



0.1. 安装 HBase

第一节 简介

HBase 的设计来源于 Google 公司的 BigTable 论文，相当于是一个数据库，其查询的实时性较高。也是 Hadoop 生态系统中的一个很重要的工具。

第二节 配置文件

hbase-env.sh

```
export JAVA_HOME=/opt/java
export HBASE_MANAGES_ZK=false
export HADOOP_HOME=/opt/hadoop
HBASE_CLASSPATH=$HBASE_CLASSPATH:$HADOOP_HOME/etc/hadoop
```

hbase-site.xml

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
<property>
  <name>hbase.rootdir</name>
  <value>hdfs://ldcluster:8020/hbase</value>
</property>
<property>
  <name>hbase.cluster.distributed</name>
  <value>true</value>
</property>
<property>
  <name>hbase.unsafe.stream.capability.enforce</name>
  <value>false</value>
</property>
<property>
  <name>hbase.zookeeper.quorum</name>
  <value>master,slave1,slave2</value>
</property>
```

```
<property>
  <name>hbase.zookeeper.property.dataDir</name>
  <value>/home/zhangyu/zookeeperdata/</value>
</property>
</configuration>
```

我在配置 HBase 时，经常出现 HMaster 进程自动退出的问题，查看日志，调试了很长的时间。多半是因为访问时权限的问题。可以在 `hdfs-site.xml` 文件夹中加入相应的权限设置选项，取消权限设置。

并且按照官网推荐，在 HBase 的配置目录下面，添加一个指向 `hdfs-site.xml` 的软连接。

启动 HBase，需要先启动 HBase 服务。

启动 HBase

```
$ start-hbase.sh
$ jps
# 使用 Jps 查看进程可以看到主节点上多了 HMaster 进程和 HRegionServer 进程
# 在从节点上多了 HRegionServer 进程
# 如果出现过几秒后 HMaster 进程结束的情况，需要查看日志进行排查
```

第三节 HBase Shell 操作

下面是一些操作运行效果的截图：

表 1: HBase 操作指南

操作	命令表达式
创建表	create "表名", "列族 1", "列族 2", ..., "列族 n"
列出表	list
获取表的描述	desc "表名"
修改表	alter "表名", 应先 disable "表名"
删除表	drop "表名", 应先 disable "表名"
判断表是否存在	exists "表名"
判断表是否是 enable	is_enabled "表名"
判断表是否是 disable	is_disabled "表名"
插入数据	put "表名", "行键", "列族名: 列名", "值"
获取数据	get "表名", "行键", "列族名: 列名"
全表扫描	scan "表名"
删除数据	delete "表名", "行键", "列族名: 列名"
查询行数	count "表名"
清空该表	truncate "表名"
更新表中数据	执行插入语句覆盖原来数据

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL 1: java
=> true
hbase(main):007:0> exists 'user'
Table user does not exist
Took 0.0070 seconds
=> false
hbase(main):008:0> exists 'users'
Table users does exist
Took 0.0088 seconds
=> true
hbase(main):009:0> scan 'users'
ROW COLUMN+CELL
0 row(s)
Took 0.0958 seconds
hbase(main):010:0> desc 'users'
Table users is ENABLED
users
COLUMN FAMILIES DESCRIPTION
{NAME => 'address', VERSIONS => '1', EVICT_BLOCKS_ON_CLOSE => 'false', NEW_VERSION_BEHAVIOR => 'false', KEEP_DELETED_CELLS => 'FALSE',
, CACHE_DATA_ON_WRITE => 'false', DATA_BLOCK_ENCODING => 'NONE', TTL => 'FOREVER', MIN_VERSIONS => '0', REPLICATION_SCOPE => '0', B
LOOMFILTER => 'ROW', CACHE_INDEX_ON_WRITE => 'false', IN_MEMORY => 'false', CACHE_BLOOMS_ON_WRITE => 'false', PREFETCH_BLOCKS_ON_OPE
N => 'false', COMPRESSION => 'NONE', BLOCKCACHE => 'true', BLOCKSIZE => '65536'}

{NAME => 'info', VERSIONS => '1', EVICT_BLOCKS_ON_CLOSE => 'false', NEW_VERSION_BEHAVIOR => 'false', KEEP_DELETED_CELLS => 'FALSE',
, CACHE_DATA_ON_WRITE => 'false', DATA_BLOCK_ENCODING => 'NONE', TTL => 'FOREVER', MIN_VERSIONS => '0', REPLICATION_SCOPE => '0', BLOO
MFILTER => 'ROW', CACHE_INDEX_ON_WRITE => 'false', IN_MEMORY => 'false', CACHE_BLOOMS_ON_WRITE => 'false', PREFETCH_BLOCKS_ON_OPEN =
> 'false', COMPRESSION => 'NONE', BLOCKCACHE => 'true', BLOCKSIZE => '65536'}

2 row(s)
QUOTAS
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL 1: java
2 row(s)
Took 0.0101 seconds
=> ["test", "users"]
hbase(main):014:0> count 'test'
3 row(s)
Took 1.2100 seconds
=> 3
hbase(main):015:0> scan 'test'
ROW COLUMN+CELL
row0 column=cf:b, timestamp=1562568254204, value=value1
row1 column=cf:a, timestamp=1562568231619, value=value2
row2 column=cf:a, timestamp=1562568241642, value=value3
3 row(s)
Took 0.0288 seconds
hbase(main):016:0> get 'test', 'row0', 'cf:b'
COLUMN CELL
cf:b timestamp=1562568254204, value=value1
1 row(s)
Took 0.0286 seconds
hbase(main):017:0> get 'test', 'row0'
COLUMN CELL
cf:b timestamp=1562568254204, value=value1
1 row(s)
Took 0.0198 seconds
hbase(main):018:0> get 'test', 'row1'
COLUMN CELL
cf:a timestamp=1562568231619, value=value2
1 row(s)
Took 0.0136 seconds
hbase(main):019:0>

```

可以看出 HBase 的操作比较简单，这是其灵活性和扩展性的一种保障。

第四节 HBase Java API 使用

HBase 的 Java API 主要包括 5 大类的操作：HBase 的配置、HBase 表的管理、列族的管理、列的管理、数据的操作。

1) `org.apache.hadoop.hbase.HBaseConfiguration`。

HBaseConfiguration 类用于管理 HBase 的配置信息，使用举例如下。

```
static Configuration cfg = 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 `ConnectionFactory` 的 `getConnection()` 方法,来调用返回子对象的成员方法。该接口用来管理 HBase 数据库的表信息。它提供的方法包括创建表,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。

创建表使用的例子如下。

```
Configuration configuration = HBaseConfiguration.create();
ConnectionFactory connection = ConnectionFactory.createConnection(
    configuration);
Admin admin = connection.getAdmin();
if(admin.tableExists(tableName)) { //如果存在要创建的表,那么先删除,再创建
    admin.disableTable(tableName);
    admin.deleteTable(tableName);
}
admin.createTable(tableDescriptor);
admin.disableTable(tableName);
HColumnDescriptor hd = new HColumnDescriptor(columnFamily);
admin.addColumn(tableName,hd);
```

3) `org.apache.hadoop.hbase.HTableDescriptor`

HTableDescriptor 包含了表的详细信息。创建表时添加列族使用的例子如下。

```
HTableDescriptor tableDescriptor = new HTableDescriptor (tableName)
    ; // 表的数据模式
tableDescriptor.addFamily (new HColumnDescriptor("name")); // 增加列族
tableDescriptor.addFamily(new HColumnDescriptor("age"));
```

```
tableDescriptor.addFamily(new HColumnDescriptor("gender"));
admin.createTable(tableDescriptor);
```

4) 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()` 的一个子对象, 来调用返回子对象的成员方法。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。例子如下。

```
Configuration configuration = HBaseConfiguration.create();
Connection connection = ConnectionFactory.createConnection(
    configuration);
Table table = connection.getTable();
ResultScanner scanner = table.getScanner(family);
```

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

Put 类用来对单元执行添加数据操作。给表里添加数据的例子如下

```
Configuration configuration = HBaseConfiguration.create();
Connection connection = ConnectionFactory.createConnection(
    configuration);
Table table = connection.getTable();
Put put = new Put ("*1111".getBytes()); // 一个 Put 代表一行数据, 行
    键为构造方法中传入的值
put.addColumn("name".getBytes(), null, "Ghander".getBytes()); // 本行数
    据的第一列
put.addColumn("age".getBytes(), null, "20".getBytes()); // 本行数据的
    第二列
put.addColumn("gender".getBytes(), null, "male".getBytes()); // 本行数
    据的第三列
put.add("score".getBytes(), "Math".getBytes(), "99".getBytes()); // 本
    行数据的第四列
table.put(put);
```

6) 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。

```
Configuration configuration = HBaseConfiguration.create();
Connection connection = ConnectionFactory.createConnection(
    configuration);
```

```
Table table = connection.getTable();
Get g = new Get(rowKey.getBytes());
Result rs = table.get(g);
```

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

Result 类用来存放 Get 或 Scan 操作后的查询结果，并以<key,value>的格式存储在映射表中。获取指定单元的数据的例子如下。

```
Configuration configuration = HBaseConfiguration.create();
Connection connection = ConnectionFactory.createConnection(
    configuration);
Table table = connection.getTable();
Get g = new Get(rowKey.getBytes());
Result rs = table.get(g);
for (KeyValue kv : rs.raw()) {
    System.out.println("rowkey:"+new String(kv.getRow()));
    System.out.println("Column Family:"+new String(kv.getFamily()
    ));
    System.out.println("Column :"+ new String(kv.getQualifier()
    ));
    System.out.println("value :"+ new String(kv.getValue()));
}
```

8) 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);
```

9) 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。获取指定单元的数据的例子如下。

```
Scan scan = new Scan();
```

```

scan.addColumn(Bytes.toBytes("columnFamily1"),Bytes.toBytes("
    column1"));
scan.setTimeRange(1,3);
scan.setBatch(10);
Configuration configuration = HBaseConfiguration.create();
Connection connection = ConnectionFactory.createConnection(
    configuration);
Table table = connection.getTable();
try {
    ResultScanner resultScanner = table.getScanner(scan);
    Result rs = resultScanner.next();
    for (; rs != null; rs = resultScanner.next()){
        for (KeyValue kv : rs.list()){
            System.out.println("-----");
            System.out.println("rowkey:" + new String(kv
                .getRow()));
            System.out.println("Column Family: " + new
                String(kv.getFamily()));
            System.out.println("Column : " + new String(
                kv.getQualifier ()));
            System.out.println("value :"+ new String(kv
                .getValue()));
        }
    }
} catch (IOException e) {
    e.printStackTrace();
}

```

下面演示一个具体的编程实例来学习如何使用 HBase Java API 解决实际问题。在本实例中，首先创建一个学生成绩表 scores，用来存储学生各门课程的考试成绩，然后向 scores 添加数据。

表 scores 的概念视图如图1所示，用学生的名字 name 作为行键，年级 grade 是一个只有一个列的列族，score 是一个列族，每一门课程都是 score 的一个列，如 english、math、Chinese 等。score 的列可以随时添加。

例如，后续学生又参加了其他课程的考试，如 computing、physics 等，那么就可以添加到 score 列族。因为每个学生参加考试的课程也会不同，所以，并不一定表中的每一个单元都会有值。在该实例中，要向学生成绩表 scores 中添加的数据如图1所示。

下面利用 Maven 和 IDEA 创建工程，在其中添加下面的依赖：

name	grade	score		
		english	math	chinese

图 1: 学生成绩表 scores 的概念视图

name	grade	score		
		english	math	chinese
dandan	6	95	100	92
sansan	6	87	95	98

图 2: 学生成绩表 scores 的数据

添加 Hbase Maven 依赖

```
<!-- https://mvnrepository.com/artifact/org.apache.hbase/hbase -->
<dependency>
    <groupId>org.apache.hbase</groupId>
    <artifactId>hbase</artifactId>
    <version>2.2.0</version>
    <type>pom</type>
</dependency>
```

新建主程序，写入下面代码框架：

主程序框架

```
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.*;
import org.apache.hadoop.hbase.client.*;
import org.apache.hadoop.hbase.util.Bytes;
public class StudentScores {
    public static Configuration configuration; //HBase 配置信息
    public static Connection connection; //HBase 连接
    public static void main (String [] args) throws IOException{
        init(); //建立连接
        createTable(); //建表
        insertData(); //添加课程成绩
        insertData(); //添加课程成绩
    }
}
```

```

        insertData(); //添加课程成绩
        getData(); //浏览课程成绩
        close(); //关闭连接
    }

    public static void init () {.....} //建立连接
    public static void close () {.....} //关闭连接
    public static void createTable () {.....} //创建表
    public static void insertData () {.....} //添加课程成绩
    public static void getData () {.....} //浏览操程成绩
}

```

然后分别实现下面的几个函数：

init 函数，用于初始化连接

```

public static void init() {
    configuration = HBaseConfiguration.create();
    configuration.set("hbase.rootdir", "hdfs://ldcluster:8020/
        hbase");
    try {
        connection = ConnectionFactory.createConnection(
            configuration);
        admin = connection.getAdmin();
    } catch (IOException e) {
        e.printStackTrace();
    }
}

```

close 函数，用于结束断开连接

```

public static void close() {
    try {
        if (admin != null) {
            admin.close();
        }
        if (connection != null) {
            connection.close();
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}

```

```

    }
}

```

createTable 函数

```

public static void createTable(String myTableName, String[] colFamily
    ) throws IOException {
    TableName tableName = TableName.valueOf(myTableName);
    if (admin.tableExists(tableName)) {
        System.out.println("The " + myTableName + "exists!");
    } else {
        HTableDescriptor hTableDescriptor = new
            HTableDescriptor(tableName);
        for (String str :
            colFamily) {
            HColumnDescriptor columnDescriptor = new
                HColumnDescriptor(str);
            hTableDescriptor.addFamily(columnDescriptor);
        }
        admin.createTable(hTableDescriptor);
    }
}

```

向表中插入数据的函数

```

public static void insertData(String tableName, String rowKey, String
    colFamily, String col, String val) throws IOException {
    Table table = connection.getTable(TableName.valueOf(tableName
    ));
    Put put = new Put(rowKey.getBytes());
    put.addColumn(colFamily.getBytes(), col.getBytes(), val.
        getBytes());
    table.put(put);
    table.close();
}

```

从表中获取数据的函数

```

public static void getData(String tableName, String rowKey, String
    colFamily, String col) throws IOException {

```

```

        Table table = connection.getTable(TableName.valueOf(tableName
        ));
        Get get = new Get(rowKey.getBytes());
        get.addColumn(colFamily.getBytes(), col.getBytes());
        Result result = table.get(get);
        System.out.println(new String(result.getValue(colFamily.
        getBytes(), col.getBytes())));
        table.close();
    }

```

完整的主函数如下：

完整的主函数

```

public static void main(String[] args) throws IOException {
    String tableName = "scores";
    String[] colFamily = {"grade", "score"};

    init();
    createTable(tableName, colFamily);
    insertData(tableName, "dandan", "grade", "", "6");
    insertData(tableName, "dandan", "score", "english", "95");
    insertData(tableName, "dandan", "score", "math", "100");
    insertData(tableName, "dandan", "score", "chinese", "92");

    insertData(tableName, "sansan", "grade", "", "6");
    insertData(tableName, "sansan", "score", "english", "87");
    insertData(tableName, "sansan", "score", "math", "95");
    insertData(tableName, "sansan", "score", "chinese", "98");

    getData(tableName, "dandan", "score", "math");
    getData(tableName, "sansan", "score", "english");
    getData(tableName, "sansan", "grade", "");

    close();
}

```

运行结果如下：

```
Run: StudentScores x
/opt/java/bin/java ...
log4j:WARN No appenders could be found for logger (org.apache.hadoop.util.Shell).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
The score exists!
100
87
6
Process finished with exit code 0
```

可以在控制台看到输出的结果，但是需要运行等待的时间较长。

```
hbase(main):009:0> scan "scores"
ROW COLUMN+CELL
dandan column=grade:, timestamp=1563416876831, value=6
dandan column=score:chinese, timestamp=1563416876859, value=92
dandan column=score:english, timestamp=1563416876839, value=95
dandan column=score:math, timestamp=1563416876852, value=100
sansan column=grade:, timestamp=1563416876876, value=6
sansan column=score:chinese, timestamp=1563416876894, value=98
sansan column=score:english, timestamp=1563416876880, value=87
sansan column=score:math, timestamp=1563416876884, value=95
2 row(s)
Took 0.0204 seconds
hbase(main):010:0>
```

在 Hbase shell 中也可以看到表的信息

总结：采用 HBase Shell 操作的实时性较好，但是灵活性不够，适合演示或小量数据。而采用 Java API 编程能够加深对 HBase 各个模块的理解，而且更加灵活，适合各种复杂大数据量情况。