Hadoop 集群（第 12 期）

——HBase 应用开发

1、HBase基本操作

1.1 基本Shell操作

HBase 为用户提供了一个**非常方便**的**使用方式**，我们称之为“**HBase Shell**”。

HBase Shell 提供了**大多数**的 **HBase 命令**，通过 HBase Shell 用户可以方便地**创建**、**删除**及**修改**表，还可以向表中**添加**数据、**列出**表中的**相关信息**等。

备注：**写错** HBase Shell 命令时用**键盘**上的“**Delete**”进行删除，“Backspace”不起作用。

在**启动** HBase 之后，用户可以通过下面的命令**进入 HBase Shell** 之中，命令如下所示：

```
hbase shell
```

成功进入之后，用户将看到如下图所示的界面：

```
[hadoop@Master ~]$ hbase shell
HBase Shell; enter 'help<RETURN>' for list of supported commands.
Type "exit<RETURN>" to leave the HBase Shell
Version 0.92.0, r1231986, Mon Jan 16 13:16:35 UTC 2012

hbase(main):001:0>
```

进入 HBase Shell 之后，输入“**help**”，可以获取 HBase Shell 所支持的命令，执行结果如下图所示，下面只是其中一部分。

```
SHELL USAGE:
Quote all names in HBase Shell such as table and column names.  Commas delimit
command parameters.  Type <RETURN> after entering a command to run it.
Dictionaries of configuration used in the creation and alteration of tables are
Ruby Hashes. They look like this:

  {'key1' => 'value1', 'key2' => 'value2', ...}

and are opened and closed with curley-braces.  Key/values are delimited by the
'=>' character combination.  Usually keys are predefined constants such as
NAME, VERSIONS, COMPRESSION, etc.  Constants do not need to be quoted.  Type
'Object.constants' to see a (messy) list of all constants in the environment.

If you are using binary keys or values and need to enter them in the shell, use
double-quote'd hexadecimal representation. For example:

  hbase> get 't1', "key\x03\x3f\xcd"
  hbase> get 't1', "key\003\023\011"
  hbase> put 't1', "test\xef\xff", 'f1:', "\x01\x33\x40"

The HBase shell is the (J)Ruby IRB with the above HBase-specific commands added.
For more on the HBase Shell, see http://hbase.apache.org/docs/current/book.html
```

具体的 HBase Shell 命令如下表 1.1-1 所示：

表 1.1-1 HBase Shell 命令

HBase Shell 命令	描　述
alter	修改列族模式
count	统计表中行的数量
create	创建表
describe	显示表相关的详细信息
delete	删除指定对象值（可以为表、行、列对应的值）
deleteall	删除指定行的所有元素
disable	使表无效
drop	删除表
enable	使表有效
exists	测试表是否存在
exit	退出 HBase Shell
get	获取行或单元（cell）的值
incr	增加指定表、行或列的值
list	列出 HBase 中存在的所有表
put	向指定的表单元添加值
tools	列出 HBase 所支持的工具
scan	通过对表的扫描来获取对应的值
status	返回 HBase 集群的状态信息
shutdown	关闭 HBase 集群（与 exit 不同）
truncate	重新创建指定表
version	返回 HBase 版本信息

需要注意 shutdown 操作与 exit 操作之间的不同，shutdown 表示关闭 HBase 集群，必须重新启动 HBase 集群才可以恢复。exit 只是退出 HBase Shell，退出之后完全可以重新进入。

使用“**status**”命令查看 HBase 的运行状态，确保 HBase 正确运行，如下所示：

```
hbase(main):003:0> status
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/hbase/lib/slf4j-log4j12-1.5.8.j
SLF4J: Found binding in [jar:file:/usr/hadoop/lib/slf4j-log4j12-1.4.3.
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an ex
3 servers, 0 dead, 0.6667 average load
```

备注：上面出现了“**SLF4J**”多绑定情况，虽然是一个警告，每次运行命令出现总是一件烦人的事儿，到 SLF4J 官网地址中找到了答案。

地址：http://www.slf4j.org/codes.html#multiple_bindings

Multiple bindings were found on the class path

SLF4J API is desinged to bind with **one** and o**nly one** underlying logging framework **at a time**. If **more than one** binding is present **on the class path**, **SLF4J** will **emit** a **warning**, listing the location of those bindings. When this happens, select the one and only one binding you wish to use, and remove the other bindings.

For example, if you have both slf4j-simple-1.6.4.jar and slf4j-nop-1.6.4.jar on the class path and you wish to use the nop (no-operation) binding, then remove slf4j-simple-1.6.4.jar from the class path.

Note The warning emitted by SLF4J is just that, a warning. SLF4J will still bind with the first framework it finds on the class path.

官方的回答其实很清楚，“SLF4J”只能在“Class Path”中出现**一次**且**只能**有**一次**，所以解决办法就是删除掉其中的一个，**但是按照上面的要求执行完之后 ZooKeeper 不能启动了，最还对于这个问题暂且搁下，待以后解决。**

使用“version”命令查看 HBase 的版本信息，执行结果如下：

```
hbase(main):004:0> version
0.92.0, r1231986, Mon Jan 16 13:16:35 UTC 2012
```

下面我们将以“一个学生成绩表”的例子来详细介绍常用的 HBase 命令及其使用方法。

表 1.1-2 **scores**

name	grad	course		
		china	math	english
xiapi	1	97	128	85
xiaoxue	2	90	120	90

这里 **grad** 对于**表**来说是一个**列**，**course** 对于**表**来说是一个**列族**，这个**列族**由**三个列**组成 **china**、**math** 和 **english**，当然我们可以根据我们的需要在 course 中建立更多的列族，如 **computer**，**physics** 等相应的列添加入 course 列族。图中需要注意的是 90 这个值，列族下面的列也是可以没有名字的。（**备注**：列族下面的列也是可以**没有**名字的。）

1）create 命令

创建一个具有**两个列族**“**grad**”和“**course**”的表“**scores**”。其中**表名**、**行**和**列**都要用**单引号**括起来，并以**逗号**隔开。

```
create　‘scores’,’grad’,’course’
```

```
hbase(main):002:0> create 'scores','grad','course'
0 row(s) in 1.1030 seconds

hbase(main):003:0>
```

2）list 命令

查看当前 HBase 中具有哪些表。

```
hbase(main):003:0> list
TABLE
scores
1 row(s) in 0.0230 seconds

hbase(main):004:0>
```

3）describe 命令

查看表“scores”的构造。

```
describe　‘scores’
```

```
hbase(main):004:0> describe 'scores'
DESCRIPTION                                                                                                  ENABLED
 {NAME => 'scores', FAMILIES => [{NAME => 'course', BLOOMFILTER => 'NONE', REPLICATION_SCOP                   true
 E => '0', VERSIONS => '3', COMPRESSION => 'NONE', MIN_VERSIONS => '0', TTL => '2147483647'
 , BLOCKSIZE => '65536', IN_MEMORY => 'false', BLOCKCACHE => 'true'}, {NAME => 'grad', BLOO
 MFILTER => 'NONE', REPLICATION_SCOPE => '0', VERSIONS => '3', COMPRESSION => 'NONE', MIN_V
 ERSIONS => '0', TTL => '2147483647', BLOCKSIZE => '65536', IN_MEMORY => 'false', BLOCKCACH
 E => 'true'}]}
1 row(s) in 0.0210 seconds

hbase(main):005:0>
```

4）put 命令

使用 put 命令向表中插入数据，参数分别为表名、行名、列名和值，其中列名前需要列族最为前缀，时间戳由系统自动生成。

```
格式：put　表名，行名，列名（[列族 :列名]），值
```

- 加入一行数据，行名称为“**xiapi**”，列族“**grad**”的列名为””（空字符串），值位 **1**。

```
put　‘scores’,’xiapi’, ’grad :’,’1’
```

```
hbase(main):005:0> put 'scores','xiapi','grad:','1'
0 row(s) in 0.0830 seconds
```

- 给“**xiapi**”这一行的数据的列族“**course**”添加一列“**<china,97>**”。

```
put　‘scores’,’xiapi’,’course:china’,’97’
```

```
hbase(main):006:0> put 'scores','xiapi','course:china','97'
0 row(s) in 0.0160 seconds
```

- 给“**xiapi**”这一行的数据的列族“**course**”添加一列“**<math,128>**”。

```
put　‘scores’,’xiapi’,’course:math,’128’
```

```
hbase(main):007:0> put 'scores','xiapi','course:math','128'
0 row(s) in 0.0060 seconds
```

- 给“**xiapi**”这一行的数据的列族“**course**”添加一列“**<english,85>**”。

```
put　‘scores’,’xiapi’,’course:english,’85’
```

```
hbase(main):008:0> put 'scores','xiapi','course:english','85'
0 row(s) in 0.0100 seconds
```

备注：按照表 1.1-2 中的数据把第二行，根据添加“xiapi”这行操作进行。

```
hbase(main):009:0> put 'scores','xiaoxue','grad:','2'
0 row(s) in 0.0070 seconds

hbase(main):010:0> put 'scores','xiaoxue','course:china','90'
0 row(s) in 0.0060 seconds

hbase(main):011:0> put 'scores','xiaoxue','course:math','120'
0 row(s) in 0.0080 seconds

hbase(main):012:0> put 'scores','xiaoxue','course:english','90'
0 row(s) in 0.0080 seconds
```

5）get 命令

- 查看表“**scores**”中的行“**xiapi**”的相关数据。

```
get　‘scores’,’xiapi’
```

```
hbase(main):013:0> get 'scores','xiapi'
COLUMN                          CELL
 course:china                   timestamp=1332155284625, value=97
 course:english                 timestamp=1332155518951, value=85
 course:math                    timestamp=1332155439064, value=128
 grad:                          timestamp=1332154957572, value=1
4 row(s) in 0.0400 seconds
```

- 查看表“**scores**”中行“**xiapi**”列“**course :math**”的值。

```
get　‘scores’,’xiapi’,’course :math’
```

或者

```
get　‘scores’,’xiapi’,{COLUMN=>’course :math’}
```

```
hbase(main):025:0> get 'scores','xiapi','course:math'
COLUMN                                CELL
 course:math                          timestamp=1332163974408, value=128
1 row(s) in 0.0260 seconds

hbase(main):026:0> get 'scores','xiapi',{COLUMN=>'course:math'}
COLUMN                                CELL
 course:math                          timestamp=1332163974408, value=128
1 row(s) in 0.0100 seconds
```

备注：COLUMN 和 COLUMNS 是不同的，scan 操作中的 COLUMNS 指定的是表的列族，get 操作中的 COLUMN 指定的是特定的列，COLUMN 的值实质上为“列族：列修饰符”。

6）scan 命令

- 查看表“**scores**”中的所有数据。

```
scan　‘scores’
```

```
hbase(main):014:0> scan 'scores'
ROW                                   COLUMN+CELL
 xiaoxue                              column=course:china, timestamp=1332155769887, value=90
 xiaoxue                              column=course:english, timestamp=1332155827973, value=90
 xiaoxue                              column=course:math, timestamp=1332155800850, value=120
 xiaoxue                              column=grad:, timestamp=1332155728187, value=2
 xiapi                                column=course:china, timestamp=1332155284625, value=97
 xiapi                                column=course:english, timestamp=1332155518951, value=85
 xiapi                                column=course:math, timestamp=1332155439064, value=128
 xiapi                                column=grad:, timestamp=1332154957572, value=1
2 row(s) in 0.0320 seconds
```

注意：

scan 命令可以指定 startrow，stoprow 来 scan 多个 row。

例如：

scan ‘user_test’，{COLUMNS =>‘info:username’，LIMIT =>10, STARTROW => ‘test’，STOPROW=>‘test2’}

- 查看表“**scores**”中列族“**course**”的所有数据。

```
scan　‘scores’,{COLUMNS => ‘course’}
```

```
hbase(main):015:0> scan 'scores',{COLUMNS=>'course'}
ROW                                   COLUMN+CELL
 xiaoxue                              column=course:china, timestamp=1332155769887, value=90
 xiaoxue                              column=course:english, timestamp=1332155827973, value=90
 xiaoxue                              column=course:math, timestamp=1332155800850, value=120
 xiapi                                column=course:china, timestamp=1332155284625, value=97
 xiapi                                column=course:english, timestamp=1332155518951, value=85
 xiapi                                column=course:math, timestamp=1332155439064, value=128
2 row(s) in 0.0280 seconds
```

7）delete 命令

删除表“**scores**”中行为“**xiaoxue**”，列族“**course**”中的“**math**”。

```
delete　‘scores’,’xiaoxue’,’course:math’
```

```
hbase(main):016:0> delete 'scores','xiaoxue','course:math'
0 row(s) in 0.0080 seconds
```

8）disbale、drop 命令

通过“**disable**”和“**drop**”命令**删除**“scores”表。

```
hbase(main):017:0> disable 'scores'
0 row(s) in 2.0850 seconds

hbase(main):018:0> drop 'scores'
0 row(s) in 1.2610 seconds

hbase(main):019:0> list
TABLE
0 row(s) in 0.0130 seconds
```

另外，在 shell 中，**常量不需要**用**引号**引起来，但**二进制**的值需要**双引号**引起来，而**其他值**则用**单引号**引起来。HBase Shell 的常量可以通过在 shell 中输入“Object.constants”。

1.2 基本API介绍

当前 HBase 的 Java API 已经比较完善了，从其所涉及的内容来讲，大体包括：关于 HBase 自身的配置管理部分、关于 Avro 部分、关于 HBase 的部分客户端部分、关于 MapReduce 部分、关于 Rest 部分、关于 Thrift 部分，以及关于 ZooKeeper 部分。其中关于 HBase 自身的配置管理部分又包括：HBase 配置、日志、I/O、Master、Regionserver、replication，以及安全性。

这里我们重点介绍与 **HBase 数据库存储管理**相关的内容，其所涉及的主要类包括：HBaseAdmin、HBaseConfiguration、HTable、HTableDescriptor、Put、Get，以及 Scanner。关于 Java API 详细文档，可以查看 HBase 官方网站。

地址：http://hbase.apache.org/apidocs/index.html

表 1.2-1 描述了这几个相关类与 HBase 数据模型之间的对应关系。

表 1.2-1 Java API 与 HBase 数据模型之间的关系

Java 类	HBase 数据模型
HBase**Admin**	数据库（database）
HBase**Configuration**	
HTable	表（table）
HTableDescriptor	列族（column family）
Put	列修饰符（column qualifier）
Get	
Scanner	

下面将详细介绍这些类的功能，以及它们之间的相互关系。

1）HBaseConfiguration 类

关系：org.apache.hadoop.hbase.HBaseConfiguration

作用：通过此类可以对 HBase 进行配置。

主要方法如表 1.2-2 所示：

表 1.2-2 HBaseConfiguration 类包含的方法

返回值	函　数	描　述
void	addRescoure(Path file)	通过给定的路径所指的文件来添加资源
void	clear()	清空所有已设置的属性
string	get(String name)	获取属性名对应的值
string	getBoolean(String name,Boolean defaultValue)	获取为 boolean 类型的属性值，如果其属性值类型不为 boolean，则返回默认属性值
void	set(String name,String value)	通过设置属性名来设置值
void	setBoolean(String name,Boolean value)	设置为 boolean 类型的属性值

用法示例：

```
HBaseConfiguration hconfig = new HBaseConfiguration();
hconfig.set(“hbase.zookeeper.property.clientPort”,”2181”);
```

此方法设置了“hbase.zookeeper.property.clientPort”属性的端口号为 2181。一般情况下，HBaseConfiguration 会使用构造函数进行初始化，然后再使用其他方法添加必要的配置。

2）HBaseAdmin 类

关系：org.apache.hadoop.hbase.client.HBaseAdmin

作用：提供了一个接口来管理 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。

主要方法如表 1.2-3 所示：

表 1.2-3 HBaseAdmin 类包含的方法

返回值	函　数	描　述
void	addColumn(String tableName, HColumnDesciptor column)	向一个已存在的表添加列
void	checkHBaseAvailabe(HBaseConfiguration conf)	静态函数，查看 HBase 是否处于运行状态
void	createTable(HTableDescriptor desc)	创建一个新表，同步操作
void	deleteTable(byte[] tableName)	删除一个已存在的表
void	enableTable(byte[] tableName)	使表处于有效状态
void	disableTable(String tableName)	使表处于无效状态
HTableDescriptor[]	listTables()	列出所有用户空间表项
void	modifyTable(byte[] tableName, HTableDescriptor htd)	修改表的模式，是异步的操作，可能需要花费一定的时间
Boolean	tableExists(String tableName)	检查表是否存在，存在则返回 true

用法示例：

```
HBaseAdmin admin = new HBaseAdmin(config);
admin.disableTable("tablename");
```

上述例子通过一个 HBaseAdmin 实例 admin 调用 disableTable 方法来使表处于无效状态。

3）HTableDescriptor 类

关系：org.apache.hadoop.hbase.HTableDescriptor

作用：HTableDescriptor 类包含了表的名字及其对应表的列族。

主要方法如表 1.2-4 所示：

表 1.2.4 HTableDescriptor 类包含的方法

返回值	函　数	描　述
void	addFamily(HColumnDescriptor)	添加一个列族
HColumnDescriptor	removeFamily(byte[] column)	移除一个列族
byte[]	getName()	获取表的名字
byte[]	getValue(byte[] key)	获取属性的值
void	setValue(String key,String value)	设置属性的值

用法示例：

```
HTableDescriptor htd =new HTableDescriptor(tablename);
htd.addFamily(new HColumnDescriptor("Family"));
```

在上述例子中，通过一个 HColumnDescriptor 实例，为 HTableDescriptor 添加了一个列族：Family。

4）HColumnDescriptor 类

关系：org.apache.hadoop.hbase.HColumnDescriptor

作用：HColumnDescriptor 维护着关于列族的信息，例如：版本号、压缩设置等。它通常在创建表或为表添加列族时候使用。列族被创建后不能直接修改，只能通过删除然后重建的方式来“修改”。并且，当列族被删除的时候，对应列族中所有保存的数据也将被同时删除。

主要方法如表 1.2-5 所示：

表 1.2-5 HColumnDescriptor 类包含的方法

返回值	函　数	描　述
byte[]	getName()	获取列族的名字
byte[]	getValue(byte[] key)	获取对应属性的值
void	setValue(String key,String value)	设置对应属性的值

用法示例：

```
HTableDescriptor htd = new HTableDescriptor(tablename);
HColumnDescriptor col = new HColumnDescriptor(“content:”);
htd.addFamily(col);
```

此示例添加了一个名为“content”的列族。

5）HTable 类

关系：org.apache.hadoop.hbase.client.HTable

作用：此表可以用来与 HBase 表进行通信。此方法对于更新操作来说是非线程安全的，也就是说，如果有多个线程尝试与单个 HTable 实例进行通信，那么写缓冲器可能会崩溃。

主要方法如表 1.2-6 所示：

表 1.2-6 HTable 类包含的方法

返回值	函　数	描　述
void	checkAndPut(byte[] row,byte[] family, byte[] qualifier, byte[] value,Put put)	自动地检查 row/family/qualifier 是否与给定的值匹配
void	close()	释放所有的资源或挂起内部缓冲区的更新
Boolean	exists(Get get)	检查 Get 实例所指定的值是否存在于列中
Result	get(Get get)	取出指定行的某些单元格所对应的值
byte[][]	getEndKeys()	取出当前已打开的表每一个区域的结束键值
ResultScanner	getScanner(byte[] family)	获取当前表的给定列族的 scanner 示例
HTableDescriptor	getTableDescriptor()	获取当前表的 HTableDescriptor 示例
byte[]	getTableName()	获取表名
static Boolean	isTableEnabled(HBaseConfiguration conf String tableName)	检查表是否有效
void	put(Put put)	向表中添加值

用户示例：

```
HTable table = new HTable(conf,Bytes.toBytes(tablename));
ResultScanner scanner = table.getScanner(family);
```

6）Put 类

关系：org.apache.hadoop.hbase.client.Put

作用：用于对单个行执行添加操作。

主要方法如表 1.2-7 所示：

表 1.2-7 Put 类包含的方法

返回值	函　数	描　述
void	add(byte[] family,byte[] qualifier,byte[] value)	将指定的列和对应的值添加到 Put 实例中
Put	add(byte[] family,byte[] qualifier,long ts,byte[] value)	将指定的列和对应的值及时间戳添加到 Put
byte[]	getRow()	获取 Put 实例的行
RowLock	getRowLock()	获取 Put 实例的行锁
long	getTimeStamp()	获取 Put 实例的时间戳
Boolean	isEmpty()	检查 familyMap 是否为空
Put	setTimeStamp(long timestamp)	设置 Put 实例的时间戳

用法示例：

```
HTable table = new HTable(conf,Bytes.toBytes(tablename));
Put p = new Put(brow); //为指定行（brow）创建一个 Put 操作
p.add(family,qualifier,value);
table.put(p);
```

7）Put 类

关系：org.apache.hadoop.hbase.client.Get

作用：用来获取单个行的相关信息。

主要方法如表 1.2-8 所示：

表 1.2-8 Get 类包含的方法

返回值	函　数	描　述
Get	addColumn(byte[] family,byte[] qualifier)	获取指定列族和列修饰符对应的列
Get	addFamily(byte[] family)	通过指定的列族获取其对应的所有列
Get	setTimeRange(long minStamp,long maxStamp)	获取指定区域的列的版本号
Get	setFilter(Filter filter)	当执行 Get 操作时设置服务器端的过滤器

用法示例：

```
HTable table = new HTable(conf,Bytes.toBytes(tablename));
```

```
Get g = new Get(Bytes.toBytes(row));
```

8）Result 类

关系：org.apache.hadoop.hbase.client.Result

作用：存储 Get 或 Scan 操作后获取表的单行值。使用此类提供的方法能够直接方便的获取值或获取各种 Map 结构（key-value 对）。

主要方法如表 1.2-9 所示：

表 1.2-9 Result 类包含的方法

返回值	函　数	描　述
Boolean	containsColumn(byte[] family, byte[] qualifier)	检查指定的列是否存在
NavigableMap<byte[],byte[]>	getFamily(byte[] family)	返回值格式为：Map<byte[],byte[]>，获取对应列族所包含的修饰符与值的键值对
byte[]	getValue(byte[] family, byte[] qualifier)	获取对应列的最新值

用法示例：

```
HTable table = new HTable(conf,Bytes.toBytes(tablename));
Get g = Get(Byte.toBytes(row));
Result rowResult = table.get(g);
Bytes[] ret = rowResult.getValue((family+":"+column));
```

9）ResultScanner 类

关系：Interface

作用：客户端获取值的接口。

主要方法如表 1.2-10 所示：

表 1.2-10 ResultScanner 类包含的方法

返回值	函　数	描　述
void	close()	获取 scanner 并释放分配给它的所有资源
Result	next()	获取下一行的值

用法示例：

```
ResultScanner scanner = table.getScanner(Bytes.toBytes(family));
for(Result rowResult:scanner){
    Bytes[] str=rowResult.getValue(family,column);
}
```

1.3 使用API向导

1）配置

HBaseConfiguration 是每一个 **hbase client** 都会使用到的对象，它代表的是 **HBase 配置**信息。它有两种构造方式：

```
public HBaseConfiguration()
public HBaseConfiguration(final Configuration c)
```

默认的构造方式会尝试从 **hbase-default.xml** 和 **hbase-site.xml** 中读取配置。如果 classpath 没有这两个文件，就需要你自己设置配置。

```
Configuration conf = new Configuration();
conf.set("hbase.zookeeper.quorum", "Slave1.Hadoop,Slave2.Hadoop,Slave3.Hadoop");
conf.set("hbase.zookeeper.property.clientPort", "2181");
HBaseConfiguration cfg = new HBaseConfiguration(conf);
```

2）创建表

创建表是通过 **HBaseAdmin** 对象来操作的。HBaseAdmin 负责表的 **META 信息处理**。HBaseAdmin 提供了 **createTable** 这个方法：

```
public void createTable(HTableDescriptor desc)
```

HTableDescriptor 代表的是表的 schema，提供的方法中比较有用的有：

- setMaxFileSize：指定最大的 region size；
- setMemStoreFlushSize：指定 memstore flush 到 HDFS 上的文件大小。

增加 family 通过 **addFamily** 方法：

```
public void addFamily(final HColumnDescriptor family)
```

HColumnDescriptor 代表的是 column 的 schema，提供的方法比较常用的有：

- setTimeToLive：指定最大的 **TTL**，单位是 ms，过期数据会被自动删除；
- setInMemory：指定是否放在内存中，对小表有用，可用于提高效率，默认关闭；
- setBloomFilter：指定是否使用 BloomFilter,可**提高随机查询效率**，默认关闭；
- setCompressionType：设定**数据压缩**类型。默认无压缩；
- setMaxVersions：指定数据最大**保存**的版本个数，默认为 3。

一个简单的例子，创建了 4 个 family 的表：

```
HBaseAdmin hAdmin = new HBaseAdmin(hbaseConfig);
HTableDescriptor t = new HTableDescriptor(tableName);
```

```
t.addFamily(new HColumnDescriptor(“f1〞 ));
t.addFamily(new HColumnDescriptor(“f2〞 ));
t.addFamily(new HColumnDescriptor(“f3〞 ));
t.addFamily(new HColumnDescriptor(“f4〞 ));
hAdmin.createTable(t);
```

3）删除表

删除表也是通过 **HBaseAdmin** 来操作，**删除表**之前**首先**要 **disable** 表。这是一个**非常耗时**的操作，所以**不建议频繁删除表**。

disableTable 和 **deleteTable** 分别用来 **disable** 和 **delete** 表。

样例代码：

```
HBaseAdmin hAdmin = new HBaseAdmin(hbaseConfig);
if (hAdmin.tableExists(tableName)) {
        hAdmin.disableTable(tableName);
        hAdmin.deleteTable(tableName);
}
```

4）查询数据

查询分为**单条随机查询**和**批量查询**。

- **单条查询**是通过 rowkey 在 table 中查询某一行的数据。HTable 提供了 get 方法来完成单条查询。
- **批量查询**是通过制定一段 rowkey 的范围来查询。HTable 提供了个 getScanner 方法来完成批量查询。

```
public Result get(final Get get)
public ResultScanner getScanner(final Scan scan)
```

Get 对象包含了一个 Get 查询需要的信息。它的构造方法有两种：

```
public Get(byte [] row)
public Get(byte [] row, RowLock rowLock)
```

Rowlock 是为了**保证读写**的**原子性**，你可以传递一个**已经存在** Rowlock，否则 HBase 会**自动生成**一个**新**的 rowlock。

Scan 对象提供了默认构造函数，**一般**使用默认构造函数。

Get/Scan 的**常用方法**有：

- addFamily/addColumn：指定需要的 family 或 column，如果没有调用任何 addFamily 或 Column，会返回所有的 columns；
- setMaxVersions：指定最大的版本个数。如果不带任何参数调用 setMaxVersions，表

示取所有的版本。如果不掉用 setMaxVersions，只会取到最新的版本；

- setTimeRange：指定最大的时间戳和最小的时间戳，只有在此范围内的 cell 才能被获取；
- setTimeStamp：指定时间戳；
- setFilter：指定 Filter 来过滤掉不需要的信息。

Scan 特有的方法：

- setStartRow：指定开始的行。如果不调用，则从表头开始；
- setStopRow：指定结束的行（不含此行）；
- setBatch：指定最多返回的 Cell 数目。用于防止一行中有过多的数据，导致 OutofMemory 错误。

ResultScanner 是 Result 的一个容器，每次调用 ResultScanner 的 next 方法，会返回 Result。

```
public Result next() throws IOException;
public Result [] next(int nbRows) throws IOException;
```

Result 代表是一行的数据。**常用方法**有：

- getRow：返回 rowkey；
- raw：返回所有的 key value 数组；
- getValue：按照 column 来获取 cell 的值。

样例代码：

```
Scan s = new Scan();
s.setMaxVersions();
ResultScanner ss = table.getScanner(s);

for(Result r:ss){
    System.out.println(new String(r.getRow()));
    for(KeyValue kv:r.raw()){
        System.out.println(new String(kv.getColumn()));
    }
}
```

5）插入数据

HTable 通过 **put** 方法来**插入**数据。

```
public void put(final Put put) throws IOException
public void put(final List<Put> puts) throws IOException
```

可以传递**单个批 Put** 对象或者 **List put** 对象来**分别实现单条插入**和**批量插入**。

Put 提供了 **3 种构造方式**：

```
public Put(byte [] row)
public Put(byte [] row, RowLock rowLock)
public Put(Put putToCopy)
```

Put 常用的方法有：

- add：增加一个 Cell；
- setTimeStamp：指定所有 cell 默认的 timestamp，如果一个 Cell 没有指定 timestamp，就会用到这个值。如果没有调用，HBase 会将当前时间作为未指定 timestamp 的 cell 的 timestamp；
- setWriteToWAL：WAL 是 Write Ahead Log 的缩写，指的是 HBase 在插入操作前是否写 Log。默认是打开，关掉会提高性能，但是如果系统出现故障(负责插入的 Region Server 挂掉)，数据可能会丢失。

另外 **HTable** 也有**两个方**法也会**影响插入**的**性能**：

- setAutoFlash：AutoFlush 指的是在每次调用 HBase 的 Put 操作，是否提交到 HBase Server。默认是 true，每次会提交。如果此时是单条插入，就会有更多的 IO，从而降低性能；
- setWriteBufferSize：Write Buffer Size 在 AutoFlush 为 false 的时候起作用，默认是 2MB，也就是当插入数据超过 2MB，就会自动提交到 Server。

样例代码：

```
HTable table = new HTable(hbaseConfig, tableName);
table.setAutoFlush(autoFlush);

List<Put> lp = new ArrayList<Put>();
int count = 10000;
byte[] buffer = new byte[1024];
Random r = new Random();

for (int i = 1; i <= count; ++i) {
        Put p = new Put(String.format("row%09d",i).getBytes());
        r.nextBytes(buffer);

        p.add(“f1〃.getBytes(), null, buffer);
        p.add(“f2〃.getBytes(), null, buffer);
        p.add(“f3〃.getBytes(), null, buffer);
        p.add(“f4〃.getBytes(), null, buffer);

        p.setWriteToWAL(wal);
        lp.add(p);
```

```
        if(i%1000==0){
            table.put(lp);
            lp.clear();
        }
    }
```

6）删除数据

HTable 通过 **delete** 方法来删除数据。

```
public void delete(final Delete delete)
```

Delete 构造方法有：

```
public Delete(byte [] row)
public Delete(byte [] row, long timestamp, RowLock rowLock)
public Delete(final Delete d)
```

Delete 常用方法有：

- deleteFamily/deleteColumns：指定要删除的 family 或者 column 的数据。如果不调用任何这样的方法，将会删除整行。

注意：如果某个 Cell 的 timestamp **高于**当前时间，这个 Cell 将**不会**被**删**除，仍然可以查出来。

样例代码：

```
HTable table = new HTable(hbaseConfig, “mytest”);
Delete d = new Delete(“row1”.getBytes());
table.delete(d);
```

7）切分表

HBaseAdmin 提供 **split** 方法来将 **table** 进行 split。

```
public void split(final String tableNameOrRegionName)
```

如果提供的 tableName，那么会将 table 所有 region 进行 split ；如果提供的 region Name，那么只会 split 这个 region。

由于 split 是一个**异步操作**，我们**并不能确切**的控制 **region** 的**个数**。

样例代码：

```
public void split(String tableName,int number,int timeout) throws Exception {

    Configuration conf = new Configuration();
    conf.set(“hbase.zookeeper.quorum”, GlobalConf.ZOOKEEPER_QUORUM);
    conf.set(“hbase.zookeeper.property.clientPort”, GlobalConf.ZOOKEEPER_PORT);

HBaseConfiguration cfg = new HBaseConfiguration(conf);
HBaseAdmin hAdmin = new HBaseAdmin(cfg);

    HTable hTable = new HTable(cfg,tableName);

    int oldsize = 0;
    t =    System.currentTimeMillis();

    while(true){
        int size = hTable.getRegionsInfo().size();
        logger.info(“the region number=”+size);

        if(size>=number ) break;

        if(size!=oldsize){
            hAdmin.split(hTable.getTableName());
            oldsize = size;
        }
        else if(System.currentTimeMillis()-t>timeout){
            break;
        }

        Thread.sleep(1000*10);
    }
}
```

2、HBase开发配置

2.1 开发环境

Java 版本：jdk-**6u31**-windows-i586.exe
Win 系统：Windows 7 旗舰版
Eclipse 软件：eclipse-jee-**indigo**-SR1-win32.zip | eclipse-jee-**helios**-SR2-win32.zip
Hadoop 软件：hadoop-1.0.0.tar.gz
HBase 软件：hbase-0.92.0.tar.gz

2.2 配置Eclipse

通过 Eclipse 创建一个新 Java 工程，右击项目根目录，选择“Properties→ Java Build Path→ Library→ Add External JARs”，将 HBase 安装文件解压后根目录下的 **hbase-0.92.0.jar**、**hbase-0.92.0-tests.jar** 和 **lib** 子目录下所有的 **jar** 包添加到本工程的 **Classpath** 下。

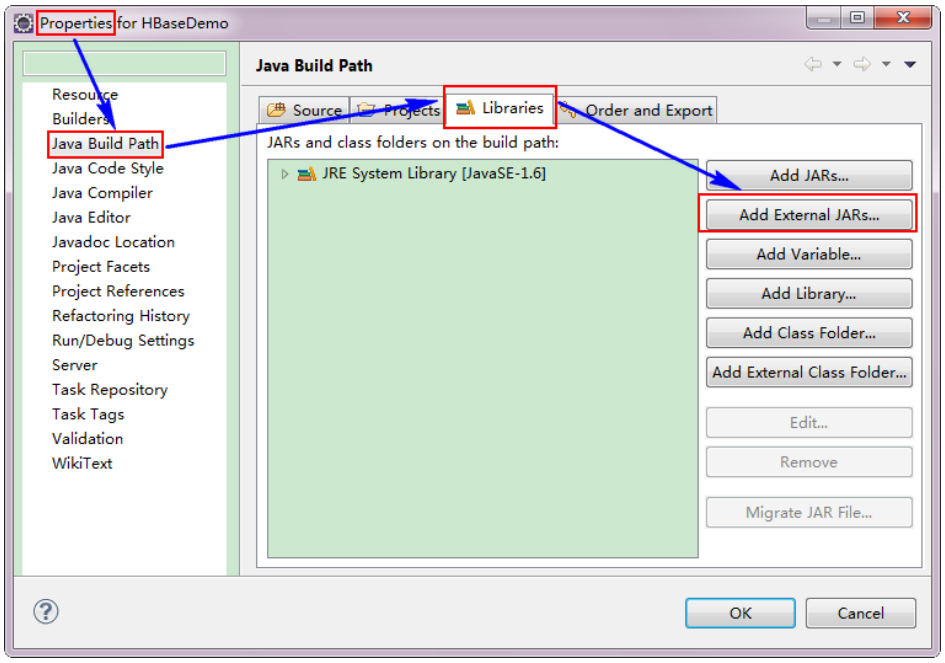

图 2.2-1 Java 工程添加额外 Jar 包

2.3 测试样例

本样例的功能是输出之前创建的表“**scores**”的列族名称：

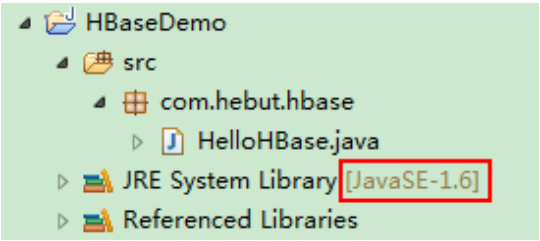

图 2.3-1 工程截图

样例代码：

```
package com.hebut.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.client.HBaseAdmin;
import org.apache.hadoop.hbase.util.Bytes;

public class HelloHBase {

    public static void main(String[] args) throws IOException {
        Configuration conf = HBaseConfiguration.create();

        conf.set("hbase.zookeeper.quorum", "192.168.1.3,192.168.1.4,192.168.1.5");
        conf.set("hbase.zookeeper.property.clientPort", "2181");

        HBaseAdmin admin = new HBaseAdmin(conf);
        HTableDescriptor tableDescriptor = admin.getTableDescriptor(Bytes
                .toBytes("scores"));

        byte[] name = tableDescriptor.getName();
        System.out.println("下面开始输出结果： ");

        System.out.println("表名： "+new String(name));

        HColumnDescriptor[] columnFamilies = tableDescriptor
                .getColumnFamilies();

        for (HColumnDescriptor d : columnFamilies) {
            System.out.println("列族名： "+d.getNameAsString());

        }
    }
}
```

运行结果：

```
12/03/20        10:29:18        INFO        zookeeper.ZooKeeper:        Client
environment:zookeeper.version=3.4.2-1221870, built on 12/21/2011 20:46 GMT
12/03/20        10:29:18        INFO        zookeeper.ZooKeeper:        Client
environment:host.name=ZD1LYV4S3T7OFLZ
12/03/20 10:29:18 INFO zookeeper.ZooKeeper: Client environment:java.version=1.6.0_31
……省略……
12/03/20 10:29:18 INFO zookeeper.ZooKeeper: Client environment:java.compiler=<NA>
12/03/20 10:29:18 INFO zookeeper.ZooKeeper: Client environment:os.name=Windows 7
12/03/20 10:29:18 INFO zookeeper.ZooKeeper: Client environment:os.arch=x86
12/03/20 10:29:18 INFO zookeeper.ZooKeeper: Client environment:os.version=6.1
```

```
12/03/20 10:29:18 INFO zookeeper.ZooKeeper: Client environment:user.name=hadoop
12/03/20 10:29:18 INFO zookeeper.ZooKeeper: Client environment:user.home=C:\Users\Administrator
12/03/20 10:29:18 INFO zookeeper.ZooKeeper: Client environment:user.dir=E:\HadoopWorkPlat\workspace\HBaseDemo
12/03/20 10:29:18 INFO zookeeper.ZooKeeper: Initiating client connection, connectString=192.168.1.4:2181,192.168.1.3:2181,192.168.5:2181 sessionTimeout=180000 watcher=hconnection
12/03/20 10:29:18 INFO zookeeper.ClientCnxn: Opening socket connection to server /192.168.1.3:2181
12/03/20 10:29:18 INFO zookeeper.RecoverableZooKeeper: The identifier of this process is 7732@ZD1LYV4S3T7OFLZ
12/03/20 10:29:18 INFO zookeeper.ClientCnxn: Socket connection established to Slave1.Hadoop/192.168.1.3:2181, initiating session
12/03/20 10:29:18 INFO zookeeper.ClientCnxn: Session establishment complete on server Slave1.Hadoop/192.168.1.3:2181, sessionid = 0x362a8d2a210004, negotiated timeout = 180000
下面开始输出结果：
表名：scores
列族名：course
列族名：grad
```

2、HBase初级案例

2.1 实战 1——HBase Java API应用

本小节主要是熟悉 HBase 开发，以及练习 HBase Java API 的基本使用，要创建的数据表如下表 2.1-1 所示。

表 2.1-1 **student**

name	info		course		
	age	sex	china	math	english
xiapi	20	boy	97	128	85
xiaoxue	19	boy	90	120	90
qingqing	18	girl	100	110	99

程序代码：

```
package com.hebut.hbase;

import java.util.ArrayList;
import java.util.List;
```

```
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.KeyValue;
import org.apache.hadoop.hbase.client.Delete;
import org.apache.hadoop.hbase.client.Get;
import org.apache.hadoop.hbase.client.HBaseAdmin;
import org.apache.hadoop.hbase.client.HTable;
import org.apache.hadoop.hbase.client.Put;
import org.apache.hadoop.hbase.client.Result;
import org.apache.hadoop.hbase.client.ResultScanner;
import org.apache.hadoop.hbase.client.Scan;
import org.apache.hadoop.hbase.util.Bytes;

public class HBaseJavaAPI {
    // 声明静态配置
    private static Configuration conf = null;
    static {
        conf = HBaseConfiguration.create();
        conf.set("hbase.zookeeper.quorum",
                "192.168.1.3,192.168.1.4,192.168.1.5");
        conf.set("hbase.zookeeper.property.clientPort", "2181");
    }

    // 创建数据库表
    public static void createTable(String tableName,
            String[] columnFamilys) throws Exception {
        // 新建一个数据库管理员
        HBaseAdmin hAdmin = new HBaseAdmin(conf);

        if (hAdmin.tableExists(tableName)) {
            System.out.println("表已经存在");
            System.exit(0);
        } else {
            // 新建一个 scores 表的描述
            HTableDescriptor tableDesc = new HTableDescriptor(tableName);
            // 在描述里添加列族
            for (String columnFamily : columnFamilys) {
                tableDesc.addFamily(new HColumnDescriptor(columnFamily));
            }
            // 根据配置好的描述建表
            hAdmin.createTable(tableDesc);
```

```
            System.out.println("创建表成功");
        }
    }

    // 删除数据库表
    public static void deleteTable(String tableName) throws Exception {
        // 新建一个数据库管理员
        HBaseAdmin hAdmin = new HBaseAdmin(conf);

        if (hAdmin.tableExists(tableName)) {
            // 关闭一个表
            hAdmin.disableTable(tableName);
            // 删除一个表
            hAdmin.deleteTable(tableName);
            System.out.println("删除表成功");

        } else {
            System.out.println("删除的表不存在");
            System.exit(0);
        }
    }

    // 添加一条数据
    public static void addRow(String tableName, String row,
    String columnFamily, String column, String value) throws Exception {
        HTable table = new HTable(conf, tableName);
        Put put = new Put(Bytes.toBytes(row));
        // 参数出分别：列族、列、值
        put.add(Bytes.toBytes(columnFamily), Bytes.toBytes(column),
                Bytes.toBytes(value));
        table.put(put);
    }

    // 删除一条数据
    public static void delRow(String tableName, String row)
    throws Exception {
        HTable table = new HTable(conf, tableName);
        Delete del = new Delete(Bytes.toBytes(row));
        table.delete(del);
    }

    // 删除多条数据
    public static void delMultiRows(String tableName, String[] rows)
            throws Exception {
```

```
        HTable table = new HTable(conf, tableName);
        List<Delete> list = new ArrayList<Delete>();

        for (String row : rows) {
            Delete del = new Delete(Bytes.toBytes(row));
            list.add(del);
        }

        table.delete(list);
    }

    // 获取一条数据
    public static void getRow(String tableName, String row)
    throws Exception {
        HTable table = new HTable(conf, tableName);
        Get get = new Get(Bytes.toBytes(row));
        Result result = table.get(get);
        // 输出结果
        for (KeyValue rowKV : result.raw()) {
            System.out.print("行名：" + new String(rowKV.getRow()) + " ");
            System.out.print("时间戳：" + rowKV.getTimestamp() + " ");
            System.out.print("列族名：" + new String(rowKV.getFamily()) + " ");
            System.out.print("列名：" + new String(rowKV.getQualifier()) + " ");
            System.out.println("值：" + new String(rowKV.getValue()) + " ");
        }
    }

    // 获取所有数据
    public static void getAllRows(String tableName) throws Exception {
        HTable table = new HTable(conf, tableName);
        Scan scan = new Scan();
        ResultScanner results = table.getScanner(scan);
        // 输出结果
        for (Result result : results) {
            for (KeyValue rowKV : result.raw()) {
                System.out.print("行名：" + new String(rowKV.getRow()) + " ");
                System.out.print("时间戳：" + rowKV.getTimestamp() + " ");
                System.out.print("列族名：" + new String(rowKV.getFamily()) + " ");
                System.out.print("列名：" + new String(rowKV.getQualifier()) + " ");
                System.out.println("值：" + new String(rowKV.getValue()) + " ");
            }
        }
    }
```

```
    // 主函数
    public static void main(String[] args) {
        try {
            String tableName = "student";

            // 第一步：创建数据库表：“student”
            String[] columnFamilys = { "info", "course" };
            HBaseJavaAPI.createTable(tableName, columnFamilys);

            // 第二步：向数据表的添加数据
            // 添加第一行数据
            HBaseJavaAPI.addRow(tableName, "xiapi", "info", "age", "20");
            HBaseJavaAPI.addRow(tableName, "xiapi", "info", "sex", "boy");
            HBaseJavaAPI.addRow(tableName, "xiapi", "course", "china", "97");
            HBaseJavaAPI.addRow(tableName, "xiapi", "course", "math", "128");
            HBaseJavaAPI.addRow(tableName, "xiapi", "course", "english", "85");
            // 添加第二行数据
            HBaseJavaAPI.addRow(tableName, "xiaoxue", "info", "age", "19");
            HBaseJavaAPI.addRow(tableName, "xiaoxue", "info", "sex", "boy");
            HBaseJavaAPI.addRow(tableName, "xiaoxue", "course", "china", "90");
            HBaseJavaAPI.addRow(tableName, "xiaoxue", "course", "math", "120");
            HBaseJavaAPI.addRow(tableName,"xiaoxue","course","english","90");
            // 添加第三行数据
            HBaseJavaAPI.addRow(tableName, "qingqing", "info", "age", "18");
            HBaseJavaAPI.addRow(tableName, "qingqing", "info", "sex", "girl");
            HBaseJavaAPI.addRow(tableName,"qingqing","course","china","100");
            HBaseJavaAPI.addRow(tableName, "qingqing", "course", "math", "100");
            HBaseJavaAPI.addRow(tableName,"qingqing","course","english","99");
            // 第三步：获取一条数据
            System.out.println("获取一条数据");
            HBaseJavaAPI.getRow(tableName, "xiapi");
            // 第四步：获取所有数据
            System.out.println("获取所有数据");
            HBaseJavaAPI.getAllRows(tableName);
            // 第五步：删除一条数据
            System.out.println("删除一条数据");
            HBaseJavaAPI.delRow(tableName, "xiapi");
            HBaseJavaAPI.getAllRows(tableName);
            // 第六步：删除多条数据
            System.out.println("删除多条数据");
            String[] rows = { "xiaoxue", "qingqing" };
            HBaseJavaAPI.delMultiRows(tableName, rows);
            HBaseJavaAPI.getAllRows(tableName);
            // 第八步：删除数据库
```

```
            System.out.println("删除数据库");
            HBaseJavaAPI.deleteTable(tableName);

        } catch (Exception err) {
            err.printStackTrace();
        }
    }
}
```

运行结果：

```
12/03/20 13:42:27 INFO zookeeper.ZooKeeper: Client environment:
zookeeper.version=3.4.2-1221870, built on 12/21/2011 20:46 GMT
12/03/20 13:42:27 INFO zookeeper.ZooKeeper: Client environment:
host.name=ZD1LYV4S3T7OFLZ
12/03/20 13:42:27 INFO zookeeper.ZooKeeper: Client environment:
java.version=1.6.0_31
12/03/20 13:42:27 INFO zookeeper.ZooKeeper: Client
.....信息.....
12/03/20 13:42:28 INFO zookeeper.ClientCnxn: EventThread shut down
创建表成功
获取一条数据
行名：xiapi 时间戳：1332251255417 列族名：course 列名：china 值：97
行名：xiapi 时间戳：1332251255432 列族名：course 列名：english 值：85
行名：xiapi 时间戳：1332251255427 列族名：course 列名：math 值：128
行名：xiapi 时间戳：1332251255400 列族名：info 列名：age 值：20
行名：xiapi 时间戳：1332251255411 列族名：info 列名：sex 值：boy
获取所有数据
行名：qingqing 时间戳：1332251255478 列族名：course 列名：china 值：100
行名：qingqing 时间戳：1332251255490 列族名：course 列名：english 值：99
行名：qingqing 时间戳：1332251255484 列族名：course 列名：math 值：100
行名：qingqing 时间戳：1332251255466 列族名：info 列名：age 值：18
行名：qingqing 时间戳：1332251255473 列族名：info 列名：sex 值：girl
行名：xiaoxue 时间戳：1332251255449 列族名：course 列名：china 值：90
行名：xiaoxue 时间戳：1332251255461 列族名：course 列名：english 值：90
行名：xiaoxue 时间戳：1332251255454 列族名：course 列名：math 值：120
行名：xiaoxue 时间戳：1332251255438 列族名：info 列名：age 值：19
行名：xiaoxue 时间戳：1332251255444 列族名：info 列名：sex 值：boy
行名：xiapi 时间戳：1332251255417 列族名：course 列名：china 值：97
行名：xiapi 时间戳：1332251255432 列族名：course 列名：english 值：85
行名：xiapi 时间戳：1332251255427 列族名：course 列名：math 值：128
行名：xiapi 时间戳：1332251255400 列族名：info 列名：age 值：20
行名：xiapi 时间戳：1332251255411 列族名：info 列名：sex 值：boy
删除一条数据
```

```
行名：qingqing 时间戳：1332251255478 列族名：course 列名：china 值：100
行名：qingqing 时间戳：1332251255490 列族名：course 列名：english 值：99
行名：qingqing 时间戳：1332251255484 列族名：course 列名：math 值：100
行名：qingqing 时间戳：1332251255466 列族名：info 列名：age 值：18
行名：qingqing 时间戳：1332251255473 列族名：info 列名：sex 值：girl
行名：xiaoxue 时间戳：1332251255449 列族名：course 列名：china 值：90
行名：xiaoxue 时间戳：1332251255461 列族名：course 列名：english 值：90
行名：xiaoxue 时间戳：1332251255454 列族名：course 列名：math 值：120
行名：xiaoxue 时间戳：1332251255438 列族名：info 列名：age 值：19
行名：xiaoxue 时间戳：1332251255444 列族名：info 列名：sex 值：boy
删除多条数据
删除数据库
12/03/20 13:42:29 INFO zookeeper.ZooKeeper: Initiating client connection,
connectString=192.168.1.4:2181,192.168.1.3:2181,192.168.1.5:2181
sessionTimeout=180000
.....信息.....
12/03/20 13:42:29 INFO zookeeper.ClientCnxn: EventThread shut down
12/03/20 13:42:29 INFO client.HBaseAdmin: Started disable of student
12/03/20 13:42:30 INFO client.HBaseAdmin: Disabled student
12/03/20 13:42:31 INFO client.HBaseAdmin: Deleted student
删除表成功
```

备注：该程序用“Run AS→Java Application”运行。

2.2 实战 2——HBase与MapReduce

从“**Hadoop 集群_第 11 期_HBase 简介及安装**”的**图 1-1Hadoop 生态系统结构**中，我们得知，在**伪分布式模式**和**全分布式模式**下 **HBase** 是**架构**在 **HDFS** 上的。因此完全可以将 **MapReduce** 编程框架和 **HBase** **结合起来使用**。也就是说，将 HBase 作为底层“存储结构”，MapReduce 调用 HBase 进行特殊的处理，这样能够**充分结合 HBase 分布式大型数据库**和 **MapReduce 并行计算的优点**。

前面几期已经对 **MapReduce 处理过程**进行了**详细的介绍**，根据这些处理过程，**HBase** 为**每个阶段**提供了**相应的类**用来**处理表数据**。

1）**InputFormat 类**：HBase 实现了 **TableInputFormatBase** 类，该类提供了对**表数据**的**大部分操作**，其子类 **TableInputFormat** 则提供了完整的实现，用于**处理表数据并生成键值对**。TableInputFormat 类将数据表按照 Region 分割成 split，既有多少个 Regions 就有多个 splits。然后将 Region 按**行键**分成<key,value>对，**key 值**对应与**行健**，**value 值**为**该行所包含的数据**。

2）**Mapper 类和 Reducer 类**：HBase 实现了 **TableMapper** 类和 **TableReducer** 类，其中 TableMapper 类并没有具体的功能，只是将输入的<key,value>对的类型**分别限定**为 Result 和 ImmutableBytesWritable。**IdentityTableMapper** 类和 **IdentityTableReducer** 类则是上述两个类的具体实现，其和 Mapper 类和 Reducer 类一样，**只是简单**地将<key,value>对输出到下一个阶段。

3）**OutputFormat 类**：HBase 实现的 **TableOutputFormat** 将输出的<key,value>对写到指定的 HBase 表中，该类**不会**对 **WAL（Write-Ahead Log）**进行操作，即如果服务器发生故障将**面临丢失数据**的**风险**。可以使用 **MultipleTableOutputFormat** 类**解决这个问题**，该类可以对是否写入 WAL 进行设置。

为了能使 Hadoop 集群上运行 HBase 的程序，还需要把相关的类文件引入 Hadoop 集群上。不然会出现下面错误：

```
java.lang.ClassNotFoundException:
org.apache.hadoop.hbase.mapreduce.TableOutputFormat
```

配置前，先用命令“stop-hbase.sh”停止 HBase 数据库，然后用命令“stop-hadoop.sh”停止 Hadoop 集群。

需要配置 **Hadoop 集群**中**每台**机器，在 **hadoop** 目录的 **conf** 子目录中，找 **hadoop-env.sh** 文件，并添加如下内容：（**备注**：我们的是“**/usr/hadoop/conf/hadoop-env.sh**”）

```
set hbase environment
export HBASE_HOME=/usr/hbase
export HADOOP_CLASSPATH=$HBASE_HOME/hbase-0.92.0.jar:
                        $HBASE_HOME/hbase-0.92.0-tests.jar:
                        $HBASE_HOME/conf:
                        $HBASE_HOME/zookeeper-3.2.0.jar
```

执行之后如下：

```
# set java environment
export JAVA_HOME=/usr/java/jdk1.6.0_31

# set hbase environment
export HBASE_HOME=/usr/hbase
export HADOOP_CLASSPATH=$HBASE_HOME/hbase-0.92.0.jar:$HBASE_HOME/hbase-0.92.0-tests.jar
export HADOOP_CLASSPATH=$HADOOP_CLASSPATH:$HBASE_HOME/conf:$HBASE_HOME/lib/zookeeper-3.4.2.jar
"hadoop-env.sh" 62L, 2537C 已写入
[hadoop@Master conf]$
```

备注：为了使显示效果更好一点，我把 HADOOP_CLASSPATH 写成了两行，但是写成两行时要注意写**第二个**时加上“**$HADOOP_CLASSPATH**”。Hadoop 集群中每台机器都弄完之后，然后用命令“start-all.sh”启动 Hadoop 集群，最后用命令“start-hbase.sh”启动 HBase 数据库。

下面将给出两个 HBase 与 MapReduce 相结合的示例。

样例示例（1）

这个例子是将 MapReduce 和 HBase 结合起来的 WordCount 程序。

在这个例子中，输入文件为：

- hbaseinput/file1.txt（它包含的内容：hello world bye world）
- hbaseinput/file2.txt（它包含的内容：hello hadoop bye hadoop）。

程序首先从文件中收集数据，在 shuffle 完成之后进行统计并计算，最后将计算结构存储到 HBase 中。

程序代码：

```
package com.hebut.hbase;

import java.io.IOException;
import java.util.Iterator;
import java.util.StringTokenizer;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.hbase.HColumnDescriptor;
import org.apache.hadoop.hbase.HTableDescriptor;
import org.apache.hadoop.hbase.client.HBaseAdmin;
import org.apache.hadoop.hbase.client.Put;
import org.apache.hadoop.hbase.mapreduce.TableOutputFormat;
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.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;

public class WordCountHBase {

    // 实现 Map 类
    public static class Map extends
            Mapper<LongWritable, Text, Text, IntWritable> {
        private final static IntWritable one = new IntWritable(1);
        private Text word = new Text();

        public void map(LongWritable key, Text value, Context context)
                throws IOException, InterruptedException {
            StringTokenizer itr = new StringTokenizer(value.toString());
            while (itr.hasMoreTokens()) {
                word.set(itr.nextToken());
                context.write(word, one);
```

```
        }
    }
}

// 实现 Reduce 类
public static class Reduce extends
        TableReducer<Text, IntWritable, NullWritable> {

    public void reduce(Text key, Iterable<IntWritable> values,
            Context context) throws IOException, InterruptedException {

        int sum = 0;

        Iterator<IntWritable> iterator = values.iterator();
        while (iterator.hasNext()) {
            sum += iterator.next().get();
        }

        // Put 实例化，每个词存一行
        Put put = new Put(Bytes.toBytes(key.toString()));
        // 列族为 content，列修饰符为 count，列值为数目
        put.add(Bytes.toBytes("content"), Bytes.toBytes("count"),
                Bytes.toBytes(String.valueOf(sum)));

        context.write(NullWritable.get(), put);
    }
}

// 创建 HBase 数据表
public static void createHBaseTable(String tableName)
throws IOException {
    // 创建表描述
    HTableDescriptor htd = new HTableDescriptor(tableName);
    // 创建列族描述
    HColumnDescriptor col = new HColumnDescriptor("content");
    htd.addFamily(col);

    // 配置 HBase
    Configuration conf = HBaseConfiguration.create();

    conf.set("hbase.zookeeper.quorum",
            "192.168.1.3,192.168.1.4,192.168.1.5");
    conf.set("hbase.zookeeper.property.clientPort", "2181");
```

```
        HBaseAdmin hAdmin = new HBaseAdmin(conf);

        if (hAdmin.tableExists(tableName)) {
            System.out.println("该数据表已经存在，正在重新创建。");
            hAdmin.disableTable(tableName);
            hAdmin.deleteTable(tableName);
        }

        System.out.println("创建表：" + tableName);
        hAdmin.createTable(htd);
    }

    public static void main(String[] args) throws Exception {
        String tableName = "wordcount";
        // 第一步：创建数据库表
        WordCountHBase.createHBaseTable(tableName);

        // 第二步：进行 MapReduce 处理
        // 配置 MapReduce
        Configuration conf = new Configuration();
        // 这几句话很关键
        conf.set("mapred.job.tracker", "192.168.1.2:9001");
        conf.set("hbase.zookeeper.quorum",
                "192.168.1.3,192.168.1.4,192.168.1.5");
        conf.set("hbase.zookeeper.property.clientPort", "2181");
        conf.set(TableOutputFormat.OUTPUT_TABLE, tableName);

        Job job = new Job(conf, "New Word Count");
        job.setJarByClass(WordCountHBase.class);

        // 设置 Map 和 Reduce 处理类
        job.setMapperClass(Map.class);
        job.setReducerClass(Reduce.class);

        // 设置输出类型
        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(IntWritable.class);

        // 设置输入和输出格式
        job.setInputFormatClass(TextInputFormat.class);
        job.setOutputFormatClass(TableOutputFormat.class);

        // 设置输入目录
        FileInputFormat.addInputPath(job, new Path("hbaseinput"));
```

```
        System.exit(job.waitForCompletion(true) ? 0 : 1);

    }
}
```

运行结果：

```
hbase(main):002:0> list
TABLE
scores
wordcount
2 row(s) in 0.0280 seconds

hbase(main):003:0> scan 'wordcount'
ROW                                   COLUMN+CELL
 bye                                  column=content:count, timestamp=1332263330988, value=2
 hadoop                               column=content:count, timestamp=1332263330988, value=2
 hello                                column=content:count, timestamp=1332263330988, value=2
 world                                column=content:count, timestamp=1332263330988, value=2
4 row(s) in 0.1680 seconds
```

备注：运行 MapReduce 和 HBase 相结合的程序时，选择“Run As →Run on Hadoop”。

样例示例（2）

HBase 索引主要用于提高 HBase 中表数据的访问速度，有效地避免了全表扫描（多数查询可以仅扫描少量索引页及数据页，而不是遍历所有的数据页）。我们知道 HBase 中的表根据行健被分成了多个 Regions，通常一个 Region 的一行都会包含的较多的数据，如果以列值作为查询条件，就只能从第一行数据开始往下查找，直到找到相关数据为止，这显然很低效。相反，如果经常被查询的列作为行健、行健作为列重新构造一张表，即可实现根据列值快速地定位相关数据所在的行，这就是索引。显然索引表仅需要包含一列，所以索引表的大小和原表比起来要小得多，如图 2-2-1 可以看出，由于索引表的单条记录所占的空间比原表要小，所以索引表的一个 Region 与原表相比，能够包含更多记录。

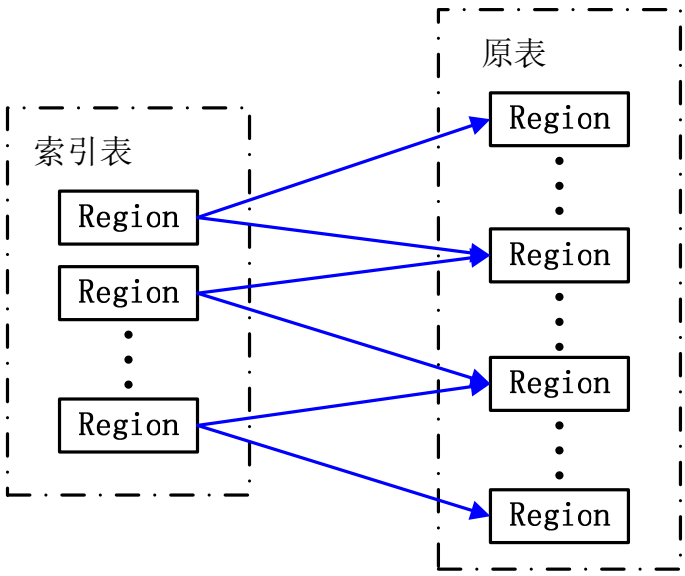

图 2.2-1 索引表与原表关系

假设 HBase 中存在一张表 heroes，里面内容如**表 2.2-1** 所示。则根据列“**info:name**”构造的索引表如**图 2.2-2** 所示，HBase 会自动将生成的索引表加入如**图 2.2-3** 所示的结构中，从而提高搜索的效率。

表 2.2-1 heroes 表的逻辑视图

行 健	列族：info		
	name	**email**	**power**
1	peter	peter@heroes.com	absorb abilities
2	hiro	hiro@heroes.com	bend time and space
3	sylar	sylar@heroes.com	hnow how things work
4	claire	claire@heroes.com	heal
5	noah	noah@heroes.com	cath the people with ablities

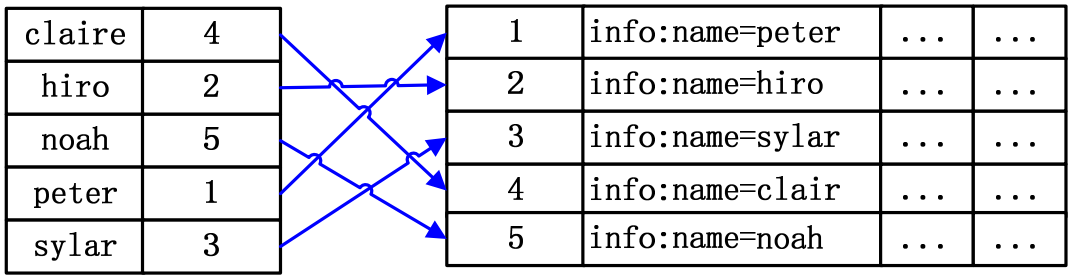

图 2.2-2 heroes 索引表示意

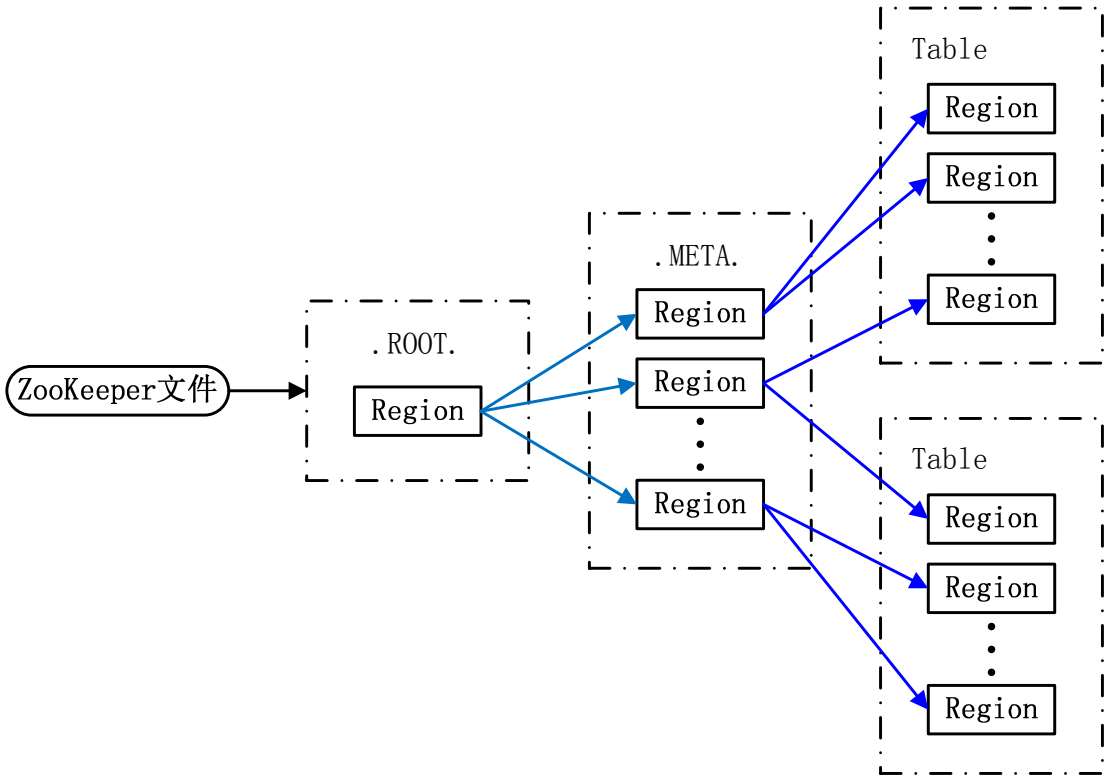

图 2.2-3 Region 定位示意图

HBase 中的表通常是非常大的，并且还可以不断增大，所以为表建立索引的工作量也是

相当大的。为了解决类似的问题，HBase 集成了 MapReduce 框架，用于对表中的大量数据进行并行处理。下面是利用 MapReduce 和 HBase 相结合完成构建索引表。

程序代码：

```
package com.hebut.hbase;

import java.io.ByteArrayOutputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.util.HashMap;

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.client.HBaseAdmin;
import org.apache.hadoop.hbase.client.HTable;
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.MultiTableOutputFormat;
import org.apache.hadoop.hbase.mapreduce.TableInputFormat;
import org.apache.hadoop.hbase.util.Base64;
import org.apache.hadoop.hbase.util.Bytes;
import org.apache.hadoop.io.Writable;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;

public class IndexBuilder {

    // 索引表唯一的一列为 INDEX_ROW，其中 INDEX 为列族
    private static final byte[] INDEX_COLUMN = Bytes.toBytes("INDEX");
    private static final byte[] INDEX_QUALIFIER = Bytes.toBytes("ROW");

    // 实现 Map 类
    public static class Map extends
        Mapper<ImmutableBytesWritable, Result, ImmutableBytesWritable, Writable> {

        // 存储了“列名”到“表名—列名”的映射
        // 前者用于获取某列的值，并作为索引表的键值；后者用户作为索引表的表名
        private HashMap<byte[], ImmutableBytesWritable> indexes;
```

```
        private byte[] family;

        // 实现 map 函数
        public void map(ImmutableBytesWritable key, Result value,
                Context context) throws IOException, InterruptedException {
            for (java.util.Map.Entry<byte[], ImmutableBytesWritable> index : indexes
                    .entrySet()) {
                // 获取列名
                byte[] qualifier = index.getKey();
                // 索引表的表名
                ImmutableBytesWritable tableName = index.getValue();
                // 根据“列族：列名”获得元素值
                byte[] newValue = value.getValue(family, qualifier);

                if (newValue != null) {
                    // 以列值作为行健，在列“INDEX：ROW”中插入行健
                    Put put = new Put(newValue);
                    put.add(INDEX_COLUMN, INDEX_QUALIFIER, key.get());

                    // 在 tableName 表上执行 put
                    // 操作使用 MultipleOutputFormat 时，
                    //第二个参数必须是 Put 和 Delete 类型
                    context.write(tableName, put);
                }
            }
        }

        // setup为Mapper中的方法，该方法只在任务初始化时执行一次
        protected void setup(Context context) throws IOException,
                InterruptedException {
            Configuration conf = context.getConfiguration();

            // 通过 Configuration.set()方法传递参数
            String tableName = conf.get("index.tablename");
            String[] fields = conf.getStrings("index.fields");

            // fields 内为需要做索引的列名
            String familyName = conf.get("index.familyname");
            family = Bytes.toBytes(familyName);

            // 初始化 indexes 方法
            indexes = new HashMap<byte[], ImmutableBytesWritable>();

            for (String field : fields) {
```

```
            // 如果给 name 做索引，则索引表的名称为“heroes-name”
            indexes.put(Bytes.toBytes(field),
            new ImmutableBytesWritable(
                    Bytes.toBytes(tableName + "-" + field)));
        }
    }
}

// 初始化示例数据表—“heroes”
public static void initHBaseTable(Configuration conf, String tableName)
        throws IOException {
    // 创建表描述
    HTableDescriptor htd = new HTableDescriptor(tableName);
    // 创建列族描述
    HColumnDescriptor col = new HColumnDescriptor("info");

    htd.addFamily(col);

    HBaseAdmin hAdmin = new HBaseAdmin(conf);

    if (hAdmin.tableExists(tableName)) {
        System.out.println("该数据表已经存在，正在重新创建。");
        hAdmin.disableTable(tableName);
        hAdmin.deleteTable(tableName);
    }

    System.out.println("创建表：" + tableName);
    // 创建表
    hAdmin.createTable(htd);
    HTable table = new HTable(conf, tableName);
    System.out.println("向表中插入数据");
    // 添加数据
    addRow(table, "1", "info", "name", "peter");
    addRow(table, "1", "info", "email", "peter@heroes.com");
    addRow(table, "1", "info", "power", "absorb abilities");

    addRow(table, "2", "info", "name", "hiro");
    addRow(table, "2", "info", "email", "hiro@heroes.com");
    addRow(table, "2", "info", "power", "bend time and space");

    addRow(table, "3", "info", "name", "sylar");
    addRow(table, "3", "info", "email", "sylar@heroes.com");
    addRow(table, "3", "info", "power", "hnow how things work");
```

```
        addRow(table, "4", "info", "name", "claire");
        addRow(table, "4", "info", "email", "claire@heroes.com");
        addRow(table, "4", "info", "power", "heal");

        addRow(table, "5", "info", "name", "noah");
        addRow(table, "5", "info", "email", "noah@heroes.com");
        addRow(table, "5", "info", "power", "cath the people with ablities");
    }

    // 添加一条数据
    private static void addRow(HTable table, String row,
    String columnFamily,String column, String value) throws IOException {
        Put put = new Put(Bytes.toBytes(row));
        // 参数出分别：列族、列、值
        put.add(Bytes.toBytes(columnFamily), Bytes.toBytes(column),
                Bytes.toBytes(value));
        table.put(put);
    }

    // 创建数据库表
    public static void createIndexTable(Configuration conf,
    String tableName) throws Exception {
        // 新建一个数据库管理员
        HBaseAdmin hAdmin = new HBaseAdmin(conf);

        if (hAdmin.tableExists(tableName)) {
            System.out.println("该数据表已经存在，正在重新创建。");
            hAdmin.disableTable(tableName);
            hAdmin.deleteTable(tableName);
        }

        // 新建一个表的描述
        HTableDescriptor tableDesc = new HTableDescriptor(tableName);
        // 在描述里添加列族
        tableDesc.addFamily(new HColumnDescriptor(INDEX_COLUMN));

        // 根据配置好的描述建表
        hAdmin.createTable(tableDesc);
        System.out.println("创建" + tableName + "表成功");
    }

    public static Job configureJob(Configuration conf, String jobName)
            throws IOException {
        Job job = new Job(conf, jobName);
```

```
        job.setJarByClass(IndexBuilder.class);

        // 设置 Map 处理类
        job.setMapperClass(Map.class);

        // 设置 Reduce 个数
        job.setNumReduceTasks(0);

        // 设置输入和输出格式
        job.setInputFormatClass(TableInputFormat.class);
        job.setOutputFormatClass(MultiTableOutputFormat.class);

        return job;
    }

    private static String convertScanToString(Scan scan)
    throws IOException {
        ByteArrayOutputStream out = new ByteArrayOutputStream();
        DataOutputStream dos = new DataOutputStream(out);
        scan.write(dos);
        return Base64.encodeBytes(out.toByteArray());
    }

    public static void main(String[] args) throws Exception {
        Configuration conf = HBaseConfiguration.create();
        conf.set("hbase.zookeeper.quorum",
                "192.168.1.3,192.168.1.4,192.168.1.5");
        conf.set("hbase.zookeeper.property.clientPort", "2181");

        String tableName = "heroes";
        String columnFamily = "info";
        String[] fields = { "name", "power" };
        // 第一步：初始化数据库表
        IndexBuilder.initHBaseTable(conf, tableName);

        // 第二步：创建索引表
        for (String field : fields) {
            IndexBuilder.createIndexTable(conf, tableName + "-" + field);
        }

        // 第三步：进行 MapReduce 处理
        conf.set("mapred.job.tracker", "192.168.1.2:9001");
        conf.set(TableInputFormat.SCAN, convertScanToString(new Scan()));
        conf.set(TableInputFormat.INPUT_TABLE, tableName);
```

```
        // 设置传递属性值
        conf.set("index.tablename", tableName);
        conf.set("index.familyname", columnFamily);
        conf.setStrings("index.fields", fields);

        Job job = IndexBuilder.configureJob(conf, "Index Builder");

        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

运行结果：

1）heroes 表数据

```
hbase(main):007:0> scan 'heroes'
ROW                                   COLUMN+CELL
 1                                    column=info:email, timestamp=1332327231947, value=peter@heroes.com
 1                                    column=info:name, timestamp=1332327231935, value=peter
 1                                    column=info:power, timestamp=1332327231952, value=absorb abilities
 2                                    column=info:email, timestamp=1332327231962, value=hiro@heroes.com
 2                                    column=info:name, timestamp=1332327231957, value=hiro
 2                                    column=info:power, timestamp=1332327231968, value=bend time and space
 3                                    column=info:email, timestamp=1332327231978, value=sylar@heroes.com
 3                                    column=info:name, timestamp=1332327231973, value=sylar
 3                                    column=info:power, timestamp=1332327231982, value=hnow how things work
 4                                    column=info:email, timestamp=1332327231992, value=claire@heroes.com
 4                                    column=info:name, timestamp=1332327231987, value=claire
 4                                    column=info:power, timestamp=1332327232000, value=heal
 5                                    column=info:email, timestamp=1332327232010, value=noah@heroes.com
 5                                    column=info:name, timestamp=1332327232004, value=noah
 5                                    column=info:power, timestamp=1332327232015, value=cath the people with ablities
5 row(s) in 1.1610 seconds
```

2）heroes-name 表数据

```
hbase(main):008:0> scan 'heroes-name'
ROW                                   COLUMN+CELL
 claire                               column=INDEX:ROW, timestamp=1332327243901, value=4
 hiro                                 column=INDEX:ROW, timestamp=1332327243901, value=2
 noah                                 column=INDEX:ROW, timestamp=1332327243901, value=5
 peter                                column=INDEX:ROW, timestamp=1332327243901, value=1
 sylar                                column=INDEX:ROW, timestamp=1332327243901, value=3
5 row(s) in 1.0970 seconds
```

3）heroes-power 表数据

```
hbase(main):009:0> scan 'heroes-power'
ROW                                   COLUMN+CELL
 absorb abilities                     column=INDEX:ROW, timestamp=1332328547497, value=1
 bend time and space                  column=INDEX:ROW, timestamp=1332328547497, value=2
 cath the people with ablities        column=INDEX:ROW, timestamp=1332328547497, value=5
 heal                                 column=INDEX:ROW, timestamp=1332328547497, value=4
 hnow how things work                 column=INDEX:ROW, timestamp=1332328547497, value=3
5 row(s) in 0.0280 seconds
```

参考文献

感谢以下文章的编写作者，没有你们的铺路，我或许会走得很艰难，参考不分先后，贡献同等珍贵。

【1】Hadoop 实战——陆嘉恒——机械工业出版社

【2】实战 Hadoop——刘鹏——电子工业出版社

【3】eclipse 写 MAPREDUCE 程序对 HBase 表进行操作之 IndexBuilder

地址：http://blog.csdn.net/liuxingjiaofu/article/details/7188375

【4】基于 Java 的 HBase 客户端编程

地址：http://tech.it168.com/a2011/0815/1232/000001232755_2.shtml

【5】HBase client API Guide

地址：http://www.spnguru.com/2010/07/hbase-client-api-guide/