实 验 报 告 内 容

一・实验内容

实验题目：HBase 编程实践

实验内容与要求：

1. 在 Hadoop 基础上安装 HBase；
2. 使用 Shell 命令（`create`: 创建表、`list`: 列出 HBase 所有的表信息、`put`: 向表、行、列指定的单元格添加数据等）；使用 HBase 常用 Java API 创建表、插入数据、浏览数据。

二・实验目的

1. 理解 HBase 在 Hadoop 体系结构中的角色。
2. 熟练使用 HBase 操作常用的 Shell 命令。
3. 熟悉 HBase 操作常用的 Java API。

三・实验过程截图及说明

1. 实验环境与前提

本实验在 Hadoop 伪分布式环境基础上部署 HBase，依赖 Zookeeper 提供协调服务。HBase 数据目录配置到 HDFS（`hdfs://namenode:9000/hbase`），并通过 `http://localhost:16010` 提供 Web 管理界面。

与本实验相关的 `docker-compose.yml` 配置片段如下：

```yaml
zookeeper:
  image: zookeeper:3.8
  container_name: zookeeper
  hostname: zookeeper
  restart: always
  ports:
    - "2181:2181"

hbase:
  image: harisekhon/hbase:latest
  container_name: hbase
  hostname: hbase
  restart: always
  ports:
    - "16010:16010"
  depends_on:
    - namenode
    - zookeeper
  environment:
    - HBASE_MANAGES_ZK=false
    - HBASE_ZOOKEEPER_QUORUM=zookeeper
    - HBASE_ROOTDIR=hdfs://namenode:9000/hbase
    - SERVICE_PRECONDITION=namenode:9000 zookeeper:2181
```

进入 `hbase shell` 的方式如下：

```bash
docker exec -it hbase hbase shell
```

2. 使用 HBase Shell 命令

本部分围绕建表、写入、查询、删除与删表构建一套可验证的命令序列，表名为 `student`，列族为 `info` 与 `score`，并用学号作为 `RowKey`。

2.1 Namespace 与建表（DDL）

```
status 'detailed'

create 'student', 'info', 'score'

describe 'student'

list
```

图 1 展示了状态查看、建表、表结构查看与列出表的输出，证明表 `student` 已创建成功。

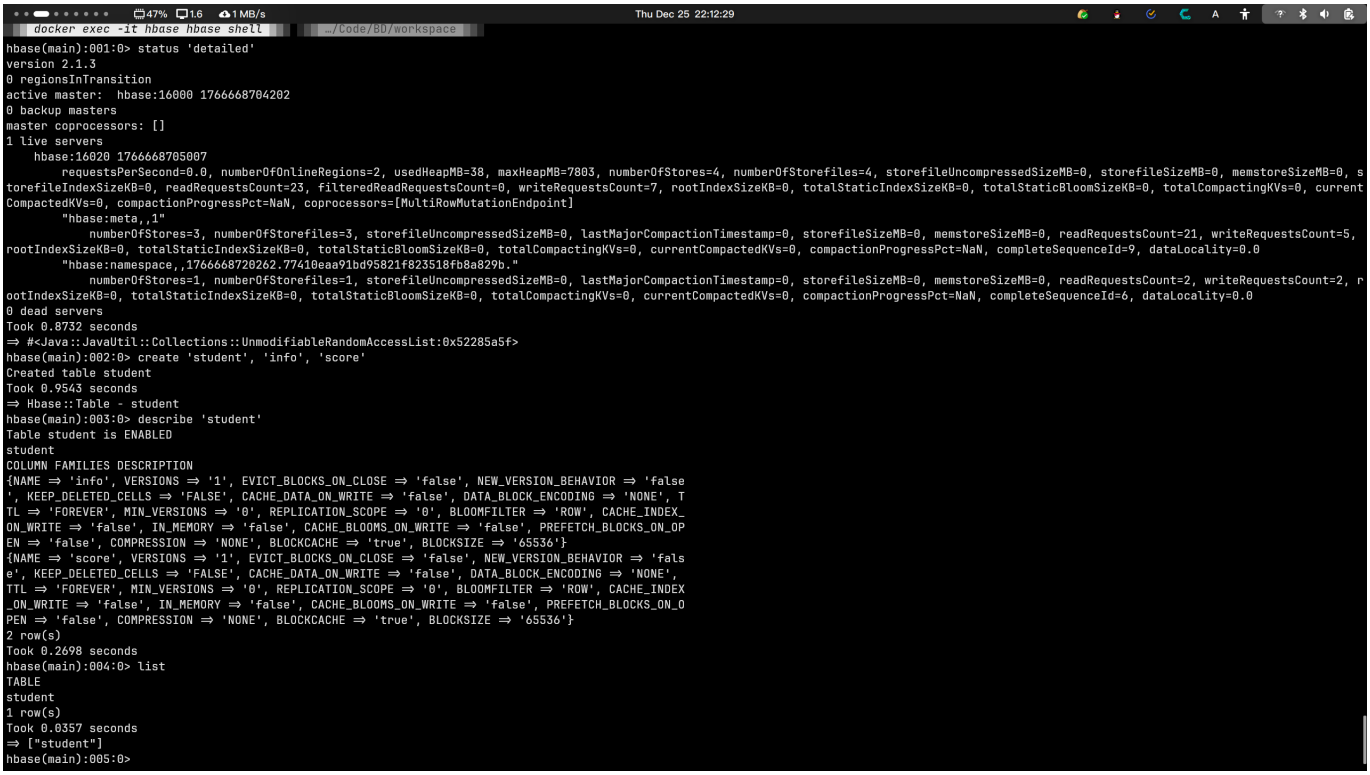

图 1 HBase Shell：状态查看与建表

2.2 数据插入（put）

```
put 'student', '1001', 'info:name', 'Alice'
put 'student', '1001', 'info:class', 'CS-1'
put 'student', '1001', 'score:math', '85'
put 'student', '1001', 'score:english', '90'

put 'student', '1002', 'info:name', 'Bob'
put 'student', '1002', 'info:class', 'CS-2'
put 'student', '1002', 'score:math', '92'
```

图 2 展示了多次 `put` 写入的执行情况，写入分别落在不同 `RowKey` 与不同列族/列上。

```
hbase(main):001:0> put 'student', '1001', 'info:name', 'Alice'
Took 0.7157 seconds
hbase(main):002:0> put 'student', '1001', 'info:class', 'CS-1'
Took 0.0059 seconds
hbase(main):003:0> put 'student', '1001', 'score:math', '85'
Took 0.0083 seconds
hbase(main):004:0> put 'student', '1001', 'score:english', '90'
Took 0.0095 seconds
hbase(main):005:0> put 'student', '1002', 'info:name', 'Bob'
Took 0.0113 seconds
hbase(main):006:0> put 'student', '1002', 'info:class', 'CS-2'
Took 0.0070 seconds
hbase(main):007:0> put 'student', '1002', 'score:math', '92'
Took 0.0059 seconds
hbase(main):008:0>
```

图 2 HBase Shell：数据插入

2.3 数据查询（scan/get）

```
scan 'student'

get 'student', '1001'

get 'student', '1001', 'info:name'

get 'student', '1002', 'score'
```

图 3 展示了 `scan` 与 `get` 的输出，可见写入的 `info` 与 `score` 列数据均可被正确读取。

```
docker exec -it hbase hbase shell    …/Code/BD/workspace
hbase(main):008:0> scan 'student'
ROW                                             COLUMN+CELL
 1001                                           column=info:class, timestamp=1766672078767, value=CS-1
 1001                                           column=info:name, timestamp=1766672073205, value=Alice
 1001                                           column=score:english, timestamp=1766672087429, value=90
 1001                                           column=score:math, timestamp=1766672083399, value=85
 1002                                           column=info:class, timestamp=1766672095523, value=CS-2
 1002                                           column=info:name, timestamp=1766672091380, value=Bob
 1002                                           column=score:math, timestamp=1766672101605, value=92
2 row(s)
Took 0.0642 seconds
hbase(main):009:0> get 'student', '1001'
COLUMN                                          CELL
 info:class                                     timestamp=1766672078767, value=CS-1
 info:name                                      timestamp=1766672073205, value=Alice
 score:english                                  timestamp=1766672087429, value=90
 score:math                                     timestamp=1766672083399, value=85
1 row(s)
Took 0.0571 seconds
hbase(main):010:0> get 'student', '1001', 'info:name'
COLUMN                                          CELL
 info:name                                      timestamp=1766672073205, value=Alice
1 row(s)
Took 0.0119 seconds
hbase(main):011:0> get 'student', '1002', 'score'
COLUMN                                          CELL
 score:math                                     timestamp=1766672101605, value=92
1 row(s)
Took 0.0152 seconds
hbase(main):012:0>
```

图 3 HBase Shell：数据查询

2.4 数据删除与清空（delete/deleteall）

```
delete 'student', '1002', 'score:math'

get 'student', '1002'

deleteall 'student', '1002'
```

图 4 展示了删除单元格后查询验证，以及对整行执行 `deleteall` 的过程与结果。

```
docker exec -it hbase hbase shell    …/Code/BD/workspace
hbase(main):012:0> delete 'student', '1002', 'score:math'
Took 0.0166 seconds
hbase(main):013:0> get 'student', '1002'
COLUMN                                          CELL
 info:class                                     timestamp=1766672095523, value=CS-2
 info:name                                      timestamp=1766672091380, value=Bob
1 row(s)
Took 0.0129 seconds
hbase(main):014:0> deleteall 'student', '1002'
Took 0.0064 seconds
hbase(main):015:0>
```

图 4 HBase Shell：数据删除与验证

2.5 删表操作（disable/drop）

```
disable 'student'

drop 'student'

list
```

图 5 展示了 `disable`、`drop` 与 `list` 的输出，证明表 `student` 已被删除。

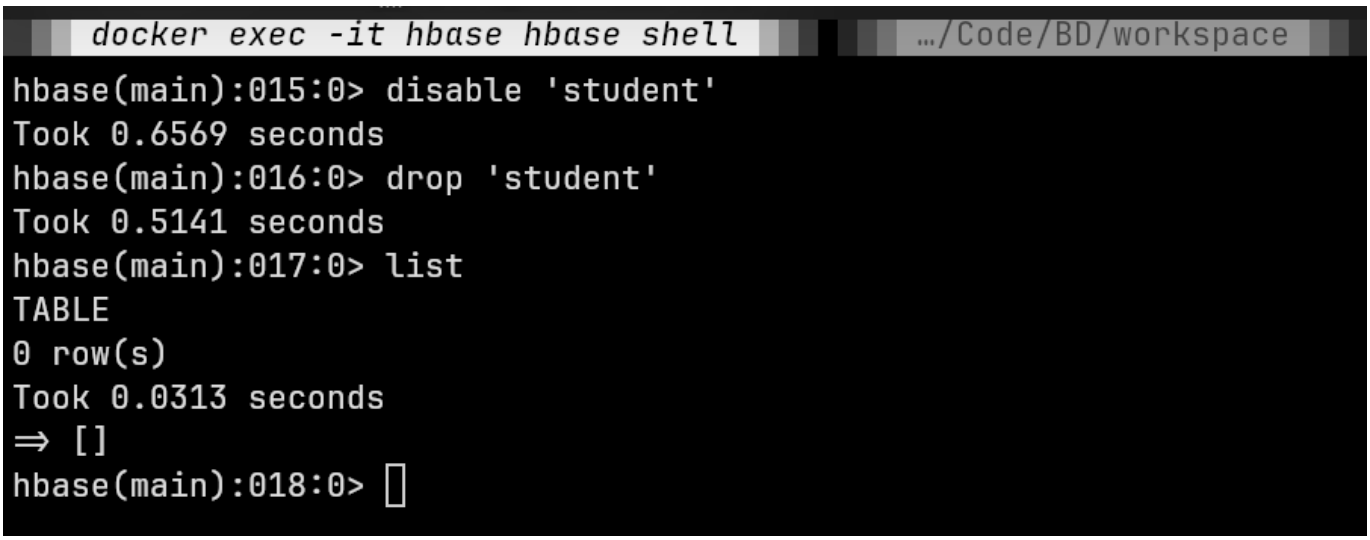

图 5 HBase Shell：删表操作

3. 使用 HBase 常用 Java API 创建表、插入数据、浏览数据

本部分使用 Java API 完成连接、建表、写入与扫描读取的闭环流程，验证程序化访问 HBase 的基本路径。示例程序 `HBaseTest.java` 如下：

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.hbase.TableName;
import org.apache.hadoop.hbase.client.*;
import org.apache.hadoop.hbase.util.Bytes;
import org.apache.hadoop.hbase.HTableDescriptor;
import org.apache.hadoop.hbase.HColumnDescriptor;

import java.io.IOException;

public class HBaseTest {
    public static void main(String[] args) {
        Connection connection = null;
        Admin admin = null;
        try {
            System.out.println(">>> 1. Initializing Configuration...");
            Configuration config = HBaseConfiguration.create();

            config.set("hbase.zookeeper.quorum", "zookeeper");
            config.set("hbase.zookeeper.property.clientPort", "2181");

            config.set("hbase.client.retries.number", "3");

            System.out.println(">>> 2. Connecting to HBase...");
            connection = ConnectionFactory.createConnection(config);
            admin = connection.getAdmin();
            System.out.println(">>> Connection Success!");

            TableName tableName = TableName.valueOf("student_java");

            System.out.println(">>> 3. Checking Table...");
            if (admin.tableExists(tableName)) {
                System.out.println(">>> Table exists, deleting it first (for
test)...");
                admin.disableTable(tableName);
                admin.deleteTable(tableName);
            }

            System.out.println(">>> Creating Table 'student_java'...");
            TableDescriptorBuilder tableBuilder =
TableDescriptorBuilder.newBuilder(tableName);
            ColumnFamilyDescriptor family =
ColumnFamilyDescriptorBuilder.newBuilder(Bytes.toBytes("info")).build();
            tableBuilder.setColumnFamily(family);
            admin.createTable(tableBuilder.build());
            System.out.println(">>> Table Created!");

            System.out.println(">>> 4. Inserting Data...");
            Table table = connection.getTable(tableName);
            Put put = new Put(Bytes.toBytes("1001"));
```

```
            put.addColumn(Bytes.toBytes("info"), Bytes.toBytes("name"),
Bytes.toBytes("    "));
            put.addColumn(Bytes.toBytes("info"), Bytes.toBytes("os"),
Bytes.toBytes("CachyOS"));
            table.put(put);
            System.out.println(">>> Insert Success!");

            System.out.println(">>> 5. Scanning Data...");
            Scan scan = new Scan();
            ResultScanner scanner = table.getScanner(scan);
            for (Result result : scanner) {
                System.out.println(">>> Found Row: " + result);
                String name = Bytes.toString(result.getValue(Bytes.toBytes("info"),
Bytes.toBytes("name")));
                String os = Bytes.toString(result.getValue(Bytes.toBytes("info"),
Bytes.toBytes("os")));
                System.out.println("    -> Name: " + name + ", OS: " + os);
            }
            scanner.close();
            table.close();

        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            try {
                if (admin != null) admin.close();
                if (connection != null) connection.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

编译与运行命令如下（在容器内执行）。其中将 HBase 相关依赖放置在 `/workspace/hbase-lib/` 并加入 `classpath`：

```
javac -cp .:$(hadoop classpath):/workspace/hbase-lib/* HBaseTest.java
java -cp .:$(hadoop classpath):/workspace/hbase-lib/* HBaseTest
```

图 6 展示了程序输出，包括连接成功、建表成功、插入成功以及扫描读取到的行数据，证明 Java API 访问链路可用。

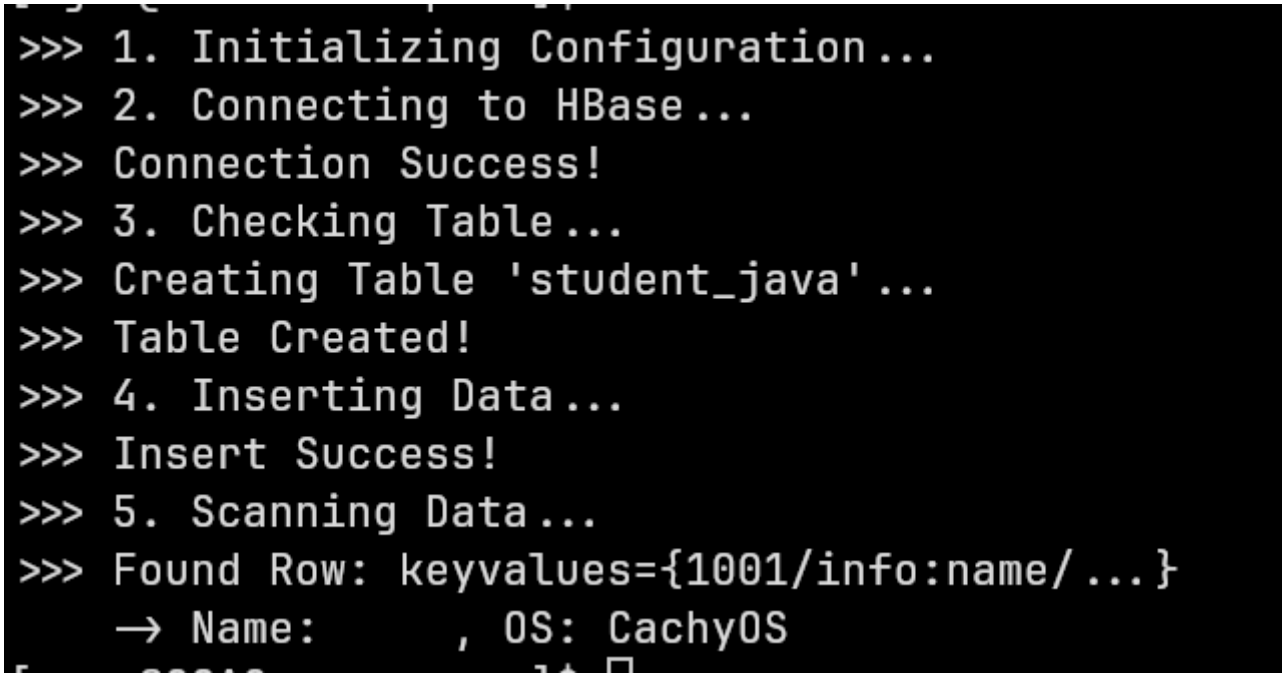

图 6 HBase Java API：建表、写入与扫描

四·实验总结及心得体会

实验总结：本实验在 Hadoop 基础上完成 HBase 部署，并通过 HBase Shell 完成建表、写入、查询、删除与删表操作；同时使用 Java API 完成连接、建表、写入与扫描读取，满足实验内容与要求。

心得体会：HBase 的数据模型以列族为核心，设计阶段需要优先稳定列族边界，再在列族内按业务演进扩展列；这与传统关系型数据库的建模习惯不同。实际操作中，`disable` 是 `drop` 的前置条件，体现了 HBase 对在线数据安全的保护；在 Java 侧，连接与 `Admin` / `Table` 的生命周期管理直接影响稳定性，建议在程序结构上显式关闭资源以避免句柄泄漏与连接堆积。