# HBase 基础

刘磊

2020 年 10 月

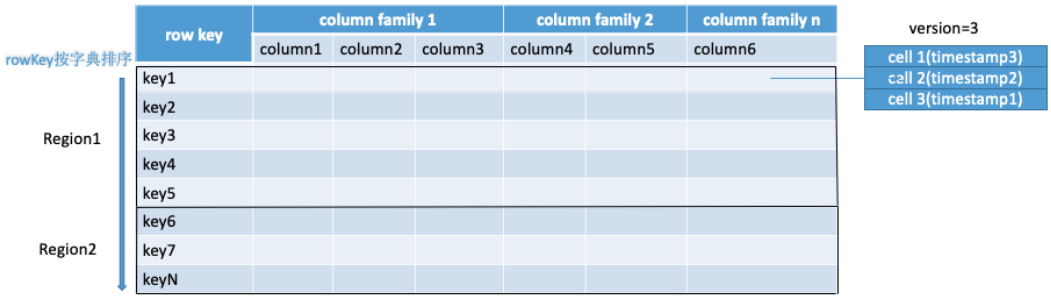
## 1. HBase 基本概念

HBase 是一个分布式的、面向列的、基于 Google Bigtable 的开源实现。名字来源于 Hadoop database，即 hadoop 数据库。不同于一般的关系数据库，它可以存储非结构化数据，而且它是基于列的而不是基于行的模式。

- 利用 Hadoop HDFS 作为其文件存储系统，
- 利用 Hadoop MapReduce 来处理 HBase 中的海量数据，
- 利用 Zookeeper 作为协同服务。

### 表结构

HBase 以表的形式存储数据。表有行和列组成。列划分为若干个列族/列簇(column family)，每个列族/列簇下面可以有多个普通列。



## 表（Table）

HBase 采用表来组织数据，表由许多行和列组成，列划分为多个列族。

## 行（Row)

在表里面，每一行代表着一个数据对象。每一行都是由一个行键（Row Key）和一个或者多个列组成的。行键是行的唯一标识，行键并没有什么特定的数据类型，以二进制的字节来存储，按字母顺序排序。

## 行键 （Row Key）

行键，每一行的主键列，每行的行键要唯一，行键的值为任意字符串(最大长度是 64KB，实际应用中长度一般为 10-100bytes)，在 HBase 内部，rowKey 保存为字节数组 byte[]。

## 列族（Column Family）

列族是每个子列的父级，每个子列都属于一个列族，一个列族包含一个或者多个相关列，创建表的时候需要指定列族，而不需要必须指定列。通过“列族名:列名”来表示某个具体的子列。HBase 中的 Schema 就是由 TableName 和 Column Family Name 构成。

## 列（Column）

列由列族（Column Family）和列限定符（Column Qualifier）联合标识，由“：”进行间隔，如 family:qualifier。

## 列限定符（Column Qualifier）

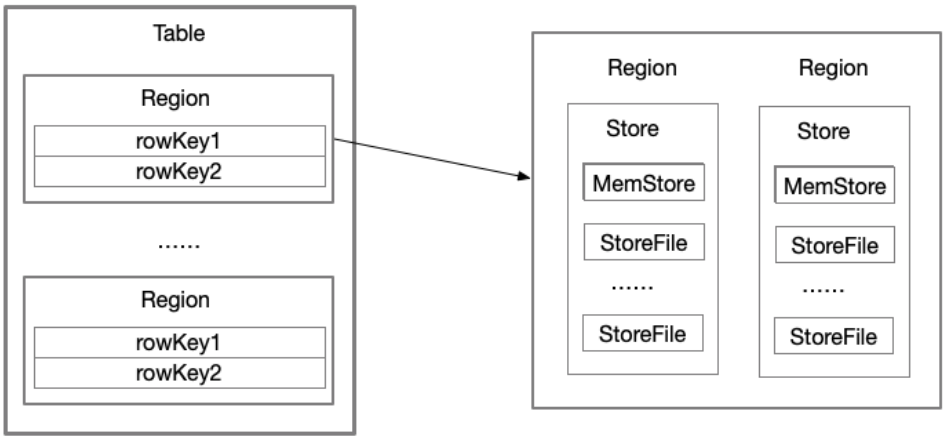
就是列族下的每个子列名称，或者称为相关列，或者称为限定符，只是翻译不同。通过 columnFamily:column 来定位某个子列。

## 存储单元 （Cell）

外观看到的每个单元格其实都可以对应着多个存储单元，默认情况下一个单元格对应着一个存储单元，一个存储单元可以存储一份数据，如果一个单元格有多个存储单元就表示一个单元格可以存储多个值。可以通过 version 来设置存储单元个数。可以通过 rowKey + columnFamily + column + timestamp 来唯一确定一个存储单元。cell 中的数据是没有类型的，全部是字节码形式存贮。

## 区域（Region）

Table 在行的方向上分割为多个 Region。Region 是按大小分割的，每个表开始只有一个 region，随着数据的增多，Region 不断增大，当增大到一个阈值的时候，Region 就会等分为两个新的 Region，之后会有越来越多的 Region。Region 是 HBase 中分布式存储和负载均衡的最小单元。不同的 Region 分布到不同的 RegionServer 上。Region 由一个或者多个 Store 组成, 每个 Store 保存一个 columnFamily, 每个 Store 又由一个 MemStore(存储在内存中)和 0 到多个 StoreFile(存储在 HDFS 上)组成。




## 时间戳（Time Stamp）

每个 cell 都保存着同一份数据的多个版本。版本通过时间戳来索引。时间戳的类型是 64 位整型。时间戳可以由 HBase(在数据写入时自动 )赋值，此时时间戳是精确到毫秒的当前系统时间。时间戳也可以由客户显式赋值。如果应用程序要避免数据版本冲突，就必须自己生成具有唯一性的时间戳。每个 cell 中，不同版本的数据按照时间倒序排序，即最新的数据排在最前面。

为了避免数据存在过多版本造成的管理 (包括存贮和索引)负担，HBase 提供了两种数据版本回收方式。一是保存数据的最后 n 个版本，二是保存最近一段时间内的版本（比如最近七天）。用户可以针对每个列族进行设置。

## 命名空间（Namespace）

命名空间指对一组表的逻辑分组，类似关系数据库中的 database，方便对表在业务上划分。HBase 系统默认定义了两个缺省的 namespace：

- hbase：系统内建表，包含 namespace 和 meta 表；
- default：用户建表时未指定 namespace 的表都创建在此。

## 2. HBase 伪分布式安装

1. 将 HBase 的安装包 hbase1.4.10.tar.gz 复制到/apps 目录下并解压缩。

```
cp ~/big_data_tools/ hbase-1.4.10-bin.tar.gz /apps
tar zxvf hbase-1.4.10-bin.tar.gz
```

重命名为 hbase，删除压缩包 hbase-1.4.10-bin.tar.gz。

```
mv /apps/hbase-1.4.10 /apps/hbase
rm /apps/hbase-1.4.10.tar.gz
```

2. 添加 HBase 的环境变量。打开用户环境变量配置文件~/.bashrc

```
vim ~/.bashrc
```

在文件末尾位置，追加 HBase 的 bin 目录路径相关配置，并保存退出。

```
# Hbase
export HBASE_HOME=/apps/hbase
export PATH=$HBASE_HOME/bin:$PATH
```

到目前位置我们安装了 Java，Hadoop 和 HBase，环境变量配置文件添加内容如下图所示

```
# Java
export JAVA_HOME=/apps/java
export PATH=$JAVA_HOME/bin:$PATH

# Hadoop
export HADOOP_HOME=/apps/hadoop
export PATH=$HADOOP_HOME/bin:$PATH

# Hbase
export HBASE_HOME=/apps/hbase
export PATH=$HBASE_HOME/bin:$PATH
```

执行 source 命令，使环境变量生效。

```
source ~/.bashrc
```

此时就可以调用 HBase 的 bin 目录下的脚本了。先来查看一下 HBase 的版本信息。

```
hbase version
```

```
lei@ubuntu:/apps$ hbase version
HBase 1.4.10
Source code repository git://apurtell-ltm4.internal.salesforce.com/Users/apurtel
l/tmp/hbase-build-2 revision=76ab087819fe82ccf6f531096e18ad1bed079651
Compiled by apurtell on Wed Jun  5 16:48:11 PDT 2019
From source with checksum a4dd28b173ece8660fc5633796deb361
```

3. 配置 HBase。切换到目录/apps/hbase/conf 下，打开文件 hbase-env.sh

```
cd /apps/hbase/conf
vim hbase-env.sh
```

在文件末尾追加下面的内容，并保存退出。

```
export JAVA_HOME=/apps/java
export HBASE_MANAGES_ZK=true
export HBASE_CLASSPATH=/apps/hbase/conf
```

说明：

- JAVA_HOME 为 Java 安装目录；
- HBASE_MANAGES_ZK 表示是否使用 HBase 自带的 Zookeeper 环境；
- HBASE_CLASSPATH 配置 HBase 配置文件的位置。

4. 打开文件 hbase-site.xml。在两个<configuration>之间添加如下内容，并保存退出。

```
 <property>
    <name>hbase.master</name>
    <value>localhost</value>
</property>
 <property>
     <name>hbase.rootdir</name>
     <value>hdfs://localhost:9000/hbase</value>
 </property>
 <property>
     <name>hbase.cluster.distributed</name>
     <value>true</value>
 </property>
 <property>
     <name>hbase.zookeeper.quorum</name>
     <value>localhost</value>
 </property>
 <property>
     <name>hbase.zookeeper.property.dataDir</name>
     <value>/data/tmp/zookeeper-hbase</value>
```

```
  </property>
```

配置项说明：

- hbase.master：HBase 主节点地址。
- hbase.rootdir：HBase 文件在 HDFS 上的存储位置。
- hbase.cluster.distributed：HBase 是否为分布式模式。
- hbase.zookeeper.quorum：配置 ZooKeeper 服务器地址。
- hbase.zookeeper.property.dataDir：HBase 在 ZooKeeper 上存储数据的位置。

注意：这里 hbase.zookeeper.property.dataDir 目录，需要提前创建。

```
mkdir -p /data/tmp/zookeeper-hbase
```

5. 启动 HBase。输入 jps，查看 Hadoop 进程是否已经启动。若未启动，先启动 Hadoop。

```
./apps/hadoop/sbin/start-all.sh
```

当 Hadoop 相关进程启动后，启动 HBase 服务。

```
/apps/hbase/bin/start-hbase.sh
```

输入 jps，查看 HBase 相关进程。

```
lei@ubuntu:/apps/hbase/bin$ jps
6802 ResourceManager
7138 NodeManager
7909 HRegionServer
7766 HMaster
8214 Jps
7705 HQuorumPeer
6556 SecondaryNameNode
6302 DataNode
6127 NameNode
```

可以看到 HBase 的三个进程 HMaster、HRegionServer、HQuorumPeer 都已启动，其中 HQuorumPeer 为 ZooKeeper 进程。

6. 测试 HBase。进入 HBase Shell 接口。

```
hbase shell
```

输入 list 的命令，查看当前有哪些 HTable 表。

```
list
```

创建一张表 tb，表中含有一个列簇 mycf。

```
create 'tb','mycf'
```

再次输入 list，列出 HBase 中的表。

```
hbase(main):001:0> list
TABLE
0 row(s) in 0.1940 seconds

=> []
hbase(main):002:0> create 'tb','mycf'
0 row(s) in 2.5340 seconds

=> Hbase::Table - tb
hbase(main):003:0> list
TABLE
tb
1 row(s) in 0.0110 seconds

=> ["tb"]
```

HBase 在 HDFS 上的存储位置是在 hbase-site.xml 设置的，可以使用 HDFS shell 命令进行查看。

```
lei@ubuntu:~$ hadoop fs -ls /
Found 6 items
drwxr-xr-x   - lei supergroup          0 2020-08-29 11:55 /hbase
drwxr-xr-x   - lei supergroup          0 2020-08-27 12:58 /input
drwxr-xr-x   - lei supergroup          0 2020-08-29 00:06 /output
-rw-r--r--   3 lei supergroup         28 2020-08-27 00:01 /testcreate
drwx------   - lei supergroup          0 2020-08-25 23:58 /tmp
drwxr-xr-x   - lei supergroup          0 2020-08-25 23:58 /user
```

## 3. HBase Shell

HBase Shell 是官方提供的一组命令，用于操作 HBase。如果配置了 HBase 的环境变量，就可以在终端中输入 hbase shell 命令进入命令行。

```
hbase shell
```

```
lei@ubuntu:~$ hbase shell
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/apps/hbase/lib/slf4j-log4j12-1.7.10.jar!/org/
slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/apps/hadoop/share/hadoop/common/lib/slf4j-log
4j12-1.7.25.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
HBase Shell
Use "help" to get list of supported commands.
Use "exit" to quit this interactive shell.
Version 1.4.10, r76ab087819fe82ccf6f531096e18ad1bed079651, Wed Jun  5 16:48:11 P
DT 2019

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

# help 命令

通过 help 命令可以查看 hbase shell 支持的所有命令，HBase 将命令分成了不同的组，

如 general, ddl, dml 等。

```
hbase(main):007:0> help
HBase Shell, version 1.4.10, r76ab087819fe82ccf6f531096e18ad1bed079651, Wed Jun
 5 16:48:11 PDT 2019
Type 'help "COMMAND"', (e.g. 'help "get"' -- the quotes are necessary) for help
on a specific command.
Commands are grouped. Type 'help "COMMAND_GROUP"', (e.g. 'help "general"') for h
elp on a command group.

COMMAND GROUPS:
  Group name: general
  Commands: processlist, status, table_help, version, whoami

  Group name: ddl
  Commands: alter, alter_async, alter_status, create, describe, disable, disable
_all, drop, drop_all, enable, enable_all, exists, get_table, is_disabled, is_ena
bled, list, list_regions, locate_region, show_filters
```

可以通过 help '命令名称'来查看命令的作用和用法。例如

```
help 'list'
```

```
hbase(main):011:0> help 'list'
List all user 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:.*'
```

## General 命令

1. 查询服务器状态

```
hbase(main):016:0> status
1 active master, 0 backup masters, 1 servers, 0 dead, 3.0000 average load
```

2. 查询版本号

```
hbase(main):017:0> version
1.4.10, r76ab087819fe82ccf6f531096e18ad1bed079651, Wed Jun  5 16:48:11 PDT 2019
```

## DDL 操作

数据定义语言（Data Defination Language，DDL）操作主要用来定义、修改和查询表。

1. 创建一个表

创建一个具有两个列族（address, info）的表 students。

```
create 'students','address','info'
```

```
hbase(main):001:0> create'students','address','info'
0 row(s) in 1.5720 seconds

=> Hbase::Table - students
```

2. 列出所有表

```
hbase(main):002:0> list
TABLE
students
tb
2 row(s) in 0.0160 seconds

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

3. 获取表的描述，自己弄清楚下图输出信息各项含义。

```
describe 'students'
```

```
hbase(main):004:0* describe 'students'
Table students is ENABLED
students
COLUMN FAMILIES DESCRIPTION
{NAME => 'address', BLOOMFILTER => 'ROW', VERSIONS => '1', IN_MEMORY => 'false',
 KEEP_DELETED_CELLS => 'FALSE', DATA_BLOCK_ENCODING => 'NONE', TTL => 'FOREVER',
 COMPRESSION => 'NONE', MIN_VERSIONS => '0', BLOCKCACHE => 'true', BLOCKSIZE =>
'65536', REPLICATION_SCOPE => '0'}
{NAME => 'info', BLOOMFILTER => 'ROW', VERSIONS => '1', IN_MEMORY => 'false', KE
EP_DELETED_CELLS => 'FALSE', DATA_BLOCK_ENCODING => 'NONE', TTL => 'FOREVER', CO
MPRESSION => 'NONE', MIN_VERSIONS => '0', BLOCKCACHE => 'true', BLOCKSIZE => '65
536', REPLICATION_SCOPE => '0'}
2 row(s) in 0.0400 seconds
```

4. 删除一个列族

删除列族'info'

```
alter 'students', {NAME => 'info', METHOD => 'delete'}
```

```
hbase(main):005:0> alter 'students', {NAME => 'info', METHOD => 'delete'}
Updating all regions with the new schema...
1/1 regions updated.
Done.
0 row(s) in 2.1760 seconds
```

```
hbase(main):001:0> describe 'students'
Table students is ENABLED
students
COLUMN FAMILIES DESCRIPTION
{NAME => 'address', BLOOMFILTER => 'ROW', VERSIONS => '1', IN_MEMORY => 'false',
 KEEP_DELETED_CELLS => 'FALSE', DATA_BLOCK_ENCODING => 'NONE', TTL => 'FOREVER',
 COMPRESSION => 'NONE', MIN_VERSIONS => '0', BLOCKCACHE => 'true', BLOCKSIZE =>
'65536', REPLICATION_SCOPE => '0'}
1 row(s) in 0.3700 seconds
```

5. 删除一个表

要彻底删除一表数据和表结构，需要先 disable，再 drop。

```
hbase(main):002:0> disable 'students'
0 row(s) in 2.2750 seconds

hbase(main):003:0> drop 'students'
0 row(s) in 1.2470 seconds

hbase(main):004:0> list
TABLE
tb
1 row(s) in 0.0110 seconds

=> ["tb"]
```

6. 查询表是否存在

```
hbase(main):005:0> exists 'student'
Table student does not exist
0 row(s) in 0.0390 seconds
```

7. 查看表是否可用

```
hbase(main):007:0> is_enabled 'tb'
true
0 row(s) in 0.0160 seconds
```

## DML 操作

DML（Data Manipulation Language，数据操作语言）操作主要用来对表的数据进行添加、修改、获取、删除和查询。

创建一个具有三个列族（name，address, info）的表 students。

1. 插入数据

给 students 表‘xiaoming‘行和’xiaowang’行插入数据

```
put 'students', 'xiaoming', 'address:province', 'zhejiang'
```

```
hbase(main):020:0> put 'students', 'xiaoming', 'address:province', 'zhejiang'
0 row(s) in 0.1340 seconds
```

```
put 'students', 'xiaoming', 'address:city', 'jinhua'
```

```
hbase(main):032:0> put 'students', 'xiaoming', 'address:city', 'jinhua'
0 row(s) in 0.0130 seconds
```

```
put 'students', 'xiaoming', 'info:age', '20'
```

```
hbase(main):022:0> put 'students', 'xiaoming', 'info:age', '20'
0 row(s) in 0.0150 seconds
```

```
put 'students', 'xiaowang', 'address:city', 'hangzhou'
```

```
hbase(main):023:0> put 'students', 'xiaowang', 'address:city', 'hangzhou'
0 row(s) in 0.0050 seconds
```

2. 获取数据

获取 students 表的 xiaoming 行的所有数据。

```
get 'students', 'xiaoming'
```

```
hbase(main):025:0> get 'students', 'xiaoming'
COLUMN                CELL
 address:province     timestamp=1598674992266, value=zhejiang
 info:age             timestamp=1598675064847, value=20
1 row(s) in 0.0120 seconds
```

获取 students 表的 xiaoming 行的 address 列族的所有数据。

```
get 'students', 'xiaoming', 'address'
```

```
hbase(main):033:0> get 'students', 'xiaoming', 'address'
COLUMN                CELL
 address:city         timestamp=1598675777966, value=jinhua
 address:province     timestamp=1598674992266, value=zhejiang
1 row(s) in 0.0060 seconds
```

获取 students 表的 xiaoming 行的 address 列族中 city 列的数据。

```
get 'students', 'xiaoming', 'address:city'
```

```
hbase(main):034:0> get 'students', 'xiaoming', 'address:city'
COLUMN                CELL
 address:city         timestamp=1598675777966, value=jinhua
1 row(s) in 0.0130 seconds
```

3. 更新一条记录

更新 students 表的 xiaowang 行、address 列族中 province 列的值。

```
put 'students', 'xiaowang', 'address:city', 'shanghai'
```

```
hbase(main):037:0> put 'students', 'xiaowang', 'address:city', 'shanghai'
0 row(s) in 0.0130 seconds
```

查看更新的结果。

```
get 'students', 'xiaowang', 'address:city'
```

```
hbase(main):038:0> get 'students', 'xiaowang', 'address:city'
COLUMN                CELL
 address:city         timestamp=1598677304953, value=shanghai
1 row(s) in 0.0050 seconds
```

4. 全表扫描

```
hbase(main):045:0> scan 'students'
ROW                   COLUMN+CELL
 xiaoming             column=address:city, timestamp=1598675777966, value=jinhua
 xiaoming             column=address:province, timestamp=1598674992266, value=zh
                      ejiang
 xiaoming             column=info:age, timestamp=1598675064847, value=20
 xiaowang             column=address:city, timestamp=1598677304953, value=shangh
                      ai
2 row(s) in 0.0190 seconds
```

5. 删除一列

删除 students 表 xiaoming 行的 address 列族的列 city。

```
delete 'students', 'xiaoming', 'address:city'
```

检查删除操作的结果。

```
get 'students', 'xiaoming'
```

```
hbase(main):076:0> delete 'students', 'xiaoming','address:city'
0 row(s) in 0.0030 seconds

hbase(main):077:0> get 'students', 'xiaoming'
COLUMN                CELL
 address:province     timestamp=1598674992266, value=zhejiang
 info:age             timestamp=1598675064847, value=20
1 row(s) in 0.0100 seconds
```

6. 删除行的所有单元格

使用“deleteall”命令删除 students 表 xiaoming 行的所有列。

```
deleteall 'students', 'xiaoming'
```

```
hbase(main):002:0> deleteall 'students', 'xiaoming'
0 row(s) in 0.0480 seconds
```

7. 统计表中的行数

```
count 'students'
```

```
hbase(main):004:0> count 'students'
1 row(s) in 0.0480 seconds

=> 1
```

8. 清空整张表

```
truncate 'students'
```

```
hbase(main):005:0> truncate 'students'
Truncating 'students' table (it may take a while):
 - Disabling table...
 - Truncating table...
0 row(s) in 3.4640 seconds
```

9. 存储多各版本的数据

创建的表，默认列族的 VERSIONS=1，也就是只会存取一个版本的列数据，当再次插入的时候，后面的值会覆盖前面的值。

```
hbase(main):006:0> describe 'students'
Table students is ENABLED
students
COLUMN FAMILIES DESCRIPTION
{NAME => 'address', BLOOMFILTER => 'ROW', VERSIONS => '1', IN_MEMORY => 'false',
 KEEP_DELETED_CELLS => 'FALSE', DATA_BLOCK_ENCODING => 'NONE', TTL => 'FOREVER',
 COMPRESSION => 'NONE', MIN_VERSIONS => '0', BLOCKCACHE => 'true', BLOCKSIZE =>
'65536', REPLICATION_SCOPE => '0'}
```

修改表结构，让表支持存储 3 个本版

```
alter 'students', {NAME=>'address', VERSIONS=>3}
```

```
hbase(main):007:0> alter 'students',{NAME=>'address',VERSIONS=>3}
Updating all regions with the new schema...
1/1 regions updated.
Done.
0 row(s) in 1.8940 seconds

hbase(main):008:0> describe 'students'
Table students is ENABLED
students
COLUMN FAMILIES DESCRIPTION
{NAME => 'address', BLOOMFILTER => 'ROW', VERSIONS => '3', IN_MEMORY => 'false',
 KEEP_DELETED_CELLS => 'FALSE', DATA_BLOCK_ENCODING => 'NONE', TTL => 'FOREVER',
 COMPRESSION => 'NONE', MIN_VERSIONS => '0', BLOCKCACHE => 'true', BLOCKSIZE =>
'65536', REPLICATION_SCOPE => '0'}
```

插入三行数据

```
put 'students', 'xiaoming', 'address:city', 'hangzhou'
put 'students', 'xiaoming', 'address:city', 'suzhou'
put 'students', 'xiaoming', 'address:city', 'shanghai'
```

一次去除三个数据

```
 get 'students','xiaoming', {COLUMN=>'address:city', VERSIONS=>3}
```

```
hbase(main):008:0> get 'students','xiaoming',{COLUMN=>'address:city',VERSIONS=>3
}
COLUMN                CELL
 address:city         timestamp=1598681335999, value=suzhou
 address:city         timestamp=1598681327308, value=shanghai
 address:city         timestamp=1598681312725, value=hangzhou
1 row(s) in 0.0130 seconds
```