status

```
hbase(main):006:0> status
1 servers, 0 dead, 5.0000 average load
hbase(main):002:0> help 'status'
hbase> status
hbase> status 'simple'
hbase> status 'summary'
hbase> status 'detailed'
hbase> status 'replication'
hbase> status 'replication', 'source'
hbase> status 'replication', 'sink'
```

alter / alter_async

```
可以修改，增加，删除表的列族信息、属性、配置等。
#对于表t1，如果t1含有f1列族，则将f1列族的版本数设为5.
# 如果t1不含f1列数，则添加f1列族到表t1上。并将f1的版本数设置为5.
hbase> alter 't1', NAME => 'f1', VERSIONS => 5
#添加或修改多个列族
hbase> alter 't1', 'f1', {NAME => 'f2', IN_MEMORY => true}, {NAME => 'f3', VERSIONS => 5}
#删除 命名空间ns1 中的 表t1 的 列族f1 的两种方法
hbase> alter 'ns1:t1', NAME => 'f1', METHOD => 'delete'
hbase> alter 'ns1:t1', 'delete' => 'f1'
#修改表t1的MAX_FILESIZE属性的值。
hbase> alter 't1', MAX_FILESIZE => '134217728'
# 修改表t1或者列族f2的配置
hbase> alter 't1', CONFIGURATION => {'hbase.hregion.scan.loadColumnFamiliesOnDemand' => 'true'}
hbase> alter 't1', {NAME => 'f2', CONFIGURATION => {'hbase.hstore.blockingStoreFiles' => '10'}}
#删除属性
hbase> alter 't1', METHOD => 'table_att_unset', NAME => 'MAX_FILESIZE'
hbase> alter 't1', METHOD => 'table_att_unset', NAME => 'coprocessor$1'
#一次性修改多个属性值
hbase> alter 't1', { NAME => 'f1', VERSIONS => 3 },
{ MAX_FILESIZE => '134217728' }, { METHOD => 'delete', NAME => 'f2' },
OWNER => 'johndoe', METADATA => { 'mykey' => 'myvalue' }
hbase(main):014:0>
```

append

```
# 向表t1的rowkey为r1的列c1的值后面添加字符串value
hbase> append 't1', 'r1', 'c1', 'value'
#表t1的引用对象t11使用append。
hbase> t11.append 'r1', 'c1', 'value'
```

count

```
#统计表t1的行数
count 't1'
#统计表t1的行数，其中参数的含义如下
# INTERVAL设置多少行显示一次及对应的rowkey，默认1000；CACHE每次去取的缓存区大小，默认是10，调整该参数可提高查询速度
# 例如，查询表t1中的行数，每10条显示一次，缓存区为1000
count 't1', INTERVAL => 10, CACHE => 1000
#对应的表应用对象的用法
hbase> t.count
hbase> t.count INTERVAL => 100000
hbase> t.count CACHE => 1000
hbase> t.count INTERVAL => 10, CACHE => 1000
```

delete / deleteall

```
# 删除命名空间ns1下的表t1的rowkey的r1的列c1，时间戳为ts1
hbase> delete 'ns1:t1', 'r1', 'c1', ts1
# 删除默认命名空间下的表t1的rowkey的r1的列c1，时间戳为ts1
hbase> delete 't1', 'r1', 'c1', ts1
#应用对象的用法
hbase> t.delete 'r1', 'c1', ts1
#删除命名空间ns1下表t1的rowkey为r1的所有数据
hbase> deleteall 'ns1:t1', 'r1'
#删除默认命名空间下表t1的rowkey为r1的所有数据
hbase> deleteall 't1', 'r1'
#删除命名空间ns1下表t1的rowkey为r1的列c1的所有数据
hbase> deleteall 't1', 'r1', 'c1'
# 删除默认命名空间下的表t1的rowkey的r1的列c1，时间戳为ts1
hbase> deleteall 't1', 'r1', 'c1', ts1
#应用对象的用法
hbase> t.deleteall 'r1'
```

delete / deleteall (cont)

```
hbase> t.deleteall 'r1', 'c1'
hbase> t.deleteall 'r1', 'c1', ts1
```

put

```
# 向命名空间ns1下表t1的rowkey为r1的列c1添加数据
hbase> put 'ns1:t1', 'r1', 'c1', 'value'
# 向默认命名空间下表t1的rowkey为r1的列c1添加数据
hbase> put 't1', 'r1', 'c1', 'value'
# 向默认命名空间下表t1的rowkey为r1的列c1添加数据，并设置时间戳为ts1
hbase> put 't1', 'r1', 'c1', 'value', ts1
# 向默认命名空间下表t1的rowkey为r1的列c1添加数据，并设置时间戳为ts1，并设置属性
hbase> put 't1', 'r1', 'c1', 'value', ts1, {ATTRIBUTES=>{'mykey'=>'-myvalue'}}
#引用对象的用法
t.put 'r1', 'c1', 'value', ts1, {ATTRIBUTES=>{'mykey'=>'myvalue'}}
```

create

```
#在命名空间ns1下，创建表t1，其中有一个列族f1，f1的版本数为5
hbase> create 'ns1:t1', {NAME => 'f1', VERSIONS => 5}
#在默认命名空间下，创建表t1，有三个列族f1,f2,f3
hbase> create 't1', {NAME => 'f1'}, {NAME => 'f2'}, {NAME => 'f3'}
#等价于
hbase> create 't1', 'f1', 'f2', 'f3'
#创建表t1，列族f1，并设置f1的版本数为1，属性TTL为2592000，属-性BLOCKCACHE为true。属性的含义在这就不解释了。
hbase> create 't1', {NAME => 'f1', VERSIONS => 1, TTL => 2592000, BLOCKCACHE => true}
# 创建表t1,列族f1，并设置f1的配置hbase.hstore.blockingStoreFiles为 10
hbase> create 't1', {NAME => 'f1', CONFIGURATION => {'hbase.h-store.blockingStoreFiles' => '10'}}
#创建表时，配置信息可以放在最后，例如：
hbase> create 'ns1:t1', 'f1', SPLITS => ['10', '20', '30', '40']
hbase> create 't1', 'f1', SPLITS => ['10', '20', '30', '40']
hbase> create 't1', 'f1', SPLITS_FILE => 'splits.txt', OWNER => 'johndoe'
hbase> create 't1', {NAME => 'f1', VERSIONS => 5}, METADATA => { 'mykey' => 'myvalue' }
hbase> # Optionally pre-split the table into NUMREGIONS, using
hbase> # SPLITALGO ("HexStringSplit", "UniformSplit" or classname)
#指定Pre-splitting的region的块数，和分割函数。
```

create (cont)

```
hbase> create 't1', 'f1', {NUMREGIONS => 15, SPLITALGO => 'HexStringSplit'}
hbase> create 't1', 'f1', {NUMREGIONS => 15, SPLITALGO => 'HexStringSplit', REGION_REPLICATION => 2, CONFIGURATION => {'hbase.hregion.scan.loadColumnFamiliesOnDemand' => 'true'}}
#也可以用另一个表t2的引用去创建一个新表t1，t1表具有t2的所有列族，并且加上f1列族。
hbase> t1 = create 't2', 'f1'
```

scan

```
# 扫描命名空间hbase下的meta表，显示出meta表的所有数据
hbase> scan 'hbase:meta'
# 扫描命名空间hbase下的meta表的列族info的列regioninfo，显示出meta表的列族info下的regioninfo列的所有数据
hbase> scan 'hbase:meta', {COLUMNS => 'info:regioninfo'}
# 扫描命名空间ns1下表t1的列族'c1'和'c2'。显示出命名空间ns1下表t-1的列族'c1'和'c2'的所有数据
hbase> scan 'ns1:t1', {COLUMNS => ['c1', 'c2']}
# 扫描命名空间ns1下表t1的列族'c1'和'c2'。显示出命名空间ns1下表t-1的列族'c1'和'c2'，且只显示前10个rowkey的数据。
hbase> scan 'ns1:t1', {COLUMNS => ['c1', 'c2'], LIMIT => 10}
# 扫描命名空间ns1下表t1的列族'c1'和'c2'。显示出命名空间ns1下表t-1的列族'c1'和'c2'，且只显示从rowkey=“xyz”开始的前10个rowkey的数据。
hbase> scan 'ns1:t1', {COLUMNS => ['c1', 'c2'], LIMIT => 10, STARTROW => 'xyz'}
# 扫描默认命名空间下表t1的列族c1时间戳从'1303668804'到'1303-668904'的数据
hbase> scan 't1', {COLUMNS => 'c1', TIMERANGE => [1303668804, 1303668904]}
# 反向显示表t1的数据
hbase> scan 't1', {REVERSED => true}
# 过滤显示表t1的数据
hbase> scan 't1', {FILTER => "(PrefixFilter ('row2') AND (QualifierFilter (>=, 'binary:xyz'))) AND (TimestampsFilter ( 123, 456))"}
# RAW为true，显示出表t1的所有数据，包括已经删除的
hbase> scan 't1', {RAW => true, VERSIONS => 10}
# 表t1的引用的扫描
hbase> t11 = get_table 't1'
hbase> t11.scan
```

get

#得到命名空间ns1下表t1的rowkey为r1的数据
hbase> get 'ns1:t1', 'r1'
#得到默认命名空间下表t1的rowkey为r1的数据
hbase> get 't1', 'r1'
#得到默认命名空间下表t1的rowkey为r1，时间戳范围在ts1和ts2之间的数据
hbase> get 't1', 'r1', {TIMERANGE => [ts1, ts2]}
#得到默认命名空间下表t1的rowkey为r1的c1列的数据
hbase> get 't1', 'r1', {COLUMN => 'c1'}
#得到默认命名空间下表t1的rowkey为r1的c1,c2,c3列的数据
hbase> get 't1', 'r1', {COLUMN => ['c1', 'c2', 'c3']}
#得到默认命名空间下表t1的rowkey为r1的c1列，时间戳为ts1的数据
hbase> get 't1', 'r1', {COLUMN => 'c1', TIMESTAMP => ts1}
#得到默认命名空间下表t1的rowkey为r1的c1列，时间戳范围为ts1到ts2，版本数为4的数据
hbase> get 't1', 'r1', {COLUMN => 'c1', TIMERANGE => [ts1, ts2], VERSIONS => 4}
#应用对象的用法
hbase> t.get 'r1'
hbase> t.get 'r1', {TIMERANGE => [ts1, ts2]}
hbase> t.get 'r1', {COLUMN => 'c1'}
hbase> t.get 'r1', {COLUMN => ['c1', 'c2', 'c3']}
hbase> t.get 'r1', {COLUMN => 'c1', TIMESTAMP => ts1}
hbase> t.get 'r1', {COLUMN => 'c1', TIMERANGE => [ts1, ts2], VERSIONS => 4}
hbase> t.get 'r1', {COLUMN => 'c1', TIMESTAMP => ts1, VERSIONS => 4}

other

truncate show_filters list drop_all drop disable_all disable whoami version describe