Apache Cassandra 内置及自定义数据类型

到目前为止，我们在使用 CQL 建表的时候使用到了一些数据类型，比如 text、timeuuid等。本文将介绍 Apache Cassandra 内置及自定义数据类型。和其他语言一样，CQL 也支持一系列灵活的数据类型，包括基本的数据类型，集合类型以及用户自定义数据类（User-Defined Types,UDTs）。下面将介绍 CQL 支持的数据类型。

数字数据类型（Numeric Data Types）

CQL 支持的数字数据类型包括整型和浮点型，这些数据类型和 Java 的标准数据类型类似。包括以下几种：

- int：32位有符号整型，和 Java 中的 int 类似；
- bigint：64位长整型，和 Java 中的 long 类似；
- smallint：16位有符号整型，和 Java 中的 short 类似，Apache Cassandra 2.2 开始引入；
- tinyint：8位有符号整型，和 Java 中的 tinyint 类似，Apache Cassandra 2.2 开始引入；
- varint：可变精度有符号整数，和 Java 中的 java.math.BigInteger 类似；
- float：32位 IEEE-754 浮点型，和 Java 中的 float 类似；
- double：64位 IEEE-754 浮点型，和 Java 中的 double 类似；
- decimal：可变精度的 decimal，和 Java 中的 java.math.BigDecimal 类似。

注意，枚举类型（enumerated types）虽然在很多语言中都提供，但是在 CQL 是不提供的。在 CQL 中存储枚举常见的方法是使用 String，使用 Enum.name() 将枚举转换成 text，然后使用 Enum.valueOf() 将 text 转换成对应的枚举类型。

文本数据类型（Textual Data Types）

CQL 中提供了两种数据类型用于存放文本类型，其中一种我们在前面的几篇文章里面已经使用过了，也就是 text。

- text, varchar：UTF-8编码的字符串，这个在 CQL 中使用的比较普遍；
- ascii：ASCII字符串。

时间和标识符数据类型（Time and Identity Data Types）

- timestamp
  ：时间可以使用64位有符号的整数表示，但是一般为了可读性，我们会选择支持 ISO 8601 标准的时间戳表示。比如下面的几种数据表示都是可以的：

2019-04-15 20:05-0700
2019-04-15 20:05:07-0700
2019-04-15 20:05:07.013-0700
2019-04-15T20:05-0700
2019-04-15T20:05:07-0700
2019-04-15T20:05:07.013+-0700

建议在使用时间戳的时候都指定时区，而不是依赖系统的时区。

- date, time：在 Apache Cassandra 2.1 版本之前只支持 timestamp 类型，里面包含了日期和时间；从 Cassandra 2.2 版本开始引入了 date 和 time 时间类型，分别表示日期和时间。和 timestamp 一样，这个也是支持 ISO 8601 标准的。
- uuid：通用唯一识别码（universally unique identifier，UUID）是128位数据类型，其实现包含了很多种类型，其中最有名的为 Type 1 和 Type 4。CQL 中的 uuid 实现是 Type 4 UUID，其实现完全是基于随机数的。UUID 的数据类似于 ab7c46ac-c194-4c71-bb03-0f64986f3daa，uuid 类型通常用作代理键，可以单独使用，也可以与其他值组合使用。由于 UUID 的长度有限，因此并不能绝对保证它们是唯一的。我们可以在 CQL 中使用 uuid() 获取 Type 4 UUID。
- timeuuid：这个是 Type 1 UUID，它的实现基于计算机的 MAC 地址，系统时间和用于防止重复的序列号。CQL 中提供了 now(), dateOf() 以及 unixTimestampOf() 等函数来操作 timeuuid 数据类型。由于这些简便的函数，timeuuid 的使用频率比 uuid 数据类型多。

集合数据类型

set

这种数据类型可以存储集合数据类型，set 里面的元素存储是无序的，但是 cql 返回的数据是有序的。set 里面可以存储前面介绍的数据类型，也可以是用户自定义数据类型，甚至是其他集合类型。

为了介绍这个类型的使用，我们使用 《Apache Cassandra 快速入门指南（Quick Start）》 文章中的 iteblog_user 表进行说明。假设我们需要在这张表里面添加 email 信息，如下：

```
cqlsh:iteblog_keyspace> ALTER TABLE iteblog_user ADD emails set<text>;
cqlsh:iteblog_keyspace> SELECT * FROM iteblog_user WHERE first_name = 'Wu';

 first_name | emails | last_name
------------+--------+-----------
         Wu |   null |      Shi

(1 rows)
```

```
cqlsh:iteblog_keyspace> UPDATE iteblog_user SET emails = {'iteblog@iteblog.com'} WHERE first
_name = 'Wu';
cqlsh:iteblog_keyspace> SELECT * FROM iteblog_user WHERE first_name = 'Wu';

 first_name | emails                 | last_name
------------+------------------------+-----------
         Wu | {'iteblog@iteblog.com'} |       Shi

(1 rows)
```

上面我们给 first_name 为 Wu 的数据添加了 email 信息。如果我们还需要往里面加一些 email 信息，可以使用下面语法进行：

```
cqlsh:iteblog_keyspace> UPDATE iteblog_user SET emails = emails + {'cassandra@iteblog.com'
} WHERE first_name = 'Wu';
cqlsh:iteblog_keyspace> SELECT * FROM iteblog_user WHERE first_name = 'Wu';

 first_name | emails                                          | last_name
------------+-------------------------------------------------+-----------
         Wu | {'cassandra@iteblog.com', 'iteblog@iteblog.com'} |       Shi

(1 rows)
```

可见 first_name 为 Wu 的记录已经添加了两条 email 信息了。当然，如果我们需要删除 email，可以使用下面语法进行：

```
cqlsh:iteblog_keyspace> UPDATE iteblog_user SET emails = emails - {'cassandra@iteblog.com'}
WHERE first_name = 'Wu';
cqlsh:iteblog_keyspace> SELECT * FROM iteblog_user WHERE first_name = 'Wu';

 first_name | emails                 | last_name
------------+------------------------+-----------
         Wu | {'iteblog@iteblog.com'} |       Shi

(1 rows)
cqlsh:iteblog_keyspace> UPDATE iteblog_user SET emails ={} WHERE first_name = 'Wu';
cqlsh:iteblog_keyspace> SELECT * FROM iteblog_user WHERE first_name = 'Wu';

 first_name | emails | last_name
------------+--------+-----------
```

```
      Wu |   null |      Shi

(1 rows)
```

上面我们使用 SET emails = emails - {'cassandra@iteblog.com'} 从用户 email 列表里面删除 email，使用 SET emails ={} 清空用户的 email。

list

list 包含了有序的列表数据，默认情况下，数据是按照插入顺序保存的。我们还是使用 iteblog_user 进行说明，比如我们想往这张表里面添加电话等信息，操作如下：

```
cqlsh:iteblog_keyspace> ALTER TABLE iteblog_user ADD phone list<text>;
cqlsh:iteblog_keyspace> UPDATE iteblog_user SET phone = ['13112345678' ] WHERE first_name
 = 'Wu';
cqlsh:iteblog_keyspace> SELECT * FROM iteblog_user WHERE first_name = 'Wu';

 first_name | emails | last_name | phone
------------+--------+-----------+-----------------
      Wu |   null |      Shi | ['13112345678']

(1 rows)
```

上面我们给 first_name 为 Wu 的记录添加了电话信息，如果需要再添加电话信息，其操作和 set 添加信息类似，如下：

```
cqlsh:iteblog_keyspace> UPDATE iteblog_user SET phone = phone + ['15511112222' ] WHERE fir
st_name = 'Wu';
cqlsh:iteblog_keyspace> SELECT * FROM iteblog_user WHERE first_name = 'Wu';

 first_name | emails | last_name | phone
------------+--------+-----------+-------------------------------
      Wu |   null |      Shi | ['13112345678', '15511112222']

(1 rows)
```

可见，新加入的电话号码被放在 list

的后面了。当然，如果我们使用下面的语句，可以往电话号码的前面添加信息：

```
cqlsh:iteblog_keyspace> UPDATE iteblog_user SET phone = ['13344448888' ] + phone WHERE fir
st_name = 'Wu';
cqlsh:iteblog_keyspace> SELECT * FROM iteblog_user WHERE first_name = 'Wu';

 first_name | emails | last_name | phone
------------+--------+-----------+-------------------------------------------------
         Wu |   null |      Shi | ['13344448888', '13112345678', '15511112222']

(1 rows)
```

我们可以使用下标从 list 数据类型里面修改数据：

```
cqlsh:iteblog_keyspace> UPDATE iteblog_user SET phone[1] = '18888888888' WHERE first_nam
e = 'Wu';
cqlsh:iteblog_keyspace> SELECT * FROM iteblog_user WHERE first_name = 'Wu';

 first_name | emails | last_name | phone
------------+--------+-----------+-------------------------------------------------
         Wu |   null |      Shi | ['13344448888', '18888888888', '15511112222']

(1 rows)
```

下标为 1 的元素被修改了。也可以使用下标删除数据：

```
cqlsh:iteblog_keyspace> DELETE phone[2] from iteblog_user WHERE first_name = 'Wu';
cqlsh:iteblog_keyspace> SELECT * FROM iteblog_user WHERE first_name = 'Wu';

 first_name | emails | last_name | phone
------------+--------+-----------+-------------------------------
         Wu |   null |      Shi | ['13344448888', '18888888888']

(1 rows)
```

当然，删除元素也可以使用 SET phone_numbers = phone_numbers - ['13344448888']，这里就不演示了。

map

map 数据类型包含了 key/value 键值对。key 和 value 可以是任何类型，除了 counter 类型。使用如下：

```
cqlsh:iteblog_keyspace> ALTER TABLE iteblog_user ADD login_sessions map<timeuuid, int>;
cqlsh:iteblog_keyspace> UPDATE iteblog_user SET login_sessions = {now(): 13, now(): 18} WHER
E first_name = 'Wu';
cqlsh:iteblog_keyspace> SELECT first_name, login_sessions FROM iteblog_user WHERE first_na
me = 'Wu';

 first_name | login_sessions
------------+-----------------------------------------------------------------------------------
         Wu | {1cc61ff0-5f8b-11e9-ac3a-5336cd8118f6: 13, 1cc61ff1-5f8b-11e9-ac3a-5336cd8118f
6: 18}

(1 rows)
```

其他简单数据类型

- boolean：值只能为 true/false，在 cql 中输入的这两个值无论大小如何写法，其输出都是 True/False；
- blob：二进制大对象（binary large object）是任意字节数组的术语简称。这个类型在存储媒体或者其他二进制数据类型时很有用，Cassandra 并不会检查其中存储的二进制数据是否有效。Cassandra 中二进制大对象是以十六进制存储的，如果我们想将任意的文本数据类型使用 blob 存储，可以使用 textAsBlob() 函数实现。
- inet：这个数据类型可以表示 IPv4 或 IPv6 网络地址。cqlsh 接受用于定义 IPv4 地址的任何合法格式，包括包含十进制，八进制或十六进制值的点或非点式表示。CQL 统一输出为 1.1.1.1 这种 ip 地址形式。
- counter：计数器数据类型是64位有符号整数，其值不能直接设置，而只能递增或递减。计数器类型的使用有一些特殊限制，它不能用作主键的一部分；如果使用计数器，则除primary key 列之外的所有列都必须是计数器。

用户自定义数据类型（User-Defined Types）

Cassandra 中如果内置的数据类型无法满足我们的需求，我们可以使用自定义数据类型的功能。比如我们想用一个字段存储用户的地址信息，然后我们需要获取地址的邮编、街道等信息，如果使用 text 来存储是不能满足我们的需求的。这时候就可以自己定义数据类型，如下：

```
cqlsh:iteblog_keyspace> CREATE TYPE address (
```

```
... street text,
... city text,
... state text,
... zip_code int);
```

上面我们定义了 address 数据类型。需要注意的是，Cassandra 中数据类型的定义是 keyspace 范围的，也就是说， address 数据类型只能在 iteblog_keyspace 里面使用。如果我们使用 DESCRIBE KEYSPACE iteblog_keyspace，可以看到 address 数据类型属于 iteblog_keyspace 的一部分。现在我们定义好了 address 数据类型，我们可以使用它了，如下：

```
cqlsh:iteblog_keyspace> ALTER TABLE iteblog_user ADD addresses map<text, frozen<address>
>;
cqlsh:iteblog_keyspace> UPDATE iteblog_user SET addresses = addresses + {'home': { street: 's
hangdi 9', city: 'Beijing', state: 'Beijing', zip_code: 100080} } WHERE first_name = 'Wu';
cqlsh:iteblog_keyspace> SELECT first_name, addresses FROM iteblog_user WHERE first_name =
'Wu';

 first_name | addresses
------------+-----------------------------------------------------------------------------------
         Wu | {'home': {street: 'shangdi 9', city: 'Beijing', state: 'Beijing', zip_code: 100080}}

(1 rows)
```

可见 我们已经成功的使用了自定义数据类型了。