作业三

一、实验目的

通过使用 Java API，实现hadoop、hbase的常见操作，包括**在Hadoop分布式文件管理系统上创建文件、删除文件、上传文件、读取文件内容，在Hbase分布式数据库上创建表、列族、插入删除数据等**，从而加深对分布式计算框架和分布式数据库的理解和应用能力。

二、实验环境

- 操作系统：Ubuntu 22.04 LTS

- 虚拟机：UTM

- 集群节点：1 个 master 节点，2 个 slave 节点（slave1, slave2）

- 网络配置：静态 IP 地址

- 软件版本：

  - Hadoop: 2.7.1

  - Hbase: 1.2.1

  - zookeeper: 3.4.10

  - jdk: 1.8.0_151

三、hdfs和hbase任务

3.1 使用Java API操作hdfs

所有操作都统一在一个源文件中 `Hdfsop.java` 中实现后编译，并打包成jar包，运行方法为：

```
java -cp $(hadoop classpath):. Hdfsop <命令> <参数1> [参数2]
```

例如：`java -cp $(hadoop classpath):. Hdfsop create /user/lfl/java/create.txt "hello hdfs"`

3.1.1 在hdfs上创建文件

```
1  // 创建文件                    Hadoop文件系统        文件路径            文件内容
2  public static void createFile(FileSystem fs, String hdfsPath, String content) throws
   Exception {
3    // 将字符串路径包装成 Hadoop 的 Path 对象
4    Path path = new Path(hdfsPath);
5
6    // 使用 try-with-resources 确保流会被自动关闭
7    try (OutputStream out = fs.create(path)) {
8      out.write(content.getBytes());
9      System.out.println("文件已创建：" + hdfsPath);
10   }
11 }
```

运行结果：

```
lfl@master:~/JavaAPI$ java -cp $(hadoop classpath):. Hdfsop create /user/lfl/java/create.txt "hello hdfs"
25/09/11 15:47:45 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-
java classes where applicable
文件已创建: /user/lfl/java/create.txt
lfl@master:~/JavaAPI$ hdfs dfs -cat /user/lfl/java/create.txt
hello hdfslfl@master:~/JavaAPI$
```

Browse Directory

/user/lfl/java								Go!
Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name	
-rw-r--r--	lfl	supergroup	10 B	2025/9/11 15:47:51	2	128 MB	create.txt	

3.1.2 删除hdfs上的文件

```
1  // 删除文件
2  public static void deleteFile(FileSystem fs, String hdfsPath) throws Exception {
3    Path path = new Path(hdfsPath);
4    if (fs.exists(path)) {
5      fs.delete(path, true);
6      System.out.println("文件已删除：" + hdfsPath);
7    } else {
8      System.out.println("文件不存在：" + hdfsPath);
9    }
10 }
```

运行结果：

```
lfl@master:~/JavaAPI$ hadoop jar Hdfsop.jar delete /user/lfl/java/create.txt
文件已删除: /user/lfl/java/create.txt
lfl@master:~/JavaAPI$ hdfs dfs -ls /user/lfl/java
lfl@master:~/JavaAPI$
```

Browse Directory

/user/lfl/java | Go!

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name

3.1.3上传文件至hdfs

```java
// 上传本地文件
public static void uploadFile(FileSystem fs, String localPath, String hdfsPath) throws
Exception {
  Path src = new Path(localPath);
  Path dst = new Path(hdfsPath);
  fs.copyFromLocalFile(src, dst);
  System.out.println("文件已上传：" + localPath + " -> " + hdfsPath);
}
```

在本地上创建文件 `upload.txt`

```
lfl@master:~/JavaAPI$ vi upload.txt
lfl@master:~/JavaAPI$ ls
Hdfsop.class   Hdfsop.jar  Hdfsop.java  upload.txt
```

运行结果：

```
lfl@master:~/JavaAPI$ hadoop jar Hdfsop.jar upload upload.txt /user/lfl/java/upload.txt
文件已上传: upload.txt -> /user/lfl/java/upload.txt
lfl@master:~/JavaAPI$ hdfs dfs -cat /user/lfl/java/upload.txt
Upload success!

lfl@master:~/JavaAPI$
```

3.1.4 读取文件内容

```java
// 读取 HDFS 文件
public static void readFile(FileSystem fs, String hdfsPath) throws Exception {
  Path path = new Path(hdfsPath);
  if (!fs.exists(path)) {
    System.out.println("文件不存在：" + hdfsPath);
    return;
  }
  try (BufferedReader br = new BufferedReader(new InputStreamReader(fs.open(path))))
  {
    System.out.println("文件内容:");
    String line;
    while ((line = br.readLine()) != null) {
      System.out.println(line);
    }
  }
}
```

运行结果：

```
lfl@master:~/JavaAPI$ hadoop jar Hdfsop.jar read /user/lfl/java/upload.txt
文件内容：
Upload success!

lfl@master:~/JavaAPI$
```

3.1.5 完整代码

```java
1  import org.apache.hadoop.conf.Configuration;
2  import org.apache.hadoop.fs.FileSystem;
3  import org.apache.hadoop.fs.Path;
4  import java.io.*;
5
6  public class Hdfsop {
7
8      // 创建文件
9      public static void createFile(FileSystem fs, String hdfsPath, String content)
   throws Exception {
10         Path path = new Path(hdfsPath);
11         try (OutputStream out = fs.create(path)) {
12             out.write(content.getBytes());
13             System.out.println("文件已创建：" + hdfsPath);
14         }
15     }
16
17     // 删除文件
18     public static void deleteFile(FileSystem fs, String hdfsPath) throws Exception {
19         Path path = new Path(hdfsPath);
20         if (fs.exists(path)) {
21             fs.delete(path, true);
22             System.out.println("文件已删除：" + hdfsPath);
23         } else {
24             System.out.println("文件不存在：" + hdfsPath);
25         }
26     }
27
28     // 上传本地文件
29     public static void uploadFile(FileSystem fs, String localPath, String hdfsPath)
   throws Exception {
30         Path src = new Path(localPath);
31         Path dst = new Path(hdfsPath);
32         fs.copyFromLocalFile(src, dst);
33         System.out.println("文件已上传：" + localPath + " -> " + hdfsPath);
34     }
35
36     // 读取 HDFS 文件
37     public static void readFile(FileSystem fs, String hdfsPath) throws Exception {
38         Path path = new Path(hdfsPath);
39         if (!fs.exists(path)) {
```

```java
                System.out.println("文件不存在：" + hdfsPath);
                return;
            }
            try (BufferedReader br = new BufferedReader(new
    InputStreamReader(fs.open(path)))) {
                System.out.println("文件内容:");
                String line;
                while ((line = br.readLine()) != null) {
                    System.out.println(line);
                }
            }
        }
    }

    public static void main(String[] args) throws Exception {
        if (args.length < 1) {
            System.out.println("用法：java Hdfsop <command> [args...]");
            System.out.println("命令列表:");
            System.out.println("  create <hdfsPath> <content>");
            System.out.println("  delete <hdfsPath>");
            System.out.println("  upload <localPath> <hdfsPath>");
            System.out.println("  read <hdfsPath>");
            System.exit(1);
        }

        Configuration conf = new Configuration();
        conf.set("fs.defaultFS", "hdfs://master:9000");
        FileSystem fs = FileSystem.get(conf);

        String command = args[0];
        switch (command) {
            case "create":
                if (args.length != 3) {
                    System.out.println("用法：create <hdfsPath> <content>");
                    return;
                }
                createFile(fs, args[1], args[2]);
                break;

            case "delete":
                if (args.length != 2) {
                    System.out.println("用法：delete <hdfsPath>");
                    return;
                }
                deleteFile(fs, args[1]);
                break;

            case "upload":
                if (args.length != 3) {
                    System.out.println("用法：upload <localPath> <hdfsPath>");
                    return;
                }
                uploadFile(fs, args[1], args[2]);
```

```
 91                    break;
 92
 93            case "read":
 94                if (args.length != 2) {
 95                    System.out.println("用法: read <hdfsPath>");
 96                    return;
 97                }
 98                readFile(fs, args[1]);
 99                break;
100
101            default:
102                System.out.println("未知命令: " + command);
103        }
104
105        fs.close();
106    }
107 }
108
```

3.2 使用Java API操作hbase

所有操作都统一在一个源文件中 `HBaseop.java` 中实现后编译，并打包成jar包，运行方法为：

```
java -cp "$(hbase classpath):$(hadoop classpath):HBaseop.jar" HBaseop <命令> <参数1> [参数2]
```

例如： `java -cp "$(hbase classpath):$(hadoop classpath):HBaseop.jar" HBaseop createTable student info score`

```
lfl@master:~/JavaAPI$ ls
HBaseop.java  Hdfsop.class  Hdfsop.jar  Hdfsop.java  upload.txt
lfl@master:~/JavaAPI$ javac -cp "$(hbase classpath):$(hadoop classpath)" HBaseop.java
lfl@master:~/JavaAPI$ sudo vi MANIFEST.MF
lfl@master:~/JavaAPI$ jar cfm HBaseop.jar MANIFEST.MF HBaseop.class
lfl@master:~/JavaAPI$ ls
HBaseop.class  HBaseop.jar  HBaseop.java  Hdfsop.class  Hdfsop.jar  Hdfsop.java  MANIFEST.MF  upload.txt
lfl@master:~/JavaAPI$
```

3.2.1 创建表、列族

```
 1  public static void createTable(Admin admin, String tableName, String... familyNames)
    throws Exception {
 2    TableName tName = TableName.valueOf(tableName);
 3
 4    if (admin.tableExists(tName)) {
 5      System.out.println("表已存在: " + tableName);
 6      return;
 7    }
 8
 9    HTableDescriptor tableDescriptor = new HTableDescriptor(tName);
10
11    for (String family : familyNames) {
12      HColumnDescriptor columnDescriptor = new HColumnDescriptor(family);
```

```
13        tableDescriptor.addFamily(columnDescriptor);
14    }
15
16    admin.createTable(tableDescriptor);
17    System.out.println("表已创建：" + tableName + "，列族：" + String.join(",",
    familyNames));
18  }
```

运行结果：

```
lfl@master:~/JavaAPI$ java -cp "$(hbase classpath):$(hadoop classpath):HBaseop.jar" HBaseop createTable student info score
表已创建：student，列族：info,score

lfl@master:~/JavaAPI$ java -cp "$(hbase classpath):$(hadoop classpath):HBaseop.jar" HBaseop createTable course info
表已创建：course，列族：info
```

访问http://master:16010查看创建的表信息：

APACHE **HBASE**   Home   Table Details   Local Logs   Log Level   Debug Dump   Metrics Dump   HBase Configuration

User Tables

2 table(s) in set.

Table	Description
course	'course', {NAME => 'info', BLOOMFILTER => 'ROW', VERSIONS => '1', IN_MEMORY => 'false', KEEP_DELETED_CELLS => 'FALSE', DATA_BLOCK_ENCODING => 'NONE', TTL => 'FOREVER', COMPRESSION => 'NONE', MIN_VERSIONS => '0', BLOCKCACHE => 'true', BLOCKSIZE => '65536', REPLICATION_SCOPE => '0'}
student	'student', {NAME => 'info', BLOOMFILTER => 'ROW', VERSIONS => '1', IN_MEMORY => 'false', KEEP_DELETED_CELLS => 'FALSE', DATA_BLOCK_ENCODING => 'NONE', TTL => 'FOREVER', COMPRESSION => 'NONE', MIN_VERSIONS => '0', BLOCKCACHE => 'true', BLOCKSIZE => '65536', REPLICATION_SCOPE => '0'}, {NAME => 'score', BLOOMFILTER => 'ROW', VERSIONS => '1', IN_MEMORY => 'false', KEEP_DELETED_CELLS => 'FALSE', DATA_BLOCK_ENCODING => 'NONE', TTL => 'FOREVER', COMPRESSION => 'NONE', MIN_VERSIONS => '0', BLOCKCACHE => 'true', BLOCKSIZE => '65536', REPLICATION_SCOPE => '0'}

使用 `habse shell` 命令查看：

```
hbase(main):017:0> list
TABLE
course
student
2 row(s) in 0.2050 seconds

=> ["course", "student"]
hbase(main):018:0>
```

3.2.2 插入数据

```
1  public static void insertData(Connection connection, String tableName, String rowKey,
2                                  String family, String column, String value) throws
   Exception {
3    try (Table table = connection.getTable(TableName.valueOf(tableName))) {
4      Put put = new Put(Bytes.toBytes(rowKey));
5      put.addColumn(Bytes.toBytes(family), Bytes.toBytes(column), Bytes.toBytes(value));
6      table.put(put);
7      System.out.println("插入成功：" + rowKey + " -> " + family + ":" + column + "=" +
   value);
8    }
9  }
```

运行结果：

在 `student` 表的列族 `info` 插入列 `name`，值为 `Alice`，键值为 `1`：

```
lfl@master:~/JavaAPI$ java -cp "$(hbase classpath):$(hadoop classpath):HBaseop.jar" HBaseop put student 1 info name Alice
```

```
插入成功：1 -> info:name=Alice
```

在 `student` 表的列族 `score` 插入列 `math`，值为 `95`，键值为 `2`：

```
lfl@master:~/JavaAPI$ java -cp "$(hbase classpath):$(hadoop classpath):HBaseop.jar" HBaseop put student 1 info Age 18
```

```
插入成功：1 -> info:Age=18
```

同样插入多条数据后,使用 `hbase shell` 命令扫描全表：

```
hbase(main):018:0> scan 'student'
ROW                              COLUMN+CELL
 1                               column=info:Age, timestamp=1758287688324, value=18
 1                               column=info:name, timestamp=1758287546003, value=Alice
 1                               column=score:math, timestamp=1758287981723, value=95
 2                               column=info:Age, timestamp=1758287894315, value=22
 2                               column=info:name, timestamp=1758287731320, value=Bob
 2                               column=score:math, timestamp=1758287988509, value=90
2 row(s) in 1.2550 seconds
```

查询数据

```
1  public static void getData(Connection connection, String tableName, String rowKey)
   throws Exception {
2    try (Table table = connection.getTable(TableName.valueOf(tableName))) {
3      Get get = new Get(Bytes.toBytes(rowKey));
4      Result result = table.get(get);
5      if (result.isEmpty()) {
6        System.out.println("未找到数据：" + rowKey);
7      } else {
8        result.listCells().forEach(cell -> {
9          String family = Bytes.toString(cell.getFamilyArray(), cell.getFamilyOffset(),
   cell.getFamilyLength());
10          String qualifier = Bytes.toString(cell.getQualifierArray(),
   cell.getQualifierOffset(), cell.getQualifierLength());
11          String value = Bytes.toString(cell.getValueArray(), cell.getValueOffset(),
   cell.getValueLength());
12          System.out.println("row=" + rowKey + ", family=" + family + ", column=" +
   qualifier + ", value=" + value);
```

```
13            });
14        }
15    }
16 }
```

运行结果：

在 `student` 表查询键值为 `1` 行：

```
lfl@master:~/JavaAPI$ java -cp "$(hbase classpath):$(hadoop classpath):HBaseop.jar" HBaseop get student 1
```

```
2025-09-19 21:20:23,772 INFO  [main-SendThread(master:2181)] zookeeper.ClientCnxn: Session establishment complete
 on server master/10.211.55.22:2181, sessionid = 0x19961f36dc90019, negotiated timeout = 40000
row=1, family=info, column=Age, value=18
row=1, family=info, column=name, value=Alice
row=1, family=score, column=math, value=95
```

在 `student` 表查询键值为 `2` 的行：

```
lfl@master:~/JavaAPI$ java -cp "$(hbase classpath):$(hadoop classpath):HBaseop.jar" HBaseop get student 2
```

```
2025-09-19 21:20:40,286 INFO  [main-SendThread(master:2181)] zookeeper.ClientCnxn: Session establishment complet
e on server master/10.211.55.22:2181, sessionid = 0x19961f36dc9001a, negotiated timeout = 40000
row=2, family=info, column=Age, value=22
row=2, family=info, column=name, value=Bob
row=2, family=score, column=math, value=90
```

3.2.5 删除数据

```
1 public static void deleteData(Connection connection, String tableName, String rowKey)
  throws Exception {
2   try (Table table = connection.getTable(TableName.valueOf(tableName))) {
3     Delete delete = new Delete(Bytes.toBytes(rowKey));
4     table.delete(delete);
5     System.out.println("已删除：" + rowKey + " from " + tableName);
6   }
7 }
```

运行结果：

在 `student` 表删除键值为 `2` 的行：

```
lfl@master:~/JavaAPI$ java -cp "$(hbase classpath):$(hadoop classpath):HBaseop.jar" HBaseop get student 2
```

```
lfl@master:~/JavaAPI$ java -cp "$(hbase classpath):$(hadoop classpath):HBaseop.jar" HBaseop delete student 2
```

```
已删除：2 from student
```

在 `hbase shell` 中再次扫描全表

```
hbase(main):022:0> scan 'student'
ROW                                         COLUMN+CELL
 1                                          column=info:Age, timestamp=1758287688324, value=18
 1                                          column=info:name, timestamp=1758287546003, value=Alice
 1                                          column=score:math, timestamp=1758287981723, value=95
1 row(s) in 0.5110 seconds
```

完整代码

```
1   import org.apache.hadoop.conf.Configuration;
```

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

public class HBaseop {

    public static void main(String[] args) throws Exception {
        if (args.length < 1) {
            System.out.println("用法：java HBaseDemo <command> [args...]");
            System.out.println("命令列表:");
            System.out.println("  createTable <tableName> <familyName>");
            System.out.println("  put <tableName> <rowKey> <family> <column> <value>");
            System.out.println("  get <tableName> <rowKey>");
            System.out.println("  delete <tableName> <rowKey>");
            System.exit(1);
        }

        String command = args[0];
        Configuration conf = HBaseConfiguration.create();

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

        try (Connection connection = ConnectionFactory.createConnection(conf);
             Admin admin = connection.getAdmin()) {

            switch (command) {
                case "createTable":
                    if (args.length < 3) {
                        System.out.println("用法：createTable <tableName> <family1> [family2] [family3] ...");
                        return;
                    }
                    String tableName = args[1];
                    String[] families = new String[args.length - 2];
                    System.arraycopy(args, 2, families, 0, families.length);
                    createTable(admin, tableName, families);
                    break;

                case "put":
                    if (args.length != 6) {
                        System.out.println("用法：put <tableName> <rowKey> <family> <column> <value>");
                        return;
                    }
                    insertData(connection, args[1], args[2], args[3], args[4], args[5]);
                    break;
```

```java
                case "get":
                    if (args.length != 3) {
                        System.out.println("用法: get <tableName> <rowKey>");
                        return;
                    }
                    getData(connection, args[1], args[2]);
                    break;

                case "delete":
                    if (args.length != 3) {
                        System.out.println("用法: delete <tableName> <rowKey>");
                        return;
                    }
                    deleteData(connection, args[1], args[2]);
                    break;

                default:
                    System.out.println("未知命令: " + command);
            }
        }
    }

    // 1. 创建表，支持多个列族
    public static void createTable(Admin admin, String tableName, String... familyNames) throws Exception {
        TableName tName = TableName.valueOf(tableName);

        if (admin.tableExists(tName)) {
            System.out.println("表已存在: " + tableName);
            return;
        }

        HTableDescriptor tableDescriptor = new HTableDescriptor(tName);

        for (String family : familyNames) {
            HColumnDescriptor columnDescriptor = new HColumnDescriptor(family);
            tableDescriptor.addFamily(columnDescriptor);
        }

        admin.createTable(tableDescriptor);
        System.out.println("表已创建: " + tableName + ", 列族: " + String.join(",", familyNames));
    }

    // 2. 插入数据
    public static void insertData(Connection connection, String tableName, String rowKey,
                                  String family, String column, String value) throws Exception {
        try (Table table = connection.getTable(TableName.valueOf(tableName))) {
            Put put = new Put(Bytes.toBytes(rowKey));
```

```
 98              put.addColumn(Bytes.toBytes(family), Bytes.toBytes(column),
     Bytes.toBytes(value));
 99              table.put(put);
100              System.out.println("插入成功：" + rowKey + " -> " + family + ":" + column
     + "=" + value);
101          }
102      }
103
104      // 3．查询数据
105      public static void getData(Connection connection, String tableName, String
     rowKey) throws Exception {
106          try (Table table = connection.getTable(TableName.valueOf(tableName))) {
107              Get get = new Get(Bytes.toBytes(rowKey));
108              Result result = table.get(get);
109              if (result.isEmpty()) {
110                  System.out.println("未找到数据：" + rowKey);
111              } else {
112                  result.listCells().forEach(cell -> {
113                      String family = Bytes.toString(cell.getFamilyArray(),
     cell.getFamilyOffset(), cell.getFamilyLength());
114                      String qualifier = Bytes.toString(cell.getQualifierArray(),
     cell.getQualifierOffset(), cell.getQualifierLength());
115                      String value = Bytes.toString(cell.getValueArray(),
     cell.getValueOffset(), cell.getValueLength());
116                      System.out.println("row=" + rowKey + ", family=" + family + ",
     column=" + qualifier + ", value=" + value);
117                  });
118              }
119          }
120      }
121
122      // 4．删除数据
123      public static void deleteData(Connection connection, String tableName, String
     rowKey) throws Exception {
124          try (Table table = connection.getTable(TableName.valueOf(tableName))) {
125              Delete delete = new Delete(Bytes.toBytes(rowKey));
126              table.delete(delete);
127              System.out.println("已删除：" + rowKey + " from " + tableName);
128          }
129      }
130 }
```

四、实验总结

本次实验中，我成功使用 Java API 实现了对hdfs和hbase的各种操作，例如创建文件、删除文件、上传文件、读取文件内容等常见hdfs处理操作，以及在hbase中创建表和列族、插入数据、查询数据、删除数据等操作。

通过本次实验，我的分布式系统开发能力得到了显著提升，不仅掌握了 Java 提供的对hdfs和hbase的API，还学会了如何在集群环境中部署和调试应用程序。同时，也对分布式计算的优势和挑战有了更深入的理解，为今后进一步学习和应用大数据技术奠定了坚实的基础。在未来的学习和工作中，我将继续探索更多大数据处理技术和框架，不断提升自己的专业技能。