HBase 高级用法

刘磊

2020 年 10 月

1. Java API

HBase 主要包括 5 大类操作：HBase 的配置、HBase 表的管理、列族的管理、列的管理、数据操作等。

1）org.apache.hadoop.hbase.HBaseConfiguration

HBaseConfiguration 类用于管理 HBase 的配置信息

```
static Configuration conf = HBaseConfiguration.create();
```

2）org.apache.hadoop.hbase.client.Admin

Admin 是 Java 接口类型，不能直接用该接口来实例化一个对象，而是必须通过调用 Connection.getAdmin() 方法，返回一个 Admin 的子对象，然后用这个 Admin 接口来操作返回的子对象方法。该接口用来管理 HBase 数据库的表信息，包括创建表，删除表，列出表项，使表有效或无效，以及添加或删除表列族成员等。

```
Connection conn = ConnectionFactory.createConnection(conf);
HBaseAdmin admin = (HBaseAdmin) conn.getAdmin();
if (admin.tableExists(tableName)) {
        System.out.println("table exists!");
} else {
        HTableDescriptor tableDesc = new
HTableDescriptor(TableName.valueOf(tableName));
        for (String cf : columnFamilys) {
                HColumnDescriptor columnDescriptor = new
HColumnDescriptor(cf);
                columnDescriptor.setMaxVersions(1);
                tableDesc.addFamily(columnDescriptor);
```

```
        }
        admin.createTable(tableDesc);
}
```

3）org.apache.hadoop.hbase.HTableDescriptor

HTableDescriptor 包含了表的详细信息，例如表中的列族。创建表时添加列族代码如下

```
HTableDescriptor tableDesc = new HTableDescriptor
(TableName.valueOf(tableName));
tableDesc.addFamily (new HColumnDescriptor("name"));// 增加列族
tableDesc.addFamily(new HColumnDescriptor("age"));
tableDesc.addFamily(new HColumnDescriptor("gender"));
```

4）org.apache.hadoop.hbase.HColumnDescriptor

HColumnDescriptor 类维护着关于列族的信息，如版本号、压缩设置等。它通常在创建表或者为表添加列族的时候使用。列族被创建后不能直接修改，只能先删除然后重新创建。列族被删除的时候，列族里面的数据也会同时被删除。

5）org.apache.hadoop.hbase.client.Table

Table 是 Java 接口类型，不可以用 Table 直接实例化一个对象，而是必须通过调用 connection.getTable()，返回 Table 的一个子对象。这个接口可以用来和 HBase 表直接通信，可以从表中获取数据、添加数据、删除数据和扫描数据。例如删除表中一行数据

```
Table table = conn.getTable(TableName.valueOf(tableName))
Delete delete = new Delete(Bytes.toBytes(rowkey));
table.delete(delete);
```

6）org.apache.hadoop.hbase.client.Put

Put 类用来对单元执行添加数据操作。给表里添加数据

```
Put p1 = new Put(Bytes.toBytes(rowkey));
p1.addColumn(Bytes.toBytes(columnFamily),Bytes.toBytes(qualifer),Byte
s.toBytes(data));
table.put(p1);
```

7）org.apache.hadoop.hbase.client.Get

Get 类用来获取单行的数据。获取指定单元的数据的例子如下。

```
Get get = new Get(Bytes.toBytes(rowkey));
Result result = table.get(get);
```

8）org.apache.hadoop.hbase.client.Result

Result 类用来存放 Get 或 Scan 操作后的查询结果。

获取指定单元的数据的例子如下。

```
Get get = new Get(Bytes.toBytes(rowkey));
Result result = table.get(get);
System.out.println("Get: " + new
string(result.getValue(columnFamily.getBytes(),qualifier.getBytes()))
);
```

9）org.apache.hadoop.hbase.client.Scan

Scan 类可以用来限定需要查找的数据，如版本号、起始行号、终止行号、列族、列限定符、返回值的数量的上限等。设置 Scan 的列族、时间戳的范围和每次最多返回的单元数目的例子如下。

```
Scan scan = new Scan();
scan.addFamily(Bytes.toBytes("columnFamily1"));
scan.setTimeRange(1,3);
scan.setBatch(1000);
```

10）org.apache.hadoop.hbase.client.ResultScanner

ResultScanner 类是客户端获取值的接口，可以用来限定需要查找的数据，如版本号、起始行号、终止行号、列族、列限定符、返回值的数量的上限等。例如获取表中所有数据

```
Scan scan = new Scan();
ResultScanner scanner = table.getScanner(scan);
for (Result result : scanner) {
        System.out.println("-----------------");
```

```
        System.out.println("rowKey: " + new String(result.getRow()));
        System.out.println("address:city " + new
String(result.getValue("address".getBytes(),"city".getBytes())));
}
```

在 Eclipse 中运行上述程序需要导入依赖的 jar 包，将 HBase 目录/apps/hbase/lib 中所有的 jar 包和/apps/hbase/lib/client-facing-thirdparty 下的所有 jar 包导入到 hbase 的工程当中。

```
package sds.hbase;

import java.io.IOException;

import org.apache.hadoop.conf.Configuration;
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 HbaseJavaApi {
        public static Configuration conf; // 管理 HBase 的配置信息
        public static Connection conn;    // 管理 HBase 连接
        public static HBaseAdmin admin;   // 管理 HBase 数据库的信息

        // 建立连接
        public static void init() throws IOException {
                conf = HBaseConfiguration.create();
                conf.set("hbase.rootdir",
"hdfs://localhost:9000/hbase");
                // 使用 eclipse 时必须添加这个，否则无法定位
                //conf.set("hbase.zookeeper.quorum", "localhost");
                try{
               conn = ConnectionFactory.createConnection(conf);
               admin = (HBaseAdmin) conn.getAdmin();
            }catch(IOException e){
               e.printStackTrace();
            }
        }
```

```
        // 创建一张表
        public static void createTable(String tableName, String[]
columnFamilys)  throws IOException {
                try {
                        if (admin.tableExists(tableName)) {
                                System.out.println("table exists!");
                        } else {
                                HTableDescriptor tableDesc = new
HTableDescriptor(TableName.valueOf(tableName));
                                for (String cf : columnFamilys) {
                                        HColumnDescriptor
columnDescriptor = new HColumnDescriptor(cf);
                                    columnDescriptor.setMaxVersions(1);

tableDesc.addFamily(columnDescriptor);
                                }
                                admin.createTable(tableDesc);
                                System.out.println("create table
success!");
                        }
                }catch (Exception e){
           e.printStackTrace();
       }
        }

        // 查看表结构
        public static void descTable(String tableName) throws
Exception {
                try(Table table =
conn.getTable(TableName.valueOf(tableName))){
                HTableDescriptor desc =table.getTableDescriptor();
                HColumnDescriptor[] columnFamilies =
desc.getColumnFamilies();
                System.out.println("columnfamilys:");
                for(HColumnDescriptor t:columnFamilies){
                    System.out.println(Bytes.toString(t.getName()));
                }
            }catch (Exception e){
                    e.printStackTrace();
            }
    }

        // 添加一条记录
```

```
        public static boolean putRow(String tableName, String rowkey,
String columnFamily, String qualifer, String data) throws IOException
{
                try(Table table =
conn.getTable(TableName.valueOf(tableName))){
                        Put p1 = new Put(Bytes.toBytes(rowkey));

        p1.addColumn(Bytes.toBytes(columnFamily),Bytes.toBytes(qualif
er),Bytes.toBytes(data));
                        table.put(p1);
                        System.out.println("put'" + rowkey + "'," +
columnFamily + ":" + qualifer + "','" + data + "'");
                }catch (Exception e){
           e.printStackTrace();
       }
                return true;
        }

        // 读取一条记录
        public static Result getRow(String tableName, String rowkey,
String columnFamily,   String qualifier)
                        throws IOException {
                try(Table table =
conn.getTable(TableName.valueOf(tableName))){
                        Get get = new Get(Bytes.toBytes(rowkey));
                        Result result = table.get(get);
                        System.out.println("Get: "
                                        + new
String(result.getValue(columnFamily.getBytes(),qualifier.getBytes()))
);
                        return result;
                }catch (Exception e){
           e.printStackTrace();
       }
                return null;
        }

        // 获取所有数据
        public static ResultScanner scan(String tableName) throws
IOException {
                try(Table table =
conn.getTable(TableName.valueOf(tableName))){
                        Scan scan = new Scan();
```

```
                        ResultScanner scanner =
table.getScanner(scan);
                        return scanner;
                }catch (Exception e){
            e.printStackTrace();
        }
                return null;
         }

         // 删除表
         public static boolean deleteTable(String tableName) throws
IOException {
                if (admin.tableExists(tableName)) {
                        try {
                                admin.disableTable(tableName);
                                admin.deleteTable(tableName);
                        } catch (IOException e) {
                                e.printStackTrace();
                                System.out.println("Delete " +
tableName + " 失败");
                        }
                }
                return true;
         }

    // 删除行
    public static boolean deleteRow(String tableName, String rowkey){
         try(Table table =
conn.getTable(TableName.valueOf(tableName))){
                Delete delete = new Delete(Bytes.toBytes(rowkey));
                table.delete(delete);
        }catch (Exception e){
            e.printStackTrace();
        }
        return true;
    }

    // 删除列簇
    public static boolean deleteColumnFamily(String tableName, String
columnFamily){
        try{
            admin.deleteColumn(tableName,columnFamily);
        }catch (Exception e){
            e.printStackTrace();
```

```
        }
        return true;
    }

    // 删除列
   public static boolean deleteQualifier(String tableName,String
rowkey, String columnFamily, String qualifer){
      try(Table table = conn.getTable(TableName.valueOf(tableName))){
         Delete delete = new Delete(Bytes.toBytes(rowkey));

delete.addColumn(Bytes.toBytes(columnFamily),Bytes.toBytes(qualifer))
;
         table.delete(delete);
      }catch (Exception e){
         e.printStackTrace();
      }
      return true;
   }

   // 关闭连接
   public static void close(){
               try {
                       if (admin != null) {
                               admin.close();
                       }
                       if (conn != null) {
                               conn.close();
                       }
               }catch (Exception e){
          e.printStackTrace();
               }
   }

        /**
         * @param args
         */
        public static void main(String[] args) throws Exception {

               // 初始化
               init();
               // 创建具有两个列族的表 students
               String [] columnFamilys = {"address","info"};
               createTable("students", columnFamilys);
               // 显示表中的列族
```

```
                descTable("students");
                // 添加数据
                putRow("students", "xiaoming", "address", "province",
"zhejiang");
                putRow("students", "xiaoming", "address", "city",
"jinhua");
                putRow("students", "xiaoming", "info", "age", "20");
                putRow("students", "xiaowang", "address", "city",
"hangzhou");
                // 读取数据
                getRow("students", "xiaoming", "info", "age");
                // 获取所有数据
                ResultScanner scanner = scan("students");
                for (Result result : scanner) {
                        System.out.println("-----------------");
                        System.out.println("rowKey: " + new
String(result.getRow()));
                        System.out.println("address:city " + new
String(result.getValue("address".getBytes(),"city".getBytes())));
                }
                // 删除表
                deleteTable("students");
                close();
        }
}
```

2. 将 tsv 文件数据存入 HBase

3. 在 HDFS 上创建目录/input/files/music/，用于存放数据文件

```
hadoop fs -mkdir /input/music/
```

4. 将/data 下的数据文件 music.txt 上传到 hdfs，

```
hadoop fs -put ~/music.txt /input/music/
```

5. 调用 `HBase` 提供的 `importtsv` 工具在 `HBase` 上创建表 `music`，并指定列族和列。命令格式：

```
Usage: importtsv -Dimporttsv.columns=a,b,c <tablename> <inputdir>
```

```
hadoop jar /apps/hbase/lib/hbase-server-1.4.10.jar importtsv -
Dimporttsv.bulk.output=tmp -
Dimporttsv.columns=HBASE_ROW_KEY,info:name,info:singer,info:gender,in
fo:ryghme,info:terminal music /input/music/
```

注意：tmp 目录的位置为/user/lei/注意将路径中的 lei 替换为你自己的用户名），不能提前存在，否则会报错。另外如果不加-Dimporttsv.bulk.output 选项需要提前创建表格，数据可以直接写入 HBase。

在 HBase 中使用 list 命令查看表是否创建成功

```
hbase(main):011:0> list
TABLE
music
students
tb
3 row(s) in 0.0130 seconds

=> ["music", "students", "tb"]
```

此时数据还没有存入 HBase，数据暂存在 HDFS 上的/user/lei/tmp（注意将路径中的 lei 替换为你自己的用户名）目录下，

```
hadoop fs -ls -R /user/lei/tmp/
```

```
lei@ubuntu:~$ hadoop fs -ls  -R /user/lei/tmp/
-rw-r--r--   1 lei supergroup          0 2020-08-31 16:01 /user/lei/tmp/_SUCCESS
drwxr-xr-x   - lei supergroup          0 2020-08-31 16:01 /user/lei/tmp/info
-rw-r--r--   1 lei supergroup       8799 2020-08-31 16:01 /user/lei/tmp/info/acb
afa3a82ed4d3fa06924b8083d380a
```

6. 要把数据存入 HBase，还需要调用 HBase 提供的 completebulkload 工具

```
hadoop jar /apps/hbase/lib/hbase-server-1.4.10.jar  completebulkload
tmp music
```

查看表中的数据

```
hbase(main):013:0* scan 'music'
ROW                   COLUMN+CELL
 10_song4_2016-1-11   column=info:gender, timestamp=1598860881999, value=woman
 10_song4_2016-1-11   column=info:name, timestamp=1598860881999, value=song4
 10_song4_2016-1-11   column=info:ryghme, timestamp=1598860881999, value=slow
 10_song4_2016-1-11   column=info:singer, timestamp=1598860881999, value=singer4
 10_song4_2016-1-11   column=info:terminal, timestamp=1598860881999, value=ios
 11_song6_2016-1-11   column=info:gender, timestamp=1598860881999, value=woman
 11_song6_2016-1-11   column=info:name, timestamp=1598860881999, value=song6
```

7. 创建表 namelist 用于存储结果

在下一节，我们将把 MapReduce 的结果存入 HBase，这里先创建好存结果的表格。

```
create 'namelist', 'details'
```

```
hbase(main):004:0> create 'namelist', 'details'
0 row(s) in 1.3830 seconds
```

3. HBase 与 MapReduce 整合

MapReduce 可以读取 HBase 中的数据作为输入，也可以将运行结果直接写入到 HBase 中。当 HBase 作为数据来源时，自定义 Mapper 需继承 TableMapper，当 HBase 作为数据流向时，自定义 Reducer 需继承 TableReducer。TableMapper 类和 TableReducer 类是 MapReduce 专门为读取写入 Hbase 数据表而定制的。另外需调用 TableMapReduceUtil 类的静态方法 initTableMapperJob 来标识作为数据来源的 HBase 表名和自定义的 Mapper 类，用 initTableReduerJob 来标识作为数据输出流向的 HBase 表名和自定义的 Reducer 类。

下面的例子演示了从我们前一节创建的表格 music 读取数据，统计歌曲播放时间，将结果存入表 namelist 中。

```
package sds.hbase;

import java.io.IOException;
import java.util.List;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.Cell;
import org.apache.hadoop.hbase.CellUtil;
import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.hbase.client.Put;
import org.apache.hadoop.hbase.client.Result;
import org.apache.hadoop.hbase.client.Scan;
import org.apache.hadoop.hbase.io.ImmutableBytesWritable;
import org.apache.hadoop.hbase.mapreduce.TableMapReduceUtil;
import org.apache.hadoop.hbase.mapreduce.TableMapper;
import org.apache.hadoop.hbase.mapreduce.TableReducer;
```

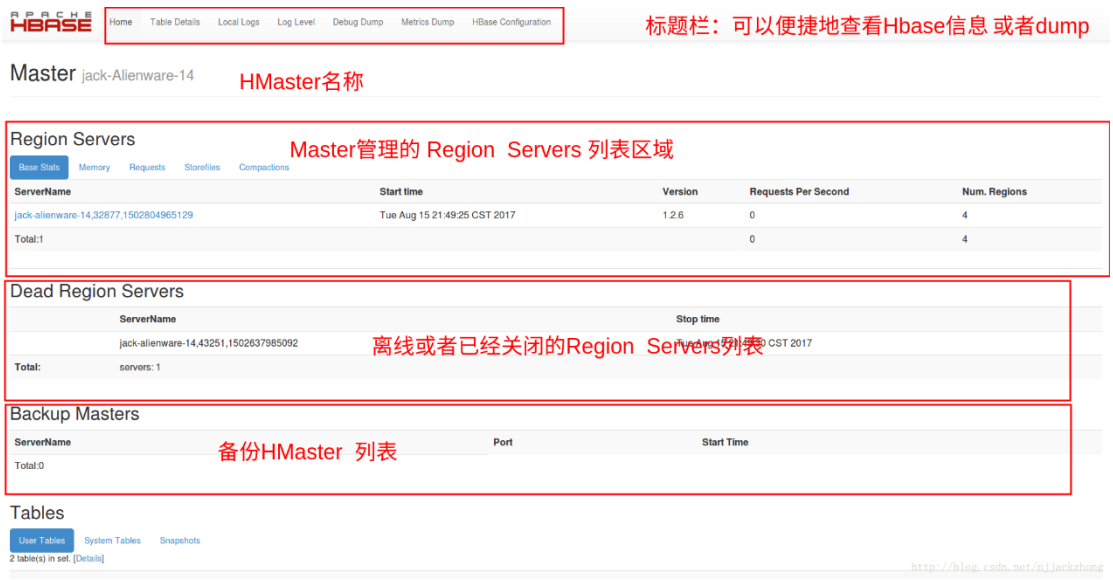
```
import org.apache.hadoop.hbase.util.Bytes;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.util.GenericOptionsParser;

public class TableMapReduceDemo {

        static class MyMapper extends TableMapper<Text, IntWritable>
{

                @Override
                protected void map(ImmutableBytesWritable key, Result
value,
                                Context context) throws IOException,
InterruptedException {
                        // 取出每行中的所有单元,实际上只扫描了一列
(info:name)
                        List<Cell> cells = value.listCells();
                        for (Cell cell : cells) {
                                context.write(
                                                new
Text(Bytes.toString(CellUtil.cloneValue(cell))),
                                                new IntWritable(1));
                        }
                }
        }

        static class MyReducer extends TableReducer<Text,
IntWritable, Text> {

                @Override
                protected void reduce(Text key, Iterable<IntWritable>
values,
                                Context context) throws IOException,
InterruptedException {
                        int playCount = 0;
                        for (IntWritable num : values) {
                                playCount += num.get();
                        }
                        // 为 Put 操作指定行键
                        Put put = new
Put(Bytes.toBytes(key.toString()));
                        // 为 Put 操作指定列和值
```

```
                        put.addColumn(Bytes.toBytes("details"),
Bytes.toBytes("rank"), Bytes.toBytes(playCount));
                        context.write(key, put);
                }

        }

        public static void main(String[] args) throws IOException,
                        ClassNotFoundException, InterruptedException {

                Configuration conf = HBaseConfiguration.create();
                conf.set("hbase.rootdir",
"hdfs://localhost:9000/hbase");
                conf.set("hbase.zookeeper.quorum", "localhost");
                Job job = Job.getInstance(conf, "top-music");

                // MapReduce 程序作业基本配置
                job.setJarByClass(TableMapReduceDemo.class);
                job.setNumReduceTasks(1);
                Scan scan = new Scan();
                scan.addColumn(Bytes.toBytes("info"),
Bytes.toBytes("name"));
                // 使用 hbase 提供的工具类来设置 job
                TableMapReduceUtil.initTableMapperJob("music", scan,
MyMapper.class,
                                Text.class, IntWritable.class, job);
                TableMapReduceUtil.initTableReducerJob("namelist",
MyReducer.class, job);
                job.waitForCompletion(true);
                System.out.println("执行成功，统计结果存于 namelist 表中。
");

        }
}
```

在 Eclipse 中创建工程运行上面的代码，由于版本的问题，使用新提供的压缩包，作为外部 jar 包导入。

4. Web UI

使用浏览器打开 http://localhost:16010 HBase Web 管理页，查看以下信息。


APACHE HBASE
Home
Table Details
Local Logs
Log Level
Debug Dump
Metrics Dump
HBase Configuration
标题栏：可以便捷地查看Hbase信息 或者dump
Master jack-Alienware-14
HMaster名称
Region Servers
Master管理的 Region Servers 列表区域
Base Stats
Memory
Requests
Storefiles
Compactions
ServerName
Start time
Version
Requests Per Second
Num. Regions
jack-alienware-14,32877,1502804965129
Tue Aug 15 21:49:25 CST 2017
1.2.6
0
4
Total:1
0
4
Dead Region Servers
ServerName
Stop time
jack-alienware-14,43251,1502637985092
Total:
servors: 1
离线或者已经关闭的Region Servers列表
Backup Masters
ServerName
Port
Start Time
Total:0
备份HMaster 列表
Tables
User Tables
System Tables
Snapshots
2 table(s) in set. [Details]
http://blog.csdn.net/njjackzhong