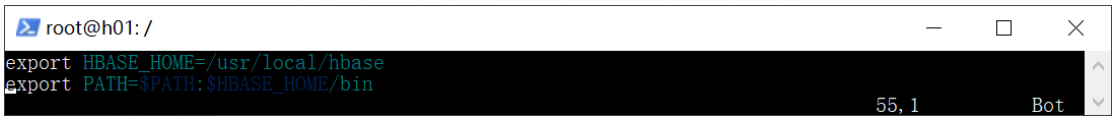
//王泽宇_181250148_金融大数据处理技术实验 3

一、实验过程

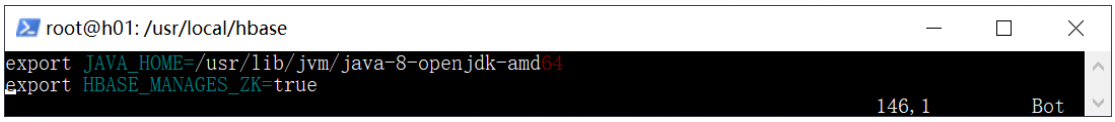
1. HBase 单机模式运行

我选择 Docker Desktop 配合先前 Hadoop MapReduce 使用的容器来进行 HBase 实验。

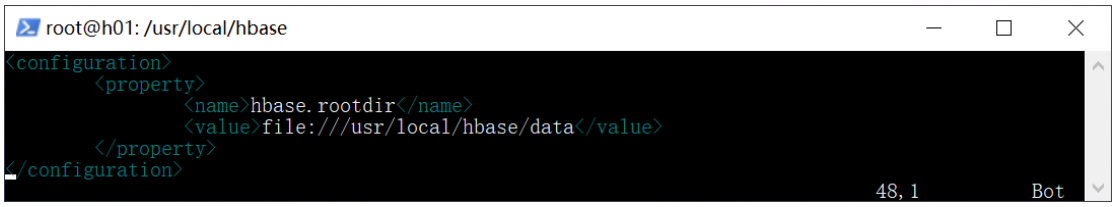
首先在容器 h01 下载并解压 HBase 2.3.3，编辑/etc/profile 文件添加环境变量。



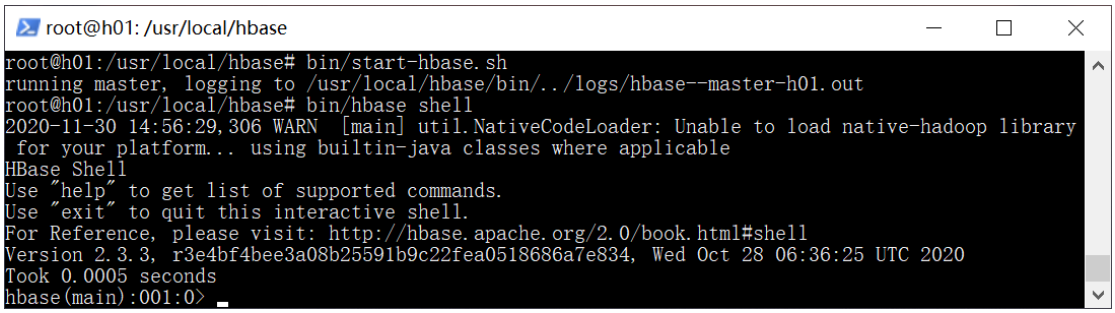
在容器 h01 的 hbase 目录下编辑 conf/hbase-env.sh，添加配置条目。



对于单机模式，将 conf/hbase-site.xml 中的 hbase.rootdir 设置为本地目录。



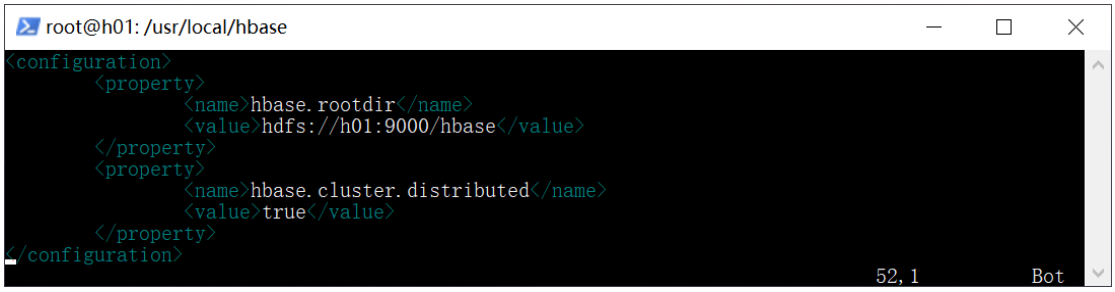
运行 bin/start-hbase.sh，再运行 bin/hbase shell。



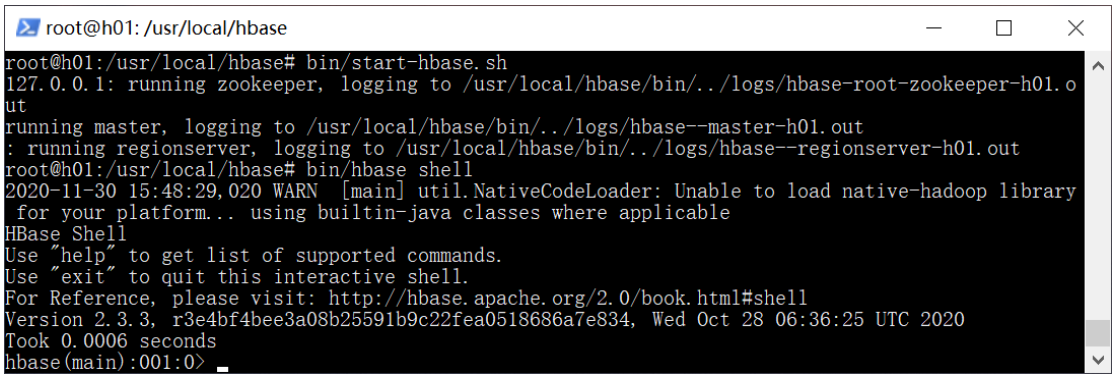
可见 shell 正常运行，HBase 单机模式配置正确。

2. HBase 伪分布模式运行

在上述基础上，对于伪分布模式，将 conf/hbase-site.xml 中的 hbase.rootdir 设置为 hdfs 目录，并且将 hbase.cluster.distributed 设为 true。



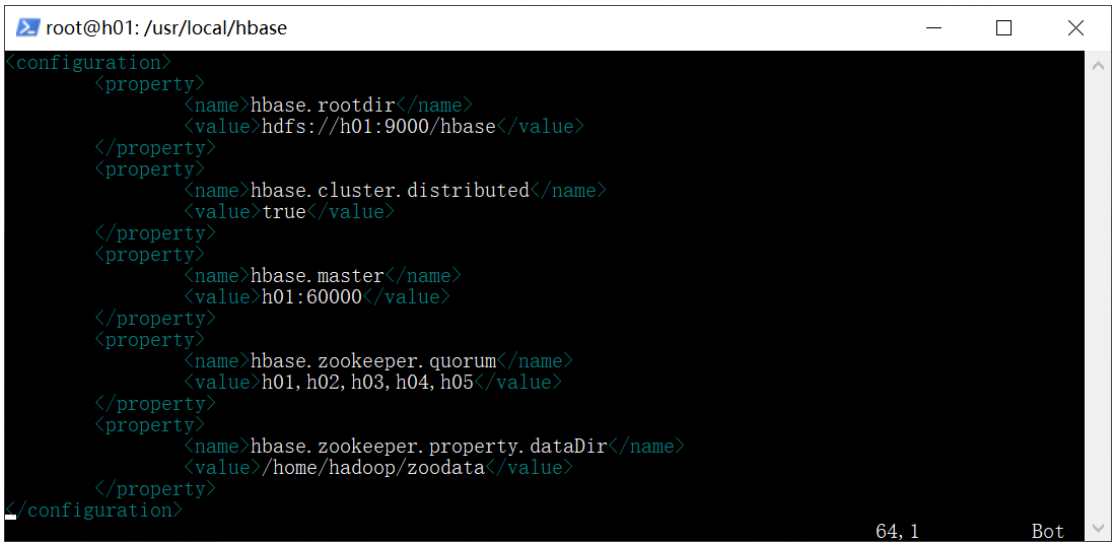
运行 bin/start-hbase.sh，再运行 bin/hbase shell。



可见 shell 正常运行，HBase 伪分布模式配置正确。

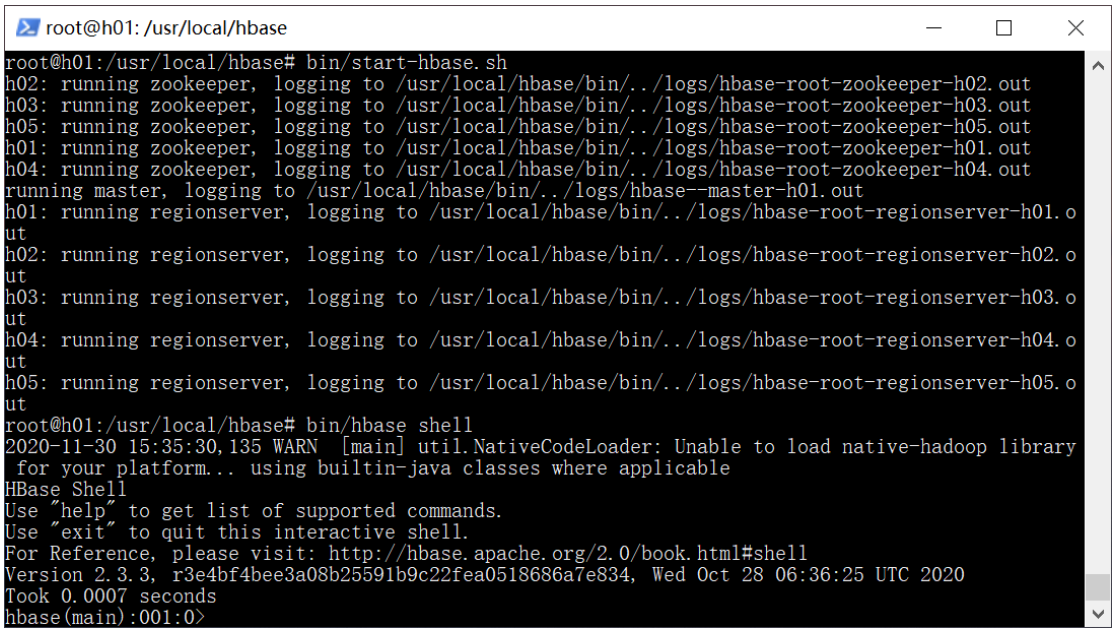
3. HBase 集群模式运行

对于集群模式，将 conf/hbase-site.xml 中的 hbase.rootdir 设置为 hdfs 目录，同时将其他参数如下设置。



编辑 conf/regionservers 文件，修改为如下内容。

使用 scp 命令将配置好的 hbase 目录从容器 h01 复制到 h02 至 h04 的相同位置。然后运行 bin/start-hbase.sh，再运行 bin/hbase shell。



可见 shell 正常运行，HBase 集群模式配置正确。

4. Java 程序编写

由于任务较为简单，所有程序逻辑都在 Main 类的 main 函数中实现。

首先初始化 configuration、connection 和 admin，连接上 HBase。

```
1.   configuration = HBaseConfiguration.create();
2.  configuration.set("hbase.rootdir", "hdfs://h01:9000/hbase");
3.  connection = ConnectionFactory.createConnection(configuration);
4.  admin = connection.getAdmin();
```

通过构造 TableDescriptorBuilder 对象，并且调用 admin.createTable()来创建具有 4 个列族的表 Students。

```
1.  TableDescriptorBuilder tableDescriptorBuilder = TableDescriptorBuilder.newBu
    ilder(TableName.valueOf("Students"));
2.  Collection<ColumnFamilyDescriptor> columnFamilies = new ArrayList();
3.  columnFamilies.add(ColumnFamilyDescriptorBuilder.newBuilder(Bytes.toBytes("I
    D")).build());
4.  columnFamilies.add(ColumnFamilyDescriptorBuilder.newBuilder(Bytes.toBytes("D
    escription")).build());
5.  columnFamilies.add(ColumnFamilyDescriptorBuilder.newBuilder(Bytes.toBytes("C
    ourses")).build());
6.  columnFamilies.add(ColumnFamilyDescriptorBuilder.newBuilder(Bytes.toBytes("H
    ome")).build());
7.  tableDescriptorBuilder.setColumnFamilies(columnFamilies);
8.  admin.createTable(tableDescriptorBuilder.build());
```

通过 connection.getTable()来获取 Table 对象。通过构造 Put 对象，并且调用

table.put()来新增表中条目。

```
Table table = connection.getTable(TableName.valueOf("Students"));

Put put = new Put("001".getBytes());
put.addColumn("Description".getBytes(), "Name".getBytes(), "Li Lei".getBytes());
put.addColumn("Description".getBytes(), "Height".getBytes(), "176".getBytes());
put.addColumn("Courses".getBytes(), "Chinese".getBytes(), "80".getBytes());
put.addColumn("Courses".getBytes(), "Math".getBytes(), "90".getBytes());
put.addColumn("Courses".getBytes(), "Physics".getBytes(), "95".getBytes());
put.addColumn("Home".getBytes(), "Province".getBytes(), "Zhejiang".getBytes());
table.put(put);

put = new Put("002".getBytes());
put.addColumn("Description".getBytes(), "Name".getBytes(), "Han Meimei".getBytes());
put.addColumn("Description".getBytes(), "Height".getBytes(), "183".getBytes());
put.addColumn("Courses".getBytes(), "Chinese".getBytes(), "88".getBytes());
put.addColumn("Courses".getBytes(), "Math".getBytes(), "77".getBytes());
put.addColumn("Courses".getBytes(), "Physics".getBytes(), "66".getBytes());
put.addColumn("Home".getBytes(), "Province".getBytes(), "Beijing".getBytes());
table.put(put);

put = new Put("003".getBytes());
put.addColumn("Description".getBytes(), "Name".getBytes(), "Xiao Ming".getBytes());
put.addColumn("Description".getBytes(), "Height".getBytes(), "162".getBytes());
put.addColumn("Courses".getBytes(), "Chinese".getBytes(), "90".getBytes());
put.addColumn("Courses".getBytes(), "Math".getBytes(), "90".getBytes());
put.addColumn("Courses".getBytes(), "Physics".getBytes(), "90".getBytes());
put.addColumn("Home".getBytes(), "Province".getBytes(), "Shanghai".getBytes());
table.put(put);
```

通过 scan 对象和 table.getScanner()获得扫描结果，遍历扫描结果并输出。

```
Scan scan = new Scan();
ResultScanner results = table.getScanner(scan);

for (Result row : results) {
    for (Cell cell : row.listCells()){
        System.out.println("RowKey:" + Bytes.toString(row.getRow())
                + " Family:" + Bytes.toString(CellUtil.cloneFamily(cell))
                + " Qualifier:" + Bytes.toString(CellUtil.cloneQualifier(cel
l))
                + " Value:" + Bytes.toString(CellUtil.cloneValue(cell))
        );
    }
}
```

通过构造 Get 对象，并且调用 table.get()获得查询结果，并输出。

```
Get get = new Get("001".getBytes());
get.addColumn("Home".getBytes(),"Province".getBytes());
Result result = table.get(get);
System.out.println("001's home is in "+new String(result.getValue("Home".get
Bytes(),"Province".getBytes()))+" Province.");
```

再次通过构造 Put 对象，并且调用 table.put()来新增表中条目，同时也新增了列。

```
put = new Put("001".getBytes());
put.addColumn("Courses".getBytes(), "English".getBytes(), "99".getBytes());

table.put(put);
```

为了新增列族，需要首先调用 disableTable()，然后通过 addColumnFamilyAsync()新增列族，最后调用 enableTableAsync()。

```
admin.disableTable(TableName.valueOf("Students"));
admin.addColumnFamilyAsync(TableName.valueOf("Students"),ColumnFamilyDescrip
torBuilder.newBuilder(Bytes.toBytes("Contact")).build());
admin.enableTableAsync(TableName.valueOf("Students"));
```

再次通过构造 Put 对象，并且调用 table.put()来新增表中条目。再扫描一次表查看修改结果。

```
put = new Put("001".getBytes());
put.addColumn("Contact".getBytes(), "Email".getBytes(), "001@hbase.com".getB
ytes());
```

```
table.put(put);

scan = new Scan();
results = table.getScanner(scan);

for (Result row : results) {
    for (Cell cell : row.listCells()){
        System.out.println("RowKey:" + Bytes.toString(row.getRow())
                + " Family:" + Bytes.toString(CellUtil.cloneFamily(cell))
                + " Qualifier:" + Bytes.toString(CellUtil.cloneQualifier(cel
l))
                + " Value:" + Bytes.toString(CellUtil.cloneValue(cell))
        );
    }
}
```

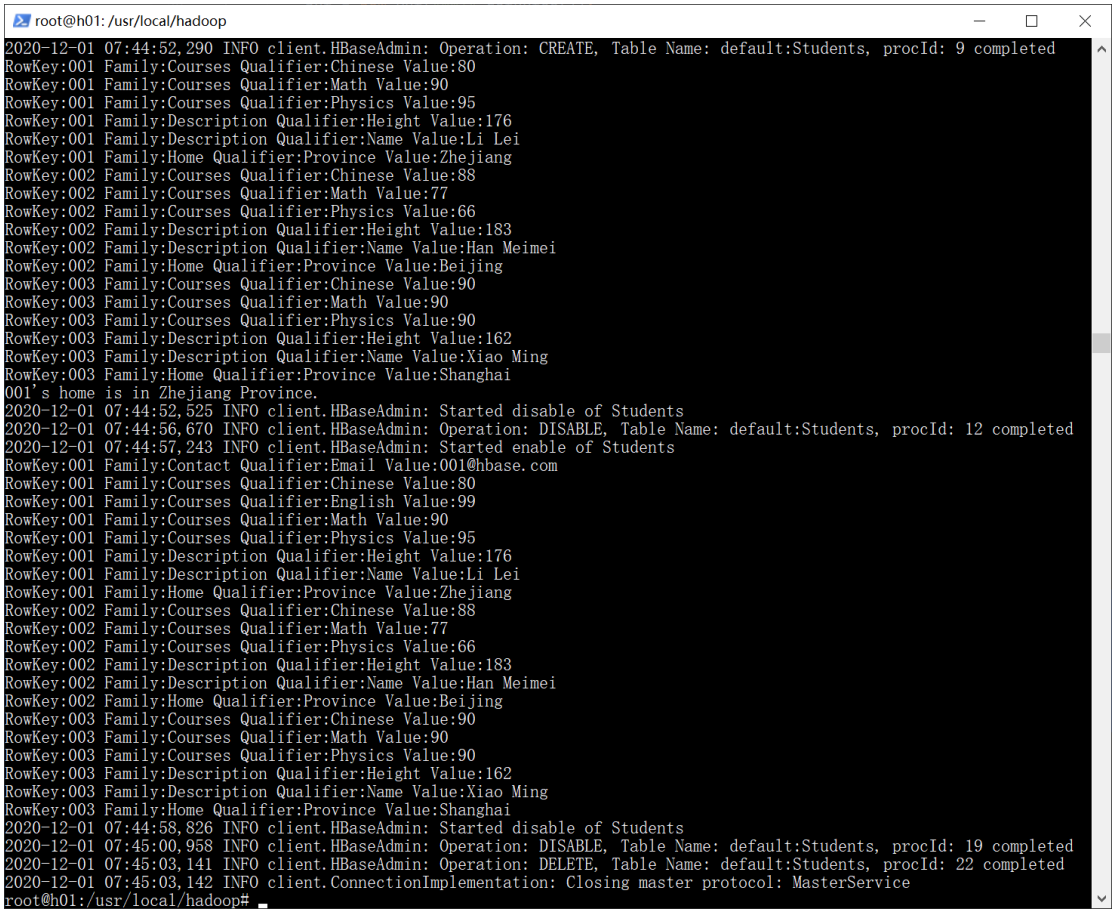
最后在 disableTable()后调用 deleteTable()来删除表，并且关闭相关连接。

```
table.close();

admin.disableTable(TableName.valueOf("Students"));
admin.deleteTable(TableName.valueOf("Students"));

admin.close();
connection.close();
```

将 Java 程序使用 Gradle 打包后在 Hadoop 运行。首先出现 ClassNotDefError，在 hadoop/etc/hadoop/hadoop-env.sh 文件中的 HADOOP_CLASSPATH 中增加 hbase/lib/* 即可。在程序运行前，需要同时打开 Hadoop 和 HBase，即分别执行 hadoop/sbin/start-all.sh 和 hbase/bin/start-hbase.sh，否则在程序执行过程中会提示 Connection refused，即相关进程连接不上。

在执行 hbase/bin/start-hbase.sh 时，出现了 HMaster 进程已存在的问题。经过几次尝试，发现即便重启容器 h01，HMaster 进程仍会自动运行，因此先使用 kill -9 指令杀掉 HMaster 进程，而后再执行 hbase/bin/start-hbase.sh，不再有错误提示。

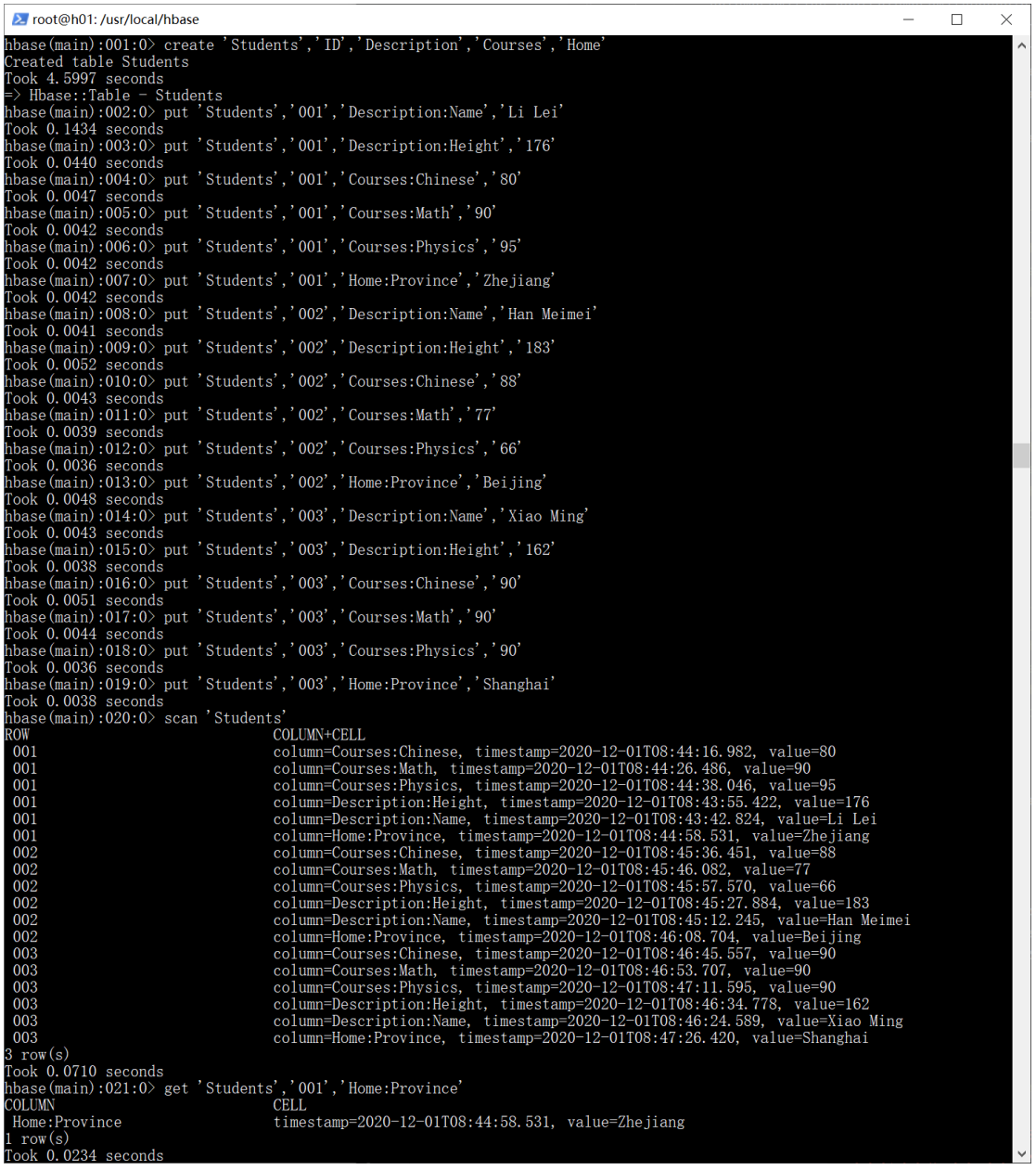
最后在 Hadoop 中执行打包好的程序，输出如下。

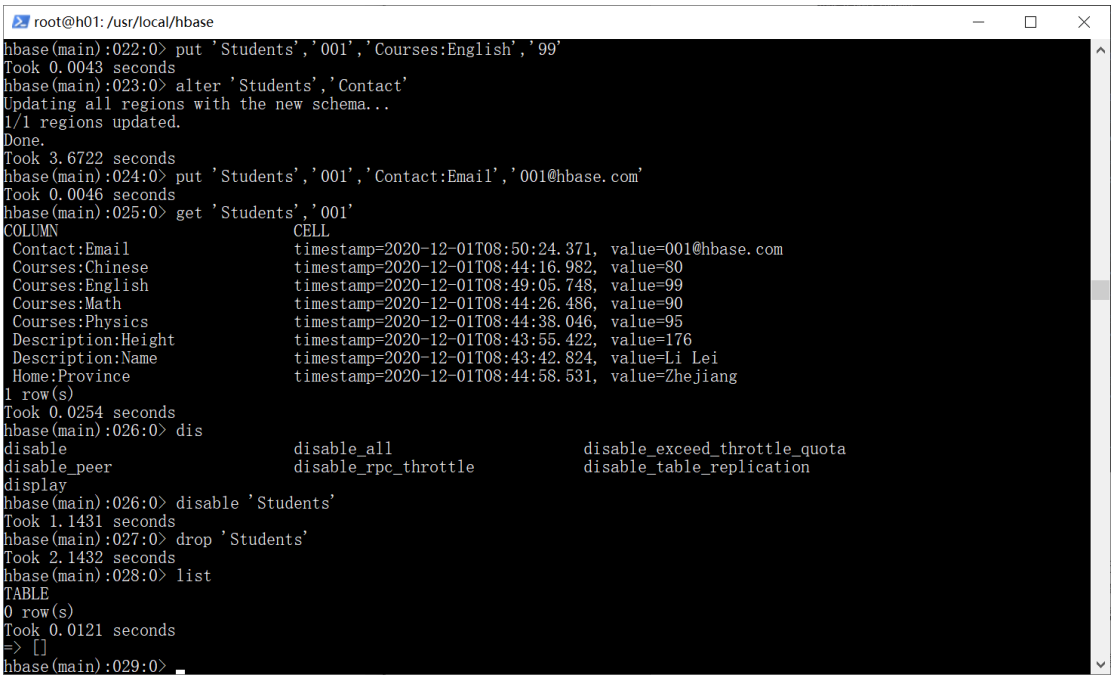


可见表 Students 创建、扫描成功，ID 为 001 的学生所在省份查询成功，观察第 2 次扫描结果，可见列族、列增加成功，最终表 Students 删除成功。

5. Shell 指令运行

使用 Shell 指令完成与 Java 程序相同的任务。


root@h01: /usr/local/hbase
hbase(main):001:0> create 'Students','ID','Description','Courses','Home'
Created table Students
Took 4.5997 seconds
=> Hbase::Table - Students
hbase(main):002:0> put 'Students','001','Description:Name','Li Lei'
Took 0.1434 seconds
hbase(main):003:0> put 'Students','001','Description:Height','176'
Took 0.0440 seconds
hbase(main):004:0> put 'Students','001','Courses:Chinese','80'
Took 0.0047 seconds
hbase(main):005:0> put 'Students','001','Courses:Math','90'
Took 0.0042 seconds
hbase(main):006:0> put 'Students','001','Courses:Physics','95'
Took 0.0042 seconds
hbase(main):007:0> put 'Students','001','Home:Province','Zhejiang'
Took 0.0042 seconds
hbase(main):008:0> put 'Students','002','Description:Name','Han Meimei'
Took 0.0041 seconds
hbase(main):009:0> put 'Students','002','Description:Height','183'
Took 0.0052 seconds
hbase(main):010:0> put 'Students','002','Courses:Chinese','88'
Took 0.0043 seconds
hbase(main):011:0> put 'Students','002','Courses:Math','77'
Took 0.0039 seconds
hbase(main):012:0> put 'Students','002','Courses:Physics','66'
Took 0.0036 seconds
hbase(main):013:0> put 'Students','002','Home:Province','Beijing'
Took 0.0048 seconds
hbase(main):014:0> put 'Students','003','Description:Name','Xiao Ming'
Took 0.0043 seconds
hbase(main):015:0> put 'Students','003','Description:Height','162'
Took 0.0038 seconds
hbase(main):016:0> put 'Students','003','Courses:Chinese','90'
Took 0.0051 seconds
hbase(main):017:0> put 'Students','003','Courses:Math','90'
Took 0.0044 seconds
hbase(main):018:0> put 'Students','003','Courses:Physics','90'
Took 0.0036 seconds
hbase(main):019:0> put 'Students','003','Home:Province','Shanghai'
Took 0.0038 seconds
hbase(main):020:0> scan 'Students'
ROW COLUMN+CELL
001 column=Courses:Chinese, timestamp=2020-12-01T08:44:16.982, value=80
001 column=Courses:Math, timestamp=2020-12-01T08:44:26.486, value=90
001 column=Courses:Physics, timestamp=2020-12-01T08:44:38.046, value=95
001 column=Description:Height, timestamp=2020-12-01T08:43:55.422, value=176
001 column=Description:Name, timestamp=2020-12-01T08:43:42.824, value=Li Lei
001 column=Home:Province, timestamp=2020-12-01T08:44:58.531, value=Zhejiang
002 column=Courses:Chinese, timestamp=2020-12-01T08:45:36.451, value=88
002 column=Courses:Math, timestamp=2020-12-01T08:45:46.082, value=77
002 column=Courses:Physics, timestamp=2020-12-01T08:45:57.570, value=66
002 column=Description:Height, timestamp=2020-12-01T08:45:27.884, value=183
002 column=Description:Name, timestamp=2020-12-01T08:45:12.245, value=Han Meimei
002 column=Home:Province, timestamp=2020-12-01T08:46:08.704, value=Beijing
003 column=Courses:Chinese, timestamp=2020-12-01T08:46:45.557, value=90
003 column=Courses:Math, timestamp=2020-12-01T08:46:53.707, value=90
003 column=Courses:Physics, timestamp=2020-12-01T08:47:11.595, value=90
003 column=Description:Height, timestamp=2020-12-01T08:46:34.778, value=162
003 column=Description:Name, timestamp=2020-12-01T08:46:24.589, value=Xiao Ming
003 column=Home:Province, timestamp=2020-12-01T08:47:26.420, value=Shanghai
3 row(s)
Took 0.0710 seconds
hbase(main):021:0> get 'Students','001','Home:Province'
COLUMN CELL
Home:Province timestamp=2020-12-01T08:44:58.531, value=Zhejiang
1 row(s)
Took 0.0234 seconds



可见运行结果均正确。

二、问题总结、解决方案及其他思考

实验过程中的所有问题及其解决方案已在实验过程中叙述，此处不再重复。由于先前实验中 Hadoop 集群环境已在本地 Docker Desktop 下配置好，本次实验中 HBase 环境的配置较为容易，没有遇到太多问题。

由于 HBase Shell 指令较为直观简洁，因此使用 Shell 指令来调用 HBase 功能也较为容易。本次实验的主要难点集中在 Java 程序中 HBase 的使用。观察 Java 代码，可见在 Java 程序中，HBase 的每个单个操作几乎都需要先新建一个对象，然后调用该对象的函数对其增加参数，然后再以该对象为参数调用表对象的函数来执行操作，非常繁琐复杂。如果 HBase Shell 指令能够像 SQL 指令一样在 Java 程序中内联，应当能有效降低程序编写和阅读的复杂度。