大数据技术基础课程实验报告

实验三：基于华为云安装 HBase、ZooKeeper 及 HBase 应用实践

付容天

学号 2020211616

班级 2020211310

计算机学院（国家示范性软件学院）

2023 年 4 月 3 日

1. 实验描述与实验目的

在本次实验中，我们需要在实验一、二搭建完毕的集群环境上，继续安装 HBase、ZooKeeper，并实践 HBase 的基本使用。

本次实验的目的是：掌握 HBase、ZooKeeper 等组件的安装与使用，以及使用 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 数据机制等相关内容。

2. 实验过程与实验分析

这部分的实验中，我按照实验指导书，先后完成了：

（1） 实验环境的检查，保证实验一、二搭建的集群环境运行正常；

（2） 下载、安装、配置 ZooKeeper 组件，包括：建立软链接、配置文件修改、修改数据目录、ZooKeeper 的拷贝等内容；

（3） 在四个节点上启动 ZooKeeper，检查是否启动成功，并通过-status 选项查看各节点是 follower 还是 leader；

（4） 下载、安装、配置 HBase 组件，包括：建立软链接、环境变量修改、配置文件修改、HBase 的拷贝等内容；

（5） 在 node1 上启动 HBase，进入 HBase Shell，使用 create 和 put 命令创建了如下图所示的数据库表格：

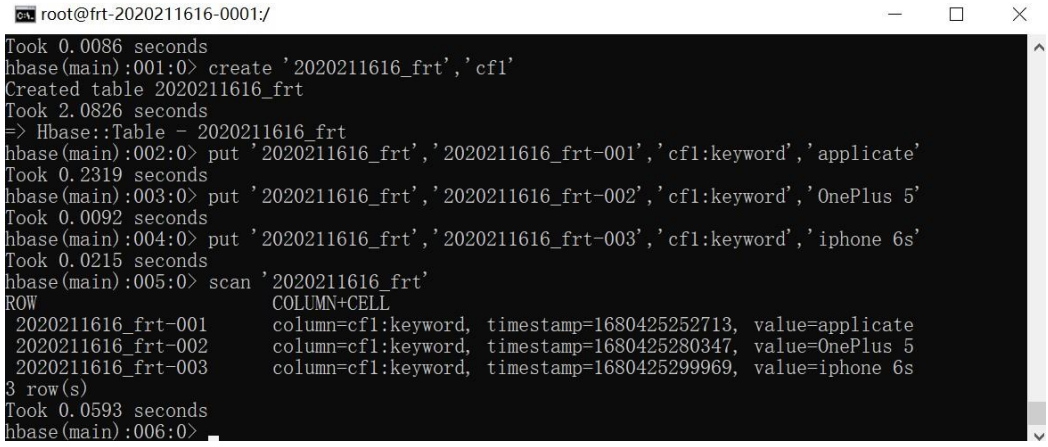


图 1：数据库表格（学号 2020211616，名字缩写 frt）

上图 1 中显示了数据库表格的结构，共有三行，且三行分别命名（即 row key）为 2020211616_frt-001、2020211616_frt-002、2020211616_frt-003，并且定义了列族 cf1（具有 keyword）。直观来看，这个表格可以表示为：

行键	cf1（列族）
	keyword
2020211616_frt-001	Applicate
2020211616_frt-002	OnePlus 5
2020211616_frt-003	iphone 6s

表 1：创建的数据库表格的形式与内容

随后，在 IDEA 中创建名为 MyHBase 的工程，进行下面的操作：

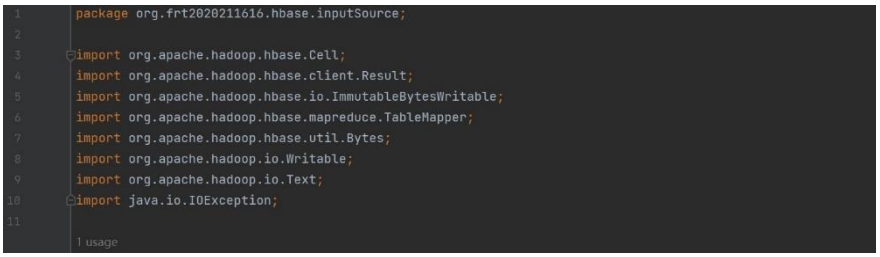
（6） 配置 pom.xml 文件，添加依赖，注意处理三个缺少依赖 jar 的包；

（7） 新建名为 org.frt2020211616.hbase.inputSource 的 package；

（8） 在新建 package 中创建类 MemberMapper，编写相应代码；

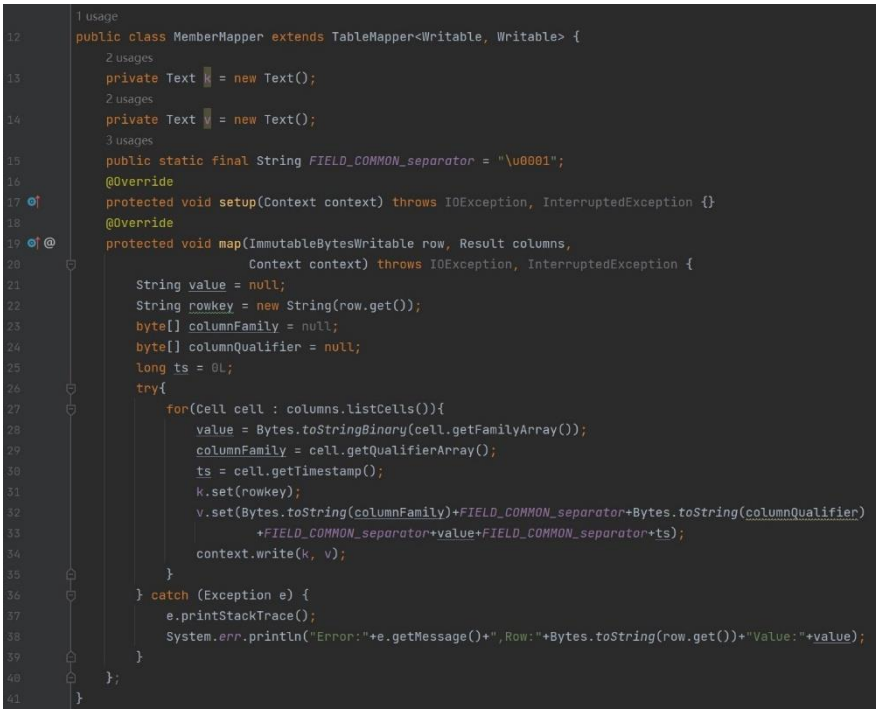
（9） 在新建 package 中创建类 Main，编写相应代码。

下面展示了两个类的源代码：



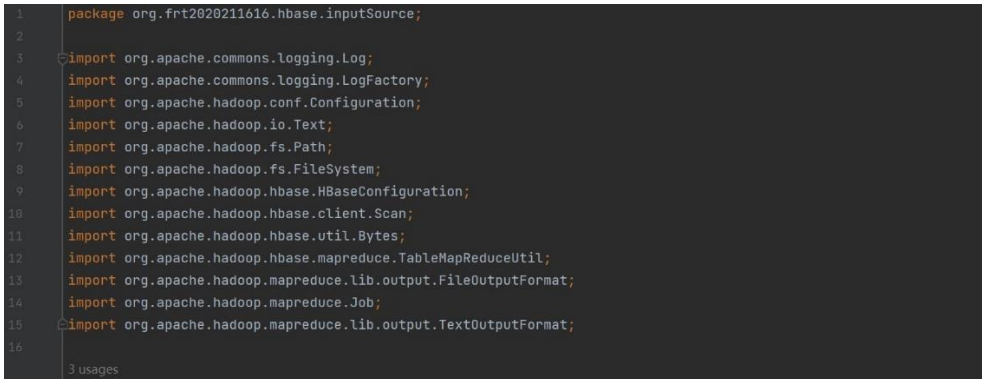
```
package org.frt2020211616.hbase.inputSource;

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;

1 usage
```

图 2：MemberMapper 类的头文件



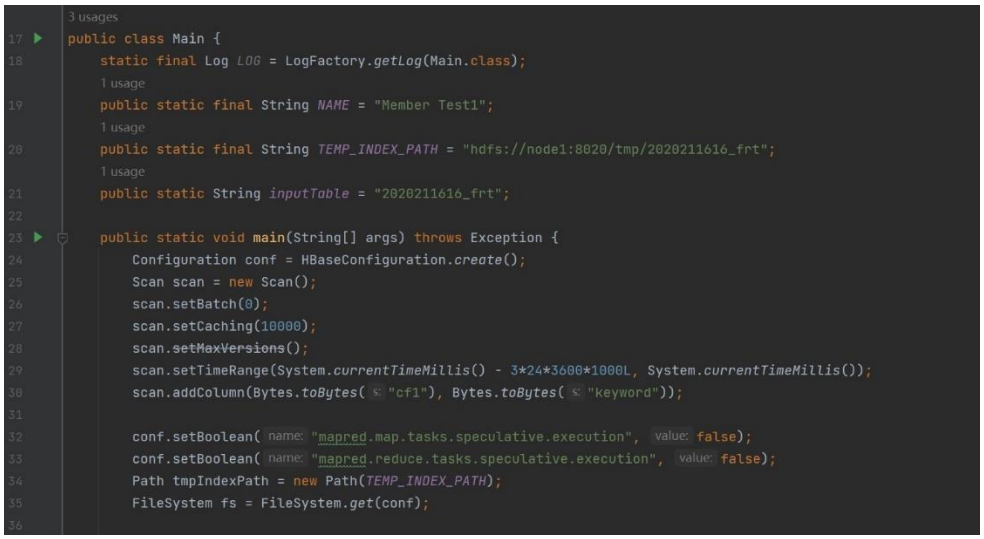
```
1 usage
public class MemberMapper extends TableMapper<Writable, Writable> {
    2 usages
    private Text k = new Text();
    2 usages
    private Text v = new Text();
    3 usages
    public static final String FIELD_COMMON_separator = "\u0001";
    @Override
    protected void setup(Context context) throws IOException, InterruptedException {}
    @Override
    protected void map(ImmutableBytesWritable row, Result columns,
                       Context context) throws IOException, InterruptedException {
        String value = null;
        String rowkey = new String(row.get());
        byte[] columnFamily = null;
        byte[] columnQualifier = null;
        long ts = 0L;
        try{
            for(Cell cell : columns.listCells()){
                value = Bytes.toStringBinary(cell.getFamilyArray());
                columnFamily = cell.getQualifierArray();
                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);
        }
    };
}
```

图 3：MemberMapper 类的源代码



```
package org.frt2020211616.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;

3 usages
```

图 4：Main 类的头文件



```
3 usages
public class Main {
    static final Log LOG = LogFactory.getLog(Main.class);
    1 usage
    public static final String NAME = "Member Test1";
    1 usage
    public static final String TEMP_INDEX_PATH = "hdfs://node1:8020/tmp/2020211616_frt";
    1 usage
    public static String inputTable = "2020211616_frt";

    public static void main(String[] args) throws Exception {
        Configuration conf = HBaseConfiguration.create();
        Scan scan = new Scan();
        scan.setBatch(0);
        scan.setCaching(10000);
        scan.setMaxVersions();
        scan.setTimeRange(System.currentTimeMillis() - 3*24*3600*1000L, System.currentTimeMillis());
        scan.addColumn(Bytes.toBytes( s: "cf1"), Bytes.toBytes( s: "keyword"));

        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);
```

图 5：Main 类的源代码

```
        if(fs.exists(tmpIndexPath)) {
            fs.delete(tmpIndexPath,  b: true);
        }

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

        TableMapReduceUtil.initTableMapperJob(inputTable, scan, MemberMapper.class, Text.class, Text.class, job);
        job.setNumReduceTasks(0);
        job.setOutputFormatClass(TextOutputFormat.class);
        FileOutputFormat.setOutputPath(job, tmpIndexPath);

        boolean success = job.waitForCompletion( verbose: true);
        System.exit(success ? 0 : 1);
    }
}
```

图 6：Main 类的源代码（续）

下面对代码进行简单的分析。图 2 展示了 MemberMapper 类依赖的头文件，主要来自 org.apache.hadoop；图 3 展示了 MemberMapper 类的源代码，该类扩展自 Mapper 类，以 HBase 中的表作为输入源进行相应的处理：图中 21 到 25 行定义了用到的变量，26 到 35 行为主要的处理逻辑（通过 for 循环读入 colunms 中的元素，处理后写入到 context 中）。

图 4 展示了 Main 类依赖的头文件，主要来自 org.apache.commons.logging 和 org.apache.hadoop 两处。图 5 和图 6 展示了 Main 类的源代码，其中 18-21

行定义了必要的信息（例如，20 行定义了表的路径），23-35 行则进行了必要的配置（例如，获得 HBase 配置信息、创建全表扫描器、添加扫描条件等操作），37-39 行解决了路径已经存在的情况（即，先删除），41-50 行先进行 mapper 类的初始化设置（例如，HBase 输入源对应表、扫描器、输出 value 类型等）、再设置输出格式和输出位置、最后开始运行作业（49 行）。

最后按照实验指导书上的内容，执行本次实验的最后部分：

（10）按照指导书上的步骤，导出完整的 jar 包，并传到 node1 上；

（11）检查环境是否启动、集群状态是否正常；

（12）在 node1 上输入指定命令，执行 jar 包，运行完毕后在 hdfs 上查看结果，如下图所示：

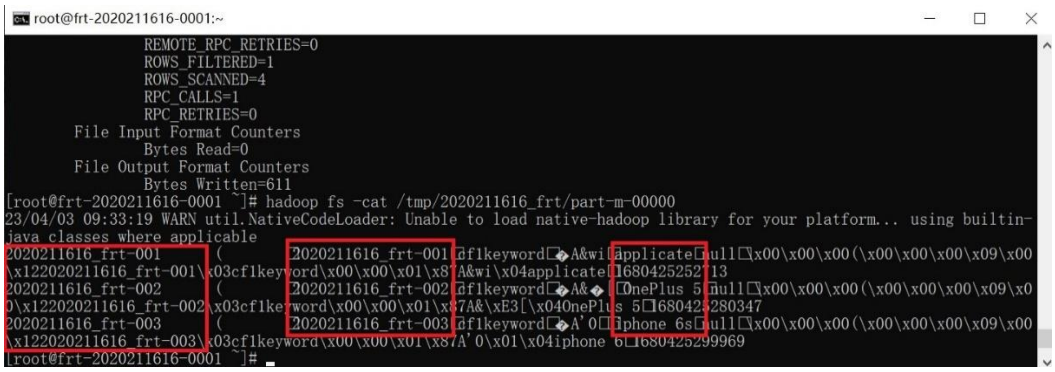


图 7：运行结果（红框处标记出了指定内容）

3. 实验问题与实验总结

本次实验中我也遇到了一些问题并进行了解决，现记录如下：

（1） 使用 JPS 查看进程时发现缺少部分进程，通过重新启动 Hadoop 集群、ZooKeeper 组件、HBase 组件三个部分，解决了这个问题；

（2） 配置 pom.xml 文件时发现依赖库加载过慢，通过增加国内的镜像源解决了这个问题。

在本次实验中，我在实验一、二的基础上安装了 ZooKeeper 和 HBase 两个组件，并进行了相应的配置，调试排除了相应的错误，最后创建了相应的表格。我还根据指导书内容编写并调试了 MyHBase 项目，并在打包为 jar 包后上传到主节点上进行运行，得到了预期的结果。总的来说，我在本次实验中复习了理论知识、进行了工程实践，圆满完成了实验任务，收获颇丰！