第 5 章 HBase API 操作

5.1 环境准备

新建项目后在 pom.xml 中添加依赖：

```
<dependency>
    <groupId>org.apache.hbase</groupId>
    <artifactId>hbase-server</artifactId>
    <version>1.3.1</version>
</dependency>

<dependency>
    <groupId>org.apache.hbase</groupId>
    <artifactId>hbase-client</artifactId>
    <version>1.3.1</version>
</dependency>

<dependency>
    <groupId>jdk.tools</groupId>
    <artifactId>jdk.tools</artifactId>
    <version>1.8</version>
    <scope>system</scope>
    <systemPath>${JAVA_HOME}/lib/tools.jar</systemPath>
</dependency>
```

5.2 HBaseAPI

5.2.1 获取 Configuration 对象：

```
public static Configuration conf;
static{
    //使用 HBaseConfiguration 的单例方法实例化
    conf = HBaseConfiguration.create();
    conf.set("hbase.zookeeper.quorum", "192.168.216.20");
    conf.set("hbase.zookeeper.property.clientPort", "2181");
}
```

5.2.2 判断表是否存在：

```
public static boolean isTableExist(String tableName) throws MasterNotRunningException,
ZooKeeperConnectionException, IOException{
    //在 HBase 中管理、访问表需要先创建 HBaseAdmin 对象
    //Connection connection = ConnectionFactory.createConnection(conf);
    //HBaseAdmin admin = (HBaseAdmin) connection.getAdmin();
```

```
    HBaseAdmin admin = new HBaseAdmin(conf);
    return admin.tableExists(tableName);
}
```

5.2.3 创建表

```
public static void createTable(String tableName, String... columnFamily) throws
MasterNotRunningException, ZooKeeperConnectionException, IOException{
    HBaseAdmin admin = new HBaseAdmin(conf);
    //判断表是否存在
    if(isTableExist(tableName)){
        System.out.println("表" + tableName + "已存在");
        //System.exit(0);
    }else{
        //创建表属性对象,表名需要转字节
        HTableDescriptor descriptor = new HTableDescriptor(TableName.valueOf(tableName));
        //创建多个列族
        for(String cf : columnFamily){
            descriptor.addFamily(new HColumnDescriptor(cf));
        }
        //根据对表的配置，创建表
        admin.createTable(descriptor);
        System.out.println("表" + tableName + "创建成功！");
    }
}
```

5.2.4 删除表

```
public static void dropTable(String tableName) throws MasterNotRunningException,
ZooKeeperConnectionException, IOException{
    HBaseAdmin admin = new HBaseAdmin(conf);
    if(isTableExist(tableName)){
        admin.disableTable(tableName);
        admin.deleteTable(tableName);
        System.out.println("表" + tableName + "删除成功！");
    }else{
        System.out.println("表" + tableName + "不存在！");
    }
}
```

5.2.5 向表中插入数据

```
public static void addRowData(String tableName, String rowKey, String columnFamily, String
column, String value) throws IOException{
    //创建 HTable 对象
    HTable hTable = new HTable(conf, tableName);
    //向表中插入数据
```

```
        Put put = new Put(Bytes.toBytes(rowKey));
        //向 Put 对象中组装数据
        put.add(Bytes.toBytes(columnFamily), Bytes.toBytes(column), Bytes.toBytes(value));
        hTable.put(put);
        hTable.close();
        System.out.println("插入数据成功");
}
```

5.2.6 删除多行数据

```
public static void deleteMultiRow(String tableName, String... rows) throws IOException{
        HTable hTable = new HTable(conf, tableName);
        List<Delete> deleteList = new ArrayList<Delete>();
        for(String row : rows){
                Delete delete = new Delete(Bytes.toBytes(row));
                deleteList.add(delete);
        }
        hTable.delete(deleteList);
        hTable.close();
}
```

5.2.7 获取所有数据

```
public static void getAllRows(String tableName) throws IOException{
        HTable hTable = new HTable(conf, tableName);
        //得到用于扫描 region 的对象
        Scan scan = new Scan();
        //使用 HTable 得到 resultcanner 实现类的对象
        ResultScanner resultScanner = hTable.getScanner(scan);
        for(Result result : resultScanner){
                Cell[] cells = result.rawCells();
                for(Cell cell : cells){
                        //得到 rowkey
                        System.out.println("行键:" + Bytes.toString(CellUtil.cloneRow(cell)));
                        //得到列族
                        System.out.println("列族" + Bytes.toString(CellUtil.cloneFamily(cell)));
                        System.out.println("列:" + Bytes.toString(CellUtil.cloneQualifier(cell)));
                        System.out.println("值:" + Bytes.toString(CellUtil.cloneValue(cell)));
                }
        }
}
```

5.2.8 获取某一行数据

```
public static void getRow(String tableName, String rowKey) throws IOException{
        HTable table = new HTable(conf, tableName);
        Get get = new Get(Bytes.toBytes(rowKey));
```

```
        //get.setMaxVersions();显示所有版本
        //get.setTimeStamp();显示指定时间戳的版本
        Result result = table.get(get);
        for(Cell cell : result.rawCells()){
            System.out.println("行键:" + Bytes.toString(result.getRow()));
            System.out.println("列族" + Bytes.toString(CellUtil.cloneFamily(cell)));
            System.out.println("列:" + Bytes.toString(CellUtil.cloneQualifier(cell)));
            System.out.println("值:" + Bytes.toString(CellUtil.cloneValue(cell)));
            System.out.println("时间戳:" + cell.getTimestamp());
        }
}
```

5.2.9 获取某一行指定“列族:列”的数据

```
public static void getRowQualifier(String tableName, String rowKey, String family, String
qualifier) throws IOException{
        HTable table = new HTable(conf, tableName);
        Get get = new Get(Bytes.toBytes(rowKey));
        get.addColumn(Bytes.toBytes(family), Bytes.toBytes(qualifier));
        Result result = table.get(get);
        for(Cell cell : result.rawCells()){
            System.out.println("行键:" + Bytes.toString(result.getRow()));
            System.out.println("列族" + Bytes.toString(CellUtil.cloneFamily(cell)));
            System.out.println("列:" + Bytes.toString(CellUtil.cloneQualifier(cell)));
            System.out.println("值:" + Bytes.toString(CellUtil.cloneValue(cell)));
        }
}
```

5.3 MapReduce

通过 HBase 的相关 JavaAPI，我们可以实现伴随 HBase 操作的 MapReduce 过程，比如使用 MapReduce 将数据从本地文件系统导入到 HBase 的表中，比如我们从 HBase 中读取一些原始数据后使用 MapReduce 做数据分析。

5.3.1 官方 HBase-MapReduce

1）查看 HBase 的 MapReduce 任务的执行

```
$ bin/hbase mapredcp
```

2）执行环境变量的导入

```
$ export HBASE_HOME=/home/admin/modules/hbase-1.3.1
$ export HADOOP_HOME=/home/admin/modules/hadoop-2.7.2
$ export HADOOP_CLASSPATH=`${HBASE_HOME}/bin/hbase mapredcp`
```

3）运行官方的 MapReduce 任务

-- 案例一：统计 Student 表中有多少行数据
$ ~/modules/hadoop-2.7.2/bin/yarn jar lib/hbase-server-1.3.1.jar rowcounter student

-- 案例二：使用 MapReduce 将本地数据导入到 HBase
1）在本地创建一个 tsv 格式的文件：fruit.tsv

```
1001	Apple	Red
1002	Pear	Yellow
1003	Pineapple	Yellow
```

2）创建 HBase 表
hbase(main):001:0> create 'fruit','info'

3）在 HDFS 中创建 input_fruit 文件夹并上传 fruit.tsv 文件
$ ~/modules/hadoop-2.7.2/bin/hdfs dfs -mkdir /input_fruit/
$ ~/modules/hadoop-2.7.2/bin/hdfs dfs -put fruit.tsv /input_fruit/

4）执行 MapReduce 到 HBase 的 fruit 表中
$ ~/modules/hadoop-2.7.2/bin/yarn jar lib/hbase-server-1.3.1.jar importtsv \
-Dimporttsv.columns=HBASE_ROW_KEY,info:name,info:color fruit \
hdfs://linux01:8020/input_fruit

5）使用 scan 命令查看导入后的结果
hbase(main):001:0> scan ‘fruit’

5.3.2 自定义 HBase-MapReduce1

目标：将 fruit 表中的一部分数据，通过 MR 迁入到 fruit_mr 表中。

分步实现：

1) 构建 ReadFruitMapper 类，用于读取 fruit 表中的数据

```
package com.z.hbase_mr;

import java.io.IOException;
import org.apache.hadoop.hbase.Cell;
import org.apache.hadoop.hbase.CellUtil;
import org.apache.hadoop.hbase.client.Put;
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;

public class ReadFruitMapper extends TableMapper<ImmutableBytesWritable, Put> {

    @Override
```

```
    protected void map(ImmutableBytesWritable key, Result value, Context context)
    throws IOException, InterruptedException {
    //将 fruit 的 name 和 color 提取出来，相当于将每一行数据读取出来放入到 Put 对象中。
        Put put = new Put(key.get());
        //遍历添加 column 行
        for(Cell cell: value.rawCells()){
            //添加/克隆列族:info
            if("info".equals(Bytes.toString(CellUtil.cloneFamily(cell)))){
                //添加/克隆列：name
                if("name".equals(Bytes.toString(CellUtil.cloneQualifier(cell)))){
                    //将该列 cell 加入到 put 对象中
                    put.add(cell);
                    //添加/克隆列:color
                }else if("color".equals(Bytes.toString(CellUtil.cloneQualifier(cell)))){
                    //向该列 cell 加入到 put 对象中
                    put.add(cell);
                }
            }
        }
        //将从 fruit 读取到的每行数据写入到 context 中作为 map 的输出
        context.write(key, put);
    }
}
```

2) 构建 WriteFruitMRReducer 类，用于将读取到的 fruit 表中的数据写入到 fruit_mr 表中

```
package com.z.hbase_mr;

import java.io.IOException;
import org.apache.hadoop.hbase.client.Put;
import org.apache.hadoop.hbase.io.ImmutableBytesWritable;
import org.apache.hadoop.hbase.mapreduce.TableReducer;
import org.apache.hadoop.io.NullWritable;

public class WriteFruitMRReducer extends TableReducer<ImmutableBytesWritable, Put,
NullWritable> {
    @Override
    protected void reduce(ImmutableBytesWritable key, Iterable<Put> values, Context context)
    throws IOException, InterruptedException {
        //读出来的每一行数据写入到 fruit_mr 表中
        for(Put put: values){
            context.write(NullWritable.get(), put);
        }
    }
}
```

3) 构建 Fruit2FruitMRRunner extends Configured implements Tool 用于组装运行 Job 任务

```
    //组装 Job
    public int run(String[] args) throws Exception {
        //得到 Configuration
        Configuration conf = this.getConf();
        //创建 Job 任务
        Job job = Job.getInstance(conf, this.getClass().getSimpleName());
        job.setJarByClass(Fruit2FruitMRRunner.class);

        //配置 Job
        Scan scan = new Scan();
        scan.setCacheBlocks(false);
        scan.setCaching(500);

        //设置 Mapper，注意导入的是 mapreduce 包下的，不是 mapred 包下的，后者是老
版本
        TableMapReduceUtil.initTableMapperJob(
        "fruit", //数据源的表名
        scan, //scan 扫描控制器
        ReadFruitMapper.class,//设置 Mapper 类
        ImmutableBytesWritable.class,//设置 Mapper 输出 key 类型
        Put.class,//设置 Mapper 输出 value 值类型
        job//设置给哪个 JOB
        );
        //设置 Reducer
        TableMapReduceUtil.initTableReducerJob("fruit_mr", WriteFruitMRReducer.class,
job);
        //设置 Reduce 数量，最少 1 个
        job.setNumReduceTasks(1);

        boolean isSuccess = job.waitForCompletion(true);
        if(!isSuccess){
            throw new IOException("Job running with error");
        }
        return isSuccess ? 0 : 1;
    }
```

4）主函数中调用运行该 Job 任务

```
public static void main( String[] args ) throws Exception{
    Configuration conf = HBaseConfiguration.create();
    int status = ToolRunner.run(conf, new Fruit2FruitMRRunner(), args);
    System.exit(status);
}
```

5）打包运行任务

$ ~/modules/hadoop-2.7.2/bin/yarn jar ~/softwares/jars/hbase-0.0.1-SNAPSHOT.jar com.z.hbase.mr1.Fruit2FruitMRRunner

提示：运行任务前，如果待数据导入的表不存在，则需要提前创建。

提示：maven 打包命令：-P local clean package 或-P dev clean package install（将第三方 jar 包一同打包，需要插件：maven-shade-plugin）

5.3.3 自定义 HBase-MapReduce2

目标：实现将 HDFS 中的数据写入到 HBase 表中。

分步实现：

1）构建 ReadFruitFromHDFSMapper 于读取 HDFS 中的文件数据

```
package com.z.hbase.mr2;

import java.io.IOException;

import org.apache.hadoop.hbase.client.Put;
import org.apache.hadoop.hbase.io.ImmutableBytesWritable;
import org.apache.hadoop.hbase.util.Bytes;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class ReadFruitFromHDFSMapper extends Mapper<LongWritable, Text,
ImmutableBytesWritable, Put> {
    @Override
    protected void map(LongWritable key, Text value, Context context) throws IOException,
InterruptedException {
        //从 HDFS 中读取的数据
        String lineValue = value.toString();
        //读取出来的每行数据使用\t 进行分割，存于 String 数组
        String[] values = lineValue.split("\t");

        //根据数据中值的含义取值
        String rowKey = values[0];
        String name = values[1];
        String color = values[2];

        //初始化 rowKey
        ImmutableBytesWritable rowKeyWritable = new
ImmutableBytesWritable(Bytes.toBytes(rowKey));

        //初始化 put 对象
```

```
            Put put = new Put(Bytes.toBytes(rowKey));

            //参数分别:列族、列、值
            put.add(Bytes.toBytes("info"), Bytes.toBytes("name"),    Bytes.toBytes(name));
            put.add(Bytes.toBytes("info"), Bytes.toBytes("color"),    Bytes.toBytes(color));

            context.write(rowKeyWritable, put);
        }
}
```

2）构建 WriteFruitMRFromTxtReducer 类

```
package com.z.hbase.mr2;

import java.io.IOException;
import org.apache.hadoop.hbase.client.Put;
import org.apache.hadoop.hbase.io.ImmutableBytesWritable;
import org.apache.hadoop.hbase.mapreduce.TableReducer;
import org.apache.hadoop.io.NullWritable;

public class WriteFruitMRFromTxtReducer extends TableReducer<ImmutableBytesWritable, Put,
NullWritable> {
    @Override
    protected void reduce(ImmutableBytesWritable key, Iterable<Put> values, Context context)
throws IOException, InterruptedException {
        //读出来的每一行数据写入到 fruit_hdfs 表中
        for(Put put: values){
            context.write(NullWritable.get(), put);
        }
    }
}
```

3）创建 Txt2FruitRunner 组装 Job

```
public int run(String[] args) throws Exception {
//得到 Configuration
Configuration conf = this.getConf();

//创建 Job 任务
Job job = Job.getInstance(conf, this.getClass().getSimpleName());
job.setJarByClass(Txt2FruitRunner.class);
Path inPath = new Path("hdfs://linux01:8020/input_fruit/fruit.tsv");
FileInputFormat.addInputPath(job, inPath);

//设置 Mapper
job.setMapperClass(ReadFruitFromHDFSMapper.class);
job.setMapOutputKeyClass(ImmutableBytesWritable.class);
```

```
job.setMapOutputValueClass(Put.class);

//设置 Reducer
TableMapReduceUtil.initTableReducerJob("fruit_mr", WriteFruitMRFromTxtReducer.class, job);

//设置 Reduce 数量，最少 1 个
job.setNumReduceTasks(1);

boolean isSuccess = job.waitForCompletion(true);
if(!isSuccess){
throw new IOException("Job running with error");
}

return isSuccess ? 0 : 1;
}
```

4）调用执行 Job

```
public static void main(String[] args) throws Exception {
        Configuration conf = HBaseConfiguration.create();
        int status = ToolRunner.run(conf, new Txt2FruitRunner(), args);
        System.exit(status);
}
```

5）打包运行

$ ~/modules/hadoop-2.7.2/bin/yarn jar ~/softwares/jars/hbase-0.0.1-SNAPSHOT.jar com.z.hbase.mr2.Txt2FruitRunner

提示：运行任务前，如果待数据导入的表不存在，则需要提前创建之。

提示：maven 打包命令：-P local clean package 或-P dev clean package install（将第三方 jar 包一同打包，需要插件：maven-shade-plugin）

5.4 与 Hive 的集成

5.4.1 HBase 与 Hive 的对比

1) Hive

(1) 数据仓库

Hive 的本质其实就相当于将 HDFS 中已经存储的文件在 Mysql 中做了一个双射关系，以方便使用 HQL 去管理查询。

(2) 用于数据分析、清洗

Hive 适用于离线的数据分析和清洗，延迟较高。

(3) 基于 HDFS、MapReduce

Hive 存储的数据依旧在 DataNode 上，编写的 HQL 语句终将是转换为 MapReduce 代码执行。

2) HBase

(1) 数据库

是一种面向列存储的非关系型数据库。

(2) 用于存储结构化和非结构话的数据
适用于单表非关系型数据的存储，不适合做关联查询，类似 JOIN 等操作。
(3) 基于 HDFS
数据持久化存储的体现形式是 Hfile，存放于 DataNode 中，被 ResionServer 以 region 的形式进行管理。
(4) 延迟较低，接入在线业务使用
面对大量的企业数据，HBase 可以直线单表大量数据的存储，同时提供了高效的数据访问速度。

5.4.2 HBase 与 Hive 集成使用

尖叫提示：HBase 与 Hive 的集成在最新的两个版本中无法兼容。所以，我们只能含着泪勇敢的重新编译：hive-hbase-handler-1.2.2.jar！！好气！！

环境准备

因为我们后续可能会在操作 Hive 的同时对 HBase 也会产生影响，所以 Hive 需要持有操作 HBase 的 Jar，那么接下来拷贝 Hive 所依赖的 Jar 包（或者使用软连接的形式）。

```
$ export HBASE_HOME=/home/admin/modules/hbase-1.3.1
$ export HIVE_HOME=/home/admin/modules/apache-hive-1.2.2-bin

$ ln -s $HBASE_HOME/lib/hbase-common-1.3.1.jar
$HIVE_HOME/lib/hbase-common-1.3.1.jar
$ ln -s $HBASE_HOME/lib/hbase-server-1.3.1.jar $HIVE_HOME/lib/hbase-server-1.3.1.jar
$ ln -s $HBASE_HOME/lib/hbase-client-1.3.1.jar $HIVE_HOME/lib/hbase-client-1.3.1.jar
$ ln -s $HBASE_HOME/lib/hbase-protocol-1.3.1.jar $HIVE_HOME/lib/hbase-protocol-1.3.1.jar
$ ln -s $HBASE_HOME/lib/hbase-it-1.3.1.jar $HIVE_HOME/lib/hbase-it-1.3.1.jar
$ ln -s $HBASE_HOME/lib/htrace-core-3.1.0-incubating.jar
$HIVE_HOME/lib/htrace-core-3.1.0-incubating.jar
$ ln -s $HBASE_HOME/lib/hbase-hadoop2-compat-1.3.1.jar
$HIVE_HOME/lib/hbase-hadoop2-compat-1.3.1.jar
$ ln -s $HBASE_HOME/lib/hbase-hadoop-compat-1.3.1.jar
$HIVE_HOME/lib/hbase-hadoop-compat-1.3.1.jar
```

同时在 hive-site.xml 中修改 zookeeper 的属性，如下：

```
<property>
    <name>hive.zookeeper.quorum</name>
    <value>linux01,linux02,linux03</value>
    <description>The list of ZooKeeper servers to talk to. This is only needed for read/write
locks.</description>
</property>
<property>
    <name>hive.zookeeper.client.port</name>
    <value>2181</value>
    <description>The port of ZooKeeper servers to talk to. This is only needed for read/write
```

```
locks.</description>
</property>
```

1) 案例一

目标：建立 Hive 表，关联 HBase 表，插入数据到 Hive 表的同时能够影响 HBase 表。

分步实现：

(1) 在 Hive 中创建表同时关联 HBase

```
CREATE TABLE hive_hbase_emp_table(
empno int,
ename string,
job string,
mgr int,
hiredate string,
sal double,
comm double,
deptno int)
STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
WITH SERDEPROPERTIES ("hbase.columns.mapping" =
":key,info:ename,info:job,info:mgr,info:hiredate,info:sal,info:comm,info:deptno")
TBLPROPERTIES ("hbase.table.name" = "hbase_emp_table");
```

尖叫提示：完成之后，可以分别进入 Hive 和 HBase 查看，都生成了对应的表

(2) 在 Hive 中创建临时中间表，用于 load 文件中的数据

尖叫提示：不能将数据直接 load 进 Hive 所关联 HBase 的那张表中

```
CREATE TABLE emp(
empno int,
ename string,
job string,
mgr int,
hiredate string,
sal double,
comm double,
deptno int)
row format delimited fields terminated by '\t';
```

(3) 向 Hive 中间表中 load 数据

```
hive> load data local inpath '/home/admin/softwares/data/emp.txt' into table emp;
```

(4) 通过 insert 命令将中间表中的数据导入到 Hive 关联 HBase 的那张表中

```
hive> insert into table hive_hbase_emp_table select * from emp;
```

(5) 查看 Hive 以及关联的 HBase 表中是否已经成功的同步插入了数据

Hive：

```
hive> select * from hive_hbase_emp_table;
```

HBase：

```
hbase> scan ‘hbase_emp_table’
```

2) 案例二

目标：在 HBase 中已经存储了某一张表 hbase_emp_table，然后在 Hive 中创建一个外部表来关联 HBase 中的 hbase_emp_table 这张表，使之可以借助 Hive 来分析 HBase 这张表中的数据。

注：该案例 2 紧跟案例 1 的脚步，所以完成此案例前，请先完成案例 1。

分步实现：

(1) 在 Hive 中创建外部表

```
CREATE EXTERNAL TABLE relevance_hbase_emp(
empno int,
ename string,
job string,
mgr int,
hiredate string,
sal double,
comm double,
deptno int)
STORED BY
'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
WITH SERDEPROPERTIES ("hbase.columns.mapping" =
":key,info:ename,info:job,info:mgr,info:hiredate,info:sal,info:comm,info:deptno")
TBLPROPERTIES ("hbase.table.name" = "hbase_emp_table");
```

(2) 关联后就可以使用 Hive 函数进行一些分析操作了

```
hive (default)> select * from relevance_hbase_emp;
```