

腾讯云数据仓库 TCHouse-C 开发指南



腾讯云

【 版权声明 】

©2013–2026 腾讯云版权所有

本文档（含所有文字、数据、图片等内容）完整的著作权归腾讯云计算（北京）有限责任公司单独所有，未经腾讯云事先明确书面许可，任何主体不得以任何形式复制、修改、使用、抄袭、传播本文档全部或部分内容。前述行为构成对腾讯云著作权的侵犯，腾讯云将依法采取措施追究法律责任。

【 商标声明 】



及其它腾讯云服务相关的商标均为腾讯云计算（北京）有限责任公司及其关联公司所有。本文档涉及的第三方主体的商标，依法由权利人所有。未经腾讯云及有关权利人书面许可，任何主体不得以任何方式对前述商标进行使用、复制、修改、传播、抄录等行为，否则将构成对腾讯云及有关权利人商标权的侵犯，腾讯云将依法采取措施追究法律责任。

【 服务声明 】

本文档意在向您介绍腾讯云全部或部分产品、服务的当时的相关概况，部分产品、服务的内容可能不时有所调整。您所购买的腾讯云产品、服务的种类、服务标准等应由您与腾讯云之间的商业合同约定，除非双方另有约定，否则，腾讯云对本文档内容不做任何明示或默示的承诺或保证。

【 联系我们 】

我们致力于为您提供个性化的售前购买咨询服务，及相应的技术售后服务，任何问题请联系 4009100100或 95716。

文档目录

开发指南

数据库引擎

表引擎

表引擎概述

MergeTree

SQL 语法参考

ClickHouse 客户端介绍

ClickHouse 集群迁移方案

开发指南

数据库引擎

最近更新时间：2026-05-25 16:50:52

默认情况下，THouse-C 使用自己的数据库引擎，该引擎提供可配置的 **表引擎** 和所有支持的 **SQL 语法**。此外，还可以选择使用 MySQL 数据库引擎。

延时引擎

在距最近一次访问间隔 `expiration_time_in_seconds` 时间段内，将表保存在内存中，仅适用于 *Log 引擎表。由于针对这类表的访问间隔较长，对保存大量小的 *Log 引擎表进行了优化。

MySQL 引擎

MySQL 引擎用于将远程的 MySQL 服务器中的表映射到 THouse-C 中，并允许对表进行 INSERT 和 SELECT 查询，以便在 THouse-C 与 MySQL 之间进行数据交换。

MySQL 数据库引擎会对其查询转换为 MySQL 语法并发送到 MySQL 服务器中，因此可执行如 SHOW TABLES 或 SHOW CREATE TABLE 之类的操作。**无法对其执行 RENAME、CREATE TABLE 和 ALTER 操作。**

CREATE DATABASE

```
CREATE DATABASE [IF NOT EXISTS] db_name [ON CLUSTER cluster]ENGINE =
MySQL('host:port', ['database' | database], 'user', 'password')
```

MySQL 数据库引擎参数说明：

参数	说明
host:port	连接的 MySQL 地址
database	连接的 MySQL 数据库
user	连接的 MySQL 用户
password	连接的 MySQL 用户密码

MySQL 和 THouse-C 支持的类型对应说明：

MySQL	THouse-C
UNSIGNED TINYINT	UInt8

TINYINT	Int8
UNSIGNED SMALLINT	UInt16
SMALLINT	Int16
UNSIGNED INT, UNSIGNED MEDIUMINT	UInt32
INT, MEDIUMINT	Int32
UNSIGNED BIGINT	UInt64
BIGINT	Int64
FLOAT	Float32
DOUBLE	Float64
DATE	日期
DATETIME, TIMESTAMP	日期时间
BINARY	固定字符串

- 其他的 MySQL 数据类型将全部转换为 **字符串**。
- 同时以上的所有类型都支持 **可为空 (类型名称)**。

使用示例

在 MySQL 中创建表:

```
mysql> USE test;Database changed
mysql> CREATE TABLE `mysql_table` (
  ->   `int_id` INT NOT NULL AUTO_INCREMENT,
  ->   `float` FLOAT NOT NULL,
  ->   PRIMARY KEY (`int_id`));Query OK, 0 rows affected (0,09 sec)
mysql> insert into mysql_table (`int_id`, `float`) VALUES (1,2);Query
OK, 1 row affected (0,00 sec)
mysql> select * from mysql_table;+-----+-----+| int_id | value |+--
-----+-----+|      1 |      2 |+-----+-----+1 row in set (0,00
sec)
```

在 TCHouse-C 中创建 MySQL 类型的数据库, 同时与 MySQL 服务器交换数据:

```
CREATE DATABASE mysql_db ENGINE = MySQL('localhost:3306', 'test',
'my_user', 'user_password')
SHOW DATABASES
┌name┐
│ default │
│ mysql_db │
│ system │
└───┘

SHOW TABLES FROM mysql_db
┌name┐
│ mysql_table │
└───┘

SELECT * FROM mysql_db.mysql_table
┌int_id┐└value┘
│ 1 │ 2 │
└───┘└───┘

INSERT INTO mysql_db.mysql_table VALUES (3,4)
SELECT * FROM mysql_db.mysql_table
┌int_id┐└value┘
│ 1 │ 2 │
│ 3 │ 4 │
└───┘└───┘
```

表引擎

表引擎概述

最近更新时间：2025-02-13 15:53:22

表引擎（即表的类型）决定了：

- 数据的存储方式和位置，写到哪里以及从哪里读取数据。
- 支持哪些查询以及如何支持
- 并发数据访问
- 索引的使用（如果存在）
- 是否可以执行多线程请求
- 数据复制参数

引擎类型

MergeTree

适用于高负载任务的最通用和功能强大的表引擎。这些引擎的共同特点是可以快速插入数据并进行后续的后台数据处理。MergeTree 系列引擎支持数据复制（使用 `Replicated*` 的引擎版本），分区和一些其他引擎不支持的其他功能。详情请参见 [MergeTree](#)。

该类型的引擎：

- MergeTree
- ReplacingMergeTree
- SummingMergeTree
- AggregatingMergeTree
- CollapsingMergeTree
- VersionedCollapsingMergeTree
- GraphiteMergeTree

日志

具有最小功能的轻量级引擎。当您需要快速写入许多小表（最多约100万行）并在以后整体读取它们时，该类型的引擎是最有效的。

该类型的引擎：

- Log
- StripeLog
- TinyLog

集成引擎

用于与其他的数据存储与处理系统集成的引擎，详情请参考 [Integrations](#)。

该类型的引擎：

- HDFS
- Hive
- JDBC
- ODBC
- Kafka
- MySQL
- PostgreSQL

用于其他特定功能的引擎

该类型的引擎：

- Distributed
- MaterializedView
- Dictionary
- Merge
- File
- Null
- Set
- Join
- URL
- View
- Memory
- Buffer

虚拟列

- 虚拟列是表引擎组成的一部分，它在对应的表引擎的源代码中定义。
- 虚拟列不能在 CREATE TABLE 中指定，并且不会包含在 SHOW CREATE TABLE 和 DESCRIBE TABLE 的查询结果中。虚拟列是只读的，所以不能向虚拟列中写入数据。
- 查询虚拟列中的数据时，必须在 SELECT 查询中包含虚拟列的名称。SELECT * 不会返回虚拟列的数据。
- 若创建的表中有一列与虚拟列的名字相同，那么虚拟列将不能再被访问。为了避免这种列名的冲突，虚拟列的名字一般都以下划线开头。

MergeTree

最近更新时间：2023-06-06 11:47:32

MergeTree 表引擎主要用于海量数据分析，支持数据分区、主键索引、稀疏索引和数据 TTL 等。

- 列式存储：只读取需要的列，节省 IO 和 CPU 资源。
- 数据分区：按照日期或其他条件将数据分割成多个部分，方便管理和查询。
- 稀疏主键索引：按照主键或排序键对数据进行排序和索引，加速范围查询。
- 次级跳过索引：根据列的最小值和最大值等统计信息跳过不符合条件的数据，进一步提高查询效率。
- 数据合并：后台定期将多个小的数据部分合并成一个大的数据部分，减少数据冗余和碎片。

MergeTree 引擎还有一些变种，如 ReplicatedMergeTree、AggregatingMergeTree 和 SummingMergeTree 等，它们在基本的 MergeTree 功能上增加了数据复制、数据聚合、数据求和等特性。

MergeTree 表引擎支持 ClickHouse 的所有 SQL 语法，但是部分功能与标准 SQL 存在差异。

具体的 MergeTree 引擎及其变种的使用方式如下表所示：

系列	表引擎	简介	备注
MergeTree	MergeTree	用于插入极大量的数据到一张表中，数据以数据片段的形式一个接着一个地快速写入，数据片段按照一定的规则进行合并。	MergeTree 表引擎的详细使用说明请参见 MergeTree 。
	ReplacingMergeTree	可以替换掉具有相同主键的旧数据。	ReplacingMergeTree 表引擎的详细使用说明请参见 ReplacingMergeTree 。
	CollapsingMergeTree	CollapsingMergeTree 表引擎用于消除 ReplacingMergeTree 表引擎的功能限制。可以显著减少存储量并 SELECT，因此提高查询效率。	CollapsingMergeTree 表引擎的详细使用说明请参见 CollapsingMergeTree 。
	VersionedCollapsingMergeTree	类似于 CollapsingMergeTree，但是可以保留最新版本的数据。	VersionedCollapsingMergeTree 表引擎的详细使用说明请参见 VersionedCollapsingMergeTree 。
	SummingMergeTree	可以对具有相同主键的数据进行求和操作。	说明 SummingMergeTree 表引擎的详细使用说明请参见 SummingMergeTree 。

AggregatingMergeTree

可以对具有相同主键的数据进行聚合操作。

AggregatingMergeTree 引擎的详细使用说明请参见 [AggregatingMergeTree](#)

ⓘ 说明：

生产环境中，通常前面还要增加一个 Replicated 前缀，表示多副本，详细使用说明请参见 [Data Replication](#)。

- ReplicatedSummingMergeTree
- ReplicatedReplacingMergeTree
- ReplicatedAggregatingMergeTree
- ReplicatedCollapsingMergeTree
- ReplicatedVersionedCollapsingMergeTree
- ReplicatedGraphiteMergeTree

SQL 语法参考

最近更新时间：2026-05-25 16:50:52

数据类型

支持整数、浮点数、字符型、日期、枚举值、数组等多种数据类型。

类型列表

类别	名称	类型标识	数据范围或描述
整数	单字节整数	Int8	[-128, 127]
	双字节整数	Int16	[-32768, 32767]
	四字节整数	Int32	[-2147483648, 2147483647]
	八字节整数	Int64	[-9223372036854775808, 9223372036854775807]
	无符号单字节整数	UInt8	[0, 255]
	无符号双字节整数	UInt16	[0, 65535]
	无符号四字节整数	UInt32	[0, 4294967295]
	无符号八字节整数	UInt64	[0, 18446744073709551615]
浮点数	单精度浮点数	Float32	浮点数有效数字6 - 7位
	双精度浮点数	Float64	浮点数有效数字15 - 16位
	自定义浮点	Decimal32(S)	浮点数有效数字 S, S 取值范围[1, 9]
		Decimal64(S)	浮点数有效数字 S, S 取值范围[10, 18]
Decimal128(S)		浮点数有效数字 S, S 取值范围[19, 38]	
字符型	任意长度字符	String	不限定字符串长度
	固定长度字符	FixedString(N)	固定长度的字符串

	唯一标识 UUID 类型	UUID	通过内置函数 generateUUIDv4 生成唯一的标识符
时间类型	日期类型	Date	存储年月日时间，格式 yyyy-MM-dd
	时间戳类型（秒级）	DateTime(timezone)	Unix 时间戳，精确到秒
	时间戳类型（自定义）	DateTime(precision, timezone)	可以指定时间精度
枚举类型	单字节枚举	Enum8	取值范围为[-128, 127]，共256个值
	双字节枚举	Enum16	取值范围为[-32768, 32767]，共65536个值
数组类型	数组类型	Array(T)	表示由 T 类型组成的数组类型，不推荐使用嵌套数组

说明

- 可以使用 UInt8 来存储布尔类型，将取值限制为0或1。
- [其他数据类型官方文档](#)。

使用举例

枚举类型应用

存储某站点用户的性别信息。

```
CREATE TABLE user (uid Int16, name String, gender Enum('male'=1, 'female'=2)) ENGINE=Memory;
INSERT INTO user VALUES (1, 'Gary', 'male'), (2, 'Jaco', 'female');
# 查询数据SELECT * FROM user;

┌uid┆name┆gender┆
├──┆──┆──┆
│ 1 │ Gary │ male  │
├──┆──┆──┆
│ 2 │ Jaco │ female│
├──┆──┆──┆

# 使用CAST函数查询枚举整数值SELECT uid, name, CAST(gender, 'Int8') FROM user;
```

```
┌uid┆name┆CAST(gender, 'Int8')┆
├──┆──┆──┆
│ 1 │ Gary │ 1 │
│ 2 │ Jaco │ 2 │
└──┆──┆──┆
```

数组类型应用

某站点记录每天登录用户的 ID，用来分析活跃用户。

```
CREATE TABLE userloginlog (logindate Date, uids Array(String))
ENGINE=TinyLog;
INSERT INTO userloginlog VALUES ('2020-01-02', ['Gary', 'Jaco']),
('2020-02-03', ['Jaco', 'Sammie']);
# 查询结果SELECT * FROM userloginlog;
```

```
┌logindate┆uids┆
├──┆──┆
│ 2020-01-02 │ ['Gary', 'Jaco'] │
│ 2020-02-03 │ ['Jaco', 'Sammie'] │
└──┆──┆
```

创建数据库或表

使用 CREATE 语句来完成数据库或表的创建。

```
CREATE DATABASE [IF NOT EXISTS] db_name [ON CLUSTER cluster] [ENGINE =
engine(...)]
CREATE TABLE [IF NOT EXISTS] [db.]table_name [ON CLUSTER cluster]
(
    name1 [type1] [DEFAULT|MATERIALIZED|ALIAS expr1] [compression_codec]
[TTL expr1],
    name2 [type2] [DEFAULT|MATERIALIZED|ALIAS expr2] [compression_codec]
[TTL expr2],
    ...
) ENGINE = engine
```

数据库和表都支持本地和分布式两种，分布式方式的创建有以下两种方法：

- 在每台 clickhouse-server 所在机器上都执行创建语句。
- 在集群的任意一个机器上使用 ON CLUSTER 语句创建库表，命令执行成功时当前 V-cluster 的各节点库表均创建成功。

当使用 `clickhouse-client` 进行查询时，若在 A 机上查询 B 机的本地表则会报错 “Table xxx doesn't exist..”。若希望集群内的所有机器都能查询某张表，推荐使用分布式表。

相关官方文档 [CREATE Queries](#)。

查询

使用 `SELECT` 语句来完成数据查询。

```
SELECT [DISTINCT] expr_list
[FROM [db.]table | (subquery) | table_function] [FINAL]
[SAMPLE sample_coeff]
[GLOBAL] [ANY|ALL] [INNER|LEFT|RIGHT|FULL|CROSS] [OUTER] JOIN
(subquery)|table USING columns_list
[PREWHERE expr]
[WHERE expr]
[GROUP BY expr_list] [WITH TOTALS]
[HAVING expr]
[ORDER BY expr_list]
[LIMIT [offset_value, ]n BY columns]
[LIMIT [n, ]m]
[UNION ALL ...]
[INTO OUTFILE filename]
[FORMAT format]
```

相关官方文档 [SELECT Queries Syntax](#)。

批量写入

使用 `INSERT INTO` 语句来完成数据写入。

```
INSERT INTO [db.]table [(c1, c2, c3)] VALUES (v11, v12, v13), (v21, v22,
v23), ...
INSERT INTO [db.]table [(c1, c2, c3)] SELECT ...
```

相关官方文档 [INSERT-INTO](#)。

删除数据

使用 `DROP` 或 `TRUNCATE` 语句来完成数据删除。

 说明

DROP 删除元数据和数据，TRUNCATE 只删除数据。

```
DROP DATABASE [IF EXISTS] db [ON CLUSTER cluster] DROP [TEMPORARY] TABLE
[IF EXISTS] [db.]name [ON CLUSTER cluster]
TRUNCATE TABLE [IF EXISTS] [db.]name [ON CLUSTER cluster]
```

修改表结构

使用 ALTER 语句来完成表结构修改。

```
# 对表的列操作
ALTER TABLE [db].name [ON CLUSTER cluster] ADD COLUMN [IF NOT EXISTS]
name [type] [default_expr] [codec] [AFTER name_after] ALTER TABLE
[db].name [ON CLUSTER cluster] DROP COLUMN [IF EXISTS] name ALTER TABLE
[db].name [ON CLUSTER cluster] CLEAR COLUMN [IF EXISTS] name IN
PARTITION partition_name ALTER TABLE [db].name [ON CLUSTER cluster]
COMMENT COLUMN [IF EXISTS] name 'comment' ALTER TABLE [db].name [ON
CLUSTER cluster] MODIFY COLUMN [IF EXISTS] name [type] [default_expr]
[TTL]

# 对表的分区操作
ALTER TABLE table_name DETACH PARTITION partition_expr ALTER TABLE
table_name DROP PARTITION partition_expr ALTER TABLE table_name CLEAR
INDEX index_name IN PARTITION partition_expr

# 对表的属性操作
ALTER TABLE table-name MODIFY TTL ttl-expression
```

相关官方文档 [ALTER](#)。

查看信息

SHOW 语句

展现数据库、处理列表、表、字典等信息。

```
SHOW DATABASES [INTO OUTFILE filename] [FORMAT format] SHOW PROCESSLIST
[INTO OUTFILE filename] [FORMAT format] SHOW [TEMPORARY] TABLES [{FROM |
IN} <db>] [LIKE '<pattern>' | WHERE expr] [LIMIT <N>] [INTO OUTFILE
```

```
<filename>] [FORMAT <format>]SHOW DICTIONARIES [FROM <db>] [LIKE  
'<pattern>'] [LIMIT <N>] [INTO OUTFILE <filename>] [FORMAT <format>]
```

相关官方文档 [SHOW Queries](#)。

DESCRIBE 语句

查看表的元数据信息。

```
DESC|DESCRIBE TABLE [db.]table [INTO OUTFILE filename] [FORMAT format]
```

函数

函数有两种类型：常规函数和聚合函数，区别是常规函数可以通过一行数据产生结果，聚合函数则需要一组数据来产生结果。

常规函数

算数函数

数据表中各字段参与数学计算函数。

函数名称	用途	使用场景
plus(a, b), a + b	计算两个字段的和	plus(table.field1, table.field2)
minus(a, b), a - b	计算两个字段的差	minus(table.field1, table.field2)
multiply(a, b), a * b	计算两个字段的积	multiply(table.field1, table.field2)
divide(a, b), a / b	计算两个字段的商	divide(table.field1, table.field2)
modulo(a, b), a % b	计算两个字段的余数	modulo(table.field1, table.field2)
abs(a)	取绝对值	abs(table.field1)
negate(a)	取相反数	negate(table.field1)

比较函数

函数名称	用途	使用场景
=, ==	判断是否相等	table.field1 = value
!=, <>	判断是否不相等	table.field1 != value
>	判断是否大于	table.field1 > value
>=	判断是否大于等于	table.field1 >= value
<	判断是否小于	table.field1 < value
<=	判断是否小于等于	table.field1 <= value

逻辑运算函数

函数名称	用途	使用场景
AND	两个条件都必须满足	-
OR	两个条件满足其中之一	-
NOT	取条件判断的相反	-

类型转换函数

转换函数可能会溢出，溢出后的数字与 C 语言中数据类型保持一致。

函数名称	用途	使用场景
toInt(8 16 32 64)	将字符型转化为整数型	toInt8('128') 结果为-127
toUInt(8 16 32 64)	将字符型转化为无符号整数型	toUInt8('128') 结果为128
toInt(8 16 32 64)OrZero	将整数字符型转化为整数型，异常时返回0	toInt8OrZero('a') 结果为0
toUInt(8 16 32 64)OrZero	将整数字符型转化为整数型，异常时返回0	toUInt8OrZero('a') 结果为0
toInt(8 16 32 64)OrNull	将整数字符型转化为整数型，异常时返回NULL	toInt8OrNull('a') 结果为NULL
toUInt(8 16 32 64)OrNull	将整数字符型转化为整数型，异常时返回NULL	toUInt8OrNull('a') 结果为NULL

浮点数类型或日期类型也有上述类似的函数。

相关官方文档 [Type Conversion Functions](#)。

日期函数

相关官方文档 [Functions for working with dates and times](#)。

字符串函数

相关官方文档 [Functions for working with strings](#)。

UUID

相关官方文档 [Functions for working with UUID](#)。

JSON 处理函数

相关官方文档 [Functions for working with JSON](#)。

聚合函数

函数名称	用途	使用场景
count	统计行数或者非 NULL 值个数	count(expr)、COUNT(DISTINCT expr)、count()、count(*)
any(x)	返回第一个遇到的值，结果不确定	any(column)
anyHeavy(x)	基于 heavy hitters 算法，返回经常出现的值。通常结果不确定	anyHeavy(column)
anyLast(x)	返回最后一个遇到的值，结果不确定	anyLast(column)
groupBitAnd	按位与	groupBitAnd(expr)
groupBitOr	按位或	groupBitOr(expr)
groupBitXor	按位异或	groupBitXor(expr)
groupBitmap	求基数 (cardinality)	groupBitmap(expr)
min(x)	求最小值	min(column)
max(x)	求最大值	max(x)
argMin(arg, val)	返回 val 最小值行的 arg 的值	argMin(c1, c2)
argMax(arg, val)	返回 val 最大值行的 arg 的值	argMax(c1, c2)

<code>sum(x)</code>	求和	<code>sum(x)</code>
<code>sumWithOverflow(x)</code>	求和，结果溢出则返回错误	<code>sumWithOverflow(x)</code>
<code>sumMap(key, value)</code>	用于数组类型，对相同 key 的 value 求和，返回两个数组的 tuple，第一个为排序后的 key，第二个为对应 key 的 value 之和	-
<code>skewPop</code>	求偏度	<code>skewPop(expr)</code>
<code>skewSamp</code>	求样本偏度	<code>skewSamp(expr)</code>
<code>kurtPop</code>	求峰度	<code>kurtPop(expr)</code>
<code>kurtSamp</code>	求样本峰度	<code>kurtSamp(expr)</code>
<code>timeSeriesGroupSum(uid, timestamp, value)</code>	对 uid 分组的时间序列对应时间点求和，求和前缺失的时间点线性插值	-
<code>timeSeriesGroupRateSum(uid, ts, val)</code>	对 uid 分组的时间序列对应时间的变化率求和	-
<code>avg(x)</code>	求平均值	-
<code>uniq</code>	计算不同值的近似个数	<code>uniq(x[, ...])</code>
<code>uniqCombined</code>	计算不同值的近似个数，相比 <code>uniq</code> 消耗的内存更少，精度更高，但是性能稍差	<code>uniqCombined(HLL_precision)(x[, ...])</code> 、 <code>uniqCombined(x[, ...])</code>
<code>uniqCombined64</code>	<code>uniqCombined</code> 的 64bit 版本，结果溢出的可能性降低	-
<code>uniqHLL12</code>	计算不同值的近似个数，不建议使用。请用 <code>uniq</code> 、 <code>uniqCombined</code>	-
<code>uniqExact</code>	计算不同值的精确个数	<code>uniqExact(x[, ...])</code>
<code>groupArray(x, groupArray(max_size)(x))</code>	返回 x 取值的数组，数组大小可由 <code>max_size</code> 指定	-
<code>groupArrayInsertAt(value, position)</code>	在数组的指定位置 <code>position</code> 插入值 <code>value</code>	-

groupArrayMovingSum	-	-
groupArrayMovingAvg	-	-
groupUniqArray(x), groupUniqArray(max_ size)(x)	-	-
quantile	-	-
quantileDeterministic	-	-
quantileExact	-	-
quantileExactWeighted	-	-
quantileTiming	-	-
quantileTimingWeighted	-	-
quantileTDigest	-	-
quantileTDigestWeighted	-	-
median	-	-
quantiles(level1, level2, ...)(x)	-	-
varSamp(x)	-	-
varPop(x)	-	-
stddevSamp(x)	-	-
stddevPop(x)	-	-
topK(N)(x)	-	-
topKWeighted	-	-
covarSamp(x, y)	-	-
covarPop(x, y)	-	-

corr(x, y)	-	-
categoricalInformationValue	-	-
simpleLinearRegression	-	-
stochasticLinearRegression	-	-
stochasticLogisticRegression	-	-
groupBitmapAnd	-	-
groupBitmapOr	-	-
groupBitmapXor	-	-

字典

一个字典是一个映射（key -> attributes），能够作为函数被用于查询，相比引用（reference）表 `JOIN` 的方式更简单和高效。

数据字典有两种，一个是内置字典，另一个是外置字典。

内置字典

支持一种 [内置字典](#) `geobase`，支持的函数可参考 [Functions for working with Yandex.Metrica dictionaries](#)。

外置字典

可以从多个数据源添加 [外置字典](#)，支持的数据源可参考 [Sources Of External Dictionaries](#)。

ClickHouse 客户端介绍

最近更新时间：2026-05-25 16:50:52

云数据仓库 TCHouse-C 本身提供两种客户端接口，分别基于 HTTP 和 TCP 协议。

基于 HTTP 协议

主要用来支持轻量级的简单操作，方便跨平台和编程语言。EMR 集群内的 `clickhouse-server` 进程会启动 8123 端口的 HTTP 服务，可以发送简单的 GET 请求检查服务是否正常。

```
$ curl http://127.0.0.1:8123.
```

还可以通过 query 参数发送请求，例如查询 testdb 中 account 表的数据。

```
$ wget -q -O- 'http://127.0.0.1:8123/?query=SELECT * from
testdb.account'1      GHua      WuHan Hubei      19902      SLiu
ShenZhen Guangzhou      19913      JPong      Chengdu Sichuan 1992
```

其他用法可参考官方文档 [HTTP Interface](#)。

基于 TCP 协议

主要在 clickhouse-client 端使用，在云数据仓库 TCHouse-C 集群内输入 clickhouse-client 命令，会输出版本信息、连接到的 clickhouse-server 地址、默认使用的数据库等。可以通过 quit、exit 或 q 等退出使用。

```
$ clickhouse-client
ClickHouse client version 19.16.12.49.
Connecting to localhost:9000 as user default.
Connected to ClickHouse server version 19.16.12 revision 54427.
```

clickhouse-client 使用的主要参数：

参数	说明
<code>-C --config-file</code>	指定客户端使用的配置文件
<code>-h --host</code>	指定 clickhouse-server 的 IP 地址
<code>--port</code>	指定 clickhouse-server 的端口地址

-u --user	用户名
--password	密码
-d --database	数据库名称
-V --version	查看客户端版本
-E --vertical	查询结果按照垂直格式显示
-q --query	非交互模式下传入的 SQL 语句
-t --time	非交互模式下显示执行时间
--log-level	客户端日志级别
--send_logs_level	指定服务端返回日志数据的级别
--server_logs_file	指定服务端日志保存路径

其他参数可参考官方文档 [Command-line Client](#)。

ClickHouse 集群迁移方案

最近更新时间：2026-04-07 15:13:41

本文介绍如何将自建 IDC 或其他云平台上的 ClickHouse 集群的数据迁移至腾讯云数据库 TCHouse-C 集群。在数据迁移之前，确保自建集群与目标集群之间网络互通。

1. 数据库表 Schema 迁移

通过执行 `SELECT DATABASE, TABLE FROM system.tables WHERE database != 'system'` 来确定数据库中所有的表。

在源数据库中通过 `database.table` 可以查询该表的 DDL 语句。

```
clickhouse-client --host $CLICKHOUSE_HOST --port $CLICKHOUSE_PORT --user
$USER --password $PASSWORD -q "SHOW CREATE database.table" >
database.table.sql
```

获取包含 DDL 语句的文件，在新集群中执行。其中，`CLICKHOUSE_HOST` 和 `CLICKHOUSE_PORT` 是源服务入口，`USER` 和 `PASSWORD` 是用户名和密码。操作步骤如下：

在新集群中创建对应的数据库：

```
clickhouse-client --host $TENCENT_CLICKHOUSE_HOST --port
$TENCENT_CLICKHOUSE_PORT --user $USER --password $PASSWORD -q "CREATE
DATABASE database"
```

创建表：

```
clickhouse-client --host $TENCENT_CLICKHOUSE_HOST --port
$TENCENT_CLICKHOUSE_PORT --user $USER --password $PASSWORD >
database.table.sql
```

其中，`TENCENT_CLICKHOUSE_HOST` 和 `TENCENT_CLICKHOUSE_PORT` 是腾讯云数据库 TCHouse-C 服务的入口，`USER` 和 `PASSWORD` 是用户名和密码。

2. 通过 Remote 函数进行数据迁移

在数据规模较少的情况下，可以通过 Remote 函数，将源集群的数据直接迁移至新集群中。执行如下 SQL：

```
INSERT INTO target_database.table
SELECT *
```

```
FROM remote ( 'source_cluster' , source_database . table , $ USER , $PASSWOR
D )
```

该方案优势在于简单且可靠性高，但较为依赖网络传输效率。若因网络问题出现超时，可能需要重新导入数据。同时在迁移过程中，可能会对源集群造成一定的读压力，建议在业务低峰期操作。

在数据规模较大时（百 GB 级及以上），可以按照更细粒度的条件分批迁移（如按分区、天、小时等），例如按天进行迁移：

```
INSERT INTO target_database . table
SELECT *
FROM remote ( 'source_cluster' , source_database . table , $ USER , $PASSWOR
D )
WHERE event_date = '2025-05-25'
```

或者按照日期区间进行迁移：

```
INSERT INTO target_database . table
SELECT *
FROM remote ( 'source_cluster' , source_database . table , $ USER , $PASSWOR
D )
WHERE event_date BETWEEN '2025-06-01' AND '2025-06-30'
```

❗ 说明：

在数据规模较大时，建议采用本地表对本地表的迁移模式，直写目标集群的分布式表可能导致内存不足或者出现 Too many parts 报错。

可通过如下 SQL 来查询数据表分区的大小：

```
SELECT
    database,
    table,
    partition,
    count ( ) ,
    formatReadableSize ( sum ( bytes_on_disk ) )
FROM system.parts
WHERE active
GROUP BY
    database,
```

```
table,  
partition  
ORDER BY  
database ASC,  
table ASC,  
partition ASC
```

3. 通过导出/导入数据文件进行数据迁移

将数据从源集群中导出到文件，并将该文件导入到新集群中。操作步骤如下：

数据导出：

```
clickhouse-client --host $CLICKHOUSE_HOST --port $CLICKHOUSE_PORT --user  
$USER --password $PASSWORD -q "SELECT * FROM database.table FORMAT CSV"  
> database.table.dat
```

数据导入：

```
clickhouse-client --host $TENCENT_CLICKHOUSE_HOST --port  
$TENCENT_CLICKHOUSE_PORT --user $USER --password $PASSWORD -q "INSERT  
INTO database.table FORMAT CSV" < database.table.dat
```

ⓘ 说明：

此处可选择多种数据格式，ClickHouse 可支持的格式请参见 [官方文档](#)。

4. 通过 JDBC 完成数据迁移

如果有后备数据源，还可以通过 JDBC 将数据重新写入到腾讯云数据仓库 TCHouse-C 集群中。关于 JDBC 写入详情请见 [官方文档](#)。