实验三：HBase、ZooKeeper 安装与 HBase 应用实践

一、实验描述

基于 Hadoop 集群环境，安装 HBase、ZooKeeper，实践 HBase 基本使用。

二、实验目的

掌握 HBase、ZooKeeper 的安装与使用，使用 MapReduce 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。

三、实验环境

1. Docker 25.0.3
2. Hadoop 3.3.6
   ZooKeeper 3.9.2
   HBase 2.5.8
3. JDK 版本：1.8.0*;
4. IDEA 2023.3.4

四、实验步骤

4.1 环境配置

步骤一：

下载 Hadoop、HBase、Zookeeper 对应的版本（可于 https://mirrors.tuna.tsinghua.edu.cn/apache/ 等镜像地址下载），将安装包放入 docker 配置文件夹中，`ls`结果如下：

```
    Length Name
    ------ ----
           hadoop_config
           hbase_config
           scripts
           zookeeper_config
  20213309 apache-zookeeper-3.9.2-bin.tar.gz
      3532 Dockerfile
 730107476 hadoop-3.3.6.tar.gz
 322479779 hbase-2.5.8-bin.tar.gz
```

查看服务器的 CPU 架构：

```
uname -m
```

根据返回结果，修改 **Dockerfile** 中对应内容为 **arm64** 或 **amd64**

```
12 # 安装JDK等
13 RUN apt update && \
14     apt install -y openjdk-8-jdk wget openssh-server nano vim net-tools iputils-ping
15 ENV JAVA_HOME=/usr/lib/jvm/java-8-openjdk-arm64
```

```
45 # 配置环境变量
46 # ENV CLASSPATH=/usr/lib/jvm/java-8-openjdk-arm64/lib
47 # hadoop
48 RUN echo "slave1" > /root/hadoop/etc/hadoop/workers
49 RUN echo "slave2" >> /root/hadoop/etc/hadoop/workers
50 RUN echo "slave3" >> /root/hadoop/etc/hadoop/workers
```

```
75 # hadoop
76 COPY hadoop_config/* /root/hadoop/etc/hadoop/
77 RUN sed -i '1i export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-arm64' /root/hadoop/etc/hadoop/hadoop-env.sh
```

```
83 # hbase
84 COPY hbase_config/* /root/hbase/conf/
85 RUN echo "export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-arm64" >> /root/hbase/conf/hbase-env.sh
86 RUN echo "export HBASE_MANAGES_ZK=false" >> /root/hbase/conf/hbase-env.sh
87 RUN echo "export HBASE_LIBRARY_PATH=/root/hadoop/lib/native" >> /root/hbase/conf/hbase-env.sh
88 RUN echo 'export HBASE_DISABLE_HADOOP_CLASSPATH_LOOKUP="true"' >> /root/hbase/conf/hbase-env.sh
```

步骤二：

创建镜像并启动容器

在文件夹中执行以下命令创建 image

```
docker build . -t hadoop
```

在四个终端中依次执行

```
docker run -it -h master --name=master -p 9870:9870 -p 16010:16010
                    --network bridge hadoop
  docker run -it -h slave1 --name=slave1 --network bridge hadoop
  docker run -it -h slave2 --name=slave2 --network bridge hadoop
  docker run -it -h slave3 --name=slave3 --network bridge hadoop
```

步骤三：

当 master、slave1、slave2、slave3 的 ip 分别为 172.17.0.2、3、4、5 时，在四个节点上分别执行：

```
inithosts.sh 1/2/3/4
```

否则：a. 根据 ifconfig 输出修改 scripts/inithosts.sh 和 zookeeper_config/zoo.cfg 中的 ip；

或 b. 手动修改 hosts 文件（注释 127.0.0.1，写入四个节点的名称及对应 ip），将 1、2、3、4 分别写入四个节点的/root/zookeeper/tmp/myid，修改/root/zookeeper/conf/zoo.cfg 中的 dataDir 如下：

```
10  # do not use /tmp for storage, /tmp here is
11  # example sakes.
12  dataDir=/root/zookeeper/tmp
13  # the port at which the clients will connec
14  clientPort=2181
15  # the maximum number of client connections.
```

文件末尾增加如下四行内容（分别对应前一步的 1、2、3、4）：

```
38  server.1=172.17.0.2:2888:3888
39  server.2=172.17.0.3:2888:3888
40  server.3=172.17.0.4:2888:3888
41  server.4=172.17.0.5:2888:3888
```

接着将所在节点的 ip 修改为 0.0.0.0；例如在 master 节点中，最终结果应如下：

```
38  server.1=0.0.0.0:2888:3888
39  server.2=172.17.0.3:2888:3888
40  server.3=172.17.0.4:2888:3888
41  server.4=172.17.0.5:2888:3888
```

在 slave1 节点中，最终结果应如下：（slave2/3 同理）

```
38  server.1=172.17.0.2:2888:3888
39  server.2=0.0.0.0:2888:3888
40  server.3=172.17.0.4:2888:3888
41  server.4=172.17.0.5:2888:3888
```

步骤四：

启动 hadoop，同实验一、二，在 master 节点依次执行

```
1. hdfs namenode -format
2. start-all.sh
```

启动 ZooKeeper，在四个节点分别执行

```
zkServer.sh start
```

执行 `zkServer.sh status`，查看状态（具体谁是 `leader` 谁是 `follower` 无所谓，不影响最终实验）

```
root@master:~# zkServer.sh status
ZooKeeper JMX enabled by default
Using config: /root/zookeeper/bin/../conf/zoo.cfg
Client port found: 2181. Client address: localhost. Client SSL: false.
Mode: follower
```

```
root@slave2:~# zkServer.sh status
ZooKeeper JMX enabled by default
Using config: /root/zookeeper/bin/../conf/zoo.cfg
Client port found: 2181. Client address: localhost. Client SSL: false.
Mode: leader
```

启动 HBase，在 master 节点执行

```
start-hbase.sh
```

执行 `jps` 查看状态

```
root@master:~# jps
224 NameNode
436 SecondaryNameNode
792 QuorumPeerMain
636 ResourceManager
1166 HMaster
1342 Jps
```

```
root@slave2:~# jps
1171 Jps
182 NodeManager
74 DataNode
442 HRegionServer
316 QuorumPeerMain
```

4.2 HBase 实践

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

执行`hbase shell`进入交互环境，创建表格：

```
create 'example_table','cf1'
```

（注意 example_table 表名称改为学号+姓名）

向表“example_table”中插入数据：（注意 rk001/rk002/rk003 改为学号+姓名+编号）

```
put 'example_table','rk001','cf1:keyword','Honor 20'
put 'example_table','rk002','cf1:keyword','Galaxy S21'
put 'example_table','rk003','cf1:keyword','Xiaomi 14'
```

查看表中数据添加情况：

```
scan 'example_table'
```

（注意 example_table 表名称改为学号+姓名）

```
root@master:~# hbase shell
HBase Shell
Use "help" to get list of supported commands.
Use "exit" to quit this interactive shell.
For Reference, please visit: http://hbase.apache.org/2.0/book.html#shell
Version 2.5.8, r37444de6531b1bdabf2e445c83d0268ab1a6f919, Thu Feb 29 15:37:32 PST 2024
Took 0.0017 seconds
hbase:001:0> create 'example_table','cf1'
Created table example_table
Took 2.7870 seconds
=> Hbase::Table - example_table
hbase:002:0> put 'example_table','rk001','cf1:keyword','Honor 20'
Took 0.2141 seconds
hbase:003:0> put 'example_table','rk002','cf1:keyword','Galaxy S21'
Took 0.0135 seconds
hbase:004:0> put 'example_table','rk003','cf1:keyword','Xiaomi 14'
Took 0.0150 seconds
hbase:005:0> scan 'example_table'
ROW                     COLUMN+CELL
 rk001                  column=cf1:keyword, timestamp=2024-04-26T11:47:55.445, value=Honor 20
 rk002                  column=cf1:keyword, timestamp=2024-04-26T11:48:06.952, value=Galaxy S21
 rk003                  column=cf1:keyword, timestamp=2024-04-26T11:48:15.741, value=Xiaomi 14
3 row(s)
Took 0.0543 seconds
```

【提交 1：数据库表格截图】（截图需要包含标记信息，未按要求扣分）

4.3 程序编写与打包

步骤一：

通过 IDEA 新建 maven 工程，编写 pom 依赖，依赖如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>org.example</groupId>
    <artifactId>MyWordCount</artifactId>
    <version>1.0-SNAPSHOT</version>

    <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>3.3.6</hadoop.version>
        <hbase.version>2.5.8</hbase.version>
```

```
    </properties>
    <dependencies>
        <dependency>
            <groupId>log4j</groupId>
            <artifactId>log4j</artifactId>
            <version>1.2.17</version>
        </dependency>
        <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>
            <type>pom</type>
        </dependency>
        <dependency>
            <groupId>org.apache.hbase</groupId>
            <artifactId>hbase</artifactId>
```

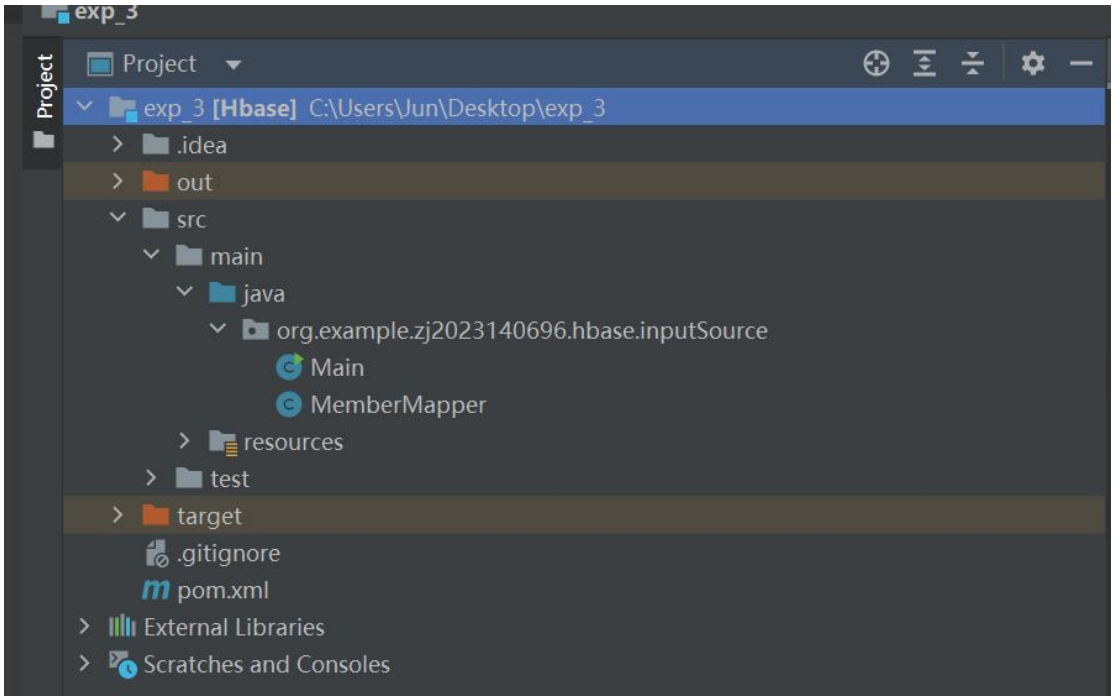
```
            <version>${hbase.version}</version>
            <type>pom</type>
        </dependency>
        <dependency>
            <groupId>org.apache.hbase</groupId>
            <artifactId>hbase-client</artifactId>
            <version>${hbase.version}</version>
        </dependency>
        <dependency>
            <groupId>org.apache.hbase</groupId>
            <artifactId>hbase-mapreduce</artifactId>
            <version>${hbase.version}</version>
        </dependency>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>RELEASE</version>
            <scope>test</scope>
        </dependency>

    </dependencies>

</project>
```

步骤二：

在 src/main/java 下新建 package，名称为 org.namenumber.hbase.inputSource（namenumber 修改为对应的姓名缩写+学号）



步骤三：

新建 MemberMapper 类和 Main 类，完成代码

MemberMapper：

```
package org.namenumber.hbase.inputSource; // TODO: namenumber  改成姓名缩写+学号

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;

		try {
			// 便利一行中所有列
			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);
            }
        }
}
```

Main：

```
package org. namenumber.hbase.inputSource; // TODO: namenumber 改成姓名缩写+学号

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.Job;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
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，将数据储存到 HDFS 中
 */
public class Main {
    public 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://master:8020/tmp/member_user";
// TODO: member_user 改成学号+姓名即表名     // HBase 作为输入源的 HBase 中的表
    public static String inputTable = "member_user"; // TODO:member_user 改成学号+姓名即表名

    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("cf1"), Bytes.toBytes("keyword"));
        // 设置 HDFS 的存储执行为 fasle
```

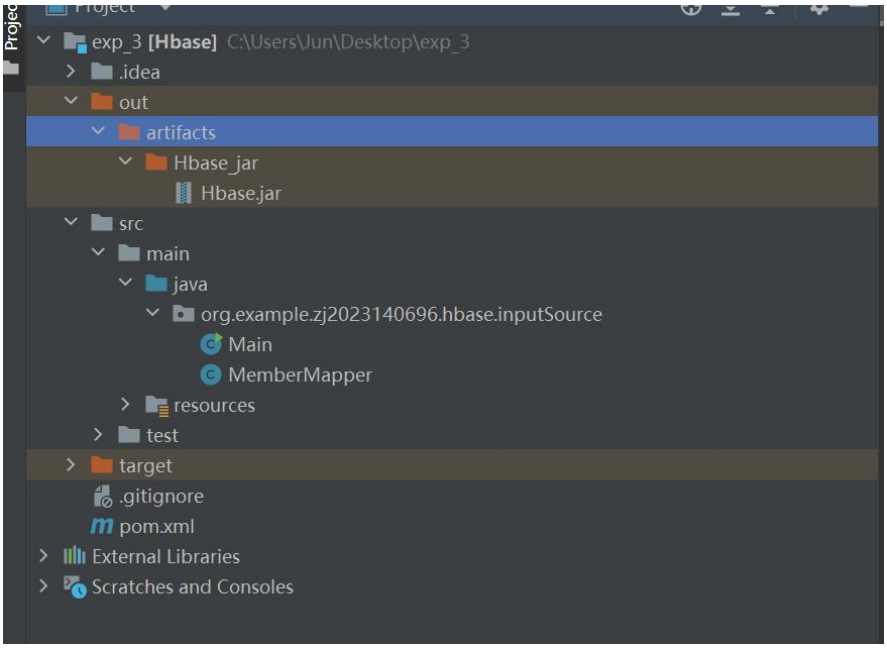
```
        conf.setBoolean("mapred.map.tasks.speculative.execution", false);
        conf.setBoolean("mapred.reduce.tasks.speculative.execution", false);
        Path tmpIndexPath = new Path(TEMP_INDEX_PATH);
        FileSystem fs = FileSystem.get(conf);
        // 判断该路径是否存在，如果存在则首先进行删除
        if (fs.exists(tmpIndexPath)) {
            fs.delete(tmpIndexPath, 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(true);
        System.exit(success ? 0 : 1);
    }
}
```

【提交 2：MemberMapper 和 Main 代码】

步骤四：

程序打包及上传，同实验二，具体见实验二报告 4.3 节

之后会在 out 文件夹下生成 jar 包



通过 docker cp 命令将 jar 包上传到 master 节点

```
[root@zj-2023140696 ~]# docker cp MyWordCount.jar master:/root/
Successfully copied 115MB to master:/root/
```

步骤五：

运行并查看结果

执行程序：

`hadoop jar hbase.jar org.namenumber.hbase.inputSource.Main` （注意 `namenumber` 改成姓名缩写+学号）

查看结果：

`hadoop fs -cat /tmp/example_table/part-m-00000` （注意 `example_table` 表名称改为学号+姓名）

```
root@master:~# hadoop fs -cat /tmp/example_table/part-m-00000
2024-04-26 11:55:13,629 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using buil
tin-java classes where applicable
rk001  |k001└cf1keyword ...Honor 20 k001└cf1keyword ...Honor 20 \x00\x00\x00\x1B\x00\x00\x00\x08\x00\x05rk001\x03c
f1keyword\x00\x00\x01\x8F\x1A;\xC3\xB5\x04Honor 20 1714132075445
rk002
|k002└cf1keyword ...Galaxy S21
|k002└cf1keyword ...Galaxy S21 \x00\x00\x00\x1B\x00\x00\x00\x0A\x00\x05rk002\x03cf1keyword\x00\x00\x01\x8F\x1A;\xF0\
xA8\x04Galaxy S21 1714132086952
rk003          |k003└cf1keyword ...Xiaomi 14      |k003└cf1keyword ...Xiaomi 14 \x00\x00\x00\x1B\x00\x00\x00\
x09\x00\x05rk003\x03cf1keyword\x00\x00\x01\x8F\x1A<\x12\xFD\x04Xiaomi 14 1714132095741
```

【提交 3：结果截图】（截图需要包含标记信息，未按要求扣分）

五、实验结果与分析

1. 提交 1、3 两张截图（数据库表和运行结果）每张图 4 分，相关文字描述 3 分。
2. 提交项目 src 文件夹，使用 tree 命令打印 src 目录的结构截图，运行”`jar tf <jar` 文件`> | grep` 或 `findstr <`学号`>`”命令截图，4 分。
3. MemberMapper 和 Main 代码描述，3 分。