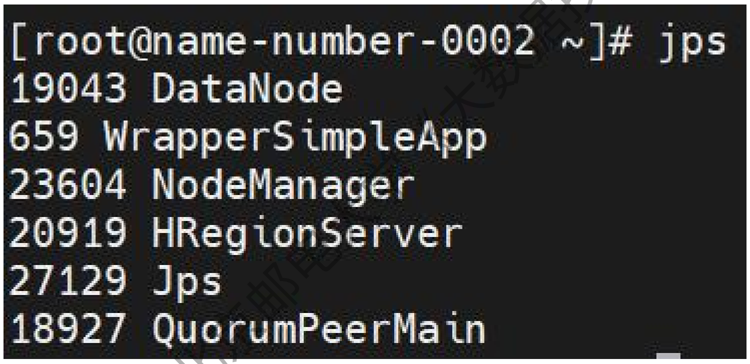
```
Jps
```

name-number-0001：



```
[root@name-number-0001 conf]# jps
30192 ResourceManager
19504 SecondaryNameNode
19300 NameNode
647 WrapperSimpleApp
2700 Jps
30668 QuorumPeerMain
25791 HMaster
```

name-number-0002：

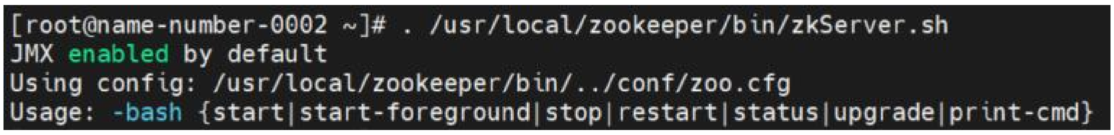


```
[root@name-number-0002 ~]# jps
19043 DataNode
659 WrapperSimpleApp
23604 NodeManager
20919 HRegionServer
27129 Jps
18927 QuorumPeerMain
```

北京邮电大学《大数据技术基础》课程教研室

### 1.1.3.4. HBase 实践

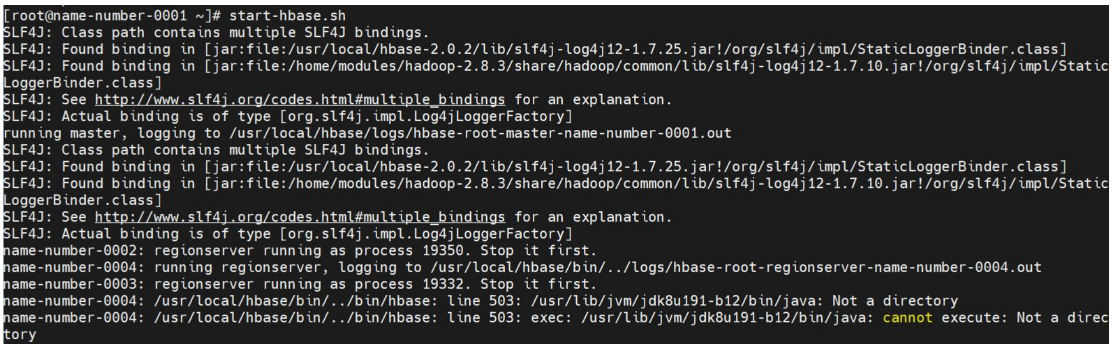
- 启动 Hadoop 集群
  在 name-number-0001 运行：
  start-dfs.sh
  start-yarn.sh
- 启动 Zookeeper 集群
  需要在 name-number-000{2..4}分别运行：. /usr/local/zookeeper/bin/zkServer.sh start



```
[root@name-number-0002 ~]# . /usr/local/zookeeper/bin/zkServer.sh
JMX enabled by default
Using config: /usr/local/zookeeper/bin/../conf/zoo.cfg
Usage: -bash {start|start-foreground|stop|restart|status|upgrade|print-cmd}
```

- 启动 HBase 集群

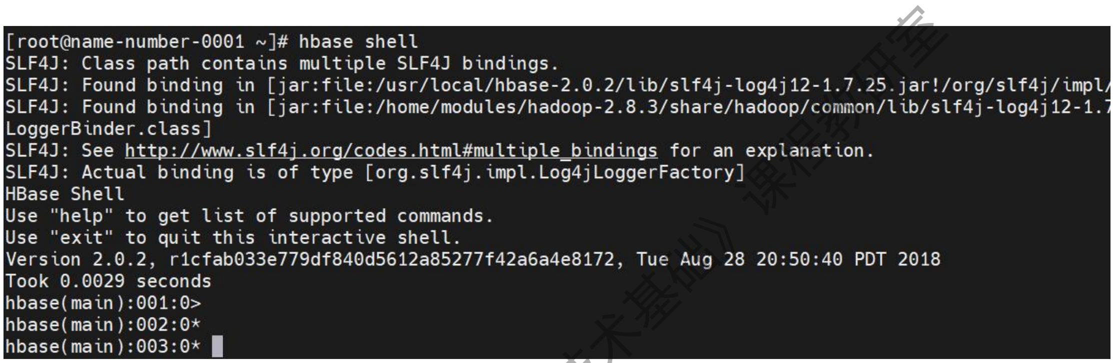
  在 name-number-0001 运行：



```
[root@name-number-0001 ~]# start-hbase.sh
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/local/hbase-2.0.2/lib/slf4j-log4j12-1.7.25.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/home/modules/hadoop-2.8.3/share/hadoop/common/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/Static
LoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
running master, logging to /usr/local/hbase/logs/hbase-root-master-name-number-0001.out
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/local/hbase-2.0.2/lib/slf4j-log4j12-1.7.25.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/home/modules/hadoop-2.8.3/share/hadoop/common/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/Static
LoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
name-number-0002: regionserver running as process 19350. Stop it first.
name-number-0004: running regionserver, logging to /usr/local/hbase/bin/../logs/hbase-root-regionserver-name-number-0004.out
name-number-0003: regionserver running as process 19332. Stop it first.
name-number-0004: /usr/local/hbase/bin/../bin/hbase: line 503: /usr/lib/jvm/jdk8u191-b12/bin/java: Not a directory
name-number-0004: /usr/local/hbase/bin/../bin/hbase: line 503: exec: /usr/lib/jvm/jdk8u191-b12/bin/java: cannot execute: Not a direc
tory
```

- 进入 HBase Shell 创建实验用表

  输入 hbase shell 进入 hbase 交互式环境：



```
[root@name-number-0001 ~]# hbase shell
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/local/hbase-2.0.2/lib/slf4j-log4j12-1.7.25.jar!/org/slf4j/impl/
SLF4J: Found binding in [jar:file:/home/modules/hadoop-2.8.3/share/hadoop/common/lib/slf4j-log4j12-1.7
LoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
HBase Shell
Use "help" to get list of supported commands.
Use "exit" to quit this interactive shell.
Version 2.0.2, r1cfab033e779df840d5612a85277f42a6a4e8172, Tue Aug 28 20:50:40 PDT 2018
Took 0.0029 seconds
hbase(main):001:0>
hbase(main):002:0*
hbase(main):003:0*
```

数据库表格设计要求：（未按要求设计扣分）

**（1）　表格命名：学号+姓名**

**（2）　行数不限定，字段名不限定**

**（3）　ROW 命名：学号+姓名+编号**

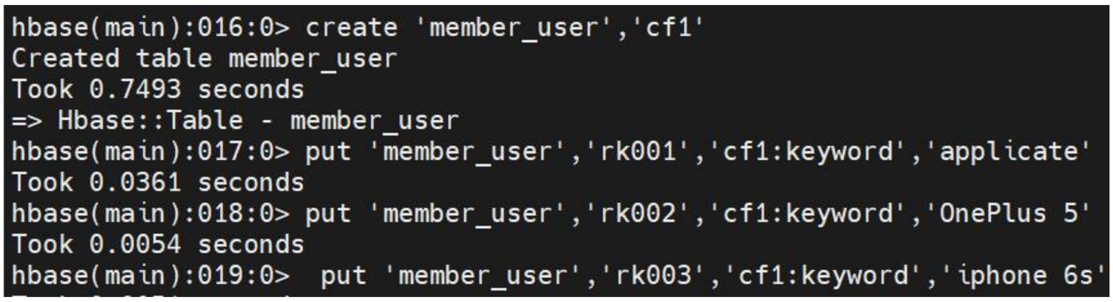
【实验报告截图要求：截图 1：数据库表格】（截图需要包含标记信息，未按要求扣分）

创建表格

```
create 'member_user','cf1'
```

向表“member_user”中插入数据

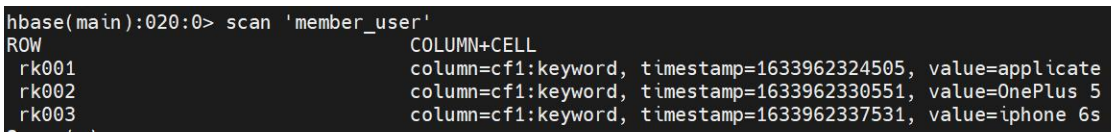
```
put 'member_user','rk001','cf1:keyword','applicate'

put 'member_user','rk002','cf1:keyword','OnePlus 5'

put 'member_user','rk003','cf1:keyword','iphone 6s'
```



```
hbase(main):016:0> create 'member_user','cf1'
Created table member_user
Took 0.7493 seconds
=> Hbase::Table - member_user
hbase(main):017:0> put 'member_user','rk001','cf1:keyword','applicate'
Took 0.0361 seconds
hbase(main):018:0> put 'member_user','rk002','cf1:keyword','OnePlus 5'
Took 0.0054 seconds
hbase(main):019:0>  put 'member_user','rk003','cf1:keyword','iphone 6s'
```

扫描整个表

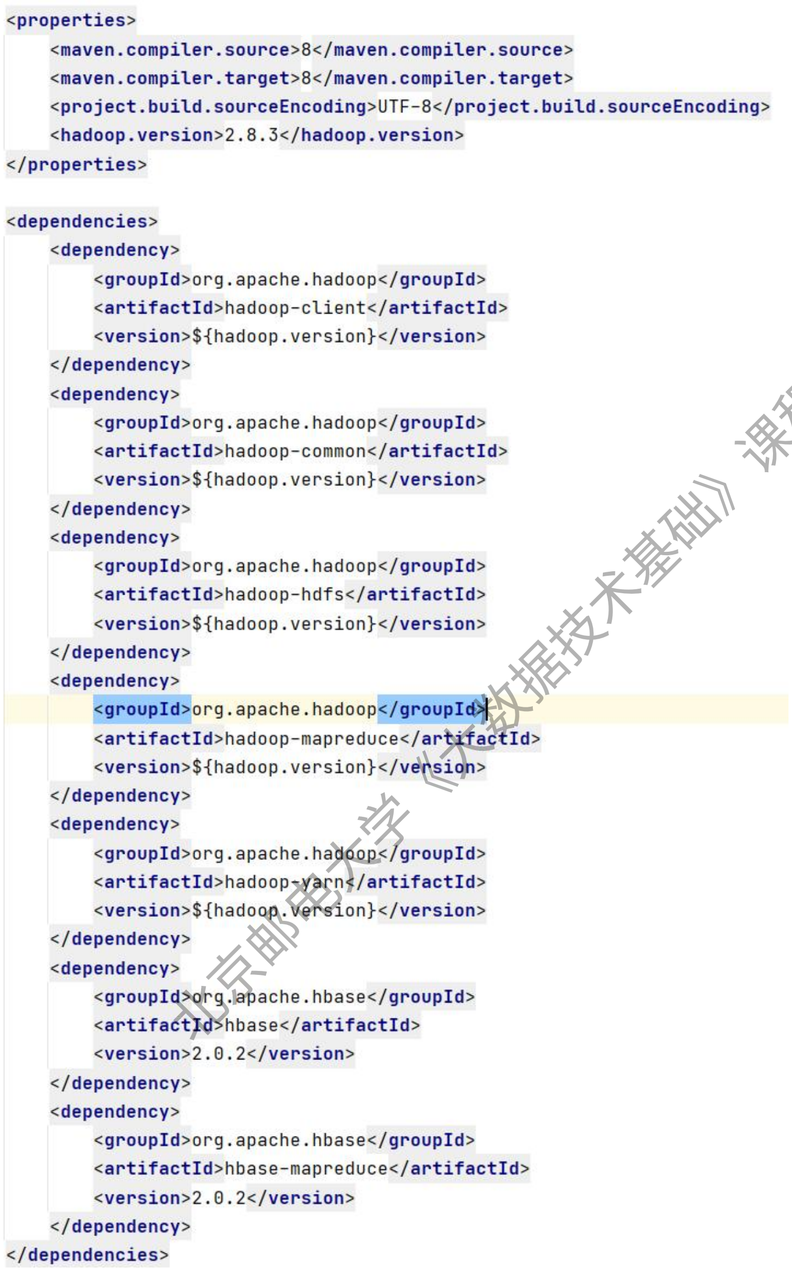
北京邮电大学《大数据技术基础》课程教研室



```
hbase(main):020:0> scan 'member_user'
ROW                           COLUMN+CELL
 rk001                        column=cf1:keyword, timestamp=1633962324505, value=applicate
 rk002                        column=cf1:keyword, timestamp=1633962330551, value=OnePlus 5
 rk003                        column=cf1:keyword, timestamp=1633962337531, value=iphone 6s
```

● 编写代码，将 Hbase 中的数据导出到 hdfs 指定目录

打开 IDEA，新建 maven 工程，工程名 MyHBase，编写 pom.xml 文件添加依赖

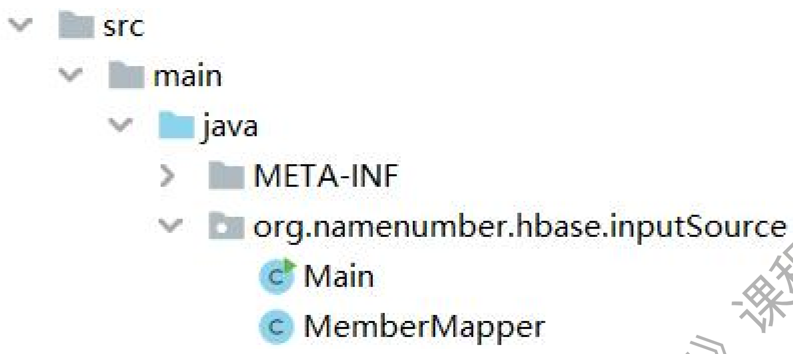


```xml
<properties>
    <maven.compiler.source>8</maven.compiler.source>
    <maven.compiler.target>8</maven.compiler.target>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <hadoop.version>2.8.3</hadoop.version>
</properties>

<dependencies>
    <dependency>
        <groupId>org.apache.hadoop</groupId>
        <artifactId>hadoop-client</artifactId>
        <version>${hadoop.version}</version>
    </dependency>
    <dependency>
        <groupId>org.apache.hadoop</groupId>
        <artifactId>hadoop-common</artifactId>
        <version>${hadoop.version}</version>
    </dependency>
    <dependency>
        <groupId>org.apache.hadoop</groupId>
        <artifactId>hadoop-hdfs</artifactId>
        <version>${hadoop.version}</version>
    </dependency>
    <dependency>
        <groupId>org.apache.hadoop</groupId>
        <artifactId>hadoop-mapreduce</artifactId>
        <version>${hadoop.version}</version>
    </dependency>
    <dependency>
        <groupId>org.apache.hadoop</groupId>
        <artifactId>hadoop-yarn</artifactId>
        <version>${hadoop.version}</version>
    </dependency>
    <dependency>
        <groupId>org.apache.hbase</groupId>
        <artifactId>hbase</artifactId>
        <version>2.0.2</version>
    </dependency>
    <dependency>
        <groupId>org.apache.hbase</groupId>
        <artifactId>hbase-mapreduce</artifactId>
        <version>2.0.2</version>
    </dependency>
</dependencies>
```

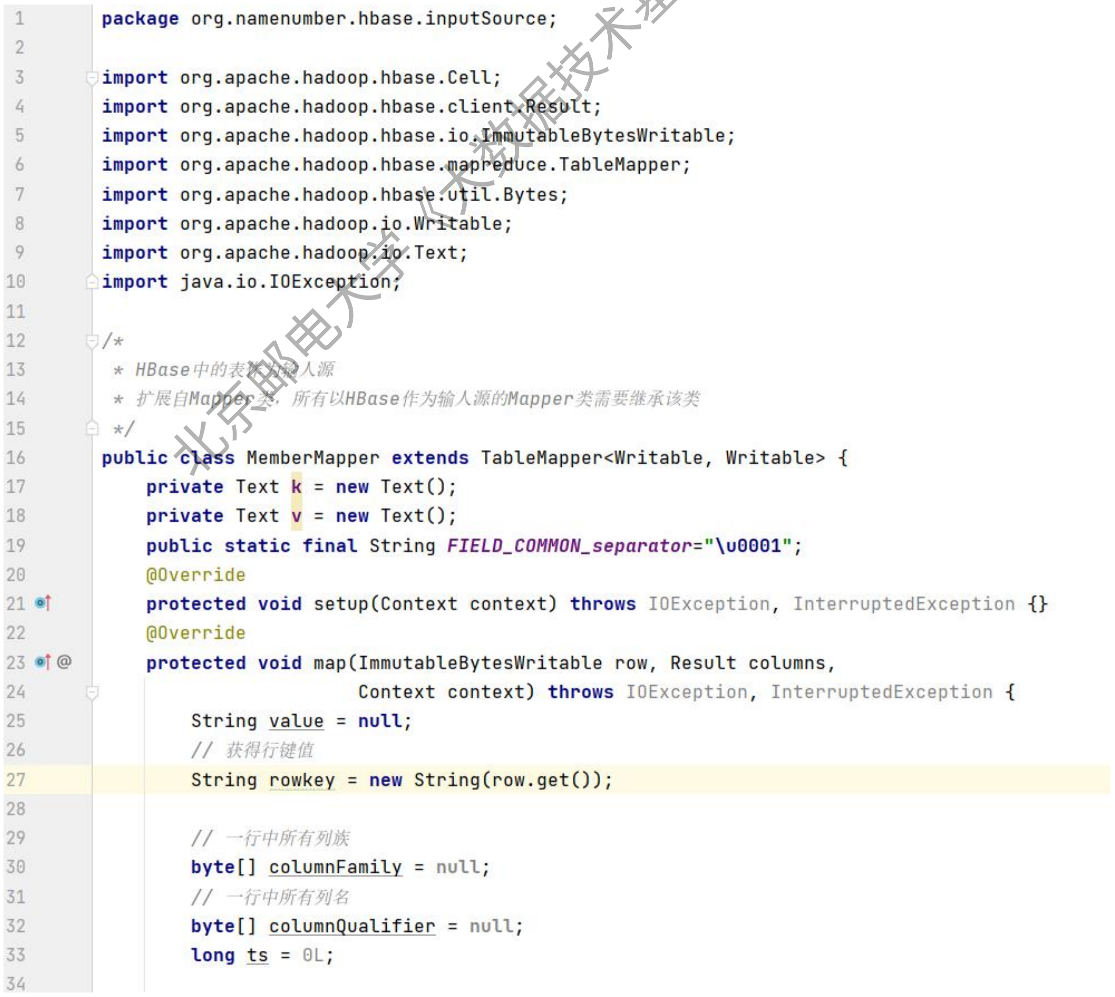
北京邮电大学《大数据技术基础》课程教研室

点击下图所示按钮自动下载依赖

在 src/java 目录下新建 package，名称 org/namenumber/hbase/inputSource（namenumber 改成对应的姓名缩写+学号）

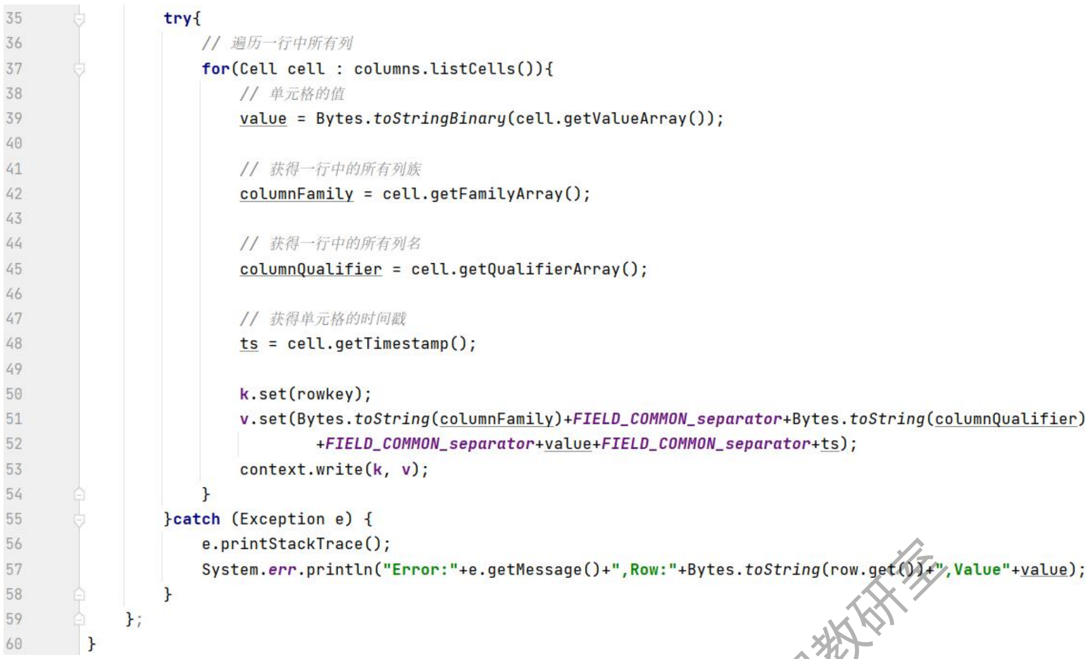


新建类 MemberMapper，完整代码如下



```java
package org.namenumber.hbase.inputSource;

import org.apache.hadoop.hbase.Cell;
import org.apache.hadoop.hbase.client.Result;
import org.apache.hadoop.hbase.io.ImmutableBytesWritable;
import org.apache.hadoop.hbase.mapreduce.TableMapper;
import org.apache.hadoop.hbase.util.Bytes;
import org.apache.hadoop.io.Writable;
import org.apache.hadoop.io.Text;
import java.io.IOException;

/*
 * HBase中的表作为输入源
 * 扩展自Mapper类，所有以HBase作为输入源的Mapper类需要继承该类
 */
public class MemberMapper extends TableMapper<Writable, Writable> {
    private Text k = new Text();
    private Text v = new Text();
    public static final String FIELD_COMMON_separator="\u0001";
    @Override
    protected void setup(Context context) throws IOException, InterruptedException {}
    @Override
    protected void map(ImmutableBytesWritable row, Result columns,
                       Context context) throws IOException, InterruptedException {
        String value = null;
        // 获得行键值
        String rowkey = new String(row.get());

        // 一行中所有列族
        byte[] columnFamily = null;
        // 一行中所有列名
        byte[] columnQualifier = null;
        long ts = 0L;

```

北京邮电大学《大数据技术基础》课程教研室

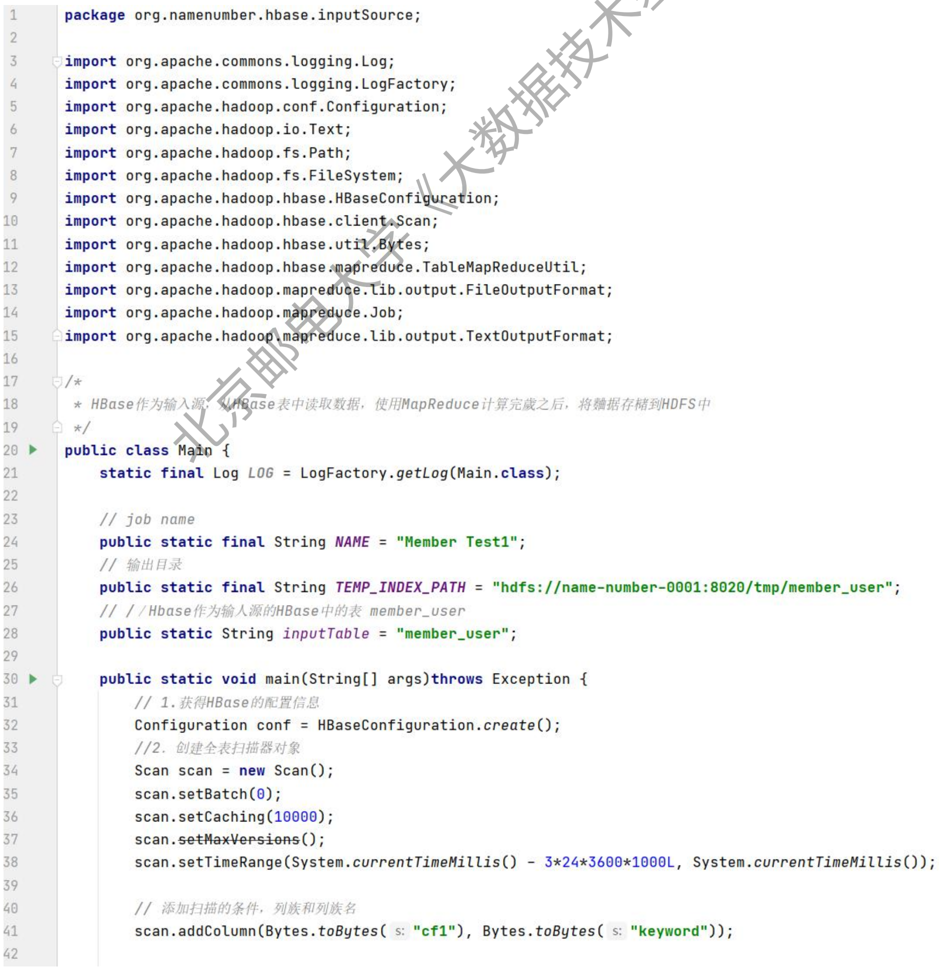


```
                try{
                    // 遍历一行中所有列
                    for(Cell cell : columns.listCells()){
                        // 单元格的值
                        value = Bytes.toStringBinary(cell.getValueArray());

                        // 获得一行中的所有列族
                        columnFamily = cell.getFamilyArray();

                        // 获得一行中的所有列名
                        columnQualifier = cell.getQualifierArray();

                        // 获得单元格的时间戳
                        ts = cell.getTimestamp();

                        k.set(rowkey);
                        v.set(Bytes.toString(columnFamily)+FIELD_COMMON_separator+Bytes.toString(columnQualifier)
                                +FIELD_COMMON_separator+value+FIELD_COMMON_separator+ts);
                        context.write(k, v);
                    }
                }catch (Exception e) {
                    e.printStackTrace();
                    System.err.println("Error:"+e.getMessage()+",Row:"+Bytes.toString(row.get())+",Value"+value);
                }
            };
        }
```

【实验报告截图要求：截图 2：完整 Mapper 代码】

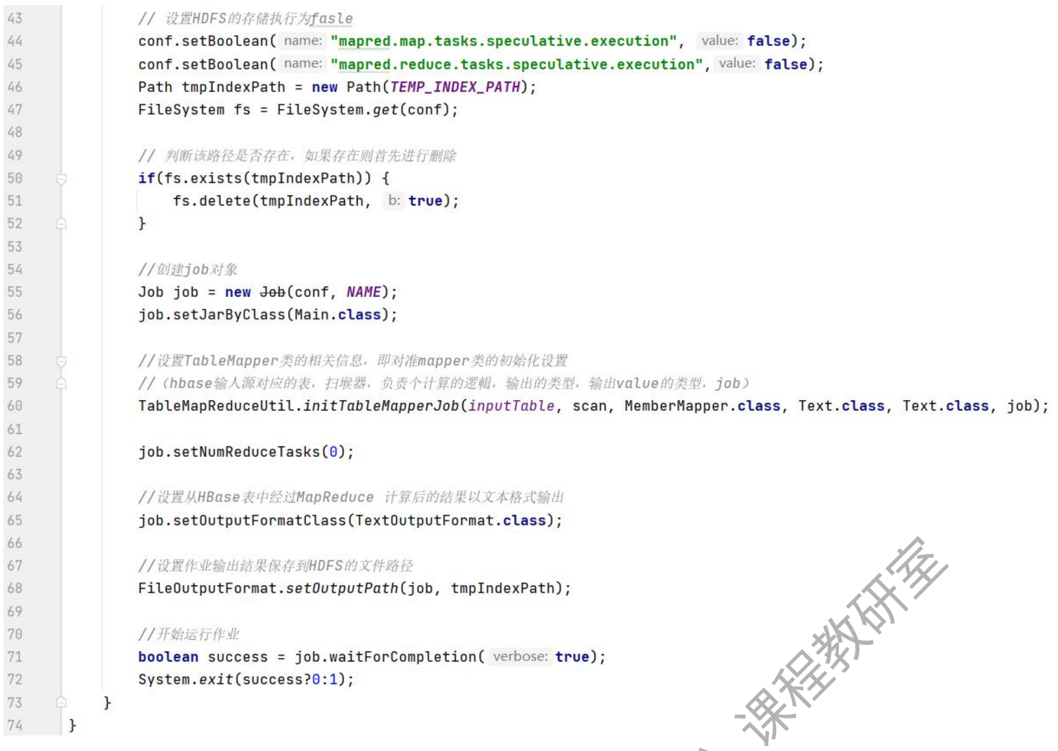
（截图需要包含标记信息）

新建类 Main，完整代码如下：



```
package org.namenumber.hbase.inputSource;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.hbase.client.Scan;
import org.apache.hadoop.hbase.util.Bytes;
import org.apache.hadoop.hbase.mapreduce.TableMapReduceUtil;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

/*
 * HBase作为输入源，从HBase表中读取数据，使用MapReduce计算完成之后，将数据存储到HDFS中
 */
public class Main {
    static final Log LOG = LogFactory.getLog(Main.class);

    // job name
    public static final String NAME = "Member Test1";
    // 输出目录
    public static final String TEMP_INDEX_PATH = "hdfs://name-number-0001:8020/tmp/member_user";
    // //Hbase作为输入源的HBase中的表 member_user
    public static String inputTable = "member_user";

    public static void main(String[] args)throws Exception {
        // 1.获得HBase的配置信息
        Configuration conf = HBaseConfiguration.create();
        //2. 创建全表扫描器对象
        Scan scan = new Scan();
        scan.setBatch(0);
        scan.setCaching(10000);
        scan.~~setMaxVersions~~();
        scan.setTimeRange(System.currentTimeMillis() - 3*24*3600*1000L, System.currentTimeMillis());

        // 添加扫描的条件，列族和列族名
        scan.addColumn(Bytes.toBytes( s: "cf1"), Bytes.toBytes( s: "keyword"));

```

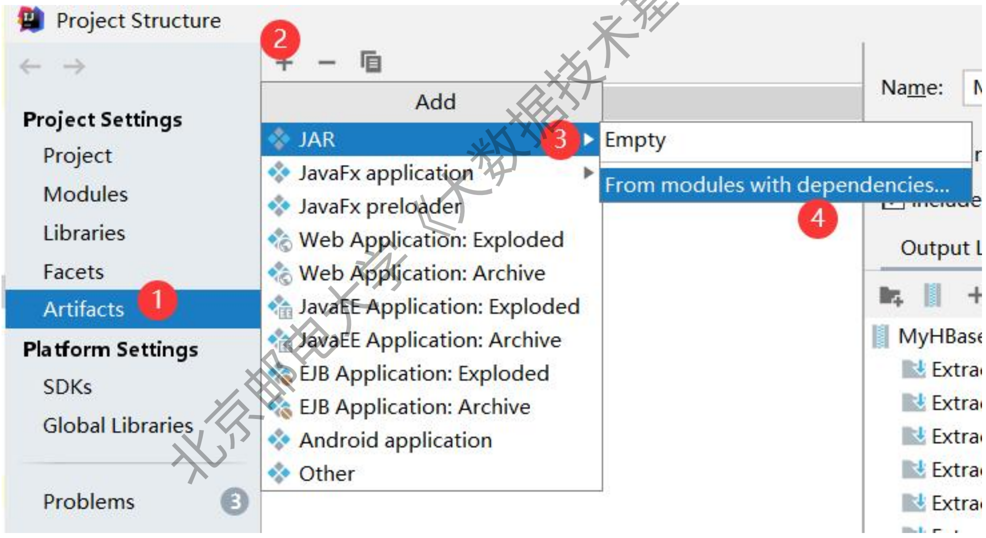
北京邮电大学《大数据技术基础》课程教研室



```
        // 设置HDFS的存储执行为fasle
        conf.setBoolean( name: "mapred.map.tasks.speculative.execution",  value: false);
        conf.setBoolean( name: "mapred.reduce.tasks.speculative.execution",  value: false);
        Path tmpIndexPath = new Path(TEMP_INDEX_PATH);
        FileSystem fs = FileSystem.get(conf);

        // 判断该路径是否存在，如果存在则首先进行删除
        if(fs.exists(tmpIndexPath)) {
            fs.delete(tmpIndexPath,  b: true);
        }

        //创建job对象
        Job job = new Job(conf, NAME);
        job.setJarByClass(Main.class);

        //设置TableMapper类的相关信息，即对准mapper类的初始化设置
        //（hbase输入源对应的表，扫描器，负责个计算的逻辑，输出的类型，输出value的类型，job）
        TableMapReduceUtil.initTableMapperJob(inputTable, scan, MemberMapper.class, Text.class, Text.class, job);

        job.setNumReduceTasks(0);

        //设置从HBase表中经过MapReduce 计算后的结果以文本格式输出
        job.setOutputFormatClass(TextOutputFormat.class);

        //设置作业输出结果保存到HDFS的文件路径
        FileOutputFormat.setOutputPath(job, tmpIndexPath);

        //开始运行作业
        boolean success = job.waitForCompletion( verbose: true);
        System.exit(success?0:1);
    }
}
```

- 打包程序,导出 jar 包

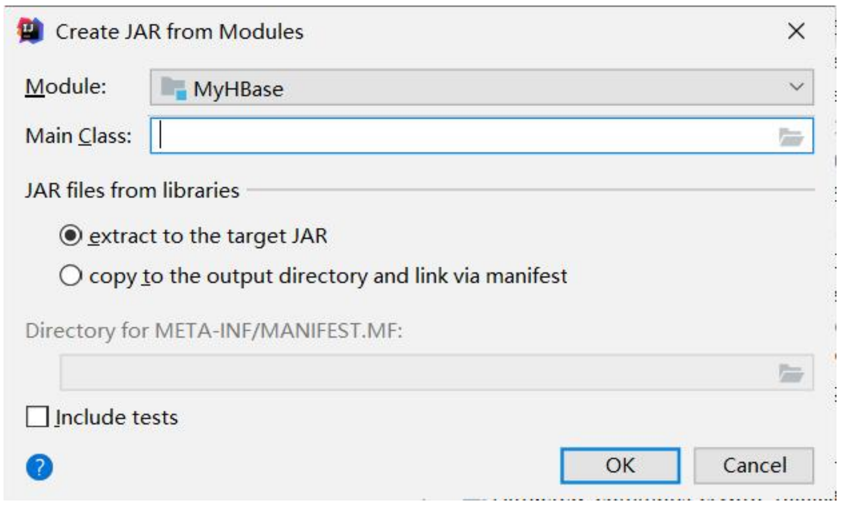
  File-Project Structure-Project Settings-Artifacts



选择 Main 类

北京邮电大学《大数据技术基础》课程教研室

华为云智能基座在线实验答疑
QQ群1：471519925（2021大数据技术基础）、QQ群2：125545864（2022-大数据技术基础）、QQ群3：295477031（2023-大数据技术基础）


Create JAR from Modules

Module: MyHBase

Main Class:

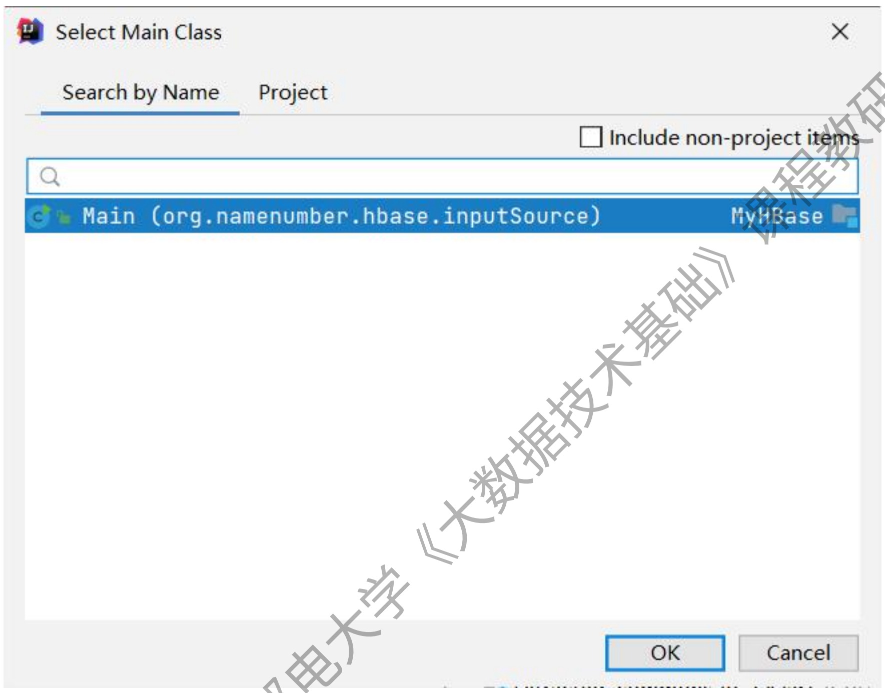
JAR files from libraries

(●) extract to the target JAR

( ) copy to the output directory and link via manifest

Directory for META-INF/MANIFEST.MF:

☐ Include tests

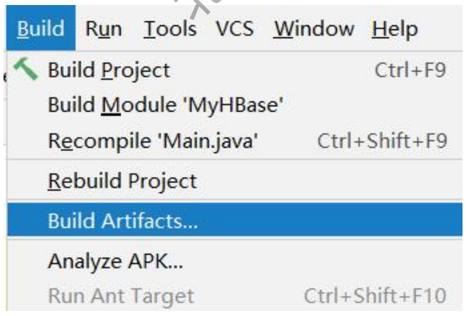

Select Main Class

☐ Include non-project items

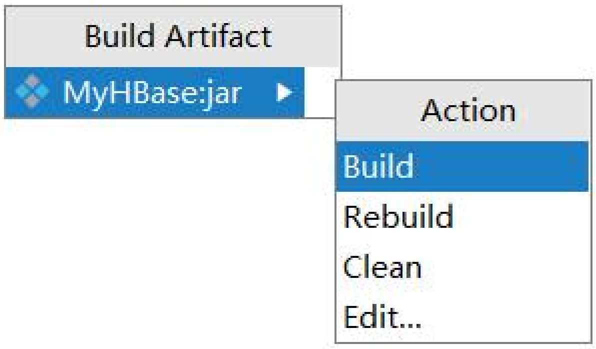
Main (org.namenumber.hbase.inputSource) MyHBase

点击 Apply 点击 OK

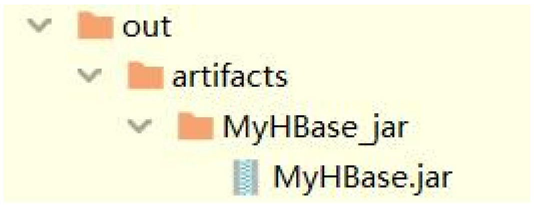
点击 Build-Build Artifacts



北京邮电大学《大数据技术基础》课程教研室

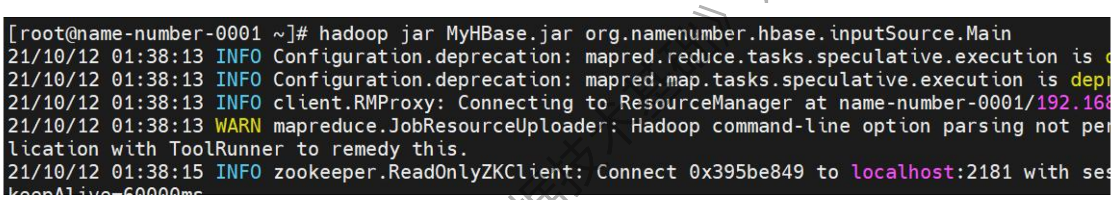


点击 Buiild，可以看到生成了 jar 包



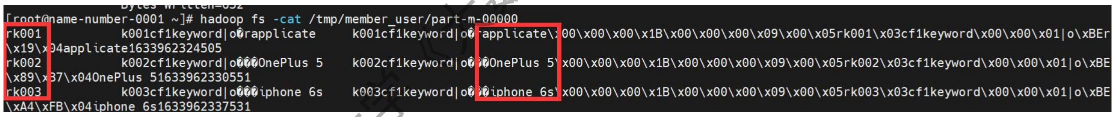
将 jar 包通过 winscp 或 scp 命令复制到服务器 name-number-0001 上

运行 jar 包



```
[root@name-number-0001 ~]# hadoop jar MyHBase.jar org.namenumber.hbase.inputSource.Main
21/10/12 01:38:13 INFO Configuration.deprecation: mapred.reduce.tasks.speculative.execution is 
21/10/12 01:38:13 INFO Configuration.deprecation: mapred.map.tasks.speculative.execution is depr
21/10/12 01:38:13 INFO client.RMProxy: Connecting to ResourceManager at name-number-0001/192.168
21/10/12 01:38:13 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not per
lication with ToolRunner to remedy this.
21/10/12 01:38:15 INFO zookeeper.ReadOnlyZKClient: Connect 0x395be849 to localhost:2181 with ses
```

查看结果



```
[root@name-number-0001 ~]# hadoop fs -cat /tmp/member_user/part-m-00000
rk001          k001cf1keyword|o�applicate     k001cf1keyword|o�applicate\x00\x00\x00\x1B\x00\x00\x00\x09\x00\x05rk001\x03cf1keyword\x00\x00\x01|o\xBEr
\x19\x04applicate1633962324505
rk002          k002cf1keyword|o���OnePlus 5   k002cf1keyword|o��OnePlus 5\x00\x00\x00\x1B\x00\x00\x00\x09\x00\x05rk002\x03cf1keyword\x00\x00\x01|o\xBE
\x89\x37\x04OnePlus 51633962330551
rk003          k003cf1keyword|o���iphone 6s   k003cf1keyword|o��iphone 6s\x00\x00\x00\x1B\x00\x00\x00\x09\x00\x05rk003\x03cf1keyword\x00\x00\x01|o\xBE
\xA4\xFB\x04iphone 6s1633962337531
```

【实验报告截图要求：截图 3：结果截图】（截图需要包含标记信息，未按要求扣分）

## 1.1.4. 实验结果与评分标准

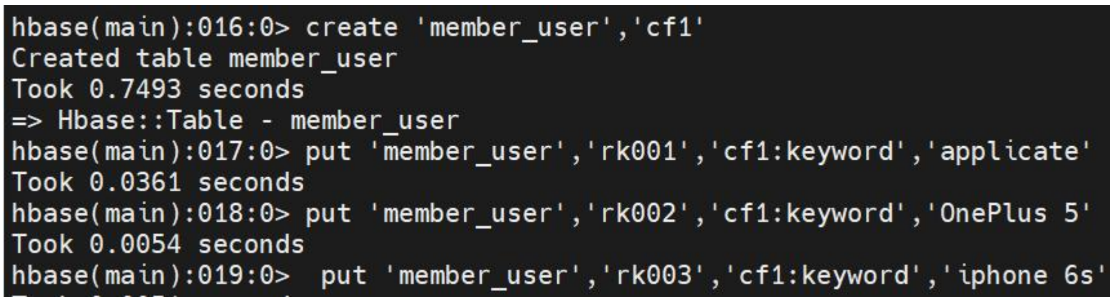
实验结束后应得到：1 个安装好 HBase 和 Zookeeper 集群，1 个 HBase 数据库和 1 个 MapReduce 程序 jar 包。

完成 MapReduce 分布式数据处理实践。

实验评分标准，提交的实验报告中应包含：

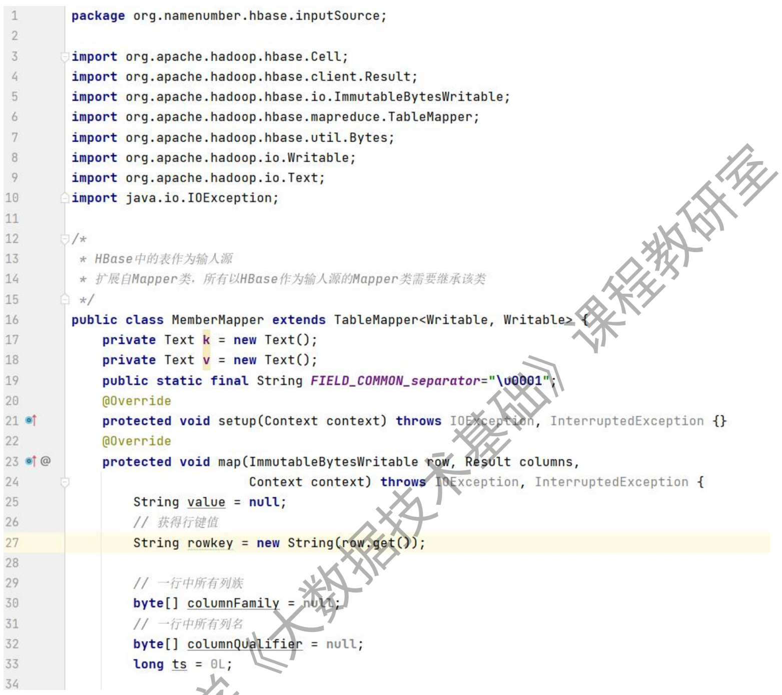
（1）HBase 数据库表构建截图，数据库表格设计要求：表格命名学号+姓名；行数不限定，字段名不限定；ROW 命名：学号+姓名+编号。

北京邮电大学《大数据技术基础》课程教研室



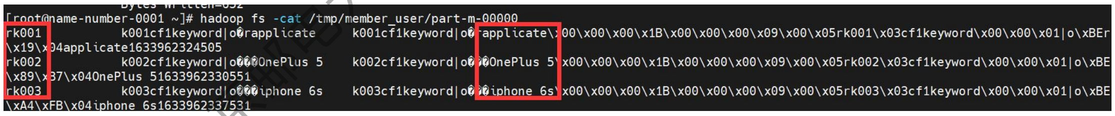
```
hbase(main):016:0> create 'member_user','cf1'
Created table member_user
Took 0.7493 seconds
=> Hbase::Table - member_user
hbase(main):017:0> put 'member_user','rk001','cf1:keyword','applicate'
Took 0.0361 seconds
hbase(main):018:0> put 'member_user','rk002','cf1:keyword','OnePlus 5'
Took 0.0054 seconds
hbase(main):019:0>  put 'member_user','rk003','cf1:keyword','iphone 6s'
```

（2）完整 Mapper 代码截图。对代码提供解释。



```java
package org.namenumber.hbase.inputSource;

import org.apache.hadoop.hbase.Cell;
import org.apache.hadoop.hbase.client.Result;
import org.apache.hadoop.hbase.io.ImmutableBytesWritable;
import org.apache.hadoop.hbase.mapreduce.TableMapper;
import org.apache.hadoop.hbase.util.Bytes;
import org.apache.hadoop.io.Writable;
import org.apache.hadoop.io.Text;
import java.io.IOException;

/*
 * HBase中的表作为输入源
 * 扩展自Mapper类，所有以HBase作为输入源的Mapper类需要继承该类
 */
public class MemberMapper extends TableMapper<Writable, Writable> {
    private Text k = new Text();
    private Text v = new Text();
    public static final String FIELD_COMMON_separator="\u0001";
    @Override
    protected void setup(Context context) throws IOException, InterruptedException {}
    @Override
    protected void map(ImmutableBytesWritable row, Result columns,
                       Context context) throws IOException, InterruptedException {
        String value = null;
        // 获得行键值
        String rowkey = new String(row.get());

        // 一行中所有列族
        byte[] columnFamily = null;
        // 一行中所有列名
        byte[] columnQualifier = null;
        long ts = 0L;

```

（3）提供运行结果截图。（截图需要包含标记信息）



```
[root@name-number-0001 ~]# hadoop fs -cat /tmp/member_user/part-m-00000
rk001           k001cf1keyword|o�rapplicate     k001cf1keyword|o�rapplicate\x00\x00\x00\x1B\x00\x00\x00\x09\x00\x05rk001\x03cf1keyword\x00\x00\x01|o\xBEr
\x19\x04applicate1633962324505
rk002           k002cf1keyword|o���OnePlus 5    k002cf1keyword|o���OnePlus 5\x00\x00\x00\x1B\x00\x00\x00\x09\x00\x05rk002\x03cf1keyword\x00\x00\x01|o\xBE
\x89\x37\x04OnePlus 51633962330551
rk003           k003cf1keyword|o���iphone 6s    k003cf1keyword|o���iphone 6s\x00\x00\x00\x1B\x00\x00\x00\x09\x00\x05rk003\x03cf1keyword\x00\x00\x01|o\xBE
\xA4\xFB\x04iphone 6s1633962337531
```

北京邮电大学《大数据技术基础》课程教研室