HBase Shell 常用操作

HBase Shell是HBase的一个命令行工具，我们可以通过它对HBase进行维护操作。我们可以使用sudo -u hbase hbase shell来进入HBase shell。 在HBase shell中，可以使用status, version和whoami分别获得当前服务的状态、版本、登录用户和验证方式。

```
> status
3 servers, 1 dead, 1.3333 average load
> version
0.98.6-cdh5.3.1, rUnknown, Tue Jan 27 16:43:50 PST 2015
> whoami
hbase (auth:SIMPLE)
groups: hbase
```

HBase shell中的帮助命令非常强大，使用help获得全部命令的列表，使用help ‘command_name‘获得某一个命令的详细信息。 例如：

```
> help 'list'
List all tables in hbase. Optional regular expression parameter could
be used to filter the output. Examples:
  hbase> list
  hbase> list 'abc.*'
  hbase> list 'ns:abc.*'
  hbase> list 'ns:.*'
```

1. 命名空间

在HBase系统中，命名空间namespace指的是一个HBase表的逻辑分组，同一个命名空间中的表有类似的用途，也用于配额和权限等设置进行安全管控。 HBase默认定义了两个系统内置的预定义命名空间：

- hbase：系统命名空间，用于包含hbase的内部表
- default：所有未指定命名空间的表都自动进入该命名空间

我们可以通过create_namespace命令来建立命名空间

```
> create_namespace 'debugo_ns'
0 row(s) in 2.0910 seconds
```

通过drop_namespace来删除命名空间

```
> drop_namespace 'debugo_ns'
0 row(s) in 1.9540 seconds
```

通过alter_namespac改变表的属性，其格式如下：

```
alter_namespace 'my_ns', {METHOD => 'set', 'PROPERTY_NAME' => 'PROPERTY_VALUE'}
```

显示命名空间以及设定的元信息：

```
> describe_namespace 'debugo_ns'
DESCRIPTION
{NAME => 'debugo_ns'}
1 row(s) in 1.9540 seconds
```

显示所有命名空间

```
> list_namespace
NAMESPACE
debugo_ns
default
hbase
3 row(s) in 0.0910 seconds
```

在HBase下建表需要使用create table_name, column_family1, 这个命令:

```
> create 'user','info'
0 row(s) in 0.9030 seconds
=> Hbse::Table - user
```

这个时候这个表是创建在default下面。如果需要在debugo_ns这个命名空间下面建表，则需要使用create namespace:table_name这种方式:

```
> create_namespace 'debugo_ns'
0 row(s) in 2.0910 seconds
create 'debugo_ns:users', 'info'
0 row(s) in 0.4640 seconds
=> Hbase::Table - debugo_ns:users
```

List命令可以列出当前HBase实例中的所有表，支持使用正则表达式来匹配。

```
> list_namespace_tables 'debugo_ns'
TABLE
users
1 row(s) in 0.0400 seconds
```

使用list_namespace_tables也可以直接输出某个命名空间下的所有表

```
> list_namespace_tables 'debugo_ns'
TABLE
users
1 row(s) in 0.0400 seconds
```

2. DDL语句

首先是建立HBase表，上面我们已经用过create命令了。它后面的第一个参数是表名，然后是一系列列簇的列表。每个列簇中可以独立指定它使用的版本数，数据有效保存时间（TTL），是否开启块缓存等信息。

```
> create 't1', {NAME => 'f1', VERSIONS => 1, TTL => 2592000, BLOCKCACHE => true}, 'f2'
```

表也可以在创建时指定它预分割(pre-splitting)的region数和split方法。在表初始建立时，HBase只分配给这个表一个region。这就意味着当我们访问这个表数据时，我们只会访问一个region server，这样就不能充分利用集群资源。HBase提供了一个工具来管理表的region数，即org.apache.hadoop.hbase.util.RegionSplitter和HBase shell中create中的split的配置项。例如：

```
> create 't2', 'f1', {NUMREGIONS => 3, SPLITALGO => 'HexStringSplit'}
```

我们通过describe 来查看这个表中的元信息：

```
> describe 't2'
DESCRIPTION                                                  ENABLED
 't2', {NAME => 'f1', DATA_BLOCK_ENCODING => 'NONE', BLOOMFILTER => 'ROW', REPLIC true ATION_SCOPE => '0',
VERSIONS => '1', COMPRESSION => 'NONE', MIN_VERSIONS => '0', TTL => 'FOREVER', KEEP_DELETED_CELLS => 'false',
BLOCKSIZE => '65536', IN_MEMOR Y => 'false', BLOCKCACHE => 'true'}
1 row(s) in 0.0690 seconds
```

通过enable和disable来启用/禁用这个表,相应的可以通过is_enabled和is_disabled来检查表是否被禁用。

```
> disable 't2'
0 row(s) in 1.4410 seconds
> enable 't2'
0 row(s) in 0.5940 seconds
> is_enabled 't2'
true
0 row(s) in 0.0400 seconds
hbase(main):042:0> is_disabled 't2'
false
0 row(s) in 0.0490 seconds
```

使用exists来检查表是否存在

```
> exists 't2'
Table t2 does exist
0 row(s) in 0.0590 seconds
```

使用alter来改变表的属性，比如改变列簇的属性, 这涉及将信息更新到所有的region。在过去的版本中，alter操作需要先把table禁用，而在当前版本已经不需要。

```
> alter 't1', {NAME => 'f1', VERSIONS => 5}
Updating all regions with the new schema...
0/1 regions updated.
1/1 regions updated.
Done.
0 row(s) in 2.3470 seconds
```

另外一个非常常用的操作是添加和删除列簇：

```
> alter 't1','f3'
Updating all regions with the new schema...
0/1 regions updated.
1/1 regions updated.
Done.
0 row(s) in 2.3130 seconds
> alter 't1', 'delete' => 'f3'
```

或者：

```
> alter 't1',{ NAME => 'f3', METHOD => 'delete'}
Updating all regions with the new schema...
0/1 regions updated.
1/1 regions updated.
Done.
0 row(s) in 2.2930 seconds
```

删除表需要先将表disable。

```
> disable 't1'
0 row(s) in 1.4310 seconds
> drop 't1'
0 row(s) in 0.2440 seconds
```

3. put与get

在HBase shell中，我们可以通过put命令来插入数据。例如我们新创建一个表，它拥有id、address和info三个列簇，并插入一些数据。列簇下的列不需要提前创建，在需要时通过 : 来指定即可。

```
> create 'member','id','address','info'
0 row(s) in 0.4570 seconds
=> Hbase::Table – member
put 'member', 'debugo','id','11'
put 'member', 'debugo','info:age','27'
put 'member', 'debugo','info:birthday','1987-04-04'
put 'member', 'debugo','info:industry', 'it'
put 'member', 'debugo','address:city','beijing'
put 'member', 'debugo','address:country','china'
put 'member', 'Sariel', 'id', '21'
put 'member', 'Sariel','info:age', '26'
put 'member', 'Sariel','info:birthday', '1988-05-09 '
put 'member', 'Sariel','info:industry', 'it'
put 'member', 'Sariel','address:city', 'beijing'
put 'member', 'Sariel','address:country', 'china'
put 'member', 'Elvis', 'id', '22'
put 'member', 'Elvis','info:age', '26'
put 'member', 'Elvis','info:birthday', '1988-09-14 '
put 'member', 'Elvis','info:industry', 'it'
put 'member', 'Elvis','address:city', 'beijing'
put 'member', 'Elvis','address:country', 'china'
```

获取一个id的所有数据

```
> get 'member', 'Sariel'
COLUMN                          CELL

 address:city                   timestamp=1425871035382, value=beijing

 address:country                timestamp=1425871035424, value=china

 id:                            timestamp=1425871035176, value=21

 info:age                       timestamp=1425871035225, value=26

 info:birthday                  timestamp=1425871035296, value=1988-05-09

 info:industry                  timestamp=1425871035334, value=it

6 row(s) in 0.0530 seconds
```

获得一个id，一个列簇（一个列）中的所有数据:

```
> get 'member', 'Sariel', 'info'
COLUMN                          CELL

 info:age                       timestamp=1425871035225, value=26

 info:birthday                  timestamp=1425871035296, value=1988-05-09

 info:industry                  timestamp=1425871035334, value=it

3 row(s) in 0.0320 seconds
> get 'member', 'Sariel', 'info:age'
COLUMN                          CELL

 info:age                       timestamp=1425871035225, value=26

1 row(s) in 0.0270 seconds
```

通过describe 'member'可以看到，默认情况下列簇只保存1个version。我们先将其修改到2,然后update一些信息。

```
> alter 'member', {NAME=> 'info', VERSIONS => 2}
Updating all regions with the new schema...
0/1 regions updated.
1/1 regions updated.
Done.
0 row(s) in 2.2580 seconds
> put 'member', 'debugo','info:age','29'
> put 'member', 'debugo','info:age','28'
> get 'member', 'debugo', {COLUMN=>'info:age', VERSIONS=>2}
COLUMN                          CELL

 info:age                       timestamp=1425884510241, value=28

 info:age                       timestamp=1425884510195, value=29

2 row(s) in 0.0400 seconds
```

4. 其他DML语句

通过delete命令，我们可以删除id为某个值的'info:age'字段，接下来的get就无视了

```
> delete 'member','debugo','info:age'
0 row(s) in 0.0420 seconds
> get 'member','debugo','info:age'
COLUMN                          CELL
0 row(s) in 0.3270 seconds
```

通过deleteall来删除整行

```
> delete 'member','debugo','info:age'
0 row(s) in 0.0420 seconds
> get 'member','debugo','info:age'
COLUMN                          CELL
0 row(s) in 0.3270 seconds
```

给'Sariel'的'info:age'字段添加，并使用incr实现递增。但需要注意的是，这个value需要是一个数值，如果使用单引号标识的字符串就无法使用incr。在使用Java API开发时，我们可以使用toBytes函数讲数值转换成byte字节。在HBase shell中我们只能通过incr来初始化这个列，

```
> delete 'member','Sariel','info:age'
0 row(s) in 0.0270 seconds
> incr 'member','Sariel','info:age',26
0 row(s) in 0.0290 seconds
> incr 'member','Sariel','info:age'
0 row(s) in 0.0290 seconds
> incr 'member','Sariel','info:age', -1
0 row(s) in 0.0230 seconds
> get 'member','Sariel','info:age'
COLUMN   CELL
 info:age   timestamp=1425890213341, value=\x00\x00\x00\x00\x00\x00\x00\x1A
1 row(s) in 0.0280 seconds
```

十六进制1A是26，通过上面增1再减1后得到的结果。下面通过count统计行数。

```
> count 'member'
2 row(s) in 0.0750 seconds
=> 2
```

通过truncate来截断表。hbase是先将掉disable掉，然后drop掉后重建表来实现truncate的功能的。

```
hbase(main):010:0> truncate 'member'
Truncating 'member' table (it may take a while):
 - Disabling table...
 - Dropping table...
 - Creating table...
0 row(s) in 2.3260 seconds
```

5. scan和filter

通过scan来对全表进行扫描。我们将之前put的数据恢复。

```
> scan 'member'
ROW               COLUMN+CELL
 Elvis            column=address:city, timestamp=1425891057211, value=
                  beijing
 Elvis            column=address:country, timestamp=1425891057258, val
                  ue=china
```

```
 Elvis              column=id:, timestamp=1425891057038, value=22
 Elvis              column=info:age, timestamp=1425891057083, value=26
 Elvis              column=info:birthday, timestamp=1425891057129, value
                    =1988-09-14
 Elvis              column=info:industry, timestamp=1425891057172, value
                    =it
 Sariel             column=address:city, timestamp=1425891056965, value=
                    beijing
 Sariel             column=address:country, timestamp=1425891057003, val
                    ue=china
 Sariel             column=id:, timestamp=1425891056767, value=21
 Sariel             column=info:age, timestamp=1425891056808, value=26
 Sariel             column=info:birthday, timestamp=1425891056883, value
                    =1988-05-09
 Sariel             column=info:industry, timestamp=1425891056924, value
                    =it
 debugo             column=address:city, timestamp=1425891056642, value=
                    beijing
 debugo             column=address:country, timestamp=1425891056726, val
                    ue=china
 debugo             column=id:, timestamp=1425891056419, value=11
 debugo             column=info:age, timestamp=1425891056499, value=27
 debugo             column=info:birthday, timestamp=1425891056547, value
                    =1987-04-04
 debugo             column=info:industry, timestamp=1425891056597, value
                    =it
3 row(s) in 0.0660 seconds3 row(s) in 0.0590 seconds
```

指定扫描其中的某个列：

```
> scan 'member', {COLUMNS=> 'info:birthday'}
```

或者整个列簇：

```
> scan 'member', {COLUMNS=> 'info'}
ROW                            COLUMN+CELL
 Elvis                         column=info:age, timestamp=1425891057083, value=26
 Elvis                         column=info:birthday, timestamp=1425891057129, value=1988-09-14
 Elvis                         column=info:industry, timestamp=1425891057172, value=it
 Sariel                        column=info:age, timestamp=1425891056808, value=26
 Sariel                        column=info:birthday, timestamp=1425891056883, value=1988-05-09
 Sariel                        column=info:industry, timestamp=1425891056924, value=it
 debugo                        column=info:age, timestamp=1425891056499, value=27
 debugo                        column=info:birthday, timestamp=1425891056547, value=1987-04-04
 debugo                        column=info:industry, timestamp=1425891056597, value=it
3 row(s) in 0.0650 seconds
```

除了列（COLUMNS）修饰词外，HBase还支持Limit（限制查询结果行数），STARTROW （ROWKEY起始行。会先根据这个key定位到region，再向后扫描）、STOPROW(结束行)、TIMERANGE（限定时间戳范围）、VERSIONS（版本数）、和FILTER（按条件过滤行）等。比如我们从Sariel这个rowkey开始，找下一个行的最新版本：

```
> scan 'member', { STARTROW => 'Sariel', LIMIT=>1, VERSIONS=>1}
ROW    COLUMN+CELL
 Sariel   column=address:city, timestamp=1425891056965, value=beijing
 Sariel   column=address:country, timestamp=1425891057003, value=china
 Sariel   column=id:, timestamp=1425891056767, value=21
 Sariel   column=info:age, timestamp=1425891056808, value=26
 Sariel   column=info:birthday, timestamp=1425891056883, value=1988-05-09
 Sariel   column=info:industry, timestamp=1425891056924, value=it
1 row(s) in 0.0410 seconds
```

Filter是一个非常强大的修饰词，可以设定一系列条件来进行过滤。比如我们要限制某个列的值等于26：

```
> scan 'member', FILTER=>"ValueFilter(=,'binary:26')"
ROW    COLUMN+CELL
 Elvis    column=info:age, timestamp=1425891057083, value=26
 Sariel   column=info:age, timestamp=1425891056808, value=26
2 row(s) in 0.0620 seconds
```

值包含6这个值：

```
> scan 'member', FILTER=>"ValueFilter(=,'substring:6')"
Elvis    column=info:age, timestamp=1425891057083, value=26
 Sariel   column=info:age, timestamp=1425891056808, value=26
2 row(s) in 0.0620 seconds
```

列名中的前缀为birthday的：

```
> scan 'member', FILTER=>"ColumnPrefixFilter('birth') "
ROW                           COLUMN+CELL
 Elvis                        column=info:birthday, timestamp=1425891057129, value=1988-09-14
 Sariel      column=info:birthday, timestamp=1425891056883, value=1988-05-09
 debugo                       column=info:birthday, timestamp=1425891056547, value=1987-04-04
3 row(s) in 0.0450 seconds
```

FILTER中支持多个过滤条件通过括号、AND和OR的条件组合。

```
> scan 'member', FILTER=>"ColumnPrefixFilter('birth') AND ValueFilter ValueFilter(=,'substring:1987')"
ROW       COLUMN+CELL
Debugo    column=info:birthday, timestamp=1425891056547, value=1987-04-04
1 row(s) in 0.0450 seconds
```

同一个rowkey的同一个column有多个version，根据timestamp来区分。而每一个列簇有多个column。而FIRSTKEYONLY仅取出每个列簇的第一个column的第一个版本。而KEYONLY则是对于每一个column只去取出key，把VALUE的信息丢弃,一般和其他filter结合使用。例如：

```
> scan 'member', FILTER=>"FirstKeyOnlyFilter()"
ROW    COLUMN+CELL
 Elvis    column=address:city, timestamp=1425891057211, value=beijing
 Sariel   column=address:city, timestamp=1425891056965, value=beijing
 debugo column=address:city, timestamp=1425891056642, value=beijing
3 row(s) in 0.0230 seconds
> scan 'member', FILTER=>"KeyOnlyFilter()"
hbase(main):055:0> scan 'member', FILTER=>"KeyOnlyFilter()"
ROW    COLUMN+CELL
Elvis    column=address:city, timestamp=1425891057211, value=
Elvis    column=id:, timestamp=1425891057038, value=
……
```

PrefixFilter是对Rowkey的前缀进行判断,这是一个非常常用的功能。

```
> scan 'member', FILTER=>"PrefixFilter('E')"
ROW    COLUMN+CELL
Elvis    column=address:city, timestamp=1425891057211, value=beijing
……
1 row(s) in 0.0460 seconds
```