流计算 Oceanus

SQL 开发指南

版权所有：腾讯云计算（北京）有限责任公司

【版权声明】

©2013-2025 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

SQL 开发指南
- 开发指南概述
- 术语和数据类型
  - 术语表
  - 数据类型
- DDL 数据定义语句
  - CREATE TABLE
  - CREATE VIEW
  - CREATE FUNCTION
- DML 数据操作语句
  - 查询语句
  - INSERT 语句
- MySQL CDC 多 Source 复用
- 整库同步（SQL）能力
- 上下游开发指南
  - 消息队列 Kafka
  - 消息队列 Upsert Kafka
  - 消息队列 Pulsar
  - 消息队列 CMQ
  - 消息队列 TDMQ RabbitMQ
  - 日志消费 CLS
  - 数据库 MySQL CDC
  - 数据库 TDSQL-MySQL
  - 数据库 PostgreSQL CDC
  - 数据库 MongoDB CDC
  - 数据库 SQLServer CDC
  - 文件系统 FileSystem
  - 数据湖 Hudi
  - 数据湖 Iceberg
  - 数据湖计算 DLC
  - 数据仓库 Hive
  - 数据仓库 Doris
  - 数据仓库 StarRocks
  - 数据仓库 TCHouse-P
  - 数据仓库 ClickHouse
  - 数据仓库 Kudu
  - 数据库 Redis
  - 数据库 MongoDB
  - 数据库 HBase
  - 数据库 Oceanbase
  - 数据库连接 JDBC
  - 数据分析引擎 Elasticsearch Service
  - 模拟上下游 Datagen Logger Blackhole
  - 自定义 Connector
- SET 控制语句
  - Flink 配置项
- 运算符和内置函数
  - 概览
  - 内置函数差异对比

- 比较函数
- 逻辑和位运算函数
- 算术函数
- 条件函数
- 字符串操作函数
- 类型转换函数
- 时间相关函数
- 聚合函数
- 时间窗口函数
- 其他函数

标识符与保留字
- 命名规则
- 保留字

版权所有：腾讯云计算（北京）有限责任公司

SQL 开发指南
开发指南概述

最近更新时间：2023-06-21 15:21:44

SQL 与其他的编程语言相比学习成本较低，使用 SQL 作业进行开发将降低数据开发人员使用 Flink 的门槛。
本章节提供在独享集群 SQL 作业的开发指南。通过阅读本章节，您将学习到以下内容：

- 术语和数据类型
- DDL 数据定义语句
- DML 数据操作语句
- MySQL CDC 多 Source 复用
- 整库同步（SQL）能力
- 上下游开发指南
- SET 控制语句
- 运算符与内置函数
- 标识符与保留字

术语和数据类型
术语表

最近更新时间：2023-09-19 15:26:31

流计算常用术语如下：

术语	详细说明
流计算	流计算是面向流式数据的计算，它从一个或多个流式数据源读取持续不断产生的数据，经过引擎中多个算子的组合进行高效计算，再根据实际需要，将结果输出至下游的多种数据目的，例如消息队列、数据库、数据仓库、存储服务等。
数据源（Source）	为流计算系统持续提供输入数据，例如腾讯云 CKafka 等。
数据目的（Sink）	流计算系统处理结果输出的地方，例如腾讯云 CKafka、云数据库 MySQL、PostgreSQL 等。
Schema	表示一个表的结构信息，例如各个列名、列类型等。对于 PostgreSQL 而言，Schema 是介于 Database 和 Table 之间的一个层级，可以理解成数据库内部的命名空间。
时间模式	指导系统处理数据时如何获取时间戳，目前支持 Event Time、Processing Time、Source Time 三种时间模式。
Event Time	Event Time 时间模式下，时间戳由输入数据的某个字段提供，可以用 WATERMARK FOR 语句指定该字段并启用 Event Time 时间模式，适用于数据源包含精确时间戳的场合。
Watermark	表示一个特定的时间点，在该时间点之前的所有数据已经得到妥善处理。 Watermark 由系统自动生成，用户可通过 WATERMARK FOR BOUNDED 语句指定时间戳的最大容差。
Processing Time	Processing Time 时间模式下，时间戳由系统自动生成并添加到数据源中（以 `PROCTIME` 命名，`SELECT *` 时不可见，使用时必须显式指定）。它以每条数据被系统处理的时间作为时间戳，因而有一定的不可控性，适用于对时间精度要求不是很高的场合。
Source Time	在 Source Time 时间模式下，可使用 Kafka 每条记录所含元数据的时间戳作为流计算处理所使用的时间戳（以 `SOURCETIME` 命名，`SELECT *` 时不可见，使用时必须显式指定），避免了输入数据没有时间戳字段时，使用 Processing Time 模式带来的不可控性。
时间窗口	定义了多个时间段以及各个时间段之间的关系（例如是否可重叠、是否固定大小）。目前系统支持 TUMBLE、HOP、SESSION 三种时间窗口。具体见 时间窗口函数 。
TencentDB for MySQL	TencentDB 是腾讯云提供的一种高性能、高可靠、可灵活伸缩的数据库托管服务。它让用户可以轻松在云端部署、使用 MySQL、PostgreSQL 等数据库。
CKafka	CKafka 是腾讯云提供的一个分布式的、高吞吐量、高可扩展性的消息系统，完全兼容0.10版本的 Kafka API。流计算目前支持 CSV 和 JSON 两种输入输出格式。
Tuple 与 Append 流	Tuple（又称为 Append）为数据流类型的一种，可以存放不含主键的流数据。用户可以不断追加新数据到这种数据流中，它不涉及到对之前已发出数据的更新操作。目前各种数据源和数据目的均支持 Append 流的输入输出。
Upsert 流	Upsert 是 Update OR Insert 的简写，由 DISTINCT、不含时间窗口的 GROUP BY 语句、不含时间范围的 JOIN 语句等查询产生，它具有主键定义，如果后续发出的数据与之前的某条数据具有相同主键，则更新该条记录为新值；反之则新增一行数据。它可以确保之前发出的数据被更新以反映最新的值。
DDL 语句	DDL 即数据定义语言（Data Definition Language），为 SQL 语言的一个子集，由 CREATE 语句组成。它可以用来定义表、视图、以及用户自定义函数（UDF）等。
DML 语句	DML 即数据操作语言（Data Manipulation Language），为 SQL 语言的一个子集，包括 INSERT 和 SELECT 语句，可以用来对数据表、视图等进行选择、变换、筛选、插入等操作。

版权所有：腾讯云计算（北京）有限责任公司

数据类型

最近更新时间：2023-06-21 15:21:45

流计算 Oceanus 采用符合 ANSI SQL 规范的定义，支持丰富的数据类型。用户在使用 `CREATE TABLE` 语句定义一个数据表时，可以用这些数据类型来定义每个字段的类型。

支持的类型列表

类型名称	使用说明
CHAR CHAR(n)	定长字符串。n 表示容纳的字符数，默认为1，即 CHAR 等价于 CHAR(1)。
VARCHAR VARCHAR(n) STRING	可变长度字符串。n 表示最多容纳的字符数，默认为1，即 VARCHAR 等价于 VARCHAR(1)。STRING 等价于 VARCHAR(2147483647)。
BINARY BINARY(n)	固定长度的二进制字符串。n 表示容纳的字节数量，默认为1，即 BINARY 等价于 BINARY(1)。
VARBINARY VARBINARY(n) BYTES	可变长度的二进制字符串。n 表示容纳的字节数量，默认为1，即 VARBINARY 等价于 VARBINARY(1)。BYTES 等价于 VARBINARY(2147483647)。
DECIMAL DECIMAL(p) DECIMAL(p, s) DEC DEC(p) DEC(p, s) NUMERIC NUMERIC(p) NUMERIC(p, s)	固定精度的实数（定点数）。 p 表示数字的总位数（精度），取值区间为[1，38]，默认值是10。 s 表示小数点右边的位数（尾数），取值区间为[0，p]，默认值是0。DEC 和 NUMERIC 是 DECIMAL 的别名，可以任意互换使用，即 DECIMAL(p, s) 等价于 DEC(p, s) 也等价于 NUMERIC(p, s)。
TINYINT	1个字节的整数。等价于 Java 的 Byte 类型，取值范围是[-128，127]。
SMALLINT	2个字节的整数。等价于 Java 的 Short 类型，取值范围是[-32768，32767]。
INT	4个字节的整数。等价于 Java 的 Integer 类型，取值范围是[-2147483648，2147483647]。
BIGINT	8个字节的整数。等价于 Java 的 Long 类型，取值范围是[-9223372036854775808，9223372036854775807]。
FLOAT	4个字节的单精度浮点数。等价于 Java 的 Float 类型。
DOUBLE	8个字节的双精度浮点数。等价于 Java 的 Double 类型。
DATE	日期类型，包含年-月-日。取值范围是[0000-01-01，9999-12-31]。
TIME TIME(p)	不含时区信息的时间类型，包含时-分-秒及纳秒信息。 取值范围是[00:00:00.000000000，23:59:59.999999999]。 p 表示秒的小数位精度，取值范围是[0，9]。如果未指定，默认为0。 此类型不支持闰秒。类似于 Java 的 LocalTime 类型。
TIMESTAMP TIMESTAMP(p) TIMESTAMP WITHOUT TIME ZONE TIMESTAMP(p) WITHOUT TIME ZONE	不含时区信息的时间戳类型，精度可以达到纳秒级别。 取值范围是[0000-01-01 00:00:00.000000000，9999-12-31 23:59:59.999999999]。 p 表示秒的小数位精度，取值范围是[0，9]。如果未指定，默认为6。 此类型不支持与 BIGINT（Java 的 Long 类型）之间相互转换。TIMESTAMP WITHOUT TIMEZONE 类型等价于 TIMESTAMP 类型。 此类型不支持闰秒。类似于 Java 的 Timestamp 类型。
TIMESTAMP WITH TIME ZONE	含时区信息的时间戳类型。

TIMESTAMP(p) WITH TIME ZONE	取值范围是[0000-01-01 00:00:00.000000000 +14:59，9999-12-31 23:59:59.999999999 -14:59]。 p 表示秒的小数位精度，取值范围是[0，9]。如果未指定，默认为6。 每条该类型的数据，都含有各自的时区信息。 此类型不支持闰秒。类似于 Java 的 OffsetDateTime 类型。
TIMESTAMP WITH LOCAL TIME ZONE TIMESTAMP(p) WITH LOCAL TIME ZONE	含 本地时区 信息的时间戳类型。 取值范围是[0000-01-01 00:00:00.000000000 +14:59，9999-12-31 23:59:59.999999999 -14:59]。 p 表示秒的小数位精度，取值范围是[0，9]。如果未指定，默认为6。 时区数据不存储在每条数据中，而是遵循全局的时区设置。 此类型不支持闰秒。类似于 Java 的 OffsetDateTime 类型。
INTERVAL YEAR INTERVAL YEAR(p) INTERVAL YEAR(p) TO MONTH INTERVAL MONTH	表示以年和月表示的一段粗粒度的时间间隔，精度为月份。 语法为+年数-月数，例如+04-02。 取值范围是[-9999-11，+9999-11]。 p 表示年的精度位数，取值范围是[1，4]，默认为2。
INTERVAL DAY INTERVAL DAY(p1) INTERVAL DAY(p1) TO HOUR INTERVAL DAY(p1) TO MINUTE INTERVAL DAY(p1) TO SECOND(p2) INTERVAL HOUR INTERVAL HOUR TO MINUTE INTERVAL HOUR TO SECOND(p2) INTERVAL MINUTE INTERVAL MINUTE TO SECOND(p2) INTERVAL SECOND INTERVAL SECOND(p2)	表示以天、时、分、秒、纳秒表示的细粒度时间间隔，最高精度为纳秒。 取值范围是[-999999 23:59:59.999999999，+999999 23:59:59.999999999]。 p1 表示天数精度的位数，p1 的取值范围是[1，6]，默认为2。 p2 表示秒的小数位数，p2 的取值范围是[0，9]，默认为6。
ARRAY<t> t ARRAY	数组类型，大小固定为2147483647。 t 表示数组中元素的类型。 ARRAY<t> 等价于 t ARRAY，例如 ARRAY<INT> 与 INT ARRAY 含义一致。
MAP<kt, vt>	键值对映射类型，其中 kt 是键（key）的类型，vt 是值（value）的类型。
MULTISET<t> t MULTISET	允许重复元素的集合类型，别称为 Bag。同样地，MULTISET<t> 等价于 t MULTISET。
ROW<n0 t0, n1 t1, ...> ROW<n0 t0 'd0', n1 t1 'd1', ...> ROW(n0 t0, n1 t1, ...) ROW(n0 t0 'd0', n1 t1 'd1', ...)	允许包含多个字段的复合类型，每个字段有自己的类型，类似于其他语言的 Struct 及 Tuple 类型。 n 表示字段名。 t 是字段的逻辑类型。 d 是字段的描述。 尖括号和圆括号的两种写法是等价的，例如 ROW(field1 INT, field2 BOOLEAN) 等同于 ROW<field1 INT, field2 BOOLEAN>。
BOOLEAN	三值布尔型，可选值为 TRUE、FALSE 和 UNKNOWN。如果不允许出现 UNKNOWN，可以定义为 BOOLEAN NOT NULL 类型。
RAW('class', 'snapshot')	可表示任意类型，例如 Flink 无法识别或者不需要识别的类型。 class 表示原始类型。 snapshot 表示 Base64 编码的序列化后的 TypeSerializerSnapshot 定义。
NULL	空值，类似 Java 等语言中的 null 值。

DDL 数据定义语句
CREATE TABLE

最近更新时间：2023-06-21 15:21:45

CREATE TABLE 语句用来描述数据源（Source）或者数据目的（Sink）表，并将其定义为一张表，以供后续语句引用。

表定义语法

语法结构

```
CREATE TABLE 表名
  (
    { <列定义> | <计算列定义> }[ , ...n]
    [ <Watermark 定义> ]
    [ <表约束定义, 例如 Primary Key 等> ][ , ...n]
  )
  [COMMENT 表的注释]
  [PARTITIONED BY (分区列名1, 分区列名2, ...)]
  WITH (键1=值1, 键2=值2, ...)
  [ LIKE 其他的某个表 [( <LIKE 子句选项> )] ]
```

符号含义

CREATE TABLE 语句创建的表，既可以作为数据源表，也可以作为数据目的表。但是如果没有对应的 Connector，则会在运行时报错。

```
<列定义>:
  列名 列类型 [ <列的约束定义> ] [COMMENT 列的注释]

<列的约束定义>:
  [CONSTRAINT 约束名] PRIMARY KEY NOT ENFORCED

<表的约束定义>:
  [CONSTRAINT 约束名] PRIMARY KEY (列名1, 列名2, ...) NOT ENFORCED

<计算列定义>:
  列名 AS 计算列表达式 [COMMENT 列的注释]

<Watermark 定义>:
  WATERMARK FOR 某个Rowtime类型的列名 AS 某个Watermark策略表达式

<LIKE 子句选项>:
{
   { INCLUDING | EXCLUDING } { ALL | CONSTRAINTS | PARTITIONS }
 | { INCLUDING | EXCLUDING | OVERWRITING } { GENERATED | OPTIONS | WATERMARKS }
}[, ...]
```

子句功能说明

计算列

计算列是一种虚拟列，它是逻辑上的定义而非数据源中实际存在的列，通常由同一个表的其他列、常量、变量、函数等计算而来。例如，如果数据源中定义了 price（商品单价）和 quantity（采购量），那么就可以新定义一个 cost（总成本）字段，即 `cost AS price * quantity`，即可在后续查询中直接使用 cost 字段。
更常见的用法是使用计算列来实现非标准时间戳的标准化。例如在一个数据源中，时间戳字段 mytime 为 Unix 格式（例如以毫秒为单位的 1599469771494），则可以通过计算列的方式 `ts AS TO_TIMESTAMP(FROM_UNIXTIME(mytime / 1000, 'yyyy-MM-dd HH:mm:ss'))`，将时间

戳处理为 Flink 可识别的 Timestamp(3) 类型。另外，如果数据源时间戳字段虽然是 Timestamp(3) 格式，但是嵌套在 JSON 的其他字段中，也可以用计算列的方式将其解析出来。

> **注意**
> - 计算列只允许在 SELECT 语句中使用。
> - 对于 INSERT 语句，目的表中的计算列会被自动忽略。

WATERMARK

Watermark 定义

Watermark 决定着 Flink 作业的时间模式（详见下文的 Event Time/Processing Time 介绍小节），定义方式：

```
WATERMARK FOR 某个Rowtime类型的列名 AS 某个Watermark策略表达式
```

例如 `WATERMARK FOR my_time_field AS my_time_field - INTERVAL '3' SECOND` 表示定义一个容差为 3 秒的 Watermark 策略。

- 某个 Rowtime 类型的列名：必须是 Flink 可识别的 `Timestamp(3)` 类型，且不是嵌套列。如果类型不对或者属于嵌套字段，则需要使用上文提到的“计算列”功能，创建一个转换后的虚拟列，作为 Rowtime 类型的列。
- 某个 Watermark 策略表达式：用于定义 Watermark 的生成策略，可以用各种表达式来描述一个 `Timestamp(3)` 类型的值，以作为每次生成 Watermark 时的依据。

定义示例如下：

```sql
CREATE TABLE StudentRecord (
    Id BIGINT,
    StudentName STRING,
    RegistrationTime TIMESTAMP(3),
    WATERMARK FOR RegistrationTime AS RegistrationTime - INTERVAL '3' MINUTE
) WITH ( ... ... );
```

Watermark 生成策略

单调递增时间戳的 Watermark 策略

如果时间戳可以确保是单调递增的，不存在乱序的情况，则可以用如下语法，以期得到最低的数据处理延迟。

```
WATERMARK FOR 某个Rowtime类型的列名 AS 某个Rowtime类型的列名
```

下面语句的含义是将每个输入数据中最大时间戳作为 Watermark 的取值。因此如果存在乱序，就会造成晚到的数据未达到 Watermark 的界限而被丢弃。例如：

```sql
WATERMARK FOR my_time AS my_time
```

有限容忍乱序的 Watermark 策略

与上述的策略不同，本策略允许数据中存在一定范围的乱序。这个乱序范围由用户自行控制，如果设置的较大，则会带来较长的延迟（数据积压、等待）；如果设置的较小，则超过阈值的数据则可能被丢弃（造成结果不准确）。

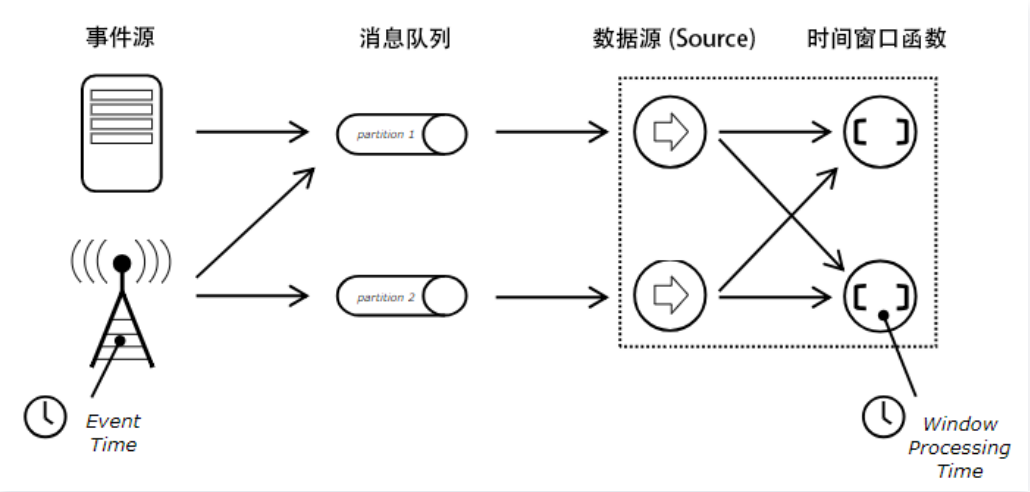
```
WATERMARK FOR 某个Rowtime类型的列名 AS 某个Rowtime类型的列名 - INTERVAL '时间长度' 时间单位
```

下面语句的含义是每个输入数据中最大的时间戳减去3秒的容差作为 Watermark 的取值。因此如果存在乱序，但是后来的数据比之前最大值相差不到3秒，也会被允许加入计算。

```sql
WATERMARK FOR my_time AS my_time - INTERVAL '3' SECOND
```

Event Time/Processing Time 介绍

对于基于窗口的操作（例如 GROUP BY、OVER、JOIN 条件中时间段的指定），流计算 Oceanus 支持 Event Time 和 Processing Time 两种时间处理模式。



- Event Time 模式使用输入数据自带的时间戳，容忍一定程度的乱序数据输入（例如，更早的数据由于各节点处理能力、网络波动等不可预知的原因，来的却更晚），这个参数可以通过 BOUNDED 的第二个参数指定，单位是毫秒。该处理模式最精确，但要求输入数据自带时间戳。目前只支持数据源中以 timestamp 类型定义的字段，未来将会支持虚拟列，可将其他类型的列应用处理函数转换为系统接受的时间戳。
- Processing Time 处理模式不要求输入数据有时间戳，而是将该条数据被处理的时间戳自动加入数据，并以 PROCTIME（必须全为大写）字段命名。该列是隐藏的，`SELECT *` 时不会出现，只有用户手动使用时才会被读取。

> **注意**
> 对于同一个任务的所有数据源，只允许采用一种时间模式。若某个使用 Event Time 模式，则必须要求所有定义的 Table Source 都定义时间戳并声明 WATERMARK 时间戳字段。

主键 PRIMARY KEY

定义表或视图时，可以声明某些字段为主键（PRIMARY KEY），表示这些字段的值不会重复且不会为 NULL（即 SQL 的 NOT NULL + UNIQUE）。主键的定义可以在列上，也可以单独使用 CONSTRAINT 语句定义。**不能对同一个表多次定义不同的主键**。Flink 因为无法保证数据源的每条数据主键不重复，目前只支持 PRIMARY KEY NOT ENFORCED 语法，即提醒用户需自行保证主键语义。

分区 PARTITIONED BY

如果在某个列上定义了 PARTITIONED BY 子句，则表明允许 Flink 对该列进行分区。主要影响 FileSystem Sink，它会根据分区的不同，为每个数据分区创建一个单独的目录。
流计算 Oceanus 不建议用户使用 FileSystem Sink，因为所有 TaskManager 运行结束后，文件系统的数据会被自动清理。

WITH 参数

WITH 参数通常用于指定数据源和数据目的的 Connector 所需参数，语法为 `'key1'='value1', 'key2' = 'value2'` 的键值对。
例如要写入 Kafka（腾讯云 CKafka 或自建 Kafka）时，需要指定服务器地址、消费的 Topic、消费的起始时间点等信息。
对于常见的上下游 Connector 的具体的使用方法，可参考 上下游开发指南 。

LIKE 子句

LIKE 子句允许用于创建表（下文称为 B 表）时，引用其他表（下文称为 A 表）的结构，这样可以大幅节省 CREATE TABLE 语句的代码量，做到代码复用。例如，把同样的数据一份写入 Kafka Sink，另一份写入 Elasticsearch Sink，还有一部分写入 MySQL，那么就可以通过 LIKE 语句来实现定义三张表，同时复用列定义的效果。
定义一张 A 表：

```
CREATE TABLE A (
    Id BIGINT,
    StudentName STRING,
    RegistrationTime TIMESTAMP(3)
) WITH ( ... 某些参数 ... );
```

再定义一个含 Watermark 的 B 表，则可以直接基于上面的表，创建一个新的表：

```
CREATE TABLE B (
    WATERMARK FOR RegistrationTime AS RegistrationTime - INTERVAL '3' MINUTE
) WITH ( ... 另一些参数 ... ) LIKE `A`;
```

默认情况下 LIKE 语句与 WITH 参数无关，所以两个表允许使用完全不同的 WITH 参数集。如果希望继承原表的 WITH 参数信息，则需要通过 **LIKE 子句选项**来实现。

LIKE 子句选项

目前 LIKE 子句提供了如下的选项，可以控制引用（继承）某个 A 表的内容：

- CONSTRAINTS：主键（PRIMARY KEY）等约束
- GENERATED：计算列
- OPTIONS：WITH 参数
- PARTITIONS：PARTITIONED BY 定义
- WATERMARKS：WATERMARK FOR 定义
- ALL：以上所有

同时，Flink 提供了三种不同的合并策略：

- INCLUDING：继承 A 表的所有指定属性，但如果 A 表和 B 表的某些定义有冲突（例如含有相同字段定义）则报错。
- EXCLUDING：B 表中**不会**包含任何 A 表中已有的指定属性。
- OVERWRITING：继承 A 表的所有指定属性，如果 A 表和 B 表定义有冲突，则 B 表的定义会覆盖 A 表的定义。

如果未提供 LIKE 子句选项，默认行为是 `INCLUDING ALL OVERWRITING OPTIONS`，即 B 表会继承 A 表的所有定义和设置，但是会覆盖掉 WITH 参数。

CREATE VIEW

最近更新时间：2023-06-21 15:21:45

用户可以使用 CREATE VIEW 语句创建视图。视图是一个虚拟表，基于某条 SELECT 语句。视图可以用在定义新的虚拟数据源（类型转换、列变换和虚拟列等），拆分过长代码等场景。

语法

```
CREATE VIEW 视图名 AS
SELECT 子句
```

示例一

创建一个名为 MyView 的视图：

```
CREATE VIEW MyView AS
SELECT s1.time_, s1.client_ip, s1.uri, s1.protocol_version, s2.status_code, s2.date_
FROM KafkaSource1 AS s1, KafkaSource2 AS s2
WHERE s1.time_ = s2.time_ AND s1.client_ip = s2.client_ip;
```

示例二

在计算中由于数据量较大、函数方法类型匹配要求等原因，必须使用 TINYINT、SMALLINT 等类型。当 Kafka 等输入类型不符合需求时，可通过 CREATE VIEW 语句配合 CAST() 类型转换函数（参见 类型转换函数），实现定义虚拟视图作为新的数据源。
通过定义一个名为 KafkaSource2 的视图，实现将 KafkaSource1 数据源中的 BIGINT 类型的 status_code 列转为 VARCHAR 类型的列，命令如下：

```
CREATE VIEW KafkaSource2 AS
SELECT
  `time_`,
  `client_ip`,
  `method`,
  CAST(`status_code` AS VARCHAR) AS status_code,
FROM KafkaSource1;
```

注意

- 不当的数据转换 CAST() 可能会导致精度损失，例如由 BIGINT 转为 INTEGER 或 TINYINT 等，请谨慎使用。
- 如果需要进行字符串（VARCHAR）和时间戳（TIMESTAMP）之间的类型转换，可参见 时间相关函数 中 TO_TIMESTAMP、DATE_FORMAT 等函数。

CREATE FUNCTION

最近更新时间：2023-06-21 15:21:46

对于 SQL 作业，用户可以上传 自定义程序包，然后在作业分析开发页的参数设置中，引用该程序包。目前支持从本地上传，也可以引用账户下现有 COS 存储中的资源（仅限相同地域）。
这里的程序包既可以用来扩展 Connector 的功能，也可以创建自定义函数（UDF）。

语法

目前流计算 Oceanus 支持 Java 和 Scala 两种语言编写的程序包。当用户上传了自定义程序包后，在界面上关联后即可用下面的 CREATE FUNCTION 语句来声明：

```
CREATE TEMPORARY SYSTEM FUNCTION 函数名
  AS '函数类全名' [LANGUAGE JAVA|SCALA]
```

其中的**函数名**可以自行定义，但不要与现有的冲突。**函数类全名**为 Java 或 Scala 类的类全名（例如 `'com.example.flink.MyCustomFunction'` ）。

命名覆盖

如果存在系统内置的同名函数时，用户使用上述语法创建的 UDF 会覆盖系统内置的函数。因此除非有意改变系统函数的功能，请不要创建与系统内置函数同名的自定义函数。

函数类型

目前 Flink 支持下面多种函数定义。

标量函数（Scalar Function）

标量函数简称 UDF，作用是将一个值转换为另一个值（一对一），例如系统内置的 `SUBSTRING` 、 `REPLACE` 等字符串操作函数，都属于标量函数。

表函数（Table Function）

表函数简称 UDTF，作用是将一个值转为表中的一行数据（一变多），这样可以在后续 JOIN 操作中作为右表。

聚合函数（Aggregate Function）

聚合函数简称 UDAGG，作用是将多行数据的一组值，聚合为一个最终值（多变一），例如系统内置的 `MAX` 、 `MIN` 、 `AVG` 等都属于聚合函数。

表聚合函数（Table Aggregate Function）

表聚合函数的作用是将多行数据的一组值，聚合为新的多行数据（多对多）。

异步表函数（Async Table Function）

异步表函数可作为一种特殊的数据源，例如可以通过它来对接外部的数据库、数据存储。

UDF 开发指南

由于 Flink 不同版本间 API 和文档迭代频繁，可参考 Flink 官方文档中的 UDF 开发指南 。目前流计算 Oceanus 兼容开源版的 Flink 1.11 版本 API。

DML 数据操作语句
查询语句

最近更新时间：2023-06-21 15:21:46

SELECT FROM

SELECT 不能单独使用，必须配合 `CREATE VIEW … AS` 或 INSERT INTO 使用，否则系统会提示没有合适的 Operator。

语法

```
SELECT 以逗号分隔的需要选中的字段
FROM 数据源或视图
WHERE 过滤条件
其他子查询
```

示例

```
SELECT s1.time_, s1.client_ip, s1.uri, s1.protocol_version, s2.status_code, s2.date_
FROM KafkaSource1 AS s1, KafkaSource2 AS s2
WHERE s1.time_ = s2.time_ AND s1.client_ip = s2.client_ip;
```

WHERE

WHERE 用来过滤查询条件（谓词），多个并列的条件可以用 AND、OR 来连接。在与外部数据库 TencentDB 的表 JOIN 时，条件的连接只支持 AND。如需使用 OR 的功能，请参见 UNION ALL。

HAVING

HAVING 用于过滤 GROUP BY 之后的结果。**WHERE 在 GROUP BY 之前过滤，而 HAVING 在 GROUP BY 分组之后过滤。**

```
SELECT SUM(amount)
FROM Orders
WHERE price > 10
GROUP BY users
HAVING SUM(amount) > 50
```

GROUP BY

在流计算 Oceanus 中，GROUP BY 用于对结果进行分组聚合。目前有含时间窗口（Window）类型的 GROUP BY 和不含窗口的 GROUP BY（即持续查询）。

- 含时间窗口（Window）类型的 GROUP BY 不会更新之前的结果，因而会产生 Append（Tuple）类型的数据流，只允许写入不带主键的 MySQL、PostgreSQL、Kafka、Elasticsearch 等数据目的。
- 不含窗口的 GROUP BY 会**更新**之前发出的记录，因而会产生 Upsert 类型的数据流，只允许写入含主键的云数据库 MySQL、PostgreSQL 数据目的表（Sink）、Elasticsearch 等，且主键必须与 GROUP BY 语句里 Upsert 字段一致。

含时间窗口的 GROUP BY

本示例定义了一个包含时间窗口的 GROUP BY 查询语句。关于时间窗口函数的使用方法，请参见 时间窗口函数 。

```
SELECT user, SUM(amount)
FROM Orders
GROUP BY TUMBLE(rowtime, INTERVAL '1' DAY), user
```

- 在 Event Time 时间模式下，使用 `WATERMARK FOR` 定义时间戳字段，那么 TUMBLE 窗口函数的第一个参数必须为该字段。HOP 和 SESSION 窗口同理。

版权所有：腾讯云计算（北京）有限责任公司

- 在 Processing Time 时间模式下，TUMBLE 窗口函数的第一个参数必须为表 `proctime()` 声明的字段。HOP 和 SESSION 窗口同理。

不含时间窗口的 GROUP BY（持续查询）

本示例定义了一个不包含时间窗口的 GROUP BY 查询语句，这种查询叫做持续查询，因为它会根据每条新到的数据来计算并决定是否更新之前发出的结果，因而会产生一个 Upsert 流。

```
SELECT a, SUM(b) as d
FROM Orders
GROUP BY a
```

> **注意**
> 这种方式可能会因为 key 的数量过大或数据过多而发生内存溢出。因而请谨慎设置对象超时时间，不要过长。

JOIN

目前流计算 Oceanus 系统只支持等值连接 Equi-JOIN，即 JOIN 条件内包含至少一条令左右表某字段相等的过滤条件。

流和流的 Inner Equi-JOIN

目前流和流的连接也分为两种：含时间范围的和不含时间范围的。前者会生成 Append（Tuple）类型的流，而后者会生成 Upsert 类型的流。

含时间范围的 Inner JOIN

含时间范围的 JOIN 也称为 **Interval Join**，它的 WHERE 条件中需要至少一个等值连接的 JOIN 条件和一个指定的时间范围。这个时间范围可以用 <、<=、>=、> 或 BETWEEN … AND 等来表示。

```
ltime = rtime
ltime >= rtime AND ltime < rtime + INTERVAL '10' MINUTE
ltime BETWEEN rtime - INTERVAL '10' SECOND AND rtime + INTERVAL '5' SECOND
```

示例：

```
SELECT *
FROM Orders o, Shipments s
WHERE o.id = s.orderId AND
o.ordertime BETWEEN s.shiptime - INTERVAL '4' HOUR AND s.shiptime
```

不含时间范围的 Inner JOIN

不含时间范围的流-流 JOIN 的特点是只要求有至少一个等值连接，而不要求指定时间范围。也就是说，它会将历史以来所有的活跃数据参与计算（可以通过指定超时时间来去除不活跃的元素）。

> **注意**
> - 可能会导致非常大的内存占用，需要谨慎使用。通常需要设置合适的对象超时时间，以及时清除失活的对象。
> - 这种查询会产生一个 Upsert 流，只能使用含主键的 MySQL、PostgreSQL、Elasticsearch 等数据目的（Sink）来接收数据。

示例：

```
SELECT *
FROM Orders INNER JOIN Product ON Orders.productId = Product.id
```

Outer Equi-JOIN

Outer Equi-JOIN 会产生 Upsert 流，因此需要注意数据目的（Sink）必须可以接受 Upsert 数据流，例如 MySQL、PostgreSQL、Elasticsearch 等。

> **注意**
> 由于不对 JOIN 的顺序做优化，JOIN 操作会依次按照 FROM 语句后定义的各表进行，因此可能会产生非常大的状态压力，甚至可能执行失败。

```
SELECT *
FROM Orders LEFT JOIN Product ON Orders.productId = Product.id

SELECT *
FROM Orders RIGHT JOIN Product ON Orders.productId = Product.id

SELECT *
FROM Orders FULL OUTER JOIN Product ON Orders.productId = Product.id
```

维表 JOIN

流计算 Oceanus 也支持流与 MySQL、PostgreSQL 数据库中维表（Temporal table，即随时间不断变化的表）的 JOIN，语法同上面介绍的完全一致，只是要求维表必须放在 JOIN 条件的右表。

```
SELECT
  o.amout, o.currency, r.rate, o.amount * r.rate
FROM
  Orders AS o
  JOIN LatestRates FOR SYSTEM_TIME AS OF o.proctime AS r
  ON r.currency = o.currency
```

> **注意**
> 一定要加入 `FOR SYSTEM_TIME AS OF` 语句，否则虽然仍然可以执行 JOIN，但是只会全量读取一次数据库，结果可能不符合预期。

与 UDTF 的 JOIN

如果用户自定义了表函数（UDTF，即 User-Defined Table Function），则可以将表函数作为 JOIN 的右表，语法与普通 JOIN 类似，只是需要加上 `LATERAL TABLE( )` 关键字，将 UDTF 包围起来。

Inner UDTF JOIN

```
SELECT users, tag
FROM Orders, LATERAL TABLE(unnest_udtf(tags)) t AS tag
```

Left Outer UDTF JOIN

```
SELECT users, tag
FROM Orders LEFT JOIN LATERAL TABLE(unnest_udtf(tags)) t AS tag ON TRUE
```

> **注意**
> 目前 Left Outer UDTF JOIN 只支持 `ON TRUE` 语法，类似于 CROSS JOIN。

与数组进行 JOIN

流计算 Oceanus 系统支持和一个已定义的数组对象（可通过 值构造函数 构造数组对象 ARRAY）做 JOIN 操作。
示例：假设 tags 是一个已定义的数组。

```
SELECT users, tag
FROM Orders CROSS JOIN UNNEST(tags) AS t (tag)
```

UNION ALL

UNION ALL 用来合并两个查询的结果。

```
SELECT *
FROM (
    (SELECT user FROM Orders WHERE a % 2 = 0)
  UNION ALL
    (SELECT user FROM Orders WHERE b = 0)
)
```

目前流计算 Oceanus 只支持 UNION ALL 而暂不支持 UNION，即不会对相同的行进行去重操作。
如果需要实现去重以达到 UNION 的效果，请配合 DISTINCT 使用。**DISTINCT 会让结果从 Append（Tuple）流变为 Upsert 流，因而只能使用含主键的 MySQL、PostgreSQL、Elasticsearch 数据目的（Sink）来接收数据。**

OVER Window 聚合

如果需要对数据流做基于滑动窗口的聚合（不使用 GROUP BY 的聚合），那么可以使用 OVER 来进行滑动窗口的聚合操作。在 OVER 中可以指定 PARTITION、ORDER、窗口范围等。
下面的示例定义了一个滑动窗口聚合查询，统计一个大小为3的滑动窗口的交易总额（amount）。其中对之前的行使用 PRECEDING，而目前还不支持 FOLLOWING。
另外 ORDER BY 后面只允许一个时间戳字段，本例使用数据源中声明的 `proctime` 字段。

```
SELECT SUM(amount) OVER (
  PARTITION BY user
  ORDER BY proctime
  ROWS BETWEEN 2 PRECEDING AND CURRENT ROW)
FROM Orders
```

```
SELECT COUNT(amount) OVER w, SUM(amount) OVER w
FROM Orders
WINDOW w AS (
  PARTITION BY user
  ORDER BY proctime
  ROWS BETWEEN 2 PRECEDING AND CURRENT ROW)
```

ORDER BY

ORDER BY 用来对查询的结果做排序，默认是 ASC（升序排列），也可以显式指定 DESC（降序排列）。

> **注意**
> 要求第一个排序项必须是升序的时间列（Event Time 时间戳或 Processing Time 时间戳，即 PROCTIME），之后的排序项可以自由指定。

```
SELECT *
FROM Orders
ORDER BY `orderTime`, `username` DESC, `userId` ASC
```

DISTINCT

DISTINCT 用来对查询结果进行去重，它必须放在 SELECT 后面。

```
SELECT DISTINCT users FROM Orders
```

- DISTINCT 会产生一个 Upsert 流，因而只有 Upsert 类型的数据目的（Sink）才可以接收其结果。而且长时间查询可能会导致内存占用过大，请谨慎使用。

- 通过设置合适的对象过期时间，可以及时清除失活对象来节省内存。

IN

可以使用 IN 关键字对判断指定集合（例如子查询）中是否存在某元素。

> **注意**
> 该操作的内存压力可能较大，请谨慎使用。

```
SELECT user, amount
FROM Orders
WHERE product IN (
    SELECT product FROM NewProducts
)
```

EXISTS

如果 EXISTS 后面子查询的结果大于或等于一行（存在数据），则返回 `true` 。

> **注意**
> 该操作的内存压力可能较大，请谨慎使用。

```
SELECT user, amount
FROM Orders
WHERE product EXISTS (
    SELECT product FROM NewProducts
)
```

ORDER BY

ORDER BY 可以对某个字段进行排序后输出。

> **注意**
> 该操作的内存压力可能较大，请谨慎使用。

```
SELECT *
FROM Orders
ORDER BY orderTime
```

Grouping Sets、Rollup、Cube

对于 Grouping Sets、Rollup、Cube 操作，产生的是一个 Upsert 流，因此只有 Upsert 类型的数据目的才可以接收其结果。

```
SELECT SUM(amount)
FROM Orders
GROUP BY GROUPING SETS ((user), (product))
```

模式匹配

目前流计算 Oceanus 支持 `MATCH_RECOGNIZE` 语句对一条输入流进行模式匹配，用户可以用 SQL 语句来描述 CEP（复杂事件处理）的逻辑。

```
SELECT T.aid, T.bid, T.cid
FROM MyTable
MATCH_RECOGNIZE (
```

```
  PARTITION BY userid
  ORDER BY proctime
  MEASURES
    A.id AS aid,
    B.id AS bid,
    C.id AS cid
  PATTERN (A B C)
  DEFINE
    A AS name = 'a',
    B AS name = 'b',
    C AS name = 'c'
) AS T
```

上述例子定义了 A、B、C 三个事件，分别表示 name 字段等于 a、b、c 时的事件。PATTERN 表示事件触发的规则，即 A、B、C 三个事件连续出现时触发。MEASURES 指定了事件触发后的输出格式。
更多内容，可参见 Flink 官方文档的 表中的模式检测 。

Top-N

Top-N 查询可以在一批流数据中，不断输出当前最新的前 N 大或者前 N 小的记录，因此输出类型为 Upsert 流，需要写入支持 Upsert 数据流的数据目的（Sink）。
Top-N 语法详情请参见 Flink 官方文档的 Top-N 查询 。

相邻数据去重

有时候上游输入的数据可能会包含连续的重复值，该查询语句可以将重复的数据删除，只保留一个。去重的语法，可参见 Flink 官方文档的 去重 。

INSERT 语句

最近更新时间：2023-06-21 15:21:46

INSERT INTO

INSERT INTO 语句必须和 SELECT 子查询联用，SELECT 的数据会写入到指定的数据目的表（Table Sink）中。

语法

```
INSERT INTO 数据目的表
SELECT 子句
```

示例

将 SELECT 查询的结果插入名为 KafkaSink1 的数据目的表（Sink）。

```
INSERT INTO KafkaSink1
SELECT s1.time_, s1.client_ip, s1.uri, s1.protocol_version, s2.status_code, s2.date_
FROM KafkaSource1 AS s1, KafkaSource2 AS s2
WHERE s1.time_ = s2.time_ AND s1.client_ip = s2.client_ip;
```

Table Sink 注意事项

选择合适的 Connector 程序包

- 如果在 WITH 参数里指定了某个 Sink，那么请务必勾选相应的【内置 Connector】，或自行上传相应的 Connector 程序包。
- 如果缺少符合条件的 Connector 程序包，作业启动时会抛出 `org.apache.flink.table.api.ValidationException: Could not find any factory` 异常信息。
- 对于读写 Kafka 的场景，推荐使用不带版本号的 `flink-connector-kafka` 程序包，并将 `connector.version` 参数设置为 `universal`，以获得最新的功能适配。**不建议**选择 `flink-connector-kafka-0.11` 等带版本号的旧版程序包。

排除计算列

对于 INSERT INTO 的数据目的表，**计算列**是不考虑在内的。例如，某个 Sink 的定义如下，那么在 INSERT INTO MySink 后的 SELECT 语句，必须包含 a (VARCHAR) 和 b (BIGINT) 两个字段，且不允许加入 c 字段，因为 c 是虚拟的计算列。

```
CREATE TABLE MySink (
   a VARCHAR,
   b BIGINT,
   c AS PROCTIME()
) WITH ( ... ... );
```

Tuple 和 Upsert 数据流的区别

确保 Sink Table 定义了合适的 WITH 参数。例如有些 Connector 只支持作为数据源，不支持作为数据目的；还有的只支持 Tuple 类型的数据流，不支持 Upsert 数据流等。

MySQL CDC 多 Source 复用

最近更新时间：2025-04-18 10:16:53

目前使用 Flink CDC Connector 做数据同步时，每个表都需要建立一个数据库连接，在多表、整库同步等场景下，对数据库实例的压力非常大，Oceanus 引入了多 Source 复用的优化来解决这种问题。

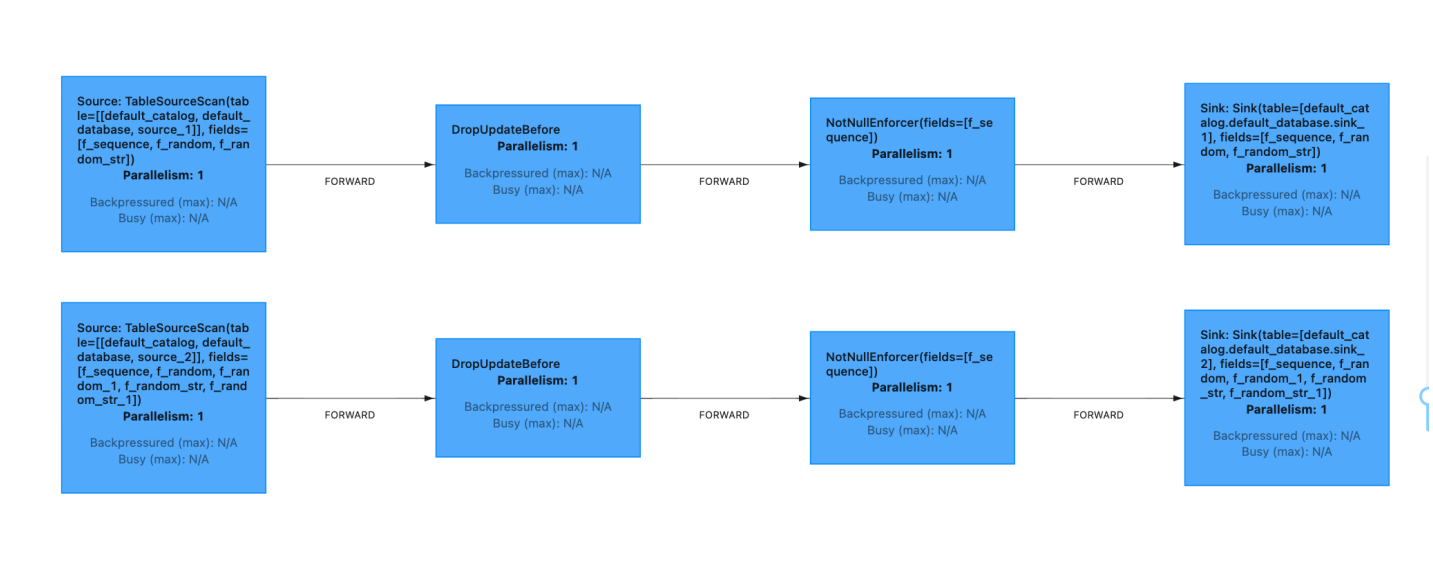
功能介绍

例如以下作业：

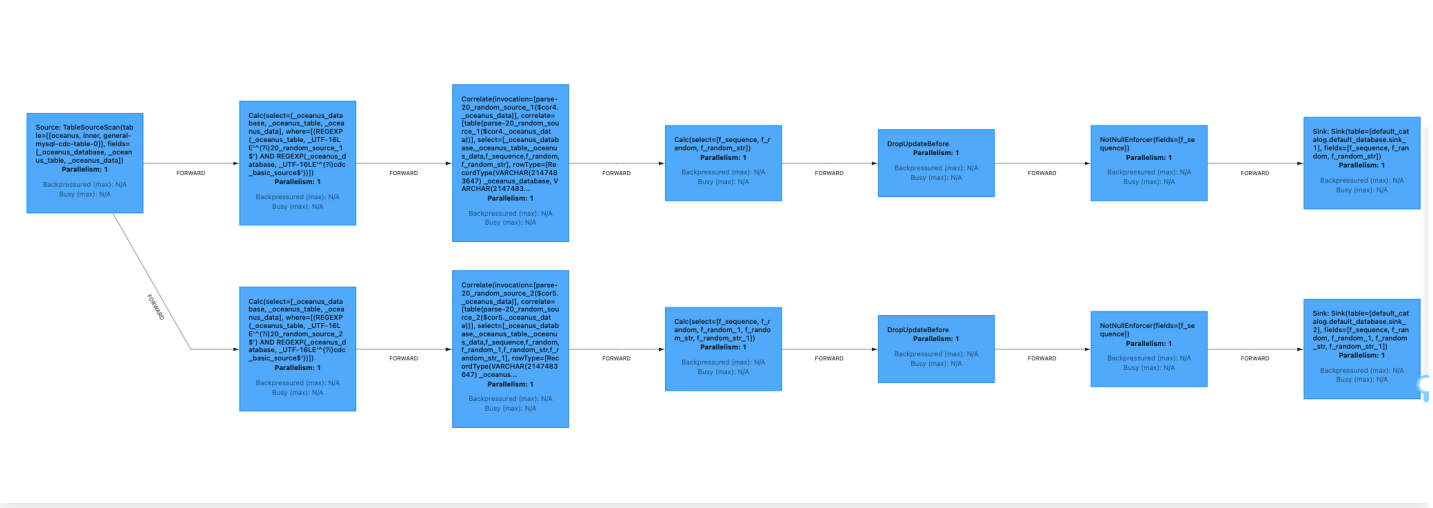
```
CREATE TABLE `source_1`
(
    `f_sequence`   INT,
    `f_random`     INT,
    `f_random_str` VARCHAR,
    PRIMARY KEY (`f_sequence`) NOT ENFORCED
) WITH (
      'connector' = 'mysql-cdc' ,
      'hostname' = 'ip1',
      'port' = '3306',
      'username' = 'xxx',
      'password' = 'xxx',
      'database-name' = 'db1',
      'table-name' = 'source_1'
);

CREATE TABLE `source_2`
(
    `f_sequence`   INT,
    `f_random`     INT,
    `f_random_1`     INT,
    `f_random_str` VARCHAR,
    `f_random_str_1` VARCHAR,
    PRIMARY KEY (`f_sequence`) NOT ENFORCED
) WITH (
      'connector' = 'mysql-cdc' ,
      'hostname' = 'ip1',
      'port' = '3306',
      'username' = 'xxx',
      'password' = 'xxx',
      'database-name' = 'db2',
      'table-name' = 'source_2'
);


CREATE TABLE `sink_1`
(
    `f_sequence`   INT,
    `f_random`     INT,
    `f_random_str` VARCHAR,
    PRIMARY KEY (`f_sequence`) NOT ENFORCED
) WITH (
      'connector' = 'logger'
);

CREATE TABLE `sink_2`
(
    `f_sequence`   INT,
```

```
    `f_random`     INT,
    `f_random_1`     INT,
    `f_random_str` VARCHAR,
    `f_random_str_1` VARCHAR,
    PRIMARY KEY (`f_sequence`) NOT ENFORCED
) WITH (
      'connector' = 'logger'
);

insert into sink_1 select * from source_1;
insert into sink_2 select * from source_2;
```

对于来自相同 db 实例的表同步，Flink 会生成两条 pipeline（如下图），每个 CDC 的 Source 都会跟 Mysql 建立一个数据库连接，在表的数量非常多的时候，作业与 Mysql 的连接会非常多，DB 端相应的压力也会非常大。



我们可以看到在开启了 Source 复用功能之后，读相同 DB 实例的 Mysql CDC Source 会合并成一个 Source（如下图），可以有效的降低对 DB 的链接压力。并且在 Mysql 增量同步阶段，多个 Source 只需要读取一份 binlog 数据即可，避免了多个 Source 重复拉取 binlog 的问题。



如何开启 CDC Source 复用功能

在 sql 作业的开头，通过 set 命令指定开启 mysql cdc source 合并功能即可。

```
SET table.optimizer.mysql-cdc-source.merge.enabled=true;
```

相关 set 参数说明

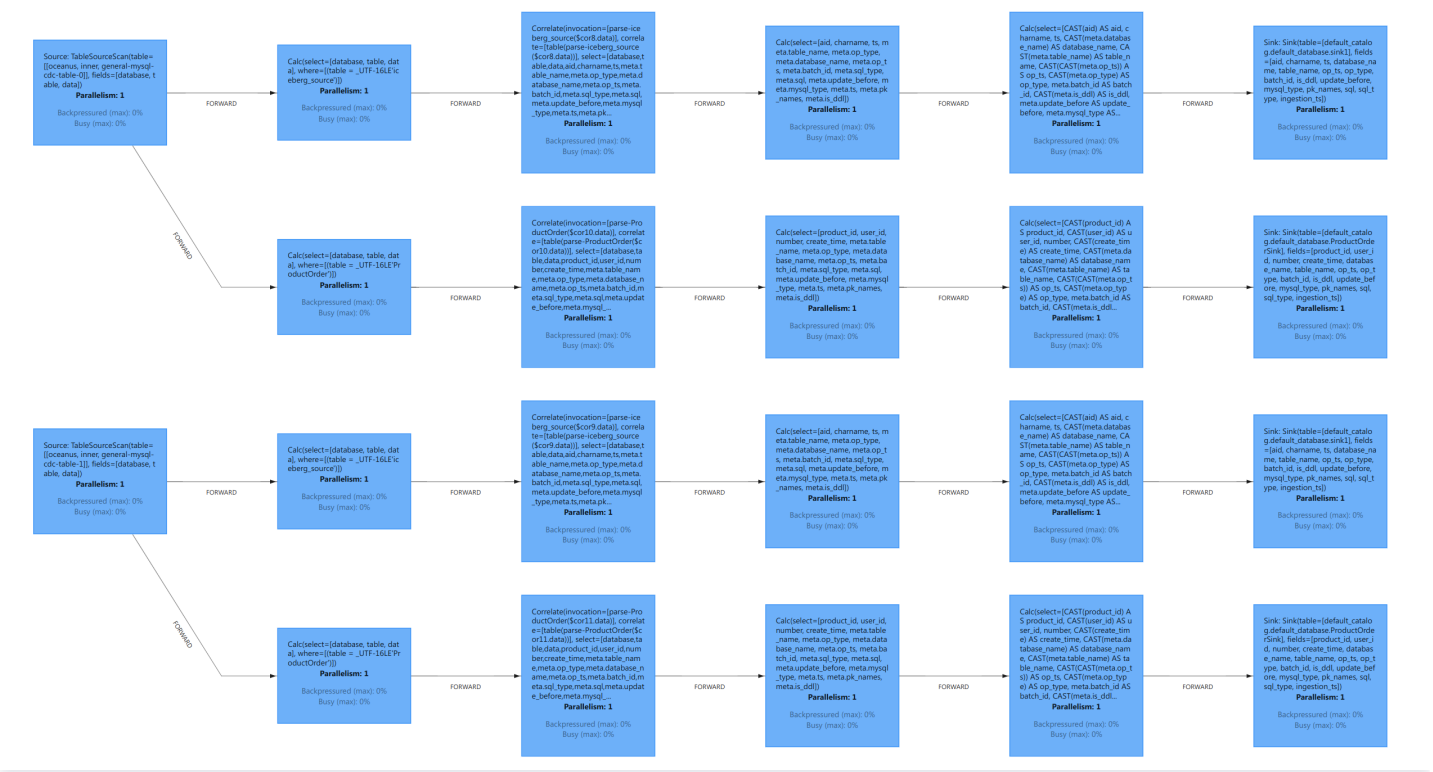
参数	默认值	含义
table.optimizer.mysql-cdc-source.merge.enabled	false	是否开启 mysql cdc source 复用功能，开启后会自动尝试 merge 同一作业内读相同 DB 的 mysql cdc source 为同一个 source。
table.optimizer.mysql-cdc-source.merge.default-group.splits	1	开启 mysql cdc source 合并功能后，相同 DB 实例 mysql cdc source 会合并成的 source 数量。对于表非常多的场景，当合并成一个 source 不能满足性能要求时，可以通过 set 命令调高这个值，oceanus 会自动将 cdc source 尽量均匀的划分成指定的 group splits 数量。

注意：
CDC Source 复用功能只能合并除 database-name、table-name 和 server-id 外其它 with 参数均相同的源表。

分组合并功能示例如下，假如 source_1 、source_2、source_3、source_4是来自同一个 DB 实例的 Mysql CDC Source 表（此处省略 source 和 sink 表定义），我们可以用如下的参数来配置 source 复用功能。

```
SET table.optimizer.mysql-cdc-source.merge.enabled=true;
SET table.optimizer.mysql-cdc-source.merge.default-group.splits=2;

insert into sink_1 select * from source_1;
insert into sink_2 select * from source_2;
insert into sink_3 select * from source_3;
insert into sink_4 select * from source_4;
```

最终的运行拓扑图如下，我们可以看到4个 cdc source 自动的划分成2个分组。



整库同步（SQL）能力

最近更新时间：2024-11-01 10:52:41

> **注意：**
> Oceanus 的整库同步（SQL）能力后续不再更新演进，如有整库同步场景或需求，请使用 WeData 产品中的 数据集成模块 或者 数据集成 产品。

适用场景

在实时湖仓的数据链路建设过程中，将传统关系型数据库（如 MySQL 等）中的**整库数据**低延迟、高吞吐的同步到下游的 OLAP 数据库（如 Doris、ClickHouse）或者归档到对应的文件系统中（如 HDFS、Hive 表），是一个非常普遍和强烈的需求。
Oceanus 提供了非常方便的 CDAS（CREATE DATABASE AS）SQL 语法，来支持将数据库中的整库数据全量导入、增量实时同步写入到用户指定的下游系统中。

语法说明

```
CREATE DATABASE IF NOT EXISTS <target_database>
[COMMENT database_comment]
[WITH (key1=val1, key2=val2, ...)] -- 指明写入目标库的参数
AS DATABASE <source_catalog>.<source_database>  -- source_database 是被同步的源数据库
INCLUDING { ALL TABLES | TABLE 'table_name' }
-- INCLUDING ALL TABLES 表示同步数据库中的所有表
-- INCLUDING TABLE 'table' 表示同步数据库中特定的表，支持正则表达式，如 'order_.*';
-- 同步多张表时，可以写成 INCLUDING TABLE 'tableA|tableB|tableC'的格式
[EXCLUDING TABLE 'table_name']
-- EXCLUDING TABLE 'table' 表示不同步数据库中特定的表，支持正则表达式，如 'order_.*';
-- 排除多张表时，可以写成 EXCLUDING TABLE 'tableA|tableB|tableC'的格式
[/*+ `OPTIONS`('key1'='val1', 'key2'='val2', ... ) */]
-- （可选，指明读取source的参数，如指定source serverId的范围，解析debezium时间戳字段类型等）
```

参数说明：

参数	解释
target_database	待写入的目标数据库名
database_comment	待写入的数据库注释
WITH参数	指明写入目标库的参数，目前会被翻译成下游 sink 表的描述参数
`<source_catalog>.<source_database>`	声明源 catalog 中需要同步的数据库
INCLUDING ALL TABLES	同步源库中的所有表
INCLUDING TABLE	同步数据库中特定的表，支持正则表达式，如 'order_.*'； 同步多张表时，可以写成 INCLUDING TABLE 'tableA|tableB'格式
EXCLUDING TABLE	表示不同步数据库中特定的表，支持正则表达式，如 'order_.*'; 排除多张表时，可以写成 EXCLUDING TABLE 'tableA|tableB'格式
OPTIONS	可选，指明读取 Source 时覆盖的参数，如指定 source serverId 的范围等

> **注意**
> 第三行 [WITH (key1=val1, key2=val2, ..)] 指明写入目标库的参数中， value 值支持将 Source 表的表名进行变量替换，使用方法是使用占位符 $tableName。
> 如下整库同步到 Doris 的示例中，写入每一个 sink 的表中 $tableName 会替换为相对应 MySQL 库中源表表名。

```
create catalog my_mysql with(...);
create database if not exists sink_db
with (
  'connector' = 'doris',
  'table.identifier' = 'db1.$tableName_doris'
  ...
)
including all tables
/*+ `OPTIONS`('server-time-zone' = 'Asia/Shanghai') */; -- 声明解析timestamp字段的时区类型
```

状态无损的加表能力（仅限 MySQL 源表）

如果您的 MySQL 数据库有较多的新增表需求，希望 Flink 可以支持**新增表的数据同步**，且**不影响现有各表的同步状态**，那么您可以使用 SET 和 OPTIONS 选项来开启该功能（本示例还开启了 MySQL CDC 的多 Source 复用 、自动拆分最后一个大分片 等高级能力，请按需使用）：

```
-- 必须设置 table.optimizer.deterministic-operator-uid-for-cdas=true 才可以启用无损加表能力
SET table.optimizer.deterministic-operator-uid-for-cdas=true;

-- 可选：开启多 Source 复用，降低资源占用，提升稳定性
SET table.optimizer.mysql-cdc-source.merge.enabled=true;

-- 请替换成实际的 CDAS 语句，但至少要保留 'scan.newly-added-table.enabled' = 'true' 的 OPTIONS 选项
create catalog my_mysql with(...);
create database if not exists sink_db
with (
  ...
)
including all tables
/*+ `OPTIONS`('scan.newly-added-table.enabled' = 'true','scan.lastchunk.optimize.enable' = 'true') */;
```

注意：

1. SET 参数、scan.newly-added-table.enabled 的 OPTIONS 需要**在作业首次启动时**就加上；否则需要丢弃现有状态，重新全量同步一次数据，后续才可无损加表。
2. 只有**当作业启动**时，才可以感知新增表的存在。因此每当有加表操作，请对作业做一个快照，然后从该快照恢复运行，此时新表才会开始同步。
3. 当表的数量多时，我们建议使用 SET table.optimizer.mysql-cdc-source.merge.enabled=true; 语句开启多 Source 复用能力，提升整体稳定性。
4. 当表有持续的大流量写入时，我们建议开启 'scan.lastchunk.optimize.enable' = 'true' 参数，以避免最后一个分片过大导致 TaskManager OOM。

MySQL-CDC 元数据字段读取

- MySQL CDC connector 除支持提取物理表的字段外，还支持了 元数据字段列表 可以提取。整库同步功能中也提供了一个 options 配置参数，可以用来控制同步所需的元数据字段。

参数	解释
oceanus.source.include-metadata.fields	需要同步的 source 表的元字段，格式为 'table_name:table_name;meta.batch_id:batch_id', 元数据字段定义通过分号;分隔，每个元数据字段格式为 metadataColumn:alias， 第一部分为实际对应的元数据 column，第二部分为重命名后的值。

- 注意：元数据字段会按照声明的顺序，追加到源表之后。

  使用实例：

```
SET table.optimizer.mysql-cdc-source.merge.enabled=true;
```

```
create catalog my_mysql with (
    'type' = 'jdbc',
    'default-database' = 'test',
    'username' = 'xx',
    'password' = 'xxx',
    'base-url' = 'xxx'
);

create database if not exists print_sink_db
comment 'test_sink'
with (
'connector' = 'print',
'print-identifier' = '$tableName')
as database `my_mysql`.`test`
including all tables
/*+ `OPTIONS`('server-time-zone' = 'Asia/Shanghai',
    'oceanus.source.include-
metadata.fields'='table_name:table_name;database_name:database_name;op_ts:op_ts;meta.table_name:meta_
table_name;meta.database_name:meta_database_name;meta.op_ts:meta_op_ts;meta.op_type:meta_op_type;meta
.batch_id:meta_batch_id;meta.is_ddl:meta_id_ddl;meta.mysql_type:meta_mysql_type;meta.update_before:me
ta_update_before;meta.pk_names:meta_pk_names;meta.sql:meta_sql;meta.sql_type:meta_sql_type;meta.ts:me
ta_ts')
    */;
```

使用方法

1. 首先注册 MySQL 的 Catalog，作为待同步的数据源表，示例如下：

```
create catalog my_mysql with (
 'type' = 'jdbc',
 'default-database' = 'test',
 'username' = 'XXX',
 'password' = 'XXX',
 'base-url' = 'jdbc:mysql://ip:port'
 -- 'jdbc.properties.tinyInt1isBit' = 'false'  -- jdbc参数,是否把tinyInt识别为bool. 默认为true.
 -- 如果表字段包含tinyint(1), 建议把 jdbc.properties.tinyInt1isBit 设置为false.
);
```

2. 对于不支持自动建表的下游系统，需要事先保证在下游系统中建立和上游 Mysql 表中一一对应的源表。
3. 使用整库同步语法指定需要同步的同步表，其中写入目标库的参数现在只支持填入下游 connector 的必要参数。
4. 可以通过 `jdbc.properties.*` 传入对应的 jdbc 参数。

同步到 Hudi 示例

```
SET table.optimizer.mysql-cdc-source.merge.enabled=true;

create catalog my_mysql with (
    'type' = 'jdbc',
    'default-database' = 'test',
    'username' = 'root',
    'password' = 'XXX',
    'base-url' = 'jdbc:mysql://ip:port'
);

create database if not exists sink_db
comment 'test_sink'
with (
```

```
'connector' = 'hudi',
'path' = 'hdfs://namenode:8020/user/hive/warehouse/$tableName_mor'
) as database `my_mysql`.`trade_log`
including all tables
/*+ `OPTIONS`('server-time-zone' = 'Asia/Shanghai') */; -- 声明解析timestamp字段的时区类型
```

同步到 ClickHouse 示例

```
SET table.optimizer.mysql-cdc-source.merge.enabled=true;

create catalog my_mysql with (
    'type' = 'jdbc',
    'default-database' = 'test',
    'username' = 'root',
    'password' = 'XXX',
    'base-url' = 'jdbc:mysql://ip:port'
);

create database if not exists sink_db
comment 'test_sink'  with (
'connector' = 'clickhouse://172.11.11.11:8123',
 -- 如果ClickHouse集群未配置账号密码可以不指定
 --'username' = 'root',                          -- ClickHouse集群用户名
 --'password' = 'root',                          -- ClickHouse集群的密码
 'database-name' = 'testdb',                     -- 数据写入目的数据库
 'table-name' = 'test_table1',                   -- 数据写入目的数据表
 'sink.batch-size' = '1000',
 'table.collapsing.field' = 'Sign'
) as database my_mysql.test_db
including all tables
/*+ OPTIONS('server-time-zone' = 'Asia/Shanghai') */;
```

同步到 Doris 示例

```
SET table.optimizer.mysql-cdc-source.merge.enabled=true;

create catalog my_mysql with (
    'type' = 'jdbc',
    'default-database' = 'test',
    'username' = 'root',
    'password' = 'XXX',
    'base-url' = 'jdbc:mysql://ip:port'
);

create database if not exists sink_db
comment 'test_sink'  with (
'connector' = 'doris',
'table.identifier' = 'trade_log.$tableName',
'username' = 'admin',
'password' = 'xxx',
'sink.batch.size' = '500',
'sink.batch.interval' = '1s',
'fenodes' = 'ip:port'
) as database `my_mysql`.`trade_log`
including all tables
/*+ `OPTIONS`('server-time-zone' = 'Asia/Shanghai') */; -- 声明解析timestamp字段的时区类型
```

同步到 Hive 示例

```
SET table.optimizer.mysql-cdc-source.merge.enabled=true;

create catalog my_mysql with (
    'type' = 'jdbc',
    'default-database' = 'test',
    'username' = 'root',
    'password' = 'XXX',
    'base-url' = 'jdbc:mysql://ip:port'
);

create database if not exists sink_db
comment 'test_sink'  with (
'connector' = 'hive',
'hive-version' = '2.3.6',
'hive-database' = 'test_100',
'hive-table' = '$tableName',
'sink.partition-commit.policy.kind' = 'metastore'
) as database `my_mysql`.`trade_log`
including all tables
/*+ `OPTIONS`('append-mode' = 'true', 'server-time-zone' = 'Asia/Shanghai') */;
-- 因为hive sink不支持变更数据，此处的hint会把原始cdc的变更数据转成成append流下发
```

作业运行拓扑图展开后如下：

同步到 Hive 自动建表示例:

- 作业需要提前配置连接 Hive 服务的 jar 包，详细配置可以参考 获取 Hive 连接配置 jar 包 。

```
SET table.optimizer.mysql-cdc-source.merge.enabled=true;

-- 注册mysql的catalog
create catalog my_mysql with (
```

```
    'type' = 'jdbc',
    'default-database' = 'test',
    'username' = 'root',
    'password' = 'XXX',
    'base-url' = 'jdbc:mysql://ip:port'
);

-- 注册hive端catalog
create catalog my_hive with (
    'type' = 'hive',
    'default-database' = 'default',
    'hive-version' = '2.3.5'
);

create database if not exists `my_hive`.`trade_log`
as database `my_mysql`.`trade_log`
including all tables
/*+ `OPTIONS`('append-mode' = 'true', 'server-time-zone' = 'Asia/Shanghai') */;
-- 因为hive sink不支持变更数据，此处的hint会把原始cdc的变更数据转成成append流下发
```

同步到 DLC 示例

```
SET  table.optimizer.mysql-cdc-source.merge.enabled = true;
SET table.optimizer.deterministic-operator-uid-for-cdas=true;

-- 注册mysql的catalog
create catalog `my_mysql` with (
  'type' = 'jdbc',
  'default-database' = 'testdb', -- 数据库名
  'username' = '${username}', -- mysql用户名
  'password' = '${password}', -- mysql密码
  'base-url' = 'jdbc:mysql://ip:3306'
);

-- 整库同步
create database if not exists `my_dlc_database`
    comment 'test db sync'
with (
    'connector' = 'iceberg-inlong', -- 固定值
    'catalog-database' = 'test', -- DLC内表所在的数据库名称
    'catalog-table' = 'my_$tableName', -- DLC内表名称，$tableName会自动替换为要同步的表名，DLC表需要提前创建
    'default-database' = 'test', -- DLC内表所在的数据库名称
    'catalog-name' = 'HYBRIS', -- 固定值
    'catalog-impl' = 'org.apache.inlong.sort.iceberg.catalog.hybris.DlcWrappedHybrisCatalog', -- 固定值
    'qcloud.dlc.managed.account.uid' = '100026378089', -- 固定值，DLC管理账号的uid
    'qcloud.dlc.secret-id' = '${secret_id}', -- DLC 用户的secretId从
https://console.cloud.tencent.com/cam/capi中获取
    'qcloud.dlc.secret-key' = '${secret_key}', -- DLC 用户的secretKey，从
https://console.cloud.tencent.com/cam/capi中获取
    'qcloud.dlc.region' = 'ap-guangzhou', -- DLC 所在地域，必须填ap-地域格式
    'qcloud.dlc.jdbc.url' = 'jdbc:dlc:dlc.internal.tencentcloudapi.com?
task_type=SparkSQLTask&database_name=test&datasource_connection_name=DataLakeCatalog&region=ap-
guangzhou&data_engine_name=${engine_name}', -- DLC jdbc接入url，格式见
https://cloud.tencent.com/document/product/1342/61547
    'uri' = 'dlc.internal.tencentcloudapi.com', -- 固定值
    'user.appid' = '${appid}', -- DLC 用户的 appid
    'request.identity.token' = '100026378089' -- 固定值，DLC内表接入的token
)
```

```
    as database `my_mysql`.`test` including  table 'user.*'
    excluding table 'user_info_20230530|user_behavior_tmp'
/* +`OPTIONS`(
 'server-time-zone' = 'Asia/Shanghai',
 'scan.newly-added-table.enabled' = 'true',
 'scan.lastchunk.optimize.enable' = 'true'
*/
;
```

使用提醒

1. Flink 1.11 版本不支持整库同步（SQL）能力。
2. 目前只支持同步 MySQL 类型数据库作为整库同步的源表。
3. 目前同步到目标端时，除 Iceberg、Elasticsearch、Hudi 和 Hive（需要提前注册 Hive Catalog）作为目标表外，其它的目标端还不支持自动建表，需要事先在目标端中建立和 Mysql 库中数据表对应的表结构。
4. 推荐搭配 MySQL CDC Source 复用功能开启，一起使用，可以降低对数据库的压力。
5. CDAS 语法没有限制下游输出的类型，理论上可以同步到**任意的**下游类型。
6. 当同步的表的数量非常多的时候，flink 生成的单个 task 的 name 会非常长，导致 metric 系统占用大量的内存，影响作业稳定性，Oceanus 针对这种情况引入了 `pipeline.task-name-length` 参数来限制 taskName 的长度，能极大的提高作业稳定性和日志可读性。（适用 Flink-1.13 和 Flink-1.14 版本）。
   可以在作业的配置中生效：

   ```
   set pipeline.task-name-length=80;
   ```

上下游开发指南
消息队列 Kafka

最近更新时间：2025-05-14 10:12:42

介绍

Kafka 数据管道是流计算系统中最常用的数据源（Source）和数据目的（Sink）。用户可以把流数据导入到 Kafka 的某个 Topic 中，通过 Flink 算子进行处理后，输出到相同或不同 Kafka 实例的另一个 Topic。
Kafka 支持同一个 Topic 多分区读写，数据可以从多个分区读入，也可以写入到多个分区，以提供更高的吞吐量，减少数据倾斜和热点。

版本说明

Flink 版本	说明
1.11	支持
1.13	支持
1.14	支持
1.16	支持

使用范围

Kafka 支持用作数据源表（Source），也可以作为 Tuple 数据流的目的表（Sink）。
Kafka 还可以与 Debezium、Canal 等联用，对 MySQL、PostgreSQL 等传统数据库的变更进行捕获和订阅，然后 Flink 即可对这些变更事件进行进一步的处理。

DDL 定义

用作数据源（Source）

JSON 格式输入

```
CREATE TABLE `kafka_json_source_table` (
  `id` INT,
  `name` STRING
) WITH (
  -- 定义 Kafka 参数
  'connector' = 'kafka',
  'topic' = 'Data-Input',  -- 替换为您要消费的 Topic
  'scan.startup.mode' = 'latest-offset', -- 可以是 latest-offset / earliest-offset / specific-offsets /
group-offsets / timestamp 的任何一种
  'properties.bootstrap.servers' = '172.28.28.13:9092',  -- 替换为您的 Kafka 连接地址
  'properties.group.id' = 'testGroup', -- 必选参数，一定要指定 Group ID

  -- 定义数据格式 (JSON 格式)
  'format' = 'json',
  'json.fail-on-missing-field' = 'false',  -- 如果设置为 false，则遇到缺失字段不会报错。
  'json.ignore-parse-errors' = 'true'    -- 如果设置为 true，则忽略任何解析报错。
);
```

CSV 格式输入

```
CREATE TABLE `kafka_csv_source_table` (
  `id` INT,
  `name` STRING
```

```
) WITH (
  -- 定义 Kafka 参数
  'connector' = 'kafka',
  'topic' = 'Data-Input',  -- 替换为您要消费的 Topic
  'scan.startup.mode' = 'latest-offset', -- 可以是 latest-offset / earliest-offset / specific-offsets /
group-offsets / timestamp 的任何一种
  'properties.bootstrap.servers' = '172.28.28.13:9092',  -- 替换为您的 Kafka 连接地址
  'properties.group.id' = 'testGroup', -- 必选参数，一定要指定 Group ID

  -- 定义数据格式 (CSV 格式)
  'format' = 'csv'
);
```

Debezium 格式输入

```
CREATE TABLE `kafka_debezium_source_table` (
  `id` INT,
  `name` STRING
) WITH (
  -- 定义 Kafka 参数
  'connector' = 'kafka',
  'topic' = 'Data-Input',  -- 替换为您要消费的 Topic
  'scan.startup.mode' = 'latest-offset', -- 可以是 latest-offset / earliest-offset / specific-offsets /
group-offsets / timestamp 的任何一种
  'properties.bootstrap.servers' = '172.28.28.13:9092',  -- 替换为您的 Kafka 连接地址
  'properties.group.id' = 'testGroup', -- 必选参数，一定要指定 Group ID

  -- 定义数据格式 (Debezium 输出的 JSON 格式)
  'format' = 'debezium-json'
);
```

Canal 格式输入

```
CREATE TABLE `kafka_source`
(
    aid              BIGINT COMMENT 'unique id',
    charname         string,
    `ts`             timestamp(6),
    origin_database STRING METADATA FROM 'value.database' VIRTUAL,
    origin_table STRING METADATA FROM 'value.table' VIRTUAL,
    origin_es TIMESTAMP(3) METADATA FROM 'value.event-timestamp' VIRTUAL,
    origin_type STRING METADATA FROM 'value.operation-type' VIRTUAL,
    `batch_id` bigint METADATA FROM 'value.batch-id' VIRTUAL,
    `is_ddl` boolean METADATA FROM 'value.is-ddl' VIRTUAL,
    origin_old ARRAY<MAP<STRING, STRING>> METADATA FROM 'value.update-before' VIRTUAL,
    `mysql_type` MAP<STRING, STRING> METADATA FROM 'value.mysql-type' VIRTUAL,
    origin_pk_names ARRAY<STRING> METADATA FROM 'value.pk-names' VIRTUAL,
    `sql` STRING METADATA FROM 'value.sql' VIRTUAL,
    origin_sql_type MAP<STRING, INT> METADATA FROM 'value.sql-type' VIRTUAL,
    `ingestion_ts` TIMESTAMP(3) METADATA FROM 'value.ingestion-timestamp' VIRTUAL
) WITH (
    'connector' = 'kafka', -- 注意选择对应的内置  Connector
    'topic' = '$TOPIC', -- 替换为您要消费的 Topic
    'properties.bootstrap.servers' = '$IP:$PORT', -- 替换为您的 Kafka 连接地址
    'properties.group.id' = 'testGroup', -- 必选参数，一定要指定 Group ID
    'scan.startup.mode' = 'latest-offset',
    'scan.topic-partition-discovery.interval' = '300s',
    'format' = 'canal-json',
```

```
    'canal-json.ignore-parse-errors' = 'false', -- 忽略 JSON 结构解析异常
    'canal-json.source.append-mode' = 'true'  -- 仅支持Flink1.13及以上版本
);
```

用作数据目的（Sink）

JSON 格式输出

```
CREATE TABLE `kafka_json_sink_table` (
  `id` INT,
  `name` STRING
) WITH (
  -- 定义 Kafka 参数
  'connector' = 'kafka',
  'topic' = 'Data-Output',  -- 替换为您要写入的 Topic
  'properties.bootstrap.servers' = '172.28.28.13:9092',  -- 替换为您的 Kafka 连接地址

  -- 定义数据格式 (JSON 格式)
  'format' = 'json',
  'json.fail-on-missing-field' = 'false',  -- 如果设置为 false，则遇到缺失字段不会报错。
  'json.ignore-parse-errors' = 'true'    -- 如果设置为 true，则忽略任何解析报错。
);
```

CSV 格式输出

```
CREATE TABLE `kafka_csv_sink_table` (
  `id` INT,
  `name` STRING
) WITH (
  -- 定义 Kafka 参数
  'connector' = 'kafka',
  'topic' = 'Data-Output',  -- 替换为您要写入的 Topic
  'properties.bootstrap.servers' = '172.28.28.13:9092',  -- 替换为您的 Kafka 连接地址

  -- 定义数据格式 (CSV 格式)
  'format' = 'csv'
);
```

Canal 格式输出

```
CREATE TABLE `kafka_canal_json_sink_table`
(
    aid              BIGINT COMMENT 'unique id',
    charname         string,
    `ts`            timestamp(6),
    origin_database STRING METADATA FROM 'value.database',
    origin_table STRING METADATA FROM 'value.table',
    origin_ts TIMESTAMP(3) METADATA FROM 'value.event-timestamp',
    `type` STRING METADATA FROM 'value.operation-type',
    `batch_id` bigint METADATA FROM 'value.batch-id',
    `isDdl` BOOLEAN METADATA FROM 'value.is-ddl',
    `old` ARRAY<MAP<STRING, STRING>> METADATA FROM 'value.update-before',
    `mysql_type` MAP<STRING, STRING> METADATA FROM 'value.mysql-type',
    `pk_names` ARRAY<STRING> METADATA FROM 'value.pk-names',
    `sql` STRING METADATA FROM 'value.sql',
    `sql_type` MAP<STRING, INT> METADATA FROM 'value.sql-type',
    `ingestion_ts` TIMESTAMP(3) METADATA FROM 'value.ingestion-timestamp'
```

```
) WITH (
    'connector' = 'kafka', -- 注意选择对应的内置 Connector
    'topic' = '$TOPIC', -- 替换为您要消费的 Topic
    'properties.bootstrap.servers' = '$IP:$PORT', -- 替换为您的 Kafka 连接地址
    'properties.group.id' = 'testGroup', -- 必选参数，一定要指定 Group ID
    'format' = 'canal-json'
);
```

WITH 参数

参数值	必填	默认值	描述
connector	是	无	固定值为 `'kafka'` 。
topic	是	无	要读写的 Kafka Topic 名。
properties.bootstrap.servers	是	无	逗号分隔的 Kafka Bootstrap 地址。
properties.group.id	作为数据源时必选	无	Kafka 消费时的 Group ID。
format	是	无	Kafka 消息的输入输出格式。目前支持 `csv` 、 `json` 、 `avro` 、 `debezium-json` 、 `canal-json` ，Flink1.13支持 `maxwell-json` 。
scan.startup.mode	否	group-offsets	Kafka consumer 的启动模式。可以是 `latest-offset` 、 `earliest-offset` 、 `specific-offsets` 、 `group-offsets` 、 `timestamp` 的任何一种。 • `'scan.startup.specific-offsets' = 'partition:0,offset:42;partition:1,offset:300'` ，使用 `'specific-offsets'` 启动模式时需要指定每个 partition 对应的 offsets。 • `'scan.startup.timestamp-millis' = '1631588815000'` ，使用 `'timestamp'` 启动模式时需要指定启动的时间戳（单位毫秒）。
scan.startup.specific-offsets	否	无	如果 `scan.startup.mode` 的值为 `'specific-offsets'` ，则必须使用本参数指定具体起始读取的偏移量。例如 `'partition:0,offset:42;partition:1,offset:300'` 。
scan.startup.timestamp-millis	否	无	如果 `scan.startup.mode` 的值为 `'timestamp'` ，则必须使用本参数来指定开始读取的时间点（毫秒为单位的 Unix 时间戳）。
sink.partitioner	否	无	Kafka 输出时所用的分区器。目前支持的分区器如下： • `fixed` ：一个 Flink 分区对应不多于一个 Kafka 分区。 • `round-robin` ：一个Flink 分区依次被分配到不同的 Kafka 分区。 • 自定义分区：也可以通过继承 `FlinkKafkaPartitioner` 类，实现该逻辑。
scan.topic-partition-discovery.interval	否	由 Flink 版本决定	Kafka 消费者定期发现动态创建的 Kafka topic 和分区的间隔时间， • `300s` ：每300秒进行一次符合配置的 topic 分区发现行为。 • `10m` ：每10分钟进行一次符合配置的 topic 分区发现行为。 配置的默认值取决于 Flink 版本， • 1.16版本以下：默认不开启动态发现新分区。 • 1.16版本及以上：默认配置为 `300s` 。

JSON 格式 WITH 参数

参数值	必填	默认值	描述

json.fail-on-missing-field	否	false	如果为 true，则遇到缺失字段时，会让作业失败。如果为 false（默认值），则只会把缺失字段设置为 null 并继续处理。
json.ignore-parse-errors	否	false	如果为 true，则遇到解析异常时，会把这个字段设置为 null 并继续处理。如果为 false，则会让作业失败。
json.timestamp-format.standard	否	SQL	指定 JSON 时间戳字段的格式，默认是 SQL（格式是 `yyyy-MM-dd HH:mm:ss.s{可选精度}` ）。也可以选择 ISO-8601，格式是 `yyyy-MM-ddTHH:mm:ss.s{可选精度}` 。

CSV 格式 WITH 参数

参数值	必填	默认值	描述
csv.field-delimiter	否	,	指定 CSV 字段分隔符，默认是半角逗号。
csv.line-delimiter	否	U&'\000A'	指定 CSV 的行分隔符，默认是换行符 `\n` ，SQL 中必须用 `U&'\000A'` 表示。如果需要使用回车符 `\r` ，SQL 中必须使用 `U&'\000D'` 表示。
csv.disable-quote-character	否	false	禁止字段包围引号。如果为 true，则 'csv.quote-character' 选项不可用。
csv.quote-character	否	"	字段包围引号，引号内部的作为整体看待。默认是 `"` 。
csv.ignore-parse-errors	否	false	忽略处理错误。对于无法解析的字段，会输出为 null。
csv.allow-comments	否	false	忽略 # 开头的注释行，并输出为空行（请务必将 csv.ignore-parse-errors 设为 true）。
csv.array-element-delimiter	否	;	数组元素的分隔符，默认是 `;` 。
csv.escape-character	否	无	指定转义符，默认禁用转义。
csv.null-literal	否	无	将指定的字符串看作 null 值。

Debezium-json 格式 WITH 参数

参数值	必填	默认值	描述
debezium-json.schema-include	否	false	设置 Debezium Kafka Connect 时，如果指定了 `'value.converter.schemas.enable'` 参数，那么 Debezium 发来的 JSON 数据里会包含 Schema 信息，该选项需要设置为 true。
debezium-json.ignore-parse-errors	否	false	忽略处理错误。对于无法解析的字段，会输出为 null。
debezium-json.timestamp-format.standard	否	SQL	指定 JSON 时间戳字段的格式，默认是 SQL（格式是 `yyyy-MM-dd HH:mm:ss.s{可选精度}` ）。也可以选择 ISO-8601，格式是 `yyyy-MM-ddTHH:mm:ss.s{可选精度}` 。

Canal 格式 WITH 参数

参数值	必填	默认值	描述
canal-json.source.append-mode	否	false	设置为 true 时支持 append 流，例如，消费 kafka canal-json 数据到 hive，该参数仅支持 Flink1.13 集群
debezium-json.ignore-parse-errors	否	false	忽略处理错误。对于无法解析的字段，会输出为 null。

canal-json.*	否	-	参考 Format Options

Canal 格式支持的元数据（仅支持 Flink1.13 版本集群）

以下元数据只能作为表定义中的只读（VIRTUAL）列，若元数据列与物理列冲突，元数据列可以使用 `meta.` 列名：

列	数据类型	描述
database	STRING NOT NULL	包含该 Row 的数据库名称
table	STRING NOT NULL	包含该 Row 的表名称
event-timestamp	TIMESTAMP_LTZ(3) NOT NULL	Row 在数据库中进行更改的时间
batch-id	BIGINT	binlog 的批 ID
is-ddl	BOOLEAN	是否 DDL 语句
mysql-type	MAP	数据表结构
update-before	ARRAY	未修改前字段的值
pk-names	ARRAY	主键字段名
sql	STRING	暂时为空
sql-type	MAP	sql_type 表的字段到 java 数据类型 ID 的映射
ingestion-timestamp	TIMESTAMP_LTZ(3) NOT NULL	收到该 ROW 并处理的当前时间
operation-type	STRING	数据库操作类型，例如 INSERT/DELETE 等

代码示例

Json 格式使用示例

```
CREATE TABLE `kafka_json_source_table` (
  `id` INT,
  `name` STRING
) WITH (
  -- 定义 Kafka 参数
  'connector' = 'kafka',
  'topic' = 'Data-Input',  -- 替换为您要消费的 Topic
  'scan.startup.mode' = 'latest-offset', -- 可以是 latest-offset / earliest-offset / specific-offsets /
group-offsets / timestamp 的任何一种
  'properties.bootstrap.servers' = '172.28.28.13:9092',  -- 替换为您的 Kafka 连接地址
  'properties.group.id' = 'testGroup', -- 必选参数，一定要指定 Group ID

  -- 定义数据格式 (JSON 格式)
  'format' = 'json',
  'json.fail-on-missing-field' = 'false',  -- 如果设置为 false，则遇到缺失字段不会报错。
  'json.ignore-parse-errors' = 'true'    -- 如果设置为 true，则忽略任何解析报错。
);
CREATE TABLE `kafka_json_sink_table` (
  `id` INT,
  `name` STRING
) WITH (
  -- 定义 Kafka 参数
  'connector' = 'kafka',
  'topic' = 'Data-Output',  -- 替换为您要写入的 Topic
  'properties.bootstrap.servers' = '172.28.28.13:9092',  -- 替换为您的 Kafka 连接地址
```

```
  -- 定义数据格式 (JSON 格式)
  'format' = 'json',
  'json.fail-on-missing-field' = 'false',  -- 如果设置为 false，则遇到缺失字段不会报错。
  'json.ignore-parse-errors' = 'true'    -- 如果设置为 true，则忽略任何解析报错。
);
insert into kafka_json_sink_table select * from kafka_json_source_table;
```

复杂嵌套 Json 格式使用示例

- json 示例

```
{
  "id": 1234567890,
  "name": "tom",
  "date": "2000-10-25",
  "obj": {
    "time1": "11:11:11",
    "str": "test",
    "lg": 1122334455
  },
  "arr": [
    "aa",
    "bb",
    "cc",
    "dd"
  ],
  "rowinarr": [
    {
      "f1": "f11",
      "f2": 111
    },
    {
      "f1": "f12",
      "f2": 222
    }
  ],
  "time": "19:19:19",
  "timestamp": "1999-01-12 14:14:14",
  "map": {
    "flink": 123
  },
  "mapinmap": {
    "inner_map": {
      "key": 234
    }
  }
}
```

- Flink SQL

```
create table kafka_source(
  id            BIGINT,
  name          STRING,
  `date`        DATE,
  obj           ROW<time1 TIME,str STRING,lg BIGINT>,
  arr           ARRAY<STRING>,
  rowinarr      ARRAY<ROW<f1 STRING,f2 INT>>,
```

```
  `time`        TIME,
  `timestamp`   TIMESTAMP(3),
  `map`         MAP<STRING,BIGINT>,
  mapinmap      MAP<STRING,MAP<STRING,INT>>
) with (
  'connector' = 'kafka',
  'topic' = 'test-topic',
  'scan.startup.mode' = 'latest-offset',
  'properties.bootstrap.servers' = '172.28.28.13:9092',
  'properties.group.id' = 'testGroup',
  'format' = 'json'
);

create table logger_sink (
  id            BIGINT,
  name          STRING,
  `date`        DATE,
  str           STRING,
  arr           ARRAY<STRING>,
  nameinarray   STRING,
  rowinarr      ARRAY<ROW<f1 STRING,f2 INT>>,
  f2            INT,
  `time`        TIME,
  `timestamp`   TIMESTAMP(3),
  `map`         MAP<STRING,BIGINT>,
  flink         BIGINT,
  mapinmap      MAP<STRING,MAP<STRING,INT>>,
  `key`         INT
) with (
    'connector' = 'logger'
);

insert into
  logger_sink
select
  id,
  name,
  `date`,
  obj.str,
  arr,
  arr[4],
  rowinarr,
  rowinarr[1].f2,
  `time`,
  `timestamp`,
  `map`,
  `map`['flink'],
  mapinmap,
  mapinmap['inner_map']['key']
from kafka_source;
```

- **输出结果**

```
+I(1234567890,tom,2000-10-25,test,[aa, bb, cc, dd],dd,[f11,111, f12,222],111,13:13:13,1999-01-12T14:14:14,{flink=123},123,{inner_map={key=234}},234)
```

> **注意：**
> 1. **各数据类型获取元素的方法：**

- map：map['key']
- array：array[index]
- row：row.key

2. array 的起始下标从 1 开始。

Json 数据类型映射

Flink SQL type	JSON type
CHAR / VARCHAR / STRING	string
BOOLEAN	boolean
BINARY / VARBINARY	string with encoding: base64
DECIMAL	number
TINYINT	number
SMALLINT	number
INT	number
BIGINT	number
FLOAT	number
DOUBLE	number
DATE	string with format: date
TIME	string with format: time
TIMESTAMP	string with format: date-time
TIMESTAMP_WITH_LOCAL_TIME_ZONE	string with format: date-time (with UTC time zone)
INTERVAL	number
ARRAY	array
MAP / MULTISET	object
ROW	object

Canal 使用示例

```
CREATE TABLE `source`
(
    `aid`           bigint,
    `charname`      string,
    `ts`            timestamp(6),
    `database_name` string METADATA FROM 'value.database_name',
    `table_name`    string METADATA FROM 'value.table_name',
    `op_ts`         timestamp(3) METADATA FROM 'value.op_ts',
    `op_type` string METADATA FROM 'value.op_type',
    `batch_id` bigint METADATA FROM 'value.batch_id',
    `is_ddl` boolean METADATA FROM 'value.is_ddl',
    `update_before` ARRAY<MAP<STRING, STRING>> METADATA FROM 'value.update_before',
    `mysql_type` MAP<STRING, STRING> METADATA FROM 'value.mysql_type',
    `pk_names` ARRAY<STRING> METADATA FROM 'value.pk_names',
    `sql` STRING METADATA FROM 'value.sql',
```

```
    `sql_type` MAP<STRING, INT> METADATA FROM 'value.sql_type',
    `ingestion_ts` TIMESTAMP(3) METADATA FROM 'value.ts',
    primary key (`aid`) not enforced
) WITH (
    'connector' = 'mysql-cdc' ,
     'append-mode' = 'true',
    'hostname' = '$IP',
    'port' = '$PORT',
    'username' = '$USERNAME',
    'password' = '$PASSWORD',
    'database-name' = 't_wr',
    'table-name' = 't1',
    'server-time-zone' = 'Asia/Shanghai',
    'server-id' = '5500-5510'
);

CREATE TABLE `kafka_canal_json_sink`
(
    aid              BIGINT COMMENT 'unique id',
    charname            string,
    `ts`            timestamp(6),
    origin_database STRING METADATA FROM 'value.database',
    origin_table STRING METADATA FROM 'value.table',
    origin_ts TIMESTAMP(3) METADATA FROM 'value.event-timestamp',
    `type` STRING METADATA FROM 'value.operation-type',
    `batch_id` bigint METADATA FROM 'value.batch-id',
    `isDdl` BOOLEAN METADATA FROM 'value.is-ddl',
    `old` ARRAY<MAP<STRING, STRING>> METADATA FROM 'value.update-before',
    `mysql_type` MAP<STRING, STRING> METADATA FROM 'value.mysql-type',
    `pk_names` ARRAY<STRING> METADATA FROM 'value.pk-names',
    `sql` STRING METADATA FROM 'value.sql',
    `sql_type` MAP<STRING, INT> METADATA FROM 'value.sql-type',
    `ingestion_ts` TIMESTAMP(3) METADATA FROM 'value.ingestion-timestamp'
)
    WITH (
        'connector' = 'kafka',
        'topic' = 'TOPIC', -- 替换为您要消费的 Topic
        'properties.bootstrap.servers' = '$IP:$PORT', -- 替换为您的 Kafka 连接地址
        'format' = 'canal-json'
);

insert into kafka_canal_json_sink select * from source;
```

```
 CREATE TABLE `source`
(
    `aid`           bigint,
    `charname`      string,
    `ts`            timestamp(3),
    origin_database STRING METADATA FROM 'value.database' VIRTUAL,
    origin_table STRING METADATA FROM 'value.table' VIRTUAL,
    origin_es TIMESTAMP(3) METADATA FROM 'value.event-timestamp' VIRTUAL,
    origin_type STRING METADATA FROM 'value.operation-type' VIRTUAL,
    `batch_id` bigint METADATA FROM 'value.batch-id' VIRTUAL,
    `is_ddl` boolean METADATA FROM 'value.is-ddl' VIRTUAL,
    origin_old ARRAY<MAP<STRING, STRING>> METADATA FROM 'value.update-before' VIRTUAL,
    `mysql_type` MAP<STRING, STRING> METADATA FROM 'value.mysql-type' VIRTUAL,
    origin_pk_names ARRAY<STRING> METADATA FROM 'value.pk-names' VIRTUAL,
    `sql` STRING METADATA FROM 'value.sql' VIRTUAL,
```

```
    origin_sql_type MAP<STRING, INT> METADATA FROM 'value.sql-type' VIRTUAL,
    `ingestion_ts` TIMESTAMP(3) METADATA FROM 'value.ingestion-timestamp' VIRTUAL,
    WATERMARK FOR `origin_es` AS `origin_es` - INTERVAL '5' SECOND
) WITH (
      'connector' = 'kafka', -- 注意选择对应的内置  Connector
      'topic' = '$TOPIC', -- 替换为您要消费的 Topic
      'properties.bootstrap.servers' = '$IP:PORT', -- 替换为您的 Kafka 连接地址
      'properties.group.id' = 'testGroup', -- 必选参数, 一定要指定 Group ID
      'scan.startup.mode' = 'latest-offset',
      'scan.topic-partition-discovery.interval' = '300s',

      'format' = 'canal-json',
      'canal-json.source.append-mode' = 'true', -- 仅支持Flink1.13
      'canal-json.ignore-parse-errors' = 'false'
);


CREATE TABLE `kafka_canal_json` (
   `aid`           bigint,
   `charname`      string,
   `ts`            timestamp(9),
   origin_database STRING,
   origin_table STRING,
   origin_es TIMESTAMP(9),
   origin_type STRING,
   `batch_id` bigint,
   `is_ddl` boolean,
   origin_old ARRAY<MAP<STRING, STRING>>,
   `mysql_type` MAP<STRING, STRING>,
   origin_pk_names ARRAY<STRING>,
   `sql` STRING,
   origin_sql_type MAP<STRING, INT>,
   `ingestion_ts` TIMESTAMP(9),
   dt STRING,
   hr STRING
) PARTITIONED BY (dt, hr)
with (
     'connector' = 'hive',
     'hive-version' = '3.1.1',
     'hive-database' = 'testdb',
     'partition.time-extractor.timestamp-pattern'='$dt $hr:00:00',
     'sink.partition-commit.trigger'='partition-time',
     'sink.partition-commit.delay'='30 min',
     'sink.partition-commit.policy.kind'='metastore,success-file'
);

insert into kafka_canal_json select *,DATE_FORMAT(`origin_es`,'yyyy-MM-
dd'),DATE_FORMAT(`origin_es`,'HH')
from `source`;
```

SASL 认证授权

SASL/PLAIN 用户名密码认证授权

1. 参考 消息队列 CKafka > 配置 ACL 策略，设置 Topic 按用户名密码访问的 SASL_PLAINTEXT 认证方式。
2. 参考 消息队列 CKafka > 添加路由策略，选择 SASL_PLAINTEXT 接入方式，并以该接入方式下的网络地址访问 Topic。
3. 作业配置 with 参数。

```
CREATE TABLE `YourTable` (
...
) WITH (
  ...
  'properties.sasl.jaas.config' = 'org.apache.kafka.common.security.plain.PlainLoginModule required
username="ckafka-xxxxxxxx#YourUserName" password="YourPassword";',
  'properties.security.protocol' = 'SASL_PLAINTEXT',
  'properties.sasl.mechanism' = 'PLAIN',
  ...
);
```

说明

1. `username` 是 实例 `ID` + `#` + 刚配置的用户名 ， `password` 是刚配置的用户密码 。
2. flink 版本 sasl 认证配置 对应包：
   - 1.16版本以下：org.apache.kafka.common.security.plain.PlainLoginModule
   - 1.16版本及以上：'org.apache.flink.kafka.shaded.org.apache.kafka.common.security.plain.PlainLoginModule'

SASL/GSSAPI Kerberos 认证授权

腾讯云 CKafka 暂时不支持 Kerberos 认证，您的自建 Kafka 如果开启了 Kerberos 认证，可参考如下步骤配置作业。

1. 获取您的自建 Kafka 集群的 Kerberos 配置文件，如果您基于腾讯云 EMR 集群自建，获取 krb5.conf、emr.keytab 文件，路径如下。

```
/etc/krb5.conf
/var/krb5kdc/emr.keytab
```

2. 对步骤1中获取的文件打 jar 包。

```
jar cvf kafka-xxx.jar krb5.conf emr.keytab
```

3. 校验 jar 的结构（可以通过 vim 命令查看 vim kafka-xxx.jar），jar 里面包含如下信息，请确保文件不缺失且结构正确。

```
META-INF/
META-INF/MANIFEST.MF
emr.keytab
krb5.conf
```

4. 在 程序包管理 页面上传 jar 包，并在作业参数配置里引用该程序包。
5. 获取 kerberos principal，用于作业 高级参数 配置。

```
klist -kt /var/krb5kdc/emr.keytab

# 输出如下所示，选取第一个即可：hadoop/172.28.28.51@EMR-OQPO48B9
KVNO Timestamp     Principal
---- ------------------ -----------------------------------------------------
  2 08/09/2021 15:34:40 hadoop/172.28.28.51@EMR-OQPO48B9
  2 08/09/2021 15:34:40 HTTP/172.28.28.51@EMR-OQPO48B9
  2 08/09/2021 15:34:40 hadoop/VM-28-51-centos@EMR-OQPO48B9
  2 08/09/2021 15:34:40 HTTP/VM-28-51-centos@EMR-OQPO48B9
```

6. 作业 with 参数配置。

```
CREATE TABLE `YourTable` (
...
```

```
) WITH (
  ...
  'properties.security.protocol' = 'SASL_PLAINTEXT',
  'properties.sasl.mechanism' = 'GSSAPI',
  'properties.sasl.kerberos.service.name' = 'hadoop',
  ...
);
```

7. 作业 高级参数 配置。

```
security.kerberos.login.principal: hadoop/172.28.2.13@EMR-4K3VR5FD
security.kerberos.login.keytab: emr.keytab
security.kerberos.login.conf: krb5.conf
security.kerberos.login.contexts: KafkaClient
fs.hdfs.hadoop.security.authentication: kerberos
```

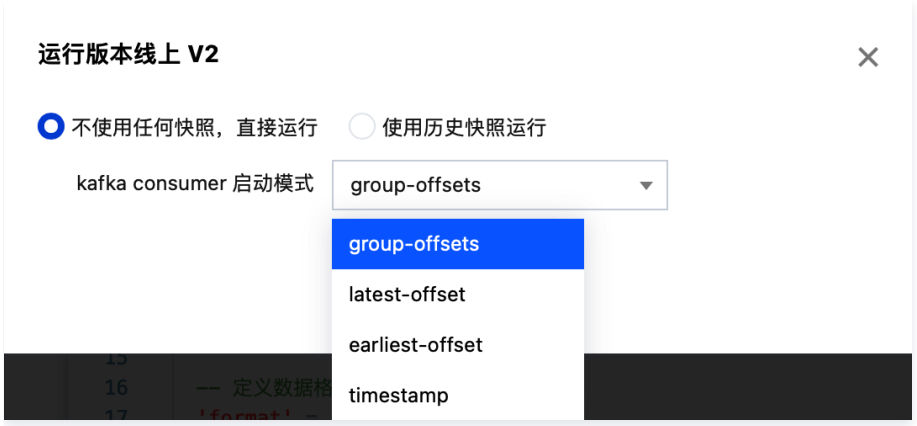
注意：
历史 Oceanus 集群可能不支持该功能，您可通过 在线客服 联系我们升级集群管控服务，以支持 Kerberos 访问。

动态指定 Source 消费模式

当使用 Kafka Connector 作为 Source 时，我们可以在启动时自定义消费模式，目前支持的消费模式有：latest-offset/earliest-offset/group-offsets/timestamp 五种启动模式。 具体用法如下：
1、在 SQL 代码中对应 With 参数部分加入如下代码定义：

```
'scan.startup.mode' = '#{CUSTOM_SCAN_MODE}',
'scan.startup.timestamp-millis' = '#{CUSTOM_TIMESTAMP}',
```

2、加入之后，在启动作业时会有选择框选择启动模式，选择“不使用任何快照，直接运行”，在“kafka consumer 启动模式”中可以选择所需启动模式。再进行启动即可实现指定模式启动。



消息队列 Upsert Kafka

最近更新时间：2023-09-25 16:12:21

介绍

Upsert Kafka 连接器支持以 upsert 方式从 Kafka topic 中读取数据并将数据写入 Kafka topic。
作为 Source，Upsert Kafka 连接器生产 changelog 流，其中每条数据记录代表一个更新或删除事件。更准确地说，数据记录中的 value 被解释为同一 key 的最后一个 value 的 UPDATE，如果有这个 key（如果不存在相应的 key，则该更新被视为 INSERT）。用表来类比，changelog 流中的数据记录被解释为 UPSERT，也称为 INSERT/UPDATE，因为任何具有相同 key 的现有行都被覆盖。另外，value 为空的消息将会被视作为 DELETE 消息。
作为 Sink，Upsert Kafka 连接器可以消费 changelog 流。它会将 INSERT/UPDATE_AFTER 数据作为正常的 Kafka 消息写入，并将 DELETE 数据以 value 为空的 Kafka 消息写入（表示对应 key 的消息被删除）。Flink 将根据主键列的值对数据进行分区，从而保证主键上的消息有序，因此同一主键上的更新/删除消息将落在同一分区中。

版本说明

Flink 版本	说明
1.11	不支持
1.13	支持
1.14	支持
1.16	支持

DDL 定义

```
CREATE TABLE kafka_upsert_sink_table (
  id INT,
  name STRING,
  PRIMARY KEY (id) NOT ENFORCED
) WITH (
  -- 定义 Upsert Kafka 参数
  'connector' = 'upsert-kafka',  -- 选择 connector
  'topic' = 'topic',  -- 替换为您要写入的 Topic
  'properties.bootstrap.servers' = '...',  -- 替换为您的 Kafka 连接地址
  'key.format' = 'json',  -- 定义 key 数据格式
  'value.format' = 'json'  -- 定义value 数据格式
);
```

说明
Upsert Kafka 确保在 DDL 中定义主键。

WITH 参数

参数	是否必选	默认值	数据类型	描述
connector	必选	(none)	String	指定要使用的连接器，Upsert Kafka 连接器使用：`'upsert-kafka'`。
topic	必选	(none)	String	用于读取和写入的 Kafka topic 名称。
properties.bootstrap.servers	必选	(none)	String	以逗号分隔的 Kafka brokers 列表。

properties.*	可选	(none)	String	该选项可以传递任意的 Kafka 参数。选项的后缀名必须匹配定义在 Kafka 参数文档 中的参数名。 Flink 会自动移除选项名中的 "properties." 前缀，并将转换后的键名以及值传入 KafkaClient。例如，您可以通过 `'properties.allow.auto.create.topics' = 'false'` 来禁止自动创建 topic。但是，某些选项，例如 `'key.deserializer'` 和 `'value.deserializer'` 是不允许通过该方式传递参数，因为 Flink 会重写这些参数的值。
key.format	必选	(none)	String	用于对 Kafka 消息中 key 部分序列化和反序列化的格式。key 字段由 PRIMARY KEY 语法指定。支持的格式包括 `'csv'`、`'json'`、`'avro'`。
key.fields-prefix	optional	(none)	String	为'key.fields'的所有字段定义自定义前缀，以避免与 `'value.fields'` 字段名称冲突。默认情况下，前缀为空。如果定义了自定义前缀，则表 schema 和 `'key.fields'` 将使用前缀名称。构建'key.fields'格式的数据类型时候，将删除前缀并使用 key format 中非前缀名称。请注意，此选项要求 `'value.fields-include'` 必须设置为 `'EXCEPT_KEY'`。
value.format	必选	(none)	String	用于对 Kafka 消息中 value 部分序列化和反序列化的格式。支持的格式包括 `'csv'`、`'json'`、`'avro'`。
value.fields-include	可选	'ALL'	String	控制哪些字段应该出现在 value 中。可取值： ALL：消息的 value 部分将包含 schema 中所有的字段，包括定义为主键的字段。 EXCEPT_KEY：记录的 value 部分包含 schema 的所有字段，定义为主键的字段除外。
sink.parallelism	可选	(none)	Integer	定义 upsert-kafka sink 算子的并行度。默认情况下，由框架确定并行度，与上游链接算子的并行度保持一致。
sink.buffer-flush.max-rows	可选	0	Integer	缓存刷新前，最多能缓存多少条记录。当 sink 收到很多同 key 上的更新时，缓存将保留同 key 的最后一条记录，因此 sink 缓存能帮助减少发往 Kafka topic 的数据量，以及避免发送潜在的 tombstone 消息。 可以通过设置为 '0' 来禁用它。默认，该选项是未开启的。注意，如果要开启 sink 缓存，需要同时设置 `'sink.buffer-flush.max-rows'` 和 `'sink.buffer-flush.interval'` 两个选项为大于零的值。
sink.buffer-flush.interval	可选	0	Duration	缓存刷新的间隔时间，超过该时间后异步线程将刷新缓存数据。当 sink 收到很多同 key 上的更新时，缓存将保留同 key 的最后一条记录，因此 sink 缓存能帮助减少发往 Kafka topic 的数据量，以及避免发送潜在的 tombstone 消息。 可以通过设置为 '0' 来禁用它。默认，该选项是未开启的。注意，如果要开启 sink 缓存，需要同时设置 `'sink.buffer-flush.max-rows'` 和 `'sink.buffer-flush.interval'` 两个选项为大于零的值。

代码示例

```
CREATE TABLE `kafka_json_source_table` (
  `id` INT,
  `name` STRING
) WITH (
  -- 定义 Kafka 参数
  'connector' = 'kafka',
  'topic' = 'Data-Input',  -- 替换为您要消费的 Topic
  'scan.startup.mode' = 'latest-offset', -- 可以是 latest-offset / earliest-offset / specific-offsets /
group-offsets / timestamp 的任何一种
  'properties.bootstrap.servers' = '172.28.28.13:9092',  -- 替换为您的 Kafka 连接地址
  'properties.group.id' = 'testGroup', -- 必选参数，一定要指定 Group ID
```

```
  -- 定义数据格式 (JSON 格式)
  'format' = 'json',
  'json.fail-on-missing-field' = 'false',  -- 如果设置为 false，则遇到缺失字段不会报错。
  'json.ignore-parse-errors' = 'true'    -- 如果设置为 true，则忽略任何解析报错。
);

CREATE TABLE kafka_upsert_sink_table (
  id INT,
  name STRING,
  PRIMARY KEY (id) NOT ENFORCED
) WITH (
  -- 定义 Upsert Kafka 参数
  'connector' = 'upsert-kafka',  -- 选择 connector
  'topic' = 'topic',  -- 替换为您要消费的 Topic
  'properties.bootstrap.servers' = '...',  -- 替换为您的 Kafka 连接地址
  'key.format' = 'json',  -- 定义 key 数据格式
  'value.format' = 'json'  -- 定义value 数据格式
);

-- 计算 pv、uv 并插入到 upsert-kafka sink
INSERT INTO kafka_upsert_sink_table
SELECT * FROM kafka_json_source_table;
```

SASL 认证授权

SASL/PLAIN 用户名密码认证授权

1. 参考 消息队列 CKafka – 配置 ACL 策略，设置 Topic 按用户名密码访问的 SASL_PLAINTEXT 认证方式。
2. 参考 消息队列 CKafka – 添加路由策略，选择 SASL_PLAINTEXT 接入方式，并以该接入方式下的网络地址访问 Topic。
3. 作业配置 with 参数。

```
CREATE TABLE `YourTable` (
...
) WITH (
  ...
  'properties.sasl.jaas.config' = 'org.apache.kafka.common.security.plain.PlainLoginModule required
username="ckafka-xxxxxxxx#YourUserName" password="YourPassword";',
  'properties.security.protocol' = 'SASL_PLAINTEXT',
  'properties.sasl.mechanism' = 'PLAIN',
  ...
);
```

> **说明**
> `username` 是 实例 `ID` + `#` + 刚配置的用户名， `password` 是刚配置的用户密码。

SASL/GSSAPI Kerberos 认证授权

腾讯云 CKafka 暂时不支持 Kerberos 认证，您的自建 Kafka 如果开启了 Kerberos 认证，可参考如下步骤配置作业。

1. 获取您的自建 Kafka 集群的 Kerberos 配置文件，如果您基于腾讯云 EMR 集群自建，获取 krb5.conf、emr.keytab 文件，路径如下。

```
/etc/krb5.conf
/var/krb5kdc/emr.keytab
```

2. 对步骤1中获取的文件打 jar 包。

```
jar cvf kafka-xxx.jar krb5.conf emr.keytab
```

3. 校验 jar 的结构（可以通过 vim 命令查看 vim kafka-xxx.jar），jar 里面包含如下信息，请确保文件不缺失且结构正确。

```
META-INF/
META-INF/MANIFEST.MF
emr.keytab
krb5.conf
```

4. 在 程序包管理 页面上传 jar 包，并在作业参数配置里引用该程序包。
5. 获取 kerberos principal，用于作业 高级参数 配置。

```
klist -kt /var/krb5kdc/emr.keytab

# 输出如下所示，选取第一个即可：hadoop/172.28.28.51@EMR-OQPO48B9
KVNO Timestamp     Principal
---- ------------------- -------------------------------------------------------
   2 08/09/2021 15:34:40 hadoop/172.28.28.51@EMR-OQPO48B9
   2 08/09/2021 15:34:40 HTTP/172.28.28.51@EMR-OQPO48B9
   2 08/09/2021 15:34:40 hadoop/VM-28-51-centos@EMR-OQPO48B9
   2 08/09/2021 15:34:40 HTTP/VM-28-51-centos@EMR-OQPO48B9
```

6. 作业 with 参数配置。

```
CREATE TABLE `YourTable` (
...
) WITH (
  ...
  'properties.security.protocol' = 'SASL_PLAINTEXT',
  'properties.sasl.mechanism' = 'GSSAPI',
  'properties.sasl.kerberos.service.name' = 'hadoop',
  ...
);
```

> **说明：**
> 参数 `properties.sasl.kerberos.service.name` 的值必须与您选取的 principal 匹配，如果您选择的为 `hadoop/${IP}@EMR-OQPO48B9`，那么取值为 hadoop。

7. 作业 高级参数 配置。

```
security.kerberos.login.principal: hadoop/172.28.2.13@EMR-4K3VR5FD
security.kerberos.login.keytab: emr.keytab
security.kerberos.login.conf: krb5.conf
security.kerberos.login.contexts: KafkaClient
fs.hdfs.hadoop.security.authentication: kerberos
```

消息队列 Pulsar

最近更新时间：2025-05-15 16:56:01

介绍

Pulsar SQL connector 允许您使用简单的 SQL 查询或 Flink Table API 从 Pulsar topic 查询数据或将数据写入 Pulsar。

版本说明

flink-connector-pulsar 基于 StreamNative/flink 适配到 flink-1.13 和 flink-1.14。使用 DataStream API 时可参考 StreamNative 文档。

Flink 版本	说明	源码
1.11	不支持	-
1.13	支持	develop-flink-1.13
1.14	支持	develop-flink-1.14
1.16	支持	-

使用范围

- 可以作为源表，以及 Tuple、Upsert 数据流的目的表。
- 不支持维表。

如何创建 Pulsar 表

以下示例展示了如何创建 Pulsar 表：

```
CREATE TABLE PulsarTable (
  `user_id` bigint,
  `item_id` bigint,
  `behavior` STRING,
  `publish_time` TIMESTAMP_LTZ(3) METADATA FROM 'publish_time' VIRTUAL
) WITH (
  'connector' = 'pulsar',
  'service-url' = 'pulsar://pulsar:6650',
  'admin-url' = 'http://pulsar:8080',
  -- 'pulsar.client.authPluginClassName' = 'org.apache.pulsar.client.impl.auth.AuthenticationToken',
  -- 'pulsar.client.authParams' =
'token:eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJ0ZXN0In0.Cpsuc4hRw3yL231Kp8iQAQkYc2SvVTfeziAMcLxxxxx',
  'topics' = 'user_behavior',
  'format' = 'json',
  'source.subscription-name' = 'flink',
  'source.start.message-id' = 'earliest'
);
```

> **说明：**
> 如果 Pulsar 集群开启了鉴权，请使用有 admin 权限的 token。

关于 token

flink-connector-pulsar 调用 Pulsar Admin APIs 监听 topic 的 partition 变化、创建订阅（subscription），如果 Pulsar 集群开启了 token 鉴权，需要 token 有 admin 权限。此外还需要有 topic 的读/写权限。

```
# 验证 admin 权限
admin_url=http://172.28.28.46:8080,172.28.28.29:8080,172.28.28.105:8080
```

```
token=eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJteS1zdXBlci11c2VyIn0.T-Bi__yGCs1lxEwdUcDKBDuJsWLO0X9LrePhnuXXXXX
namespace=public/default
pulsar-admin --admin-url ${admin_url} --auth-params token:${token} --auth-plugin
org.apache.pulsar.client.impl.auth.AuthenticationToken topics list ${namespace}

# 验证 topic 读权限
service_url=pulsar://172.28.28.46:6650,172.28.28.29:6650,172.28.28.105:6650
token=eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJteS1zdXBlci11c2VyIn0.T-Bi__yGCs1lxEwdUcDKBDuJsWLO0X9LrePhnuXXXXX
namespace=public/default
topic=xxx
subscription=yyy
pulsar-client --url ${service_url} --auth-params token:${token} --auth-plugin
org.apache.pulsar.client.impl.auth.AuthenticationToken consume -s ${subscription} -n 10
persistent://${namespace}/${topic} -p Earliest
```

Connector 参数

参数值	必填	默认值	数据类型	描述
connector	必选	(none)	String	指定要使用的连接器，Apache Pulsar 连接器使用：`'pulsar'`、`'upsert-pulsar'`
admin-url	必选	(none)	String	Pulsar 管理端地址 HTTP 地址，例如 `http://my-broker.example.com:8080` 或者 `https://my-broker.example.com:8443`
service-url	必选	(none)	String	Pulsar 服务端 URL • pulsar client 连接 pulsar 集群需要 pulsar 协议的 URL。例如：`pulsar://localhost:6650` • 多个 broker 节点的 URL：`pulsar://localhost:6550,localhost:6651,localhost:6652` • 生产环境集群一般以域名方式访问，例如：`pulsar://pulsar.us-west.example.com:6650` • 开启 TLS 验证的 URL：`pulsar+ssl://pulsar.us-west.example.com:6651`
topics	必选	(none)	String	用于读取和写入的 Apache Pulsar 的 topic 名称 • 可以是单个 topic 名称，也可以是以“;”分割的 topic 列表，例如：`topic-1;topic-2` • 可以指定一组 topic 或者分区或者是两者都有，例如：`topic-a-partition-0;topic-a-partition-2;some-topic2` • 如果同时指定了某个 topic 和其下属的分区，那么将会自动将两者合并，仅使用外层的 topic，例如：`some-topic1;some-topic1-partition-0` 等价于 `some-topic1`
pulsar.client.authPluginClassName	可选	(none)	String	鉴权插件类名，token 鉴权填写：`org.apache.pulsar.client.impl.auth.AuthenticationToken`
pulsar.client.authParams	可选	(none)	String	鉴权参数，token 鉴权填写格式为：`token:xxxx`
explicit	可选	true	Boolean	是否为 explicit flink 表，用于 pulsar catalog。参考 pulsar catalog 介绍

key.fields	可选	（none）	List<String>	Pulsar 消息中的 key 字段对应的 flink 表物理字段。注意这些字段与 primary key 没有关系
key.format	可选	(none)	String	用于对 Pulsar 消息中 key 部分序列化和反序列化的格式。 支持的格式包括 `'csv'` 、 `'json'` 、 `'avro'` 等。请参考 格式 页面以获取更多详细信息和格式参数
format	可选	(none)	String	用于对 pulsar 消息中 value 部分序列化和反序列化的格式。 支持的格式包括 `'csv'` 、 `'json'` 、 `'avro'` 。请参考 格式 页面以获取更多详细信息和格式参数 选项 `format` 和 `value.format` 必选其一，format 具有高优先级
value.format	可选	(none)	String	用于对 pulsar 消息中 value 部分序列化和反序列化的格式 支持的格式包括 `'csv'` 、 `'json'` 、 `'avro'` 。请参考 格式 页面以获取更多详细信息和格式参数 选项 `format` 和 `value.format` 必选其一，format 具有高优先级
sink.topic-routing-mode	可选	round-robin	Enum	Topic 路由策略。可用选项为 `round-robin` 和 `message-key-hash` 。默认情况下，它设置为 `round-robin` 。如果要使用自定义 topic 路由策略，请使用 `sink.custom-topic-router` 选项设置
sink.custom-topic-router	可选	（none）	String	自定义 topic 路由策略完整类名，如果设置了该选项，请不要设置 `sink.topic-routing-mode`
sink.message-delay-interval	可选	0	Duration	延迟消息投递间隔，示例： `10ms`、`1s`、`1min` 。延迟消息投递功能使得消息被延迟消费，参考 pulsar 文档 Delayed message delivery
pulsar.sink.deliveryGuarantee	可选	none	Enum	Pulsar sink 的数据一致性保证，可选 `none` 、 `at-least-once` 、 `exactly-once` 。 `exactly-once` 需要 Pulsar 集群支持事务
pulsar.sink.transactionTimeoutMillis	可选	10800000	Long	单位毫秒，Pulsar 事务超时时间，默认为 3 小时（10800000 毫秒），注意必须保证事务超时时间大于 checkpoint 时间间隔
pulsar.producer.batchingEnabled	可选	false	Boolean	是否开启分批写
pulsar.producer.batchingMaxMessages	可选	1000	Int	Pulsar 批次写入最大消息条数
source.start.message-id	可选	(none)	String	指定 source 消费起始位点，可选值为 `earliest` , `latest` 或消息 ID (格式 `ledgerId:entryId:partitionId` , 例如 "12:2:-1")
source.start.publish-time	可选	(none)	Long	指定 source 消费起始点的发布时间戳（unix 毫秒级时间戳）
source.subscription-name	可选	flink-sql-connector-pulsar-<RANDOM>	String	Pulsar 消息订阅名字。默认为 `flink-sql-connector-pulsar-<RANDOM>` ，其中 `RANDOM` 为长度为 5 的随机字母
source.subscription-type	可选	Exclusive	Enum	Pulsar 消息订阅类型，目前只支持 `Exclusive` 、 `Shared` ，消息订阅类型的更多信息请参考 Subscription types
source.stop.at-message-id	可选	(none)	String	指定 source 消费结束位点，可选值 `never` , `latest` 或消息 ID (格式 `ledgerId:entryId:partitionId` , 例如 "12:2:-1")

source.stop.at-publish-time	可选	(none)	Long	指定 source 消费结束点的发布时间戳（unix 毫秒级时间戳）
source.stop.after-message-id	可选	(none)	String	指定 source 消费结束位点的消息 ID（包含该消息），格式 `ledgerId:entryId:partitionId`，例如 "12:2:-1"
pulsar.source.partitionDiscoveryIntervalMs	可选	30000	Long	单位毫秒，pulsar source 自动探测新增 partition 的时间间隔。设置为 0 或者负值禁用 partition 探测
pulsar.admin.requestRetries	可选	5	Int	Pulsar admin Rest API 调用失败重试次数
pulsar.client.*	可选	(none)	-	该选项可以传递任意的 pulsar client 参数
pulsar.admin.*	可选	(none)	-	该选项可以传递任意的 pulsar 管理端参数
pulsar.sink..*	可选	(none)	-	该选项可以传递其他的 pulsar sink connector 参数
pulsar.producer.*	可选	(none)	-	该选项可以传递任意的 pulsar producer API 配置参数
pulsar.source..*	可选	(none)	-	该选项可以传递其他的 pulsar source connector 参数
pulsar.consumer.*	可选	(none)	-	该选项可以传递任意的 pulsar consumer API 配置参数

可用元数据

元数据 Key	数据类型	R/W	描述
topic	STRING NOT NULL	R	Pulsar 消息的 Topic name 字段。
message_size	INT NOT NULL	R	Pulsar 消息大小。
producer_name	STRING NOT NULL	R	Pulsar 消息的 Producer name 字段。
message_id	BYTES NOT NULL	R	Pulsar 消息的 Message ID 字段。
sequenceId	BIGINT NOT NULL	R	Pulsar 消息的 Sequence ID 字段。
publish_time	TIMESTAMP_LTZ(3) NOT NULL	R	Pulsar 消息的 Publish time 字段。
event_time	TIMESTAMP_LTZ(3) NOT NULL	R/W	Pulsar 消息的 Properties 字段。
properties	MAP<STRING, STRING> NOT NULL	R/W	Pulsar 消息的 Event time 字段。

说明：

- `R/W` 列定义了一个元数据是可读的（`R`）还是可写的（`W`）。只读列必须声明为 `VIRTUAL` 以在 `INSERT INTO` 操作中排除它们。
- Pulsar 消息的字段列表参考 pulsar 官网文档 Messages。

数据类型映射

Pulsar schema	Flink format
AVRO	avro
JSON	json
PROTOBUF	Not supported yet
PROTOBUF_NATIVE	Not supported yet
AUTO_CONSUME	Not supported yet

AUTO_PUBLISH	Not supported yet
NONE/BYTES	raw
BOOLEAN	raw
STRING	raw
DOUBLE	raw
FLOAT	raw
INT8	raw
INT16	raw
INT32	raw
INT64	raw
LOCAL_DATE	Not supported yet
LOCAL_TIME	Not supported yet
LOCAL_DATE_TIME	Not supported yet

PulsarCatalog

PulsarCatalog 支持将 Pulsar 集群配置为 Flink 表的元数据存储。

Explicit 表和 native 表

PulsarCatalog 定义了两种类型的表: `explicit` 表 and `native` tables。

- `explicit` 表是使用 CREATE 语句或 table API 显式创建的表，它类似于其他 SQL connector 中的常用模式。您可以创建表，然后从表中查询数据或向表中写入数据。
- `native` 表由 PulsarCatalog 自动创建。PulsarCatalog 扫描 Pulsar 集群中的所有非系统主题，然后将每个 topic 映射成 Flink 表，而不使用 CREATE 语句。

Explicit 表

PulsarCatalog 使用 topic schema 的 `schemaInfo` 字段存储 `explicit` 表的元数据信息。对于每个 `explicit` 表，PulsarCatalog 在默认为 `__flink_catalog` 的 tenant 下创建一个占位 topic。可通过 `catalog-tenant` 选项设置 tenant。Flink 的 database 映射为该 tenant 下同样名字的 namespace。然后创建一个名为 `table_<FLINK_TABLE_NAME>` 的 topic，该 topic 的 schema 存储了 Flink 表的 schema 元数据信息。
例如，如果您创建了database 为 `testdb` ，表为 `users` 的 flink 表，那么 PulsarCatalog 创建 tenant 为 `__flink_catalog` ，namespace 为 `testdb` 的 topic `table_users` 。
Topic `table_users` 之所以称之为占位 topic 是因为它没有任何 producer 或者 consumer，您可以使用占位 topic 的 schema 来存储 Flink 表的元数据信息。
您可以使用 `pulsar-admin` 命令行工具来获取 topic 的元数据信息：

```
pulsar-admin schemas get persistent://<tenant>/<namespace>/<topic>
```

Native 表

`native` 表没有任何占位 topic，PulsarCatalog 把 topic 的 schema 映射为 Flink 表的 schema。关于 Pulsar schema，参考 Pulsar 官网文档 Understand schema 。

Pulsar schema	Flink data type	Flink format	Work or not
AVRO	It is decided by the Avro format.	avro	Yes
JSON	It is decided by the JSON format.	json	Yes

PROTOBUF	Not supported yet	/	No
PROTOBUF_NATIVE	It is decided by the Protobuf definition.	Not supported yet	No
AUTO_CONSUME	Not supported yet	/	No
AUTO_PUBLISH	Not supported yet	/	No
NONE/BYTES	DataTypes.BYTES()	raw	Yes
BOOLEAN	DataTypes.BOOLEAN()	raw	Yes
LOCAL_DATE	DataTypes.DATE()	/	No
LOCAL_TIME	DataTypes.TIME()	/	No
LOCAL_DATE_TIME	DataTypes.TIMESTAMP(3)	/	No
STRING	DataTypes.STRING()	raw	Yes
DOUBLE	DataTypes.DOUBLE()	raw	Yes
FLOAT	DataTypes.FLOAT()	raw	Yes
INT8	DataTypes.TINYINT()	raw	Yes
INT16	DataTypes.SMALLINT()	raw	Yes
INT32	DataTypes.INT()	raw	Yes
INT64	DataTypes.BIGINT()	raw	Yes

说明：
尽管对于 Pulsar schema 的 `LOCAL_DATE`，`LOCAL_TIME` 和 `LOCAL_DATE_TIME` 有相应的 flink 数据类型，但 flink 基于这几种 Pulsar schema 无法解析数据，因此自动 schema 映射会失败。

Explicit 和 native 表对比

使用 `native` 表，您可以从现有 Pulsar topic 中查询数据。PulsarCatalog 自动读取 topic 的 schema，并决定使用哪种 format 来解码/编码。但是，`native` 表不支持 `watermark` 和主键，因此，不能使用 `native` 表进行基于事件时间的窗口聚合。`native` 表将 `tenant/namespace` 映射到 flink 的 database，topic 名字映射为 flink 表名。
要完全管理表，可以使用 `explicit` 表定义 `watermark`、指定元数据字段和指定自定义格式。其用法类似于在 `GenericInMemoryCatalog` 中创建 Pulsar 表。您可以将 `explicit` 表绑定到 Pulsar 的 topic，每个 Pulsar topic 可以绑定到多个 Flink 表（包括 `native` 表）。

PulsarCatalog 参数

Key	Default	Type	Description	Required
catalog-admin-url	"http://localhost:8080"	String	Pulsar 管理端地址 HTTP 地址，例如 `http://my-broker.example.com:8080` 或者 `https://my-broker.example.com:8443`。	Yes
catalog-auth-params	(none)	String	访问 Pulsar 集群的认证参数。	-
catalog-auth-plugin	(none)	String	访问 Pulsar 集群的认证 plugin 名字。	-
catalog-service-url	"pulsar://localhost:6650"	String	Pulsar 服务端 URL • pulsar client 连接 pulsar 集群需要 pulsar 协议的 URL。例如：`pulsar://localhost:6650`	Yes

			• 多个 broker 节点的 URL：`pulsar://localhost:6550,localhost:6651,localhost:6652` • 生产环境集群一般以域名方式访问，例如：`pulsar://pulsar.us-west.example.com:6650` 。 • 开启 TLS 验证的 URL：`pulsar+ssl://pulsar.us-west.example.com:6651` 。	
catalog-tenant	"__flink_catalog"	String	存储表信息的 Pulsar tenant。	-
default-database	"default_database"	String	PulsarCatalog 的默认 database，不存在时会自动创建。	-

PulsarCatalog 示例

```
CREATE CATALOG pulsar WITH (
  'type' = 'pulsar-catalog',
  -- 'catalog-auth-plugin' = 'org.apache.pulsar.client.impl.auth.AuthenticationToken',
  -- 'catalog-auth-params' = 'token:eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJteS1zdXBlci11c2VyIn0.T-Bi__yGCs11xEwdUcDKBDuJsWLO0X9LrePhnuxxxxx',
  'catalog-admin-url' = '<ADMIN_URL>',
  'catalog-service-url' = '<SERVICE_URL>'
);
```

完整示例

Pulsar source 和 sink 示例

如下示例展示了 Pulsar 数据源写入 Pulsar 数据目的， `exactly-once` 的数据一致性保证，事务超时时间 2 分钟（注意必须保证事务超时时间大于 checkpoint 时间间隔）。

```
CREATE TABLE `pulsar_source` (
  `user_id` bigint,
  `item_id` bigint,
  `behavior` STRING
) WITH (
  'connector' = 'pulsar',
  'service-url' = 'pulsar://pulsar:6650',
  'admin-url' = 'http://pulsar:8080',
  -- 'pulsar.client.authPluginClassName' = 'org.apache.pulsar.client.impl.auth.AuthenticationToken',
  -- 'pulsar.client.authParams' =
'token:eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJ0ZXN0In0.Cpsuc4hRw3yL231Kp8iQAQkYc2SvVTfeziAMcLxxxxx',
  'topics' = 'topic_source',
  'format' = 'json',
  'source.subscription-name' = 'flink',
  'source.start.message-id' = 'earliest'
);

CREATE TABLE `pulsar_sink` (
  `user_id` bigint,
  `item_id` bigint,
  `behavior` STRING
) WITH (
  'connector' = 'pulsar',
  'service-url' = 'pulsar://pulsar:6650',
  'admin-url' = 'http://pulsar:8080',
  -- 'pulsar.client.authPluginClassName' = 'org.apache.pulsar.client.impl.auth.AuthenticationToken',
```

```
  -- 'pulsar.client.authParams' =
'token:eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJ0ZXN0In0.Cpsuc4hRw3yL231Kp8iQAQkYc2SvVTfeziAMcLxxxxx',
  'topics' = 'topic_sink',
  'format' = 'json',
  'pulsar.sink.deliveryGuarantee' = 'exactly-once',
  'pulsar.sink.transactionTimeoutMillis' = '120000'
);

INSERT INTO `pulsar_sink` SELECT * FROM `pulsar_source`;
```

Upsert Pulsar sink 示例

如下示例展示了通过 `'upsert-pulsar'` 写入 Pulsar 目的表。注意，使用时需要设置主键，并在with参数中添加 `'key.format'` 。

```
CREATE TABLE `gen_source` (
  `user_id` bigint,
  `item_id` bigint,
  `behavior` STRING
) WITH (
  'connector' = 'datagen',
  'rows-per-second' = '10',
  'fields.user_id.kind' = 'sequence',
  'fields.user_id.start' = '1',
  'fields.user_id.end' = '100000'
);

CREATE TABLE `pulsar_sink` (
  `user_id` bigint,
  `item_id` bigint,
  `behavior` STRING,
  PRIMARY KEY(user_id) NOT ENFORCED
) WITH (
    'connector' = 'upsert-pulsar',
    'service-url' = 'pulsar://pulsar:6650',
    'admin-url' = 'http://pulsar:8080',
    -- 'pulsar.client.authPluginClassName' = 'org.apache.pulsar.client.impl.auth.AuthenticationToken',
    -- 'pulsar.client.authParams' =
'token:eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJ0ZXN0In0.Cpsuc4hRw3yL231Kp8iQAQkYc2SvVTfeziAMcLxxxxx',
    'topics' = 'topic_sink',
    'key.format' = 'json',
    'format' = 'json',
    'pulsar.sink.deliveryGuarantee' = 'exactly-once',
    'pulsar.sink.transactionTimeoutMillis' = '120000'
);

INSERT INTO `pulsar_sink` SELECT * FROM `gen_source`;
```

PulsarCatalog 示例

Explicit 表示例

```
CREATE CATALOG `pulsar` WITH (
  'type' = 'pulsar-catalog',
  -- 'catalog-auth-plugin' = 'org.apache.pulsar.client.impl.auth.AuthenticationToken',
  -- 'catalog-auth-params' = 'token:eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJteS1zdXBlci11c2VyIn0.T-
Bi__yGCs1lxEwdUcDKBDuJsWLO0X9LrePhnuxxxxx',
  'catalog-admin-url' = 'http://pulsar:8080',
  'catalog-service-url' = 'pulsar://pulsar:6650'
```

```
);
INSERT INTO `pulsar`.`default_database`.`pulsar_sink` SELECT * FROM
`pulsar`.`default_database`.`pulsar_source`;
```

其中的 `pulsar_source` 和 `pulsar_sink` 表用下面语句创建（可以放到同一个 SQL 作业里）。

```
CREATE TABLE IF NOT EXISTS `pulsar`.`default_database`.`pulsar_source` (
  `user_id` bigint,
  `item_id` bigint,
  `behavior` STRING
) WITH (
  'connector' = 'pulsar',
  'service-url' = 'pulsar://pulsar:6650',
  'admin-url' = 'http://pulsar:8080',
  -- 'pulsar.client.authPluginClassName' = 'org.apache.pulsar.client.impl.auth.AuthenticationToken',
  -- 'pulsar.client.authParams' =
'token:eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJ0ZXN0In0.Cpsuc4hRw3yL231Kp8iQAQkYc2SvVTfeziAMcLxxxxx',
  'topics' = 'topic_source',
  'format' = 'json',
  'source.subscription-name' = 'flink',
  'source.start.message-id' = 'earliest'
);

CREATE TABLE IF NOT EXISTS `pulsar`.`default_database`.`pulsar_sink` (
  `user_id` bigint,
  `item_id` bigint,
  `behavior` STRING
) WITH (
  'connector' = 'pulsar',
  'service-url' = 'pulsar://pulsar:6650',
  'admin-url' = 'http://pulsar:8080',
  -- 'pulsar.client.authPluginClassName' = 'org.apache.pulsar.client.impl.auth.AuthenticationToken',
  -- 'pulsar.client.authParams' =
'token:eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJ0ZXN0In0.Cpsuc4hRw3yL231Kp8iQAQkYc2SvVTfeziAMcLxxxxx',
  'topics' = 'topic_sink',
  'format' = 'json',
  'pulsar.sink.deliveryGuarantee' = 'exactly-once',
  'pulsar.sink.transactionTimeoutMillis' = '120000'
);
```

Native 表示例

1. 准备 topic schema 的 json 格式文件，命名为 schema.json。

```
{
  "schema": "{\"type\":\"record\",\"name\":\"userBehavior\",\"namespace\":\"my.example\",\"fields\":
[{\"name\":\"user_id\",\"type\":\"long\"},{\"name\":\"item_id\",\"type\":\"long\"},
{\"name\":\"behavior\",\"type\":\"string\"}]}",
  "type": "JSON",
  "properties": {}
}
```

2. Pulsar admin 命令行工具设置 topic 的 schema。

```
# 设置 schema
bin/pulsar-admin schemas upload -f ./schema.json topic_source
bin/pulsar-admin schemas upload -f ./schema.json topic_sink
```

```
# 检查 schema
bin/pulsar-admin schemas get topic_source
bin/pulsar-admin schemas get topic_sink
```

3. 作业示例。其中 flink 表的 database `public/default` 格式为 `tenant/namespace`，为 Pulsar 集群的默认值。

```
CREATE CATALOG `pulsar` WITH (
  'type' = 'pulsar-catalog',
  -- 'catalog-auth-plugin' = 'org.apache.pulsar.client.impl.auth.AuthenticationToken',
  -- 'catalog-auth-params' = 'token:eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJteS1zdXBlci11c2VyIn0.T-
Bi__yGCs1lxEwdUcDKBDuJsWLO0X9LrePhnuxxxxx',
  'catalog-admin-url' = 'http://pulsar:8080',
  'catalog-service-url' = 'pulsar://pulsar:6650'
);

INSERT INTO `pulsar`.`public/default`.`topic_sink` SELECT * FROM
`pulsar`.`public/default`.`topic_source`;
```

常见问题

事务未开启异常

```
java.lang.NullPointerException: You haven't enable transaction in Pulsar client.
```

参考 How to use transactions? 开启集群事务功能。

Pulsar sink exactly-once 默认写入消息，作业异常重启后，topic 数据无法消费

现象原因：这是由于作业异常重启前的事务可能未提交，当 topic 存在 OPEN 状态的事务时，阻塞了该 topic 后续写入的数据的读取。可用命令 `pulsar-admin transactions slow-transactions -t 1s` 查看 OPEN 状态的事务。当 OPEN 状态的事务提交或者回滚后，topic 后续写入的数据即可读取。
规避建议：使用 with 参数设置合适的事务超时时间（pulsar 默认为 3 小时），例如 `'pulsar.sink.transactionTimeoutMillis' = '120000'` 设置事务超时时间为 2 分钟。注意必须保证事务超时时间大于 checkpoint 时间间隔。

Pulsar source 在批量消息场景快照恢复失败

如果您遇到错误 `java.lang.IllegalArgumentException: We only support normal message id currently`，这是因为 Pulsar 写入端开启了批次写，当前 Pulsar source 不支持批量写消息的状态恢复。Oceanus Pulsar sink 默认不开启批量写。

```
Caused by: java.lang.IllegalArgumentException: We only support normal message id currently.
```

Pulsar source 消费起始位点与 subscription 的关系

- 如果 topic 不存在 subscription，则按起始位点对应的消息 ID 创建 subscription。
- 如果 topic 存在 subscription，则起始位点不起作用。
- 若不是从快照恢复，则从 subscription 的 cursor 开始消费。如果是从快照恢复，则从快照记录的消息 ID 的下一条消息开始消费（通过重置 subscription 的 cursor 的方式，参考 PulsarOrderedPartitionSplitReader#beforeCreatingConsumer）。

Pulsar source 无法使用 `NonDurable` 订阅模式

- PulsarSourceEnumerator#createSubscription 会先创建了 `Durable` 的 subscription。
- PulsarPartitionSplitReaderBase#createPulsarConsumer 再以 `NonDurable` 模式消费数据，报错 。`Durable subscription with the same name already exists` 。

参数值	必填	默认值	数据类型	描述

pulsar.consumer.subscriptionMode	可选	Durable	Enum	Pulsar 消息订阅模式，可选 `Durable` 、 `NonDurable` 。 `Durable` 模式下，cursor 是持久的，它保留消息并持久化当前位置。如果 broker 从故障中重新启动，它可以从持久存储（bookie）中恢复 cursor，以便消息可以从上次消费的位置继续消费。 `NonDurable` 模式下，一旦 broker 停止，cursor 将丢失且无法恢复，因此无法从上次消费的位置继续消费。更多信息参考 Subscription modes 。

MessageId 注意事项

参考 消息存储原理与 ID 规则，消息 ID 可比较大小，例如 `174:1:0 > 174:1:-1` 。

Pulsar source 根据消息 publish-time 消费数据失败

原因：如果连接的 broker 没有提供 topic 的 namespace 信息，那么根据 publish-time 获取消息 ID 的 Rest API 接口会返回。 `HTTP 307 Temporary Redirect` flink connector 中使用的 pulsar-client api 则会返回 `HTTP 500 Server Error` 错误，导致作业无法启动。可用 get-message-by-id 的 Rest API 接口查看错误情况。

```
## 1662480195714 为毫秒级的消息 publish-time
curl http://${adminUrl}:8080/admin/v2/persistent/public/default/${topic}/messageid/1662480195714
```

> **说明：**
> `source.stop.at-publish-time` 可以使用，因为不涉及到 publish-time 查找消息 ID 的 Rest API 调用。

建议：可通过适当调大 Rest API 调用失败的重试次数（选项 `pulsar.admin.requestRetries` ，默认 5）来规避该问题。

Pulsar source 设置了结束位点作业在消费完数据后没有停止

解决方案：请关闭 partition 自动探测功能，设置选项 `'pulsar.source.partitionDiscoveryIntervalMs' ='0'`

消息队列 CMQ

最近更新时间：2023-06-21 15:21:48

介绍

消息队列 CMQ（Cloud Message Queue，以下简称 CMQ）是基于腾讯自研消息引擎的分布式消息队列系统，可以用作数据源（Source）和数据目的（Sink）。用户可以把流数据导入到 CMQ 的某个 Queue 中，通过 Flink 算子进行处理后，输出到相同或不同 CMQ 示例的另一个 Queue。

版本说明

Flink 版本	说明
1.11	支持
1.13	支持
1.14	不支持
1.16	不支持

使用范围

CMQ 支持用作数据源表（Source），也可以作为 Tuple 数据流的目的表（Sink），暂不支持 Upsert 数据流。

DDL 定义

用作数据源（Source）

JSON 格式输入

```
CREATE TABLE `cmq_source_json_table` (
    `id` INT,
    `name`STRING,
    PRIMARY KEY (`id`) NOT ENFORCED -- 如果想做到数据去重的操作，则需要指定 PK，按照主键来区分不同的数据
) WITH (
    'connector' = 'cmq',                -- 必须为 'cmq'
    'hosts' = 'http://cmq-nameserver-vpc-gz.api.tencentyun.com',     -- cmq 所在地域的 nameServer
    'queue' = 'queue_name',          -- cmq 的队列名
    'secret-id' = 'xxxx',              -- 账号 secretId
    'secret-key' = 'xxxx',              -- 账号 secretKey
    'sign-method' = 'HmacSHA1',         -- 签名的方式
    'format' = 'json',                  -- 定义数据格式（JSON 格式）
    'json.fail-on-missing-field' = 'false',  -- 如果设置为 false，则遇到缺失字段不会报错。
    'json.ignore-parse-errors' = 'true',     -- 如果设置为 true，则忽略任何解析报错。
    'batch-size' = '16',                -- 批量消费消息的个数
    'request-timeout' = '5000ms',    -- 请求的超时时间
    'polling-wait-timeout'= '10s',   -- 获取不到数据情况下的等待时间
    'key-alive-timeout'= '5min'      -- 含 primary key 的消息，CMQ 去重的有效时间
);
```

CSV 格式输入

```
CREATE TABLE `cmq_source_csv_table` (
    `id` int,
    `name` STRING,
    PRIMARY KEY (`id`) NOT ENFORCED -- 如果想做到数据去重的操作，则需要指定 PK，按照这个主键来区分不同的数据
) WITH (
    'connector' = 'cmq',              -- 必须为 'cmq'
    'hosts' = 'http://cmq-nameserver-vpc-gz.api.tencentyun.com',     -- cmq 所在地域的 nameServer
```

```
    'queue' = 'queue_name',      -- cmq 的队列名
    'secret-id' = 'xxxx',          -- 账号 secretId
    'secret-key' = 'xxxx',          -- 账号 secretKey
    'sign-method' = 'HmacSHA1',      -- 签名的方式
    'format' = 'csv',               -- 定义数据格式（CSV 格式）
    'batch-size' = '16',            -- 批量消费消息的个数/批量发送消息的个数
    'request-timeout' = '5000ms',    -- 请求的超时时间
    'polling-wait-timeout'= '10s',  -- 获取不到数据情况下的等待时间
    'key-alive-timeout'= '5min'      -- 含 primary key 的消息，CMQ 去重的有效时间
);
```

用作数据目的（Sink）

JSON 格式输出

```
CREATE TABLE `cmq_sink_json_table` (
    `id` int,
    `name` STRING
) WITH (
    'connector' = 'cmq',                        -- 必须为 'cmq'
    'hosts' = 'http://cmq-nameserver-vpc-gz.api.tencentyun.com',    -- cmq 所在地域的 nameServer
    'queue' = 'queue_name',                -- cmq 的队列名
    'secret-id' = 'xxxx',                  -- 账号 secretId
    'secret-key' = 'xxxx',                 -- 账号 secretKey
    'sign-method' = 'HmacSHA1',                 -- 签名的方式
    'format' = 'json',                     -- 定义数据格式（JSON 格式）
    'json.fail-on-missing-field' = 'false',  -- 如果设置为 false，则遇到缺失字段不会报错。
    'json.ignore-parse-errors' = 'true',         -- 如果设置为 true，则忽略任何解析报错。
    'batch-size' = '16',                        -- 批量发送消息的个数
    'request-timeout' = '5000ms',               -- 请求的超时时间
    'retry-times' = '3',                        -- 发送消息的重试次数
    'max-block-timeout' = '0s'                  -- 批量发送数据的最大等待时间
);
```

CSV 格式输出

```
CREATE TABLE `cmq_sink_csv_table` (
    `id` int,
    `name` STRING
) WITH (
    'connector' = 'cmq',                        -- 必须为 'cmq'
    'hosts' = 'http://cmq-nameserver-vpc-gz.api.tencentyun.com',    -- cmq 所在地域的 nameServer
    'queue' = 'queue_name',                -- cmq 的队列名
    'secret-id' = 'xxxx',                    -- 账号 secretId
    'secret-key' = 'xxxx',                  -- 账号 secretKey
    'sign-method' = 'HmacSHA1',             -- 签名的方式
    'format' = 'csv',                       -- 定义数据格式（CSV 格式）
    'batch-size' = '16',                    -- 批量发送消息的个数
    'request-timeout' = '5000ms',           -- 请求的超时时间
    'retry-times' = '3',                    -- 发送消息的重试次数
    'max-block-timeout' = '0s'              -- 批量发送数据的最大等待时间
);
```

WITH 参数

通用 WITH 参数

参数值	必填	默认值	描述
connector	是	无	必须指定为 `'cmq'` 。
hosts	是	无	队列所在地域的 nameServer，具体的地址可以参考 TCP SDK 。
queue	是	无	cmq 对应的队列名。
secret-id	是	无	账号 secretId。
secret-key	是	无	账号 secretKey。
sign-method	否	HmacSHA1	账号的签名方式。
format	是	无	CMQ 消息的输入输出格式。目前支持 `'csv'` 、 `'json'` 。
batch-size	否	16	批量发送/接收消息的个数。
request-timeout	否	5000ms	请求的超时时间。
polling-wait-timeout	否	10s	获取不到数据情况下的等待时间。
key-alive-timeout	否	60s	含 primary key 的消息，CMQ 去重的有效时间。该参数设计是为了同一条数据不会重复消费，不保证全局不重复。
retry-times	否	3	发送消息的重试次数。
max-block-timeout	否	0s	批量发送数据的最大等待时间， `'0s'` 表示不等待，有数据就直接发送。

JSON 格式 WITH 参数

参数值	必填	默认值	描述
json.fail-on-missing-field	否	false	如果为 true，则遇到缺失字段时，会让作业失败。如果为 false（默认值），则只会把缺失字段设置为 null 并继续处理。
json.ignore-parse-errors	否	false	如果为 true，则遇到解析异常时，会把这个字段设置为 null 并继续处理。如果为 false，则会让作业失败。
json.timestamp-format.standard	否	SQL	指定 JSON 时间戳字段的格式，默认是 SQL（格式是 `yyyy-MM-dd HH:mm:ss.s{可选精度}` ）。也可以选择 ISO-8601，格式是 `yyyy-MM-ddTHH:mm:ss.s{可选精度}` 。

CSV 格式 WITH 参数

参数值	必填	默认值	描述
csv.field-delimiter	否	,	指定 CSV 字段分隔符，默认是半角逗号。
csv.line-delimiter	否	U&'\000A'	指定 CSV 的行分隔符，默认是换行符 `\n` ，SQL 中必须用 `U&'\000A'` 表示。如果需要使用回车符 `\r` ，SQL 中必须使用 `U&'\000D'` 表示。
csv.disable-quote-character	否	false	禁止字段包围引号。如果为 true，则 'csv.quote-character' 选项不可用。
csv.quote-character	否	"	字段包围引号，引号内部的作为整体看待。默认是 `''` 。
csv.ignore-parse-errors	否	false	忽略处理错误。对于无法解析的字段，会输出为 null。

csv.allow-comments	否	false	忽略 # 开头的注释行，并输出为空行（请务必将 csv.ignore-parse-errors 设为 true）。
csv.array-element-delimiter	否	;	数组元素的分隔符，默认是 `;` 。
csv.escape-character	否	无	指定转义符，默认禁用转义。
csv.null-literal	否	无	将指定的字符串看作 null 值。

代码示例

```
CREATE TABLE `cmq_source_json_table` (
    `id` int,
    `name` STRING,
    PRIMARY KEY (`id`) NOT ENFORCED -- 如果想做到数据去重的操作，则需要指定 PK，按照主键来区分不同的数据
) WITH (
    'connector' = 'cmq',                -- 必须为 'cmq'
    'hosts' = 'http://cmq-nameserver-vpc-gz.api.tencentyun.com',    -- cmq 所在地域的 nameServer
    'queue' = 'queue_name',         -- cmq 的队列名
    'secret-id' = 'xxxx',             -- 账号 secretId
    'secret-key' = 'xxxx',              -- 账号 secretKey
    'sign-method' = 'HmacSHA1',         -- 签名的方式
    'format' = 'json',                  -- 定义数据格式（JSON 格式）
    'json.fail-on-missing-field' = 'false',  -- 如果设置为 false，则遇到缺失字段不会报错。
    'json.ignore-parse-errors' = 'true',     -- 如果设置为 true，则忽略任何解析报错。
    'batch-size' = '16',                -- 批量消费消息的个数
    'request-timeout' = '5000ms',   -- 请求的超时时间
    'polling-wait-timeout'= '10s',  -- 获取不到数据情况下的等待时间
    'key-alive-timeout'= '5min'     -- 含 primary key 的消息，CMQ 去重的有效时间
);
CREATE TABLE `cmq_sink_json_table` (
    `id` int,
    `name` STRING
) WITH (
    'connector' = 'cmq',                        -- 必须为 'cmq'
    'hosts' = 'http://cmq-nameserver-vpc-gz.api.tencentyun.com',    -- cmq 所在地域的 nameServer
    'queue' = 'queue_name',                 -- cmq 的队列名
    'secret-id' = 'xxxx',                   -- 账号 secretId
    'secret-key' = 'xxxx',                  -- 账号 secretKey
    'sign-method' = 'HmacSHA1',                 -- 签名的方式
    'format' = 'json',                      -- 定义数据格式（JSON 格式）
    'json.fail-on-missing-field' = 'false',  -- 如果设置为 false，则遇到缺失字段不会报错。
    'json.ignore-parse-errors' = 'true',         -- 如果设置为 true，则忽略任何解析报错。
    'batch-size' = '16',                        -- 批量发送消息的个数
    'request-timeout' = '5000ms',               -- 请求的超时时间
    'retry-times' = '3',                        -- 发送消息的重试次数
    'max-block-timeout' = '0s'                  -- 批量发送数据的最大等待时间
);
insert into cmq_sink_json_table select * from cmq_source_json_table;
```

注意事项

CMQ 作为数据源（Source）使用的时候，需要考虑如下几点：

1. 数据去重操作，可以设置主键（PRIMARY KEY）来指定去重 key，在一定的时间范围内可以做到数据去重，这个时间范围用户可以自定义，需要注意这个时间范围设置的越长，消耗的内存越多。
2. 强烈建议设置 CMQ 的消息隐藏时间大于 Flink 任务的 checkpoint 时间，否则消费的消息会再次被消费到，影响消息处理性能。

消息队列 TDMQ RabbitMQ

最近更新时间：2023-06-21 15:21:48

介绍

消息队列 TDMQ RabbitMQ（TDMQ for RabbitMQ，以下简称 RMQ）是一款腾讯自主研发的消息队列服务，支持 AMQP 0-9-1 协议，完全兼容开源 RabbitMQ 的各个组件，可以用作数据目的（Sink）。用户可以通过 Flink 算子把流数据导入到 RMQ 的某个 Queue 中。

版本说明

Flink 版本	说明
1.11	支持
1.13	支持
1.14	不支持
1.16	不支持

使用范围

RMQ 支持用作数据目的表（Sink），暂不支持 Upsert 数据流。

DDL 定义

用作数据目的（Sink）

JSON 格式输出

```
CREATE TABLE `rmq_sink_json_table` (
    `id` int,
    `name` STRING
) WITH (
      'connector' = 'rabbitmq',                                -- 必须为 'rabbitmq'
    'host' = 'xxxx',                                               -- rabbitmq host
    'port' = 'xxxx',                                           -- rabbitmq 端口
    'vhost' = '/',                                                 -- 虚拟主机
    'username' = 'xxxx',                                           -- 用户名
    'password' = 'xxxx',                                       -- 用户密码
    'exchange' = 'exchange',                        -- 交换机名
      'routing-key' = 'Key',                                       -- 绑定 Key
      'format' = 'json',                                             -- 定义数据格式（JSON 格式）
    'json.fail-on-missing-field' = 'false',    -- 如果设置为 false，则遇到缺失字段不会报错
    'json.ignore-parse-errors' = 'true'        -- 如果设置为 true，则忽略任何解析报错
);
```

CSV 格式输出

```
CREATE TABLE `rmq_sink_csv_table` (
    `id` int,
    `name` STRING
) WITH (
      'connector' = 'rabbitmq',                                -- 必须为 'rabbitmq'
    'host' = 'xxxx',                                               -- rabbitmq host
    'port' = 'xxxx',                                           -- rabbitmq 端口
    'vhost' = '/',                                                 -- 虚拟主机
    'username' = 'xxxx',                                           -- 用户名
    'password' = 'xxxx',                                       -- 用户密码
```

```
    'exchange' = 'exchange',                    -- 交换机名
      'routing-key' = 'Key',                                  -- 绑定 Key
      'format' = 'csv'                                          -- 定义数据格式（CSV 格式）
);
```

WITH 参数

通用 WITH 参数

参数值	必填	默认值	描述
connector	是	无	必须指定为 `'rabbitmq'`。
host	是	无	队列所在 host。
port	是	5672	rabbitmq 端口。
vhost	是	/	虚拟主机。
username	是	guest	角色名。
password	是	guest	角色密码。
queue	否	无	队列名。
exchange	是	无	交换机名。
routing-key	否	无	交换机绑定 Key。
delivery-mode	否	1	消息是否持久化，1：非持久化 2：持久化。
expiration	否	86400000	消息过期时间，默认一天（单位：毫秒）。
network-recovery-interval	否	30s	网络恢复间隔。
automatic-recovery	否	true	rabbitmq 自动连接，默认自动连接。
topology-recovery	否	true	rabbitmq 拓扑恢复，默认自动恢复。
connection-timeout	否	30s	连接超时时间，默认30s。
requested-frame-max	否	0	最初请求的最大通信帧大小，以字节为单位。0意味着无限制。
requested-heartbeat	否	60s	请求心跳超时。
prefetch-count	否	0	服务器发送的消息的最大数量，0 意味着无限制。（仅 1.13 支持）
delivery-timeout	否	30s	提交队列超时时间。（仅 1.13 支持）
format	是	无	RMQ 消息的输入输出格式。目前支持 `'csv'`、`'json'`。

JSON 格式 WITH 参数

参数值	必填	默认值	描述
json.fail-on-missing-field	否	false	如果为 true，则遇到缺失字段时，会让作业失败。如果为 false（默认值），则只会把缺失字段设置为 null 并继续处理。
json.ignore-parse-errors	否	false	如果为 true，则遇到解析异常时，会把这个字段设置为 null 并继续处理。如果为 false，则会让作业失败。

json.timestamp-format.standard	否	SQL	指定 JSON 时间戳字段的格式，默认是 SQL（格式是 `yyyy-MM-dd HH:mm:ss.s{可选精度}` ）。也可以选择 ISO-8601，格式是 `yyyy-MM-ddTHH:mm:ss.s{可选精度}` 。

CSV 格式 WITH 参数

参数值	必填	默认值	描述
csv.field-delimiter	否	,	指定 CSV 字段分隔符，默认是半角逗号。
csv.line-delimiter	否	U&'\000A'	指定 CSV 的行分隔符，默认是换行符 `\n` ，SQL 中必须用 `U&'\000A'` 表示。如果需要使用回车符 `\r` ，SQL 中必须使用 `U&'\000D'` 表示。（仅 1.11 支持）
csv.disable-quote-character	否	false	禁止字段包围引号。如果为 true，则 'csv.quote-character' 选项不可用。
csv.quote-character	否	"	字段包围引号，引号内部的作为整体看待。默认是 `''` 。
csv.ignore-parse-errors	否	false	忽略处理错误。对于无法解析的字段，会输出为 null。
csv.allow-comments	否	false	忽略 # 开头的注释行，并输出为空行（请务必将 csv.ignore-parse-errors 设为 true）。
csv.array-element-delimiter	否	;	数组元素的分隔符，默认是 `;` 。
csv.escape-character	否	无	指定转义符，默认禁用转义。
csv.null-literal	否	无	将指定的字符串看作 null 值。

代码示例

```
-- 提示：请将参数替换成所属集群的信息
CREATE TABLE `rabbitmq_source_json_table` (`id` INT, `name` STRING) WITH (
  'connector' = 'jdbc',
  'url' = 'jdbc:mysql://host:port/database?rewriteBatchedStatements=true&serverTimezone=Asia/Shanghai',
  'table-name' = 'source_table_name',
  'username' = 'username',
  'password' = 'password'
);

CREATE TABLE `rabbitmq_sink_json_table` (`id` INT, `name` STRING) WITH (
  'connector' = 'rabbitmq',
  'host' = 'host',
  'port' = 'port',
  'vhost' = 'vhost',
  'username' = 'username',
  'password' = 'password',
  'queue' = 'queue-name',
  'exchange'='exchange',
  'routing-key'='key',
  'format' = 'json'
);
insert into rabbitmq_sink_json_table select * from rabbitmq_source_json_table;
```

注意

RMQ 作为数据目的表（Sink）使用的时候，需要注意往 RMQ 写入数据时有小概率会重复写入。

日志消费 CLS

最近更新时间：2025-05-08 19:47:22

介绍

CLS（Cloud Log Service）日志服务，可以作为 Oceanus 的数据源（Source），用户可以将 CLS 日志主题，通过 CLS Kafka协议消费 功能，将日志导出到 Oceanus，参与后续流计算。

版本说明

Flink 版本	说明
1.11	支持
1.13	支持
1.14	不支持
1.16	支持

使用范围

CLS 支持用作数据源表（Source）。

CLS日志主题

建立 CLS Source 表前，需要创建CLS日志主题，具体步骤如下：

1. 打开 CLS 日志主题 页面中选择**日志主题 > 创建日志主题**，创建日志主题。

2. 日志主题创建成功后，在列表页单击新建日志主题的**日志主题名称/ID** 进入到详情页面。

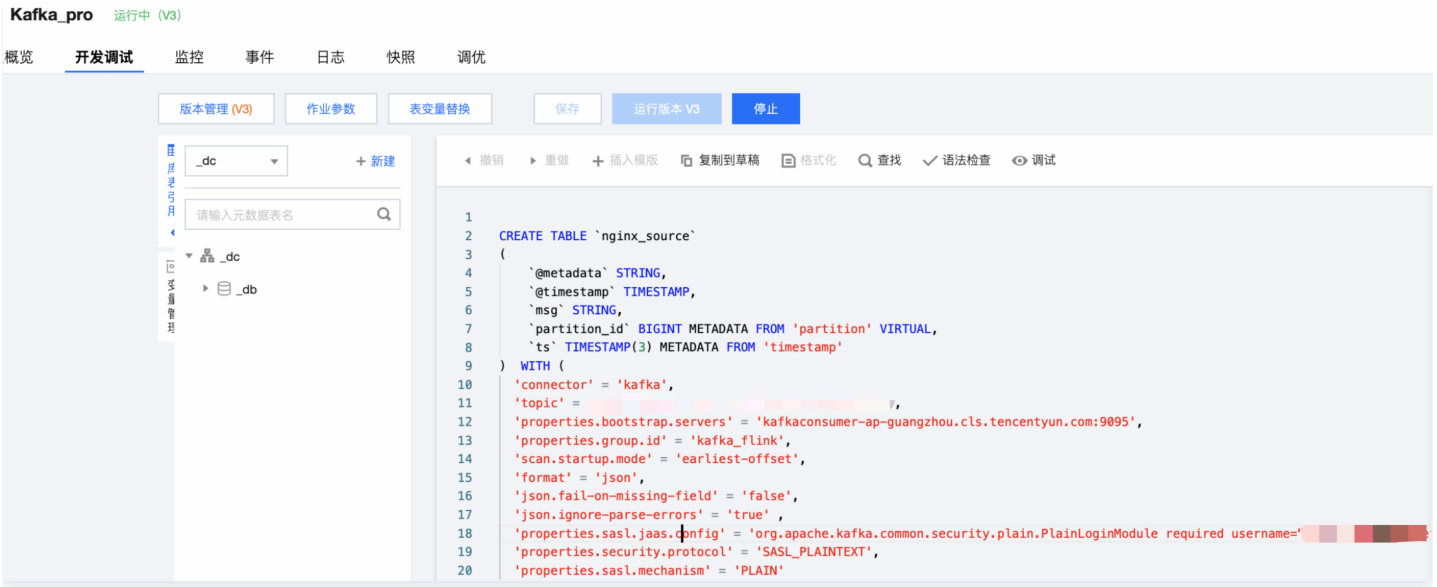
3. 在详情页，切换到**Kafka协议消费**页签，打开Kafka协议消费的功能。

开启后的状态如下：

若有不清楚的地方，可以参考 Kafka 协议消费 。

Oceanus消费CLS日志

在Oceanus控制台新建作业。




```
-- 建表语句如下
CREATE TABLE `nginx_source` (
  -- 日志中字段
  `@metadata` STRING,
  `@timestamp` TIMESTAMP,
```

```
  `agent` STRING,
  `ecs` STRING,
  `host` STRING,
  `input` STRING,
  `log` STRING,
  `message` STRING,
  `partition_id` BIGINT METADATA
  FROM
    'partition' VIRTUAL,
    -- kafka分区
    `ts` TIMESTAMP(3) METADATA
  FROM
    'timestamp'
) WITH (
  'connector' = 'kafka',
  -- cls kafka协议消费控制台给出的主题名称，例如XXXXXX-633a268c-XXXX-4a4c-XXXX-7a9a1a7baXXXX,可在控制台复制
  'topic' = '您的消费主题',
  -- 服务地址+端口，外网端口9096，内网端口9095,例子是内网消费，请根据您的实际情况填写
  'properties.bootstrap.servers' = 'kafkaconsumer-${region}.cls.tencentyun.com:9095',
  -- 请替换为您的消费组名称
  'properties.group.id' = '您的消费组名称',
  'scan.startup.mode' = 'earliest-offset',
  'format' = 'json',
  'json.fail-on-missing-field' = 'false',
  'json.ignore-parse-errors' = 'true',
  -- 用户名是日志集合ID，例如ca5cXXXXdd2e-4ac0af12-92d4b677d2c6
  -- 密码是用户的SecretId#SecretKey组合的字符串，比如AKIDWrwkHYYHjvqhz1mHVS8YhXXXX#XXXXuXtymIXT0Lac注意不要丢失#。建议使用子账号密钥,为子账号授权时,遵循最小权限原则,即子账号的访问策略中的action、resource都配置为最小范围,可以满足操作即可,注意jaas.config最后有;分号,不填写会报错.
  'properties.sasl.jaas.config' = 'org.apache.kafka.common.security.plain.PlainLoginModule required username="${logsetID}" password="${SecretId}#${SecretKey}";',
  'properties.security.protocol' = 'SASL_PLAINTEXT',
  'properties.sasl.mechanism' = 'PLAIN'
);
```

说明：

flink 版本 sasl 认证配置 对应包：

- 1.16版本以下：org.apache.kafka.common.security.plain.PlainLoginModule
- 1.16版本及以上：org.apache.flink.kafka.shaded.org.apache.kafka.common.security.plain.PlainLoginModule

注意事项

前提条件及相关限制请参见 Kafka 协议消费 。

数据库 MySQL CDC

最近更新时间：2025-06-09 11:02:02

介绍

MySQL 的 CDC 源表，支持对 MySQL 数据库的全量和增量读取，并保证 Exactly Once 语义。MySQL CDC 底层使用了 Debezium 来实现对变更数据流的实时抓取（Change Data Capture）。

MySQL CDC 1.x 工作机制

1. 获取一个全局读锁，从而阻塞住其他数据库客户端的写操作。
2. 开启一个可重复读语义的事务，来保证后续在同一个事务内读操作都是在一个一致性快照中完成的。
3. 读取 Binlog 的当前位置。
4. 读取连接器中配置的数据库和表的模式（schema）信息。
5. 释放全局读锁，允许其他的数据库客户端对数据库进行写操作。
6. 扫描全表，当全表数据读取完后，会从第3步中得到的 Binlog 位置获取增量的变更记录。

Flink 作业运行期间会周期性执行快照，记录下 Binlog 位置，当作业崩溃恢复时，便会从之前记录的 Binlog 点继续处理，从而保证 Exactly once 语义。

MySQL CDC 2.x 工作机制

1. MySQL 表需要有主键，如果是联合主键则会选择数据表中的第一个主键作为全量阶段的 splitKey，其用来将数据分为多个分片(Chunk)。
2. 全量阶段使用无锁算法，无需给表加锁。
3. 整个同步过程分为两个阶段，全量阶段并发读取分片数据，全量阶段结束之后进入增量阶段，整个过程都支持 Checkpoint 从而保证 Exactly once 语义。

版本说明

Flink 版本	说明
1.11	• 支持 MySQL 版本为 5.6
1.13	• 支持 MySQL 版本为 5.6, 5.7, 8.0.x • 默认配置，需要 source 表有主键。如果 source 表没有主键，需要 with 参数需要设置 `'scan.incremental.snapshot.enabled' = 'false'` • 内置 debezium 版本为1.5.4.Final
1.14	• 支持 MySQL 版本为 5.6, 5.7, 8.0.x • 默认配置，需要 source 表有主键。如果 source 表没有主键，需要 with 参数需要设置 `'scan.incremental.snapshot.enabled' = 'false'` • 内置 debezium 版本为1.5.4.Final
1.16	• 支持 MySQL 版本为 5.6, 5.7, 8.0.x • 默认配置，需要 source 表有主键。如果 source 表没有主键，需要 with 参数需要设置 `'scan.incremental.snapshot.enabled' = 'false'` • 内置 debezium 版本为1.5.4.Final

使用范围

MySQL CDC 只支持作为源表。

DDL 定义

```
CREATE TABLE `mysql_cdc_source_table` (
  `id` INT,
  `name` STRING,
  PRIMARY KEY (`id`) NOT ENFORCED -- 如果要同步的数据库表定义了主键，则这里也需要定义
```

```
) WITH (
  'connector' = 'mysql-cdc',       -- 固定值 'mysql-cdc'
  'hostname' = '192.168.10.22',    -- 数据库的 IP
  'port' = '3306',                 -- 数据库的访问端口
  'username' = 'debezium',         -- 数据库访问的用户名（需要提供 SHOW DATABASES、REPLICATION SLAVE、
REPLICATION CLIENT、SELECT 和 RELOAD 权限）
  'password' = 'hello@world!',     -- 数据库访问的密码
  'database-name' = 'YourDatabase',   -- 需要同步的数据库
  'table-name' = 'YourTable'       -- 需要同步的数据表名
);
```

WITH 参数

参数	说明	是否必填	备注
connector	源表类型	是	固定值为 `mysql-cdc`
hostname	MySQL 数据库的 IP 地址或者 Hostname	是	-
port	MySQL 数据库服务的端口号	否	默认值为3306
username	MySQL 数据库服务的用户名	是	有特定权限（包括 SELECT、RELOAD、SHOW DATABASES、REPLICATION SLAVE 和 REPLICATION CLIENT）的 MySQL 用户
password	MySQL 数据库服务的密码	是	-
database-name	MySQL 数据库名称	是	数据库名称支持正则表达式以读取多个数据库的数据
table-name	MySQL 表名	是	表名支持正则表达式以读取多个表的数据
server-id	数据库客户端的一个 ID	否	该 ID 必须是 MySQL 集群中全局唯一的。建议针对同一个数据库的每个作业都设置不同的 ID 范围值，例如 `5400-5405` 。默认会随机生成一个6400 - Integer.MAX_VALUE 的值
server-time-zone	数据库在使用的会话时区	否	例如 Asia/Shanghai，该参数控制了 MySQL 中的 TIMESTAMP 类型如何转成 STRING 类型
append-mode	开启 append 流模式	否	Flink1.13及以上版本支持, 例如：将 mysql-cdc 数据以 append 的方式同步到 hive
filter-duplicate-pair-records	过滤未在 Flink DDL 语句中定义的源表字段变更记录	否	例如 MySQL 源表有 a, b, c, d 四个字段，而用户在 Flink SQL 建表时只定义了 a, b 两个字段；开启该参数后，仅涉及 c 或 d 字段的变更记录会被忽略，不会输出到下游，可减少计算量和处理压力
scan.lastchunk.optimize.enable	对全量阶段的最后一个分片做重划分	否	如果全量同步期间，源表持续有大量写入和变更，则可能导致最后一个分片过大，引起 TaskManager OOM 崩溃重启。 开启本功能后（值设置为 true），Flink 会自动将过大的最后一个分片分成若干的小分片，提升作业的稳定性
debezium.min.row.count.to.stream.results	当表的条数大于该值时，会使用分批读取模式	否	默认值为1000。Flink 采用以下方式读取 MySQL 源表数据： • 全量读取：直接将整个表的数据读取到内存里。优点是速度快，缺点是会消耗对应大小的内存，如果源表数据量非常大，可能会有 OOM 风险 • 分批读取：分多次读取，每次读取一定数量的行数，直到读取完所有数据。优点是读取数据量比较大的表没有 OOM 风险，缺点是读取速度相对较慢

debezium.snapshot.fetch.size	在 Snapshot 阶段，每次读取 MySQL 源表数据行数的最大值	否	仅当分批读取模式时，该参数生效
debezium.skipped.operations	需要过滤的 oplog 操作。操作包括 c 表示插入，u 表示更新，d 表示删除。默认情况下，不跳过任何操作，以逗号分隔	否	-
scan.incremental.snapshot.enabled	增量快照	否	默认为 true
scan.incremental.snapshot.chunk.size	当读取表的快照时，表快照捕获的表的块大小(行数)	否	默认为 8096
scan.lazy-calculate-splits.enabled	全量阶段JM中数据分片懒加载避免数据量太大，分片数据太多导致JM OOM	否	默认为 true
scan.newly-added-table.enabled	动态加表	否	默认为 false
scan.split-key.mode	联合主键作为 splitkey 的模式	否	取值为 default / specific；其中 default 为默认逻辑，采用联合主键的第一个字段作为 split key；设置为 specific 需要设置 scan.split-key.specific-column 指定联合主键中的某个字段
scan.split-key.specific-column	指定联合主键中某个字段作为 splitkey	否	当 scan.split-key.mode 为 specific 时必填。取值为联合主键中某个字段名
scan.startup.mode	MySQL CDC 消费者可选的启动模式	否	合法的模式为 "initial"（默认），"earliest-offset"，"latest-offset"，"specific-offset" 和 "timestamp"
scan.startup.specific-offset.file	在 "specific-offset" 启动模式下，启动位点的 binlog 文件位置	否	-
scan.startup.specific-offset.pos	在 "specific-offset" 启动模式下，启动位点的 binlog 文件位置	否	-
scan.startup.specific-offset.gtid-set	在 "specific-offset" 启动模式下，启动位点的 GTID 集合	否	-
scan.startup.timestamp-millis	在 "timestamp" 启动模式下，启动时间的毫秒时间戳	否	-
scan.startup.specific-offset.skip-events	在指定的启动位点后需要跳过的事件数量	否	-
scan.startup.specific-offset.skip-rows	在指定的启动位点后需要跳过的数据行数量	否	-
connect.timeout	尝试连接到 MySQL 数据库服务器后在超时之前等待的最长时间	否	默认 30s
connect.max-retries	建立MySQL连接尝试最大的次数	否	默认 3
connection.pool.size	连接池大小	否	默认 20
jdbc.properties.*	自定义JDBC URL参数，例如：`'jdbc.properties.useSSL' = 'false'`	否	默认 20
heartbeat.interval	发送心跳事件的时间间隔，用于跟踪最新可用的binlog偏移量，一般用于解决慢表的问题(更新缓慢的数据表)	否	默认 20

debezium.*	Debezium 属性参数	否	从更细粒度控制 Debezium 客户端的行为。例如 `'debezium.snapshot.mode' = 'never'`，详情请参见 配置属性

可用元数据（Flink1.13 及以上版本可使用）

支持的元数据列：

列	数据类型	描述
database_name/meta.database_name	STRING NOT NULL	包含该 Row 的数据库名称
table_name/meta.table_name	STRING NOT NULL	包含该 Row 的表名称
op_ts/meta.op_ts	TIMESTAMP_LTZ(3) NOT NULL	Row 在数据库中进行更改的时间
meta.batch_id	BIGINT	binlog 的批 id
meta.is_ddl	BOOLEAN	是否 DDL 语句
meta.mysql_type	MAP	数据表结构
meta.update_before	ARRAY	未修改前字段的值
meta.pk_names	ARRAY	主键字段名
meta.sql	STRING	暂时为空
meta.sql_type	MAP	sql_type 表的字段到 Java 数据类型 ID 的映射
meta.ts	TIMESTAMP_LTZ(3) NOT NULL	收到该 ROW 并处理的当前时间
meta.op_type	STRING	数据库操作类型，例如 INSERT/DELETE 等
meta.file	STRING	全量阶段时为空。增量阶段时为数据来自的 binlog 文件名，例如 mysql-bin.000101
meta.pos	BIGINT	全量阶段时为0。增量阶段时为数据来自的 binlog 文件偏移，例如 143127802
meta.gtid	STRING	全量阶段时为 null。增量阶段时为数据对应的 gtid 值，例如 3d3c4464-c320-11e9-8b3a-6c92bf62891a:66486240

使用示例

```
CREATE TABLE `mysql_cdc_source_table` (
      `id` INT,
      `name` STRING,
      `database_name` string METADATA FROM 'database_name',
      `table_name`    string METADATA FROM 'table_name',
      `op_ts`         timestamp(3) METADATA FROM 'op_ts',
      `op_type` string METADATA FROM 'meta.op_type',
      `batch_id` bigint METADATA FROM 'meta.batch_id',
      `is_ddl` boolean METADATA FROM 'meta.is_ddl',
      `update_before` ARRAY<MAP<STRING, STRING>> METADATA FROM 'meta.update_before',
      `mysql_type` MAP<STRING, STRING> METADATA FROM 'meta.mysql_type',
      `pk_names` ARRAY<STRING> METADATA FROM 'meta.pk_names',
      `sql` STRING METADATA FROM 'meta.sql',
      `sql_type` MAP<STRING, INT> METADATA FROM 'meta.sql_type',
```

```
    `ingestion_ts` TIMESTAMP(3) METADATA FROM 'meta.ts',
    PRIMARY KEY (`id`) NOT ENFORCED -- 如果要同步的数据库表定义了主键，则这里也需要定义
) WITH (
    'connector' = 'mysql-cdc',      -- 固定值 'mysql-cdc'
    'hostname' = '192.168.10.22',   -- 数据库的 IP
    'port' = '3306',                -- 数据库的访问端口
    'username' = 'debezium',        -- 数据库访问的用户名（需要提供 SHOW DATABASES、REPLICATION SLAVE、
REPLICATION CLIENT、SELECT 和 RELOAD 权限）
    'password' = 'hello@world!',    -- 数据库访问的密码
    'database-name' = 'YourDatabase',   -- 需要同步的数据库
    'table-name' = 'YourTable'      -- 需要同步的数据表名
);
```

MySQL 分库分表读取方式

目前 Oceanus 已支持 MySQL 分库分表的读取。

如果 MySQL 是一个分库分表的数据库，分成了 A_1、A_2、A_3 ... 等多个表，**且所有表的 schema 一致**，则可以通过 table-name 选项，指定一个正则表达式来匹配读取多张表，例如设置 table-name 为 **A_.***，监控所有 **A_** 前缀的表。**database-name 选项也支持该功能**。

> **说明**
> 如果 database-name 和 table-name 设置为正则匹配的话，需要使用 `()` 把正则式包围起来。

类型映射

MySQL 的 CDC 和 Flink 字段类型对应关系如下：

MySQL type	Flink SQL type	NOTE
TINYINT	TINYINT	-
SMALLINT TINYINT UNSIGNED TINYINT UNSIGNED ZEROFILL	SMALLINT	-
INT MEDIUMINT SMALLINT UNSIGNED SMALLINT UNSIGNED ZEROFILL	INT	-
BIGINT INT UNSIGNED INT UNSIGNED ZEROFILL MEDIUMINT UNSIGNED MEDIUMINT UNSIGNED ZEROFILL	BIGINT	-
BIGINT UNSIGNED BIGINT UNSIGNED ZEROFILL SERIAL	DECIMAL(20, 0)	-
FLOAT FLOAT UNSIGNED FLOAT UNSIGNED ZEROFILL	FLOAT	-
REAL REAL UNSIGNED REAL UNSIGNED ZEROFILL DOUBLE DOUBLE UNSIGNED DOUBLE UNSIGNED ZEROFILL DOUBLE PRECISION DOUBLE PRECISION UNSIGNED	DOUBLE	-

DOUBLE PRECISION UNSIGNED ZEROFILL		
NUMERIC(p, s) NUMERIC(p, s) UNSIGNED NUMERIC(p, s) UNSIGNED ZEROFILL DECIMAL(p, s) DECIMAL(p, s) UNSIGNED DECIMAL(p, s) UNSIGNED ZEROFILL FIXED(p, s) FIXED(p, s) UNSIGNED FIXED(p, s) UNSIGNED ZEROFILL where p <= 38	DECIMAL(p, s)	–
NUMERIC(p, s) NUMERIC(p, s) UNSIGNED NUMERIC(p, s) UNSIGNED ZEROFILL DECIMAL(p, s) DECIMAL(p, s) UNSIGNED DECIMAL(p, s) UNSIGNED ZEROFILL FIXED(p, s) FIXED(p, s) UNSIGNED FIXED(p, s) UNSIGNED ZEROFILL where 38 < p <= 65	STRING	MySQL 中 DECIMAL 数据类型的精度最高为 65，而 Flink 中 DECIMAL 的精度限制为 38。 所以如果您定义一个精度大于38的十进制列，您应该把它映射到STRING，以避免精度损失
BOOLEAN TINYINT(1) BIT(1)	BOOLEAN	–
DATE	DATE	–
TIME [(p)]	TIME [(p)]	–
TIMESTAMP [(p)] DATETIME [(p)]	TIMESTAMP [(p)]	–
CHAR(n)	CHAR(n)	–
VARCHAR(n)	VARCHAR(n)	–
BIT(n)	BINARY(⌈n/8⌉)	–
BINARY(n)	BINARY(n)	–
VARBINARY(N)	VARBINARY(N)	–
TINYTEXT TEXT MEDIUMTEXT LONGTEXT	STRING	–
TINYBLOB BLOB MEDIUMBLOB LONGBLOB	BYTES	对于 MySQL 中的 BLOB 数据类型，仅支持长度不大于 2,147,483,647(2 ** 31 – 1) 的 blob
YEAR	INT	–
ENUM	STRING	–
JSON	STRING	JSON 数据类型会在 Flink 中转换为 JSON 格式的 STRING
SET	ARRAY<STRING>	由于 MySQL 中的 SET 数据类型是一个可以有零个或多个值的字符串对象，所以它应该总是映射到一个字符串数组

GEOMETRY POINT LINESTRING POLYGON MULTIPOINT MULTILINESTRING MULTIPOLYGON GEOMETRYCOLLECTION	STRING	MySQL 中的空间数据类型会被转换成固定 Json 格式的 STRING

代码示例

```
CREATE TABLE `mysql_cdc_source_table` (
  `id` INT,
  `name` STRING,
  PRIMARY KEY (`id`) NOT ENFORCED -- 如果要同步的数据库表定义了主键，则这里也需要定义
) WITH (
  'connector' = 'mysql-cdc',       -- 固定值 'mysql-cdc'
  'hostname' = '192.168.10.22',    -- 数据库的 IP
  'port' = '3306',                 -- 数据库的访问端口
  'username' = 'debezium',         -- 数据库访问的用户名（需要提供 SHOW DATABASES、REPLICATION SLAVE、
REPLICATION CLIENT、SELECT 和 RELOAD 权限）
  'password' = 'hello@world!',     -- 数据库访问的密码
  'database-name' = 'YourDatabase',    -- 需要同步的数据库
  'table-name' = 'YourTable'       -- 需要同步的数据表名
);

CREATE TABLE `print_table` (
  `id` INT,
  `name` STRING
) WITH (
 'connector' = 'print'
);
insert into print_table select * from mysql_cdc_source_table;
```

注意事项

Checkpoint 相关

- 使用 CDC 1.0 ('scan.incremental.snapshot.enabled' = 'false') 时，需要做的额外参数配置。
  由于 CDC 1.0 读取全量数据阶段无法做 checkpoint，当需要同步的表较多、数据较大时，可能会导致多次快照失败，从而引发作业失败。可以通过作业高级参数 `execution.checkpointing.tolerable-failed-checkpoints: 100` 调整 checkpoint 失败的容忍次数。
- 使用 CDC 2.0，且作业的默认并行度大于 1 时，必须开启 checkpoint。
  CDC 读取完全量数据后，需要等待一个 checkpoint 完成后才能进入增量阶段。

关于使用 CDC 1.0 的风险告知

当表没有主键时，只能通过使用 WITH 参数 `'scan.incremental.snapshot.enabled' = 'false'` 开启 CDC 1.0 模式，会存在以下风险：

1. 默认会使用 FTWRL (flush table with read lock)。
2. 虽然 FTWRL 只会持有短暂的时间，但由于 FTWRL 的机制，可能会导致锁库。
3. FTWRL 可能会出现的情况如下：
   - 会等待正在执行的 update/select 操作完成。
   - 在等待 update/select 完成的期间，会导致数据库不可用，即阻塞新加入的 SELECT 查询，这是 MySQL Query Cache 机制导致的。

如果同时启动多个不同的 MySQL CDC 1.0 的 source，大概率会碰到上述情况。

用户权限

用于同步的源数据库的用户必须拥有以下权限 SHOW DATABASES、REPLICATION SLAVE、REPLICATION CLIENT、SELECT 和 RELOAD。

```
GRANT SELECT, SHOW DATABASES, REPLICATION SLAVE, REPLICATION CLIENT ON *.* TO 'user' IDENTIFIED BY
'password';
FLUSH PRIVILEGES;
```

全局读锁

上述的工作原理中，MySQL CDC 1.x 中可以看到第一步就会获取一个全局读锁，用于获取 schema 和 Binlog 位置。这里会阻塞其他客户端的写入，因此仍可能对线上业务造成影响。 若可以接受 At Least Once 语义，可通过设置 WITH 参数 `'debezium.snapshot.locking.mode' = 'none'` 跳过这个阶段。

联合主键设置

例如，下面的 DDL 设置了 `index1` 、 `index2` 、 `index3` 、 `index4` 4个字段为联合主键索引，要和 `PRIMARY KEY` 定义保持一致，顺序不会影响正常的同步。

```
CREATE TABLE db_order_dim (
  `index1` STRING,
  `index2` STRING,
  `index3` STRING,
  `index4` STRING,
  `field5` STRING,
  `field6` STRING,
  PRIMARY KEY(`index1`, `index2`, `index3`, `index4`) NOT enforced
) WITH (
  ...
);
```

server-id 定义

不建议显式指定 `server-id` ，因为 Oceanus 平台会为每个表自动生成随机 `server-id` 值（范围是 `6400 - 2147483647` ），以避免不同作业读取同一个库可能出现的 `server-id` 冲突问题。
如果必须要手动指定 `server-id` 值，对于 CDC 2.x 版本，建议设置为范围值，例如 `5400-5405` ，因为每个并行读取器应该有一个唯一的服务器 ID，所以 `server-id` 必须是 `5400-5405` 这样的范围，且范围必须大于并行度。但对于 CDC 1.x 版本，只能设置单个 `server-id` 值，不支持范围设定。
指定 `server-id` 有以下两种方式：

1. `mysql-cdc` DDL 的 WITH 参数中指定。
2. 使用 SQL Hints 来指定 `server-id` 。

```
SELECT * FROM source_table /*+ OPTIONS('server-id'='5401-5404') */ ;
```

设置 MySQL 会话超时

全量阶段读取大型数据库的时候可能会超时，您可以对 MySQL 做一些配置来避免这个问题。

- `interactive_timeout` 服务器在关闭交互式连接之前等待其活动的秒数。请参见 MySQL 文档 。
- `wait_timeout` 服务器在关闭非交互式连接之前等待其活动的秒数。请参见 MySQL 文档 。

增量阶段也有可能因为 TaskManager 负载过高导致心跳失效，服务端主动断开连接（EOFException）。可以在 MySQL 服务端执行下面的 SQL 语句，调大超时时间：

```
SET GLOBAL slave_net_timeout = 120;
SET GLOBAL thread_pool_idle_timeout = 120;
```

JobManager 关键日志说明

在 CDC 2.x 模式下，每个表的同步都要经历**分片划分**、**全量快照同步**、**增量修正**、**纯增量同步**等阶段。由于前三个阶段资源占用大、耗时久，Oceanus 在日志和指标方面做了加强，协助用户洞察和分析作业的运行情况。

1. 分片划分与分配

开始划分分片：搜索 `into chunks` 关键字，例如 `Start splitting table cdc_basic_source.random_source_1 into chunks` 或者 `Start lazily splitting table cdc_basic_source.random_source_1 into chunks`
结束划分分片: 搜索 `chunks, time cost` 关键字，例如 `Split table cdc_basic_source.random_source_1 into 14 chunks, time cost: 994ms.`

2. 全量快照同步

查看全量分片分配进度：开启 `DEBUG` 级别日志，搜索 `Current assigned splits for` 关键字，即可查看每个表的总分片数和分配进度。
结束全量分片分配：搜索 `finished. Total split number` 关键字，例如 `Split assignment for cdc_basic_source.random_source_1 finished. Total split number: 14`
结束全量阶段：搜索 `Assigner status changes from INITIAL_ASSIGNING to INITIAL_ASSIGNING_FINISHED` 日志。

3. 增量修正

开始增量修正阶段：搜索 `Initial assigning finished as there are no more splits. Creating binlog split` 或者 `Newly added assigning finished as there are no more splits. Waking up binlog reader` 日志。

4. 纯增量同步

参见下一节 “TaskManager 关键日志说明”。

TaskManager 关键日志说明

进入纯增量同步阶段：搜索 `has entered pure binlog phase` 日志。例如 `Table cdc_basic_source.random_source_2 has entered pure binlog phase.`
表的 Schema 变更：搜索 `Received schema change event` 日志。
EOFException 异常：如果作业重启，且提示异常是 `EOFException`，则可根据提示语，调整 MySQL 服务端的超时参数。同时也可以减少总并行度，升级每个 TaskManager 的规格，以减少内存和 CPU 压力过高导致的超时问题的发生概率。

监控指标说明

Oceanus 为 MySQL CDC Connector 也增加了很多实用的统计指标。单击 Flink UI 的运行图中的 MySQL CDC 源算子，即可搜索并查看指标：
logpos：获取当前消费到的 Binlog 位点，可以协助定位消费卡顿等问题。
numberOfInsertRecords：获取输出的 +I 消息数。
numberOfDeleteRecords：获取输出的 −D 消息数。
numberOfUpdateBeforeRecords：获取输出的 −U 消息数。
numberOfUpdateAfterRecords：获取输出的 +U 消息数。

数据库 TDSQL-MySQL

最近更新时间：2023-06-21 15:21:49

介绍

tdsql-subscribe connector 是针对腾讯云 TDSQL-MySQL 数据订阅的专有 connector，通过 数据订阅 功能接入 TDSQL-MySQL 的增量 binlog 数据，使用前请确保数据订阅任务已经配置成功。

> **注意**
> 数据库 TDSQL Connector 目前处于 Beta 版本，如有需求请您 工单 联系我们。

版本说明

Flink 版本	说明
1.11	不支持
1.13	支持
1.14	不支持
1.16	不支持

使用范围

tdsql-subscribe connector 支持用作数据源表（Source），不可以作为数据流的目的表（Sink）。

DDL 定义

当 tdsql-subscribe connector 作为 source 时，with 参数大部分与 Kafka connector 参数类似，连接参数都可以在订阅任务中找到。
值得注意的是，在使用 tdsql-subscribe connector 时， `format` 必须指定为 `protobuf` 格式，因为数据订阅中，发送到 Kafka 的消息格式为 `protobuf`； 相较于通常使用的 Kafka connector，tdsql-subscribe connector 会多了一些认证信息，认证信息也是源于订阅任务。

用作数据源（Source）

protobuf 格式输入

```
CREATE TABLE `DataInput` (
      `id` INT,
      `name` VARCHAR,
       `age` INT
) WITH (
    'connector' = 'tdsql-subscribe',   -- 注意选择对应的内置 Connector
    'tdsql.database.name' = 'test_case_2022_06_0*', -- 对订阅消息进行过滤，消费数据库名满足
test_case_2022_06_0* 正则的订阅数据
    'tdsql.table.name' = 'test_0*', -- 对订阅消息进行过滤，消费数据表满足 test_0* 正则的订阅数据
    'topic' = 'topic-subs-5xop97nffk-tdsqlshard-xxx',  -- 替换为订阅任务消费的 Topic
    'scan.startup.mode' = 'latest-offset', -- 可以是 latest-offset / earliest-offset / specific-offsets
/ group-offsets 的任何一种
    'properties.bootstrap.servers' = 'guangzhou-kafka-2.cdb-dts.tencentcs.com.cn:3212',  -- 替换为您的订阅
任务 Kafka 连接地址
    'properties.group.id' = 'consumer-grp-subs-xxx-kk',
    'format' = 'protobuf', -- 只能是protobuf格式
    'properties.security.protocol'='SASL_PLAINTEXT', -- 认证协议
    'properties.sasl.mechanism'='SCRAM-SHA-512', -- 认证方式
    'properties.sasl.jaas.config'='org.apache.kafka.common.security.scram.ScramLoginModule required
username="account-subs-xxx-username" password="psw";' --用户名和密码
);
CREATE TABLE `jdbc_upsert_sink_table` (
```

```
    id INT PRIMARY KEY NOT ENFORCED,
    name STRING,
    age INT
) WITH (
    -- 指定数据库连接参数
    'connector' = 'jdbc',
    'url' = 'jdbc:mysql://172.28.28.138:3306/testdb', -- 请替换为您的实际 MySQL 连接参数
    'table-name' = 'sink', -- 需要写入的数据表
    'username' = 'user',       -- 数据库访问的用户名（需要提供 INSERT 权限）
    'password' = 'psw'  -- 数据库访问的密码
);
INSERT INTO jdbc_upsert_sink_table SELECT * FROM DataInput;
```

WITH 参数

参数值	必填	默认值	描述
connector	是	无	固定值为 `'tdsql-subscribe'`
topic	是	无	要读的 Kafka Topic 名
properties.bootstrap.servers	是	无	逗号分隔的 Kafka Bootstrap 地址
properties.group.id	是	无	Kafka 消费时的 Group ID
format	是	无	Kafka 消息的输入格式。目前只支持 `protobuf`
scan.startup.mode	否	group-offsets	Kafka consumer 的启动模式。 • 可以是 `latest-offset`、`earliest-offset`、`specific-offsets`、`group-offsets`、`timestamp` 的任何一种 `'scan.startup.specific-offsets' = 'partition:0,offset:42;partition:1,offset:300'`，使用 `'specific-offsets'` 启动模式时需要指定每个 partition 对应的 offsets。 • `'scan.startup.timestamp-miles' = '1631588815000'`，使用 `'timestamp'` 启动模式时需要指定启动的时间戳（单位毫秒）
scan.startup.specific-offsets	否	无	如果 `scan.startup.mode` 的值为 `'specific-offsets'`，则必须使用本参数指定具体起始读取的偏移量。例如 `'partition:0,offset:42;partition:1,offset:300'`
scan.startup.timestamp-millis	否	无	如果 `scan.startup.mode` 的值为 `'timestamp'`，则必须使用本参数来指定开始读取的时间点（毫秒为单位的 Unix 时间戳）
tdsql.database.name	否	无	tdsql 数据库名称，配置此参数，可以消费 tdsql 指定数据库的 binlog，前提是订阅任务包含了此数据库的 binlog。参数支持正则，例如 `test_case_2022_06_0*`
tdsql.table.name	否	无	tdsql 数据表名称，配置此参数，可以消费 tdsql 指定数据表的 binlog，前提是订阅任务包含了此数据表的 binlog。参数支持正则，例如 `test_0*`，`test_1,test_2`

说明

如果需要配置 `tdsql.database.name` 或 `tdsql.table.name` 参数，订阅任务建议配置 **订阅全实例**，如果有多个 Oceanus 任务消费不同 tdsql 数据库表时，多个 Oceanus 任务需要使用订阅任务的不同消费组，可在订阅任务中创建不同消费组。

注意事项

1. 如果订阅任务配置了多库多表或同库多表，需要保证表结构是相同的，才能正确接入订阅任务的数据。

2. 源表中文编码目前仅支持 `utf8` 和 `gbk` 两种。

数据库 PostgreSQL CDC

最近更新时间：2025-06-09 11:02:02

介绍

Postgres 的 CDC 源表（即 Postgres 的流式源表）用于依次读取 PostgreSQL 数据库全量快照数据和变更数据，保证不多读也不少读一条数据。即使发生故障，也能采用 Exactly Once 方式处理。

版本说明

Flink 版本	说明
1.11	• 支持
1.13	• 支持 • 内置 debezium 版本为1.9.7. Final
1.14	• 不支持
1.16	• 支持 • 内置 debezium 版本为1.9.7. Final

使用范围

PostgreSQL CDC 只支持作为源表。支持的 PostgreSQL 版本为9.6及以上版本。

DDL 定义

```
CREATE TABLE postgres_cdc_source_table (
  id INT,
  name STRING,
  PRIMARY KEY (`id`) NOT ENFORCED -- 如果要同步的数据库表定义了主键，则这里也需要定义
) WITH (
  'connector' = 'postgres-cdc',             -- 固定值 'postgres-cdc'
  'hostname' = 'yourHostname',              -- 数据库的 IP
  'port' = '5432',                          -- 数据库的访问端口
  'username' = 'yourUserName',              -- 数据库访问的用户名（需要提供 REPLICATION、LOGIN、SCHEMA、
DATABASE、SELECT权限）
  'password' = 'yourPassWord',              -- 数据库访问的密码
  'database-name' = 'yourDatabaseName',     -- 需要同步的数据库
  'schema-name' = 'yourSchemaName',         -- 需要同步的数据表所属schema （支持正则表达式）
  'table-name' = 'yourTableName',           -- 需要同步的数据表名 （支持正则表达式）
  'slot.name' = 'customslotname'            -- 定义一个唯一slot名称，可以包含小写字母、数字和下划线字符
  'scan.incremental.snapshot.enabled' = 'true'  -- 开启增量快照，默认关闭
);
```

WITH 参数

参数	说明	是否必填	备注
connector	源表类型	是	固定值为 `postgres-cdc`
hostname	Postgres 数据库的 IP 地址或者 Hostname	是	-
username	Postgres 数据库服务的用户名	是	有特定权限（包括 REPLICATION、LOGIN、SCHEMA、DATABASE、SELECT）的 Postgres 用户
password	Postgres 数据库服务的密码	是	-
database-name	Postgres 数据库名称	是	-

schema-name	Postgres Schema 名称	是	Schema 名称支持正则表达式以读取多个 Schema 的数据
table-name	Postgres 表名	是	表名支持正则表达式以读取多个表的数据
port	Postgres 数据库服务的端口号	否	默认值为5432
slot.name	Postgres 复制槽名称	是	只能包含小写字母、数字和下划线
scan.incremental.snapshot.enabled	是否开启增量快照	否	默认值为false
decoding.plugin.name	Postgres Logical Decoding 插件名称	否	根据 Postgres 服务上安装的插件确定。支持的插件列表如下： • decoderbufs（默认值） • wal2json • wal2json_rds • wal2json_streaming • wal2json_rds_streaming • pgoutput
debezium.	Debezium 属性参数	否	从更细粒度控制 Debezium 客户端的行为。详情请参见 配置属性

类型映射

Postgres CDC 和 Flink 字段类型对应关系如下：

Postgres CDC 字段类型	Flink 字段类型
SMALLINT	SMALLINT
INT2	
SMALLSERIAL	
SERIAL2	
INTEGER	INT
SERIAL	
BIGINT	BIGINT
BIGSERIAL	
REAL	FLOAT
FLOAT4	
FLOAT8	DOUBLE
DOUBLE PRECISION	
NUMERIC(p, s)	DECIMAL(p, s)
DECIMAL(p, s)	
BOOLEAN	BOOLEAN
DATE	DATE
TIME [(p)] [WITHOUT TIMEZONE]	TIME [(p)] [WITHOUT TIMEZONE]
TIMESTAMP [(p)] [WITHOUT TIMEZONE]	TIMESTAMP [(p)] [WITHOUT TIMEZONE]

版权所有：腾讯云计算（北京）有限责任公司

CHAR(n)	STRING
CHARACTER(n)	
VARCHAR(n)	
CHARACTER VARYING(n)	
TEXT	
BYTEA	BYTES

代码示例

```
CREATE TABLE postgres_cdc_source_table (
  id INT,
  name STRING,
  PRIMARY KEY (`id`) NOT ENFORCED -- 如果要同步的数据库表定义了主键，则这里也需要定义
) WITH (
  'connector' = 'postgres-cdc',              -- 固定值 'postgres-cdc'
  'hostname' = 'yourHostname',               -- 数据库的 IP
  'port' = '5432',                           -- 数据库的访问端口
  'username' = 'yourUserName',               -- 数据库访问的用户名（需要提供 REPLICATION、LOGIN、SCHEMA、
DATABASE、SELECT权限）
  'password' = 'yourPassWord',               -- 数据库访问的密码
  'database-name' = 'yourDatabaseName',      -- 需要同步的数据库
  'schema-name' = 'yourSchemaName',          -- 需要同步的数据表所属schema （支持正则表达式）
  'table-name' = 'yourTableName',            -- 需要同步的数据表名 （支持正则表达式）
  'debezium.slot.name' = 'customslotname'    -- 定义一个唯一slot名称，可以包含小写字母、数字和下划线字符
);

CREATE TABLE `print_table` (
  `id` INT,
  `name` STRING
) WITH (
 'connector' = 'print'
);
insert into print_table select * from postgres_cdc_source_table;
```

注意事项

用户权限

用来同步的用户至少具有 REPLICATION、LOGIN、SCHEMA、DATABASE、SELECT 权限。

```
CREATE ROLE debezium_user REPLICATION LOGIN;
GRANT USAGE ON SCHEMA schema_name TO debezium_user;
GRANT USAGE ON DATABASE schema_name TO debezium_user;
GRANT SELECT ON scheam_name.table_name, scheam_name.table_name TO debezium_user;
```

数据库 MongoDB CDC

最近更新时间：2025-06-09 11:02:02

介绍

MongoDB 的 CDC 源表（即 MongoDB 的流式源表），Connector 会自动跟踪 MongoDB 副本集或分片集群，以获取数据库和集合中的文档更改。

版本说明

Flink 版本	说明
1.11	• 不支持
1.13	• 支持 • 内置的 debezium 版本是1.5.4. Final
1.14	• 支持 • 内置的 debezium 版本是1.5.4. Final
1.16	• 支持 • 内置的 debezium 版本是1.5.4. Final

使用范围

MongoDB CDC 只支持作为源表，MongoDB CDC 支持4.0、4.2、5.0版本，MongoDB 集群必须是副本集或者分片集群。

DDL 定义

```
-- register a MongoDB table 'products' in Flink SQL
CREATE TABLE mongo_cdc_source_table (
  _id STRING, // must be declared
  name STRING,
  weight DECIMAL(10,3),
  tags ARRAY<STRING>, -- array
  price ROW<amount DECIMAL(10,2), currency STRING>, -- embedded document
  suppliers ARRAY<ROW<name STRING, address STRING>>, -- embedded documents
  PRIMARY KEY(_id) NOT ENFORCED
) WITH (
  'connector' = 'mongodb-cdc',
  'hosts' = 'localhost:27017,localhost:27018,localhost:27019',
  'username' = 'flinkuser',
  'password' = 'flinkpw',
  'database' = 'inventory',
  'collection' = 'products'
);
```

WITH 参数

参数	说明	是否必填	备注
connector	源表类型	是	固定值为 `mongodb-cdc`
hosts	MongoDB 数据库的 IP 端口对	是	–
username	MongoDB 数据库服务的用户名	是	–
password	MongoDB 数据库服务的密码	是	–

database	MongoDB 数据库名称	是	正则表达式下要求 MongoDB 4.0版本及以上
collection	MongoDB Collection 名称	是	正则表达式下要求 MongoDB 4.0版本及以上，且表达式中必须包含数据库名称，例如要包含 default库中所有"test_"开头的表，需要配置为"default.test_.*"。
filter-duplicate-pair-records	过滤未在 Flink DDL 语句中定义的源表字段变更记录	否	例如 MongoDB 源表有 a, b, c, d 四个字段，而用户在 Flink SQL 建表时只定义了 a, b 两个字段；开启该参数后，仅涉及 c 或 d 字段的变更记录会被忽略，不会输出到下游，可减少计算量和处理压力
connection.options	MongoDB 的 连接选项 。有多个时，使用&连接，例如 `replicaSet=test&connectTimeoutMS=300000`	否	-
errors.tolerance	是否忽略错误记录，接受 none 或者 all。如果设置为 all， 忽略错误记录	否	none
errors.log.enable	是否需要把错误操作打印到日志文件	否	默认值为 true
copy.existing	是否复制库中原有的数据，如果在复制期间对数据有更改，会在数据复制完成后应用更改	否	默认值为 true
copy.existing.pipeline	当复制原有数据的时候，可以通过这个参数设置筛选条件。例如 `[{"$match": {"closed": "false"}}]` ，只会复制 closed 为 false 的 记录。用法参考 $match (aggregation)	否	-
copy.existing.max.threads	执行数据复制时要使用的线程数	否	默认值为 Processors Count
copy.existing.queue.size	复制数据时要使用的队列的最大大小	否	默认值为16000
poll.max.batch.size	每次拉取数据的最大数量。默认情况下，1.5秒的拉取间隔下，最多拉取1000条变更数据	否	默认值为1000
poll.await.time.ms	拉取数据的时间间隔。默认情况下，1.5秒的拉取间隔下，最多拉取1000条变更数据	否	默认值为1500
heartbeat.interval.ms	发送心跳消息时间间隔，以毫秒为单位。使用0禁用	否	默认值为0

说明

Note：当数据流变化慢的时候，建议把 heartbeat.interval.ms 设置为一个合适的值，心跳会推送 resumeToken，防止当 Flink job 从 checkpoint 或者 savepoint 恢复的时候，resumeToken 已经过期。

类型映射

MongoDB 字段类型	Flink 字段类型
-	TINYINT

–	SMALLINT
Int	INT
Long	BIGINT
–	FLOAT
Double	DOUBLE
Decimal128	DECIMAL(p, s)
Boolean	BOOLEAN
DateTimestamp	DATE
DateTimestamp	TIME
Date	TIMESTAMP(3)TIMESTAMP_LTZ(3)
Timestamp	TIMESTAMP(0)TIMESTAMP_LTZ(0)
String ObjectId UUID Symbol MD5 JavaScript Regex	STRING
BinData	BYTES
Object	ROW
Array	ARRAY
DBPointer	ROW<$ref STRING, $id STRING>
GeoJSON	Point : ROW<type STRING, coordinates ARRAY<DOUBLE>> Line : ROW<type STRING, coordinates ARRAY<ARRAY< DOUBLE>>> ...

代码示例

```
CREATE TABLE mongo_cdc_source_table (
  _id STRING, // must be declared
  name STRING,
  weight DECIMAL(10,3),
  tags ARRAY<STRING>, -- array
  price ROW<amount DECIMAL(10,2), currency STRING>, -- embedded document
  suppliers ARRAY<ROW<name STRING, address STRING>>, -- embedded documents
  PRIMARY KEY(_id) NOT ENFORCED
) WITH (
  'connector' = 'mongodb-cdc',
  'hosts' = 'localhost:27017,localhost:27018,localhost:27019',
  'username' = 'flinkuser',
  'password' = 'flinkpw',
  'database' = 'inventory',
  'collection' = 'products'
);
CREATE TABLE `print_table` (
  `id` STRING,
  `name` STRING,
```

```
  `currency` STRING
) WITH (
 'connector' = 'print'
);
insert into print_table select _id, name, price.currency from mongo_cdc_source_table;
```

注意事项

用户权限

MongoDB 的 User 必须有 changeStream 和 read 权限。

```
use admin;
db.createUser(
 {
   user: "flinkuser",
   pwd: "flinkpw",
   roles: [
      { role: "read", db: "admin" },
      { role: "readAnyDatabase", db: "admin" }
   ]
 }
);
```

并行度

任务的并行度只支持为1。

数据库 SQLServer CDC

最近更新时间：2025-06-09 11:02:02

介绍

SQLServer 的 CDC 源表（即 SQLServer 的流式源表），SQLServer CDC Connector 允许从 SQLServer 数据库读取全量数据和增量数据。本文档介绍如何设置 SQLServer CDC Connector。目前 sqlserver connector 标识分为sqlserver-cdc 和 sqlserver-cdc-2.4，前者是 cdc 1.0版本，存量读取阶段不支持 checkpoint 和不支持并行读取存量数据，后者是 cdc 2.x 版本，支持存量数据并行读取和存量读取阶段生成快照。

版本说明

Flink 版本	说明
1.11	• 不支持
1.13	• 支持 sqlserver-cdc • 内置的 debezium 版本是1.5.4. Final
1.14	• 支持 sqlserver-cdc-2.4 • 内置的 debezium 版本是1.9.7. Final
1.16	• 支持 sqlserver-cdc-2.4 • 内置的 debezium 版本是1.9.7. Final

使用范围

SQLServer CDC 只支持作为源表。

SQLServer 数据库配置

SQLServer 源表需要启用变更数据捕获。

1. 启用数据库变更捕获，参考云数据库 SQLServer 变更数据捕获 。
2. 为 SQLServer 源表启用变更数据捕获，参考文档 sys.sp_cdc_enable_table 。

```
USE MyDB
GO

EXEC sys.sp_cdc_enable_table
@source_schema = N'dbo',      -- 指定源表所属的 schema 名
@source_name   = N'MyTable', -- 指定需要读取的源表名
@role_name     = NULL,  -- 指定一个角色 MyRole，您可以向该角色添加要对源表的捕获列授予 SELECT 权限的用户。
sysadmin 或 db_owner 角色的用户也可以访问指定的源表。将 @role_name 的值设置为 NULL，以仅允许使用 sysadmin 或
db_owner 中的用户。
@filegroup_name = NULL, -- 用于为捕获实例创建的更改表的文件组，可以为 NULL。 如果指定，则必须为当前数据库定义
filegroup_name 。 如果为 NULL，则使用默认文件组。
@supports_net_changes = 0 -- 是否为此捕获实例启用对查询净更改的支持。 如果表具有主键或表具有使用 @index_name 参
数标识的唯一索引，则supports_net_changes 默认值为 1。 否则，该参数默认为 0。如果为 0，则只生成查询所有更改的支持函
数。如果为 1，则还会生成查询净更改所需的函数。如果 supports_net_changes 设置为 1，则必须指定 index_name ，或者源
表必须具有定义的主键。
GO
```

3. 检查源表是否启动变更数据捕获，参考文档 sys.sp_cdc_help_change_data_capture 。

```
USE MyDB
GO

EXEC sys.sp_cdc_help_change_data_capture
```

```
GO
```

sqlserver-cdc DDL 定义

```sql
-- register a SqlServer table 'orders' in Flink SQL
CREATE TABLE orders (
    id INT,
    order_date DATE,
    purchaser INT,
    quantity INT,
    product_id INT,
    PRIMARY KEY (id) NOT ENFORCED
) WITH (
    'connector' = 'sqlserver-cdc',
    'hostname' = 'localhost',
    'port' = '1433',
    'username' = 'sa',
    'password' = 'Password!',
    'database-name' = 'inventory',
    'schema-name' = 'dbo',
    'table-name' = 'orders'
);

-- read snapshot and binlogs from orders table
SELECT * FROM orders;
```

sqlserver-cdc WITH 参数

参数	必填	默认值	类型	描述
connector	是	无	String	固定值为 `sqlserver-cdc`
hostname	是	无	String	SQLServer 数据库 IP 地址或主机名
username	是	无	String	SQLServer 数据库用户名
password	是	无	String	SQLServer 数据库密码
database-name	是	无	String	SQLServer 源表所属数据库名
schema-name	是	无	String	SQLServer 源表所属 schema 名。支持 Java 正则表达式，例如 (dbo.*) 可匹配 dbo、dbo1、dbo_test，对于正则表达式，强烈建议放置于括号内，以防止与 table-name 组合时出现错误
table-name	是	无	String	SQLServer 源表名。支持 Java 正则表达式，对于正则表达式，强烈建议放置于括号内，以防止与 schema-name 组合时出现错误
port	否	1433	Integer	SQLServer 数据库端口
server-time-zone	否	UTC	String	SQLServer 数据库会话时区设置，例如 'Asia/Shanghai'
debezium.*	否	无	String	Debezium 属性参数，从更细粒度控制 Debezium 客户端的行为。例如 'debezium.snapshot.mode' = 'initial_only'，详情参见 Debezium's SQLServer Connector properties

sqlserver-cdc 可用元数据

元数据名	数据类型	描述
table_name	STRING NOT NULL	包含该 Row 的表名称
schema_name	STRING NOT NULL	包含该 Row 的 schema 名称
database_name	STRING NOT NULL	包含该 Row 的数据库名称
op_ts	TIMESTAMP_LTZ(3) NOT NULL	Row 在数据库中进行更改的时间。全量阶段数据，该字段值为 0

元数据使用示例：

```
CREATE TABLE products (
    table_name STRING METADATA  FROM 'table_name' VIRTUAL,
    schema_name STRING METADATA  FROM 'schema_name' VIRTUAL,
    db_name STRING METADATA FROM 'database_name' VIRTUAL,
    operation_ts TIMESTAMP_LTZ(3) METADATA FROM 'op_ts' VIRTUAL,
    id INT NOT NULL,
    name STRING,
    description STRING,
    weight DECIMAL(10,3)
) WITH (
    'connector' = 'sqlserver-cdc',
    'hostname' = 'localhost',
    'port' = '1433',
    'username' = 'sa',
    'password' = 'Password!',
    'database-name' = 'inventory',
    'schema-name' = 'dbo',
    'table-name' = 'products'
);
```

类型映射

SQLServer 类型	Flink SQL 类型
char(n)	CHAR(n)
varchar(n) nvarchar(n) nchar(n)	VARCHAR(n)
text ntext xml	STRING
decimal(p, s) money smallmoney	DECIMAL(p, s)
numeric	NUMERIC
float real	DOUBLE
bit	BOOLEAN
int	INT
tinyint	SMALLINT
smallint	SMALLINT
bigint	BIGINT
date	DATE
time(n)	TIME(n)
datetime2 datetime smalldatetime	TIMESTAMP(n)

版权所有：腾讯云计算（北京）有限责任公司

datetimeoffset	TIMESTAMP_LTZ(3)

sqlserver-cdc 注意事项

全量阶段不能执行 checkpoint

全量阶段由于没有可恢复的位点，SQLServer CDC 无法执行 checkpoint。为了不执行 checkpoint，SqlServer CDC 将等待直到 checkpoint 超时失败。checkpoint 超时失败默认情况下将触发 Flink 作业的 failover。因此，如果数据库表很大，建议添加以下 Flink 配置，以避免由于 checkpoint 超时而发生故障切换：

```
execution.checkpointing.interval: 10min
execution.checkpointing.tolerable-failed-checkpoints: 100
restart-strategy: fixed-delay
restart-strategy.fixed-delay.attempts: 2147483647
```

单线程读取

SQLServer CDC 源无法并行读取，因为只有一个任务可以接收变更事件。

sqlserver-cdc-2.4 DDL 定义

```sql
-- register a SqlServer table 'orders' in Flink SQL
CREATE TABLE orders (
    id INT,
    order_date DATE,
    purchaser INT,
    quantity INT,
    product_id INT,
    PRIMARY KEY (id) NOT ENFORCED
) WITH (
    'connector' = 'sqlserver-cdc-2.4',
    'hostname' = 'localhost',
    'port' = '1433',
    'username' = 'sa',
    'password' = 'Password!',
    'database-name' = 'inventory',
    'table-name' = 'dbo.orders'
);

-- read snapshot and binlogs from orders table
SELECT * FROM orders;
```

sqlserver-cdc-2.4 WITH 参数

参数	必填	默认值	类型	描述
connector	是	无	String	固定值为 `sqlserver-cdc-2.4`
hostname	是	无	String	SQLServer 数据库 IP 地址或主机名
username	是	无	String	SQLServer 数据库用户名
password	是	无	String	SQLServer 数据库密码

database-name	是	无	String	SQLServer 源表所属数据库名
table-name	是	无	String	SQLServer 源表名组成格式schema.table。例子 dbo.tbl
port	否	1433	Integer	SQLServer 数据库端口
server-time-zone	否	UTC	String	SQLServer 数据库会话时区设置，例如 'Asia/Shanghai'
chunk-meta.group.size	否	1000	Integer	Chunk Meta 的 Group 大小，如果 Meta 大小超出 Group 大小，则 Meta 会被分成多个 Group。
chunk-key.even-distribution.factor.lower-bound	否	0.05d	Double	分块键分布因子的下限值。该分布因子用于判断表数据是否均匀分布。当数据分布均匀时，表的分块操作会采用均匀计算优化；而当数据分布不均匀时，则会触发分块查询。分布因子的计算公式为：(MAX(id) - MIN(id) + 1) / 行数。
chunk-key.even-distribution.factor.upper-bound	否	1000.0d	Double	分块键分布因子的上限值。该分布因子用于判断表数据是否均匀分布。当数据分布均匀时，表的分块操作会采用均匀计算优化；而当数据分布不均匀时，则会触发分块查询。分布因子的计算公式为：(MAX(id) - MIN(id) + 1) / 行数。
scan.incremental.snapshot.chunk.key-column	否	无	String	表快照的分块键（chunk key）。在读取表快照时，被捕获的表会按照块键拆分为多个数据块。默认情况下，块键是主键的第一列，且该列必须是主键的一部分。
debezium.*	否	无	String	Debezium 属性参数，从更细粒度控制 Debezium 客户端的行为。例如 'debezium.snapshot.mode' = 'initial_only'，详情参见 Debezium's SQLServer Connector properties

sqlserver-cdc-2.4 可用元数据

元数据名	数据类型	描述
table_name	STRING NOT NULL	包含该 Row 的表名称
schema_name	STRING NOT NULL	包含该 Row 的 schema 名称
database_name	STRING NOT NULL	包含该 Row 的数据库名称
op_ts	TIMESTAMP_LTZ(3) NOT NULL	Row 在数据库中进行更改的时间。全量阶段数据，该字段值为 0

元数据使用示例：

```
CREATE TABLE products (
    table_name STRING METADATA  FROM 'table_name' VIRTUAL,
    schema_name STRING METADATA  FROM 'schema_name' VIRTUAL,
    db_name STRING METADATA FROM 'database_name' VIRTUAL,
    operation_ts TIMESTAMP_LTZ(3) METADATA FROM 'op_ts' VIRTUAL,
    id INT NOT NULL,
    name STRING,
    description STRING,
    weight DECIMAL(10,3)
) WITH (
    'connector' = 'sqlserver-cdc-2.4',
    'hostname' = 'localhost',
    'port' = '1433',
    'username' = 'sa',
    'password' = 'Password!',
    'database-name' = 'inventory',
    'table-name' = 'dbo.products'
```

```
);
```

sqlserver-cdc-2.4 类型映射

SQLServer 类型	Flink SQL 类型
char(n)	CHAR(n)
varchar(n) nvarchar(n) nchar(n)	VARCHAR(n)
text ntext xml	STRING
decimal(p, s) money smallmoney	DECIMAL(p, s)
numeric	NUMERIC
float real	DOUBLE
bit	BOOLEAN
int	INT
tinyint	SMALLINT
smallint	SMALLINT
bigint	BIGINT
date	DATE
time(n)	TIME(n)
datetime2 datetime smalldatetime	TIMESTAMP(n)
datetimeoffset	TIMESTAMP_LTZ(3)

版权所有：腾讯云计算（北京）有限责任公司

文件系统 FileSystem

最近更新时间：2024-10-11 15:19:01

介绍

FileSystem connector 提供了对 `HDFS` 和 COS 等常见文件系统的写入支持。

版本说明

Flink 版本	说明
1.11	支持
1.13	支持，支持常见的 lzo、snappy 压缩算法
1.14	支持写入到 HDFS，不支持 lzo、snappy 压缩算法
1.16	支持写入到 HDFS，不支持 lzo、snappy 压缩算法

使用范围

FileSystem 支持作为 Append-Only 数据流的目的表 (Sink)，目前还不支持 Upsert 数据流的目的表。FileSystem 目前支持以下格式的数据写入：

- CSV
- JSON
- Avro
- Parquet
- Orc

> **说明**
> 目前使用数据格式 Avro、Parquet、Orc 写入时，需要 手动上传额外的 jar 包 才能使用。

DDL 定义

用作数据目的

```
CREATE TABLE `hdfs_sink_table` (
    `id` INT,
    `name` STRING,
    `part1` INT,
    `part2` INT
) PARTITIONED BY (part1, part2) WITH (
    'connector' = 'filesystem',
    'path' = 'hdfs://HDFS10000/data/', -- cosn://${buketName}/path/to/store/data
    'format' = 'json',
    'sink.rolling-policy.file-size' = '1M',
    'sink.rolling-policy.rollover-interval' = '10 min',
    'sink.partition-commit.delay' = '1 s',
    'sink.partition-commit.policy.kind' = 'success-file'
);
```

WITH 参数

参数值	必填	默认值	描述
path	是	无	文件写入的路径。

sink.rolling-policy.file-size	否	128MB	文件最大大小。当当前写入的文件大小达到设置的阈值时，当前写入的文件将被关闭，并打开一个新的文件进行写入。
sink.rolling-policy.rollover-interval	否	30min	文件最大持续写入时间。当当前写入的文件写入的时间超过了设置的阈值时，当前写入的文件将被关闭，并打开一个新的文件进行写入。
sink.rolling-policy.check-interval	否	1min	文件检查间隔。FileSystem 按照这个间隔检查文件的写入时间是否已经满足了关闭条件，并将满足条件的文件进行关闭。
sink.partition-commit.trigger	否	process-time	分区关闭策略。可选值包括： • process-time：当分区创建超过一定时间之后将这个分区关闭，分区创建时间为分区创建时的物理时间。 • partition-time：当分区创建超过一定时间之后将这个分区关闭，分区创建时间从分区中抽取出来。partition-time 依赖于 watermark 生成，需要配合 watermark 才能支持自动分区发现。当 watermark 时间超过了 `从分区抽取的时间` 与 `delay 参数配置时间` 之和后会提交分区。
sink.partition-commit.delay	否	0s	分区关闭延迟。当分区在创建超过一定时间之后将被关闭。
partition.time-extractor.kind	否	default	分区时间抽取方式。这个配置仅当 sink.partition-commit.trigger 配置为 partition-time 时生效。如果用户有自定义的分区时间抽取方法，配置为 custom。
partition.time-extractor.class	否	无	分区时间抽取类，这个类必须实现 PartitionTimeExtractor 接口。
partition.time-extractor.timestamp-pattern	否	无	分区时间戳的抽取格式。需要写成 yyyy-MM-dd HH:mm:ss 的形式，并用 Hive 表中相应的分区字段做占位符替换。默认支持第一个字段为 yyyy-mm-dd hh:mm:ss。 • 如果时间戳应该从单个分区字段 'dt' 提取，可以配置 '$dt'。 • 如果时间戳应该从多个分区字段中提取，例如 'year'、'month'、'day' 和 'hour'，可以配置 '$year-$month-$day $hour:00:00'。 • 如果时间戳应该从两个分区字段 'dt' 和 'hour' 提取，可以配置 '$dt $hour:00:00'。
sink.partition-commit.policy.kind	是	无	用于提交分区的策略。可选值包括： • success-file：当分区关闭时将在分区对应的目录下生成一个 _success 的文件。 • custom：用户实现的自定义分区提交策略。
sink.partition-commit.policy.class	否	无	分区提交类，这个类必须实现 PartitionCommitPolicy。

HDFS 配置

在 HDFS 上创建数据目录后，需为目录开启写权限，才可成功写入数据。流计算 Oceanus 写入 HDFS 的 user 是 flink。进行配置前，需要先登录 EMR 集群下载 Hadoop 集群的 hdfs-site.xml 文件，以获取下列配置中所需的参数值，参考 登录集群 。
HDFS 路径的形式为 `hdfs://${dfs.nameservices}/${path}` ， `${dfs.nameservices}` 的值可在 hdfs-site.xml 中查找， `${path}` 为要写入的数据目录。

- 若目标 Hadoop 集群只有单个 Master，仅需要为 path 参数传入 HDFS 路径即可，无需使用高级参数。
- 若目标 Hadoop 集群为高可用的双 Master 集群，为 path 参数传入 HDFS 路径后，还需要在作业参数的 高级参数 中对两个 Master 的地址和端口进行配置。以下是一个配置示例，相应的参数值都可在 hdfs-site.xml 中查找并替换。

```
fs.hdfs.dfs.nameservices: HDFS12345
fs.hdfs.dfs.ha.namenodes.HDFS12345: nn2,nn1
fs.hdfs.dfs.namenode.http-address.HDFS12345.nn1: 172.27.2.57:4008
fs.hdfs.dfs.namenode.https-address.HDFS12345.nn1: 172.27.2.57:4009
fs.hdfs.dfs.namenode.rpc-address.HDFS12345.nn1: 172.27.2.57:4007
fs.hdfs.dfs.namenode.http-address.HDFS12345.nn2: 172.27.1.218:4008
fs.hdfs.dfs.namenode.https-address.HDFS12345.nn2: 172.27.1.218:4009
fs.hdfs.dfs.namenode.rpc-address.HDFS12345.nn2: 172.27.1.218:4007
```

```
fs.hdfs.dfs.client.failover.proxy.provider.HDFS12345:
org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvider
```

说明

Flink 作业默认以 flink 用户操作 HDFS，若没有 HDFS 路径的写入权限，可通过作业 高级参数 设置为有权限的用户，或者设置为超级用户 hadoop。

```
containerized.taskmanager.env.HADOOP_USER_NAME: hadoop
containerized.master.env.HADOOP_USER_NAME: hadoop
```

COS 配置

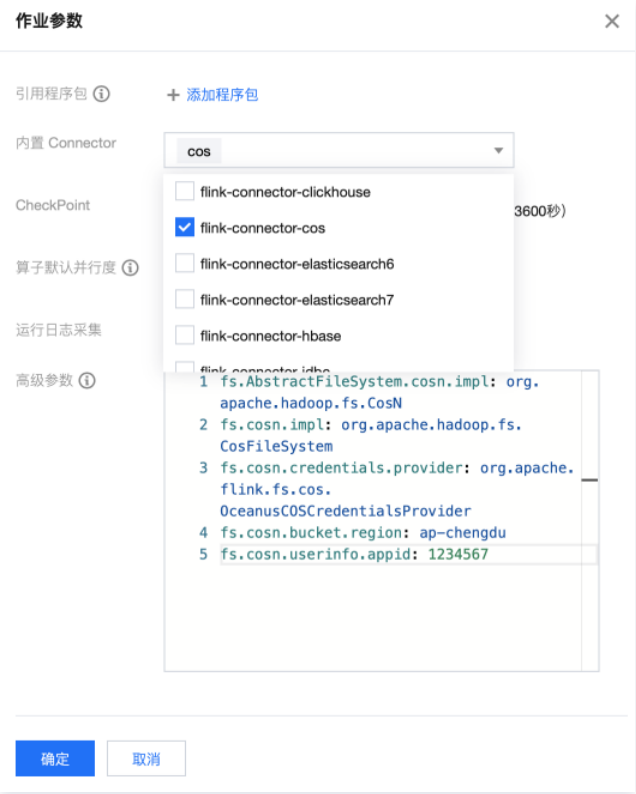
注意：

当写入 COS 时，Oceanus 作业所运行的地域必须和 COS 在同一个地域中。

在作业参数的 高级参数 中对 COS 的地址进行配置。流计算 Oceanus 写入 COS 的 user 是 flink。需要的配置项如下，其中地域的取值可参考 对象存储-地域和访问域名 。

```
fs.AbstractFileSystem.cosn.impl: org.apache.hadoop.fs.CosN
fs.cosn.impl: org.apache.hadoop.fs.CosFileSystem
fs.cosn.credentials.provider: org.apache.flink.fs.cos.OceanusCOSCredentialsProvider
fs.cosn.bucket.region: COS 所在的地域
fs.cosn.userinfo.appid: COS 所属用户的 appid
```

对于 Jar 作业，在使用 COS 作为数据写入的文件系统时，用户需要在内置 Connector 中勾选 flink-connector-cos，配置示意图：



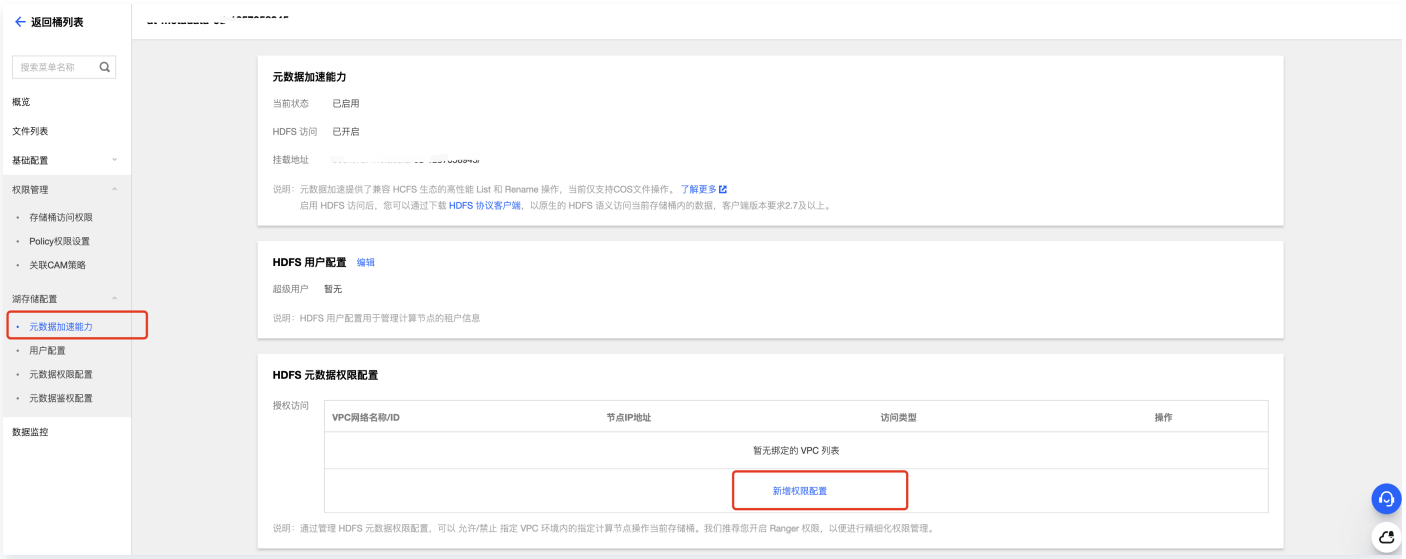
注意：

Flink-1.16已在镜像中内置 flink-connector-cos，无需上述两步。

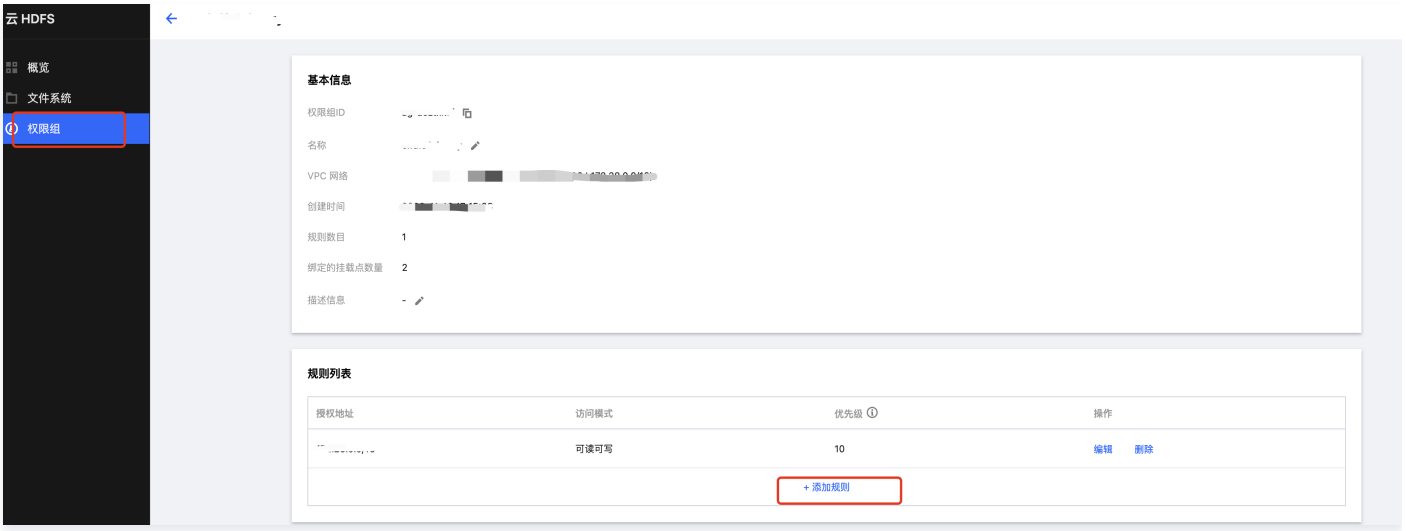
元数据加速 COS 桶、CHDFS

1. 配置 COS 元数据加速桶或 CHDFS 权限。

   在 COS 对应的加速桶 > 湖存储配置 > 元数据权限配置中：



   在 CHDFS 产品配置：



2. 依赖下载。

   在 Oceanus 的依赖管理上传对应 Jar 包，详情可参见 依赖管理 。

   - 元数据加速 COS 桶

     下载对应 Flink 版本的 Connector。

     点此下载 flink-1.13-connector-cos.jar

     点此下载 flink-1.14-connector-cos.jar

   - CHDFS

     点此下载 flink-chdfs-hadoop-1.10.0-0.1.4.jar

3. 作业 高级参数 配置。

   - 元数据加速 COS 桶

```
fs.cosn.trsf.fs.AbstractFileSystem.ofs.impl: com.qcloud.chdfs.fs.CHDFSDelegateFSAdapter
fs.cosn.trsf.fs.ofs.impl: com.qcloud.chdfs.fs.CHDFSHadoopFileSystemAdapter
```

```
fs.cosn.trsf.fs.ofs.tmp.cache.dir: /tmp/chdfs/
fs.cosn.trsf.fs.ofs.user.appid: COS 所属用户的 appid
fs.cosn.trsf.fs.ofs.bucket.region: COS 所在的地域
fs.cosn.trsf.fs.ofs.upload.flag: true

fs.AbstractFileSystem.cosn.impl: org.apache.hadoop.fs.CosN
fs.cosn.impl: org.apache.hadoop.fs.CosFileSystem
fs.cosn.credentials.provider: org.apache.flink.fs.cos.OceanusCOSCredentialsProvider
fs.cosn.bucket.region: COS 所在的地域
fs.cosn.userinfo.appid: COS 所属用户的 appid

containerized.taskmanager.env.HADOOP_USER_NAME: hadoop
containerized.master.env.HADOOP_USER_NAME: hadoop
```

> **注意：**
> **Flink-1.16 默认支持元数据加速桶，无需下载Jar包，高级参数只需填写如下两项，其他参数已自动填充。**
> containerized.taskmanager.env.HADOOP_USER_NAME: hadoop
> containerized.master.env.HADOOP_USER_NAME: hadoop

- CHDFS

```
fs.AbstractFileSystem.ofs.impl: com.qcloud.chdfs.fs.CHDFSDelegateFSAdapter
fs.ofs.impl: com.qcloud.chdfs.fs.CHDFSHadoopFileSystemAdapter
fs.ofs.tmp.cache.dir: /tmp/chdfs/
fs.ofs.upload.flag: true
fs.ofs.user.appid: CHDFS 所属用户的 appid
fs.ofs.bucket.region: CHDFS 所在的地域

containerized.taskmanager.env.HADOOP_USER_NAME: hadoop
containerized.master.env.HADOOP_USER_NAME: hadoop
```

手动上传对应 Jar 包

1. 先下载对应 Jar 包到本地。
   不同 Flink 版本下载地址：Flink 1.11 ，Flink 1.13，Flink 1.14 。
2. 在 Oceanus 的程序包管理上传对应 Jar 包，详情可参见 程序包管理 。
3. 进入对应作业的开发调试界面，打开作业参数侧栏。

在作业参数的引用程序包栏单击**添加程序包**，选择在第2步上传的 Jar 包，单击**确定**保存作业参数配置。

4. 发布作业。

HDFS Kerberos 认证授权

1. 登录集群 Master 节点，获取 krb5.conf、emr.keytab、core-site.xml、hdfs-site.xml 文件，路径如下。

```
/etc/krb5.conf
/var/krb5kdc/emr.keytab
/usr/local/service/hadoop/etc/hadoop/core-site.xml
/usr/local/service/hadoop/etc/hadoop/hdfs-site.xml
```

2. 对步骤1中获取的文件打 jar 包。

```
jar cvf hdfs-xxx.jar krb5.conf emr.keytab core-site.xml hdfs-site.xml
```

3. 校验 jar 的结构（可以通过 vim 命令查看 vim hdfs-xxx.jar），jar 里面包含如下信息，请确保文件不缺失且结构正确。

```
META-INF/
META-INF/MANIFEST.MF
emr.keytab
krb5.conf
hdfs-site.xml
core-site.xml
```

4. 在 程序包管理 页面上传 jar 包，并在作业参数配置里引用该程序包。
5. 获取 kerberos principal，用于作业 高级参数 配置。

```
klist -kt /var/krb5kdc/emr.keytab

# 输出如下所示，选取第一个即可：hadoop/172.28.28.51@EMR-OQPO48B9
KVNO Timestamp         Principal
---- ------------------- ------------------------------------------------------
   2 08/09/2021 15:34:40 hadoop/172.28.28.51@EMR-OQPO48B9
   2 08/09/2021 15:34:40 HTTP/172.28.28.51@EMR-OQPO48B9
   2 08/09/2021 15:34:40 hadoop/VM-28-51-centos@EMR-OQPO48B9
   2 08/09/2021 15:34:40 HTTP/VM-28-51-centos@EMR-OQPO48B9
```

6. 作业 高级参数 配置。

```
containerized.taskmanager.env.HADOOP_USER_NAME: hadoop
containerized.master.env.HADOOP_USER_NAME: hadoop
security.kerberos.login.principal: hadoop/172.28.28.51@EMR-OQPO48B9
security.kerberos.login.keytab: emr.keytab
security.kerberos.login.conf: krb5.conf
```

如果是 Flink-1.13 版本，需要在高级参数额外增加如下参数，其中参数的值需要为对应 hdfs-site.xml 中的值。

```
fs.hdfs.dfs.nameservices: HDFS17995
fs.hdfs.dfs.ha.namenodes.HDFS17995: nn2,nn1
fs.hdfs.dfs.namenode.http-address.HDFS17995.nn1: 172.28.28.214:4008
fs.hdfs.dfs.namenode.https-address.HDFS17995.nn1: 172.28.28.214:4009
fs.hdfs.dfs.namenode.rpc-address.HDFS17995.nn1: 172.28.28.214:4007
fs.hdfs.dfs.namenode.http-address.HDFS17995.nn2: 172.28.28.224:4008
fs.hdfs.dfs.namenode.https-address.HDFS17995.nn2: 172.28.28.224:4009
fs.hdfs.dfs.namenode.rpc-address.HDFS17995.nn2: 172.28.28.224:4007
fs.hdfs.dfs.client.failover.proxy.provider.HDFS17995:
org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvider
fs.hdfs.hadoop.security.authentication: kerberos
```

> **注意：**
> 历史 Oceanus 集群可能不支持该功能，您可通过 在线客服 联系我们升级集群管控服务，以支持 Kerberos 访问。

代码示例

```
CREATE TABLE datagen_source_table (
  id INT,
  name STRING,
  part1 INT,
  part2 INT
) WITH (
  'connector' = 'datagen',
  'rows-per-second'='1',  -- 每秒产生的数据条数
  'fields.part1.min'='1',
  'fields.part1.max'='2',
  'fields.part2.min'='1',
  'fields.part2.max'='2'
);

CREATE TABLE hdfs_sink_table (
    id INT,
    name STRING,
```

```
    part1 INT,
    part2 INT
) PARTITIONED BY (part1, part2) WITH (
    'connector' = 'filesystem',
    'path' = 'hdfs://HDFS10000/data/',
    'format' = 'json',
    'sink.rolling-policy.file-size' = '1M',
    'sink.rolling-policy.rollover-interval' = '10 min',
    'sink.partition-commit.delay' = '1 s',
    'sink.partition-commit.policy.kind' = 'success-file'
);

INSERT INTO hdfs_sink_table
SELECT id, name, part1, part2
FROM datagen_source_table;
```

compressible-fs connector使用说明

- 只支持在 flink 1.13版本使用。
- 支持对于 csv 和 json 两种 format 的写入，其它诸如 avro、parquet、orc 文件格式已经自带压缩功能。
- 支持 LzopCodec、OceanusSnappyCodec 两种压缩算法。
- 支持写入 hdfs 和 cos 文件，使用方法和 filesystem 一致。

用作数据目的

```
CREATE TABLE `hdfs_sink_table` (
    `id` INT,
    `name` STRING,
    `part1` INT,
    `part2` INT
) PARTITIONED BY (part1, part2) WITH (
    'connector' = 'compressible-fs',
    'hadoop.compression.codec' = 'LzopCodec',
    'path' = 'hdfs://HDFS10000/data/',
    'format' = 'json',
    'sink.rolling-policy.file-size' = '1M',
    'sink.rolling-policy.rollover-interval' = '10 min',
    'sink.partition-commit.delay' = '1 s',
    'sink.partition-commit.policy.kind' = 'success-file'
);
```

WITH 参数

除上文中 filesystem connector 支持的参数外，compressible-fs 额外特有的参数有以下三个：

参数值	必填	默认值	描述
hadoop.compression.codec	否	无	使用的压缩算法，可选值为 LzopCodec 和 OceanusSnappyCodec，不指定时，按照默认的文件格式写入。其中 OceanusSnappyCodec 是由于 snappy 库版本原因，对于 SnappyCodec 的封装，结果完全同 SnappyCodec
filename.suffix	否	无	最终写入文件名，如果没有声明，则会按照支持的压缩算法生成特定的后缀名，如果采用了非 lzop 和 snappy 压缩算法且未声明该值，则文件后缀为空
filepath.contain.partition-key	否	false	写入分区文件时，最终的写入路径是否包括分区字段，默认不包括。例如，假设写入一个按天分区dt=12和按小时分区ht=24的分区路径，默认的分区路径为12/24 而非 dt=12/ht=24

数据湖 Hudi

最近更新时间：2023-07-04 16:32:52

版本说明

Flink 版本	说明
1.11	不支持
1.13	支持 Source 和 Sink
1.14	不支持
1.16	不支持

使用范围

可以作为 Source/Sink 使用。

DDL 定义

用作数据目的：

```
CREATE TABLE hudi_sink
(
    uuid        VARCHAR(20) PRIMARY KEY NOT ENFORCED,
    name        VARCHAR(10),
    age         INT,
    ts          TIMESTAMP(3),
    `partition` VARCHAR(20)
) WITH (
    'connector' = 'hudi'
    , 'path' = 'hdfs://HDFS1000/data/hudi/mor'
    , 'table.type' = 'MERGE_ON_READ'  --  MERGE_ON_READ 表，默认值为 COPY_ON_WRITE
    , 'write.tasks' = '3' -- 默认为4
    , 'compaction.tasks' = '4' -- 默认为4
    --  , 'hive_sync.enable' = 'true'  -- 默认值为false
    --  , 'hive_sync.db' = 'default'
    --  , 'hive_sync.table' = 'datagen_mor_1'
    --  , 'hive_sync.mode' = 'jdbc'
    --  , 'hive_sync.username' = ''
    --  , 'hive_sync.password' = ''
    --  , 'hive_sync.jdbc_url' = 'jdbc:hive2://172.28.1.185:7001'
    --  , 'hive_sync.metastore.uris' = 'thrift://172.28.1.185:7004'
);
```

作为数据源：

```
CREATE TABLE `source`
(
    uuid        VARCHAR(20) PRIMARY KEY NOT ENFORCED,
    name        VARCHAR(10),
    age         INT,
    ts          TIMESTAMP(3),
    `partition` VARCHAR(20)
) WITH (
      'connector' = 'hudi'
```

```
      , 'path' = 'hdfs://172.28.28.202:4007/path/hudidata'
      , 'table.type' = 'MERGE_ON_READ'  -- MOR 表，目前无法读取增量数据
      , 'read.tasks' = '1'  -- 读task的并行度,默认值为4
      , 'hoodie.datasource.query.type' = 'snapshot' -- 默认值为snapshot，可选值为 read_optimized,
incremental
      , 'read.streaming.enabled' = 'true'  -- this option enable the streaming read
      , 'read.start-commit' = 'earliest' -- specifies the start commit instant time, the commit time
format should be 'yyyyMMddHHmmss'
      , 'read.streaming.check-interval' = '4'
      );
```

WITH 参数

通用参数

参数值	必填	默认值	描述
connector	是	无	必须填 hudi
path	是	无	数据的存储路径（如果存储到 HDFS，格式为 hdfs://；存储为 COS 为 COSN://$bucket/$path）

作为 Sink 的参数

参数值	必填	默认值	描述
table.type	否	COPY_ON_WRITE	Hudi 表类型，可选值为 COPY_ON_WRITE 或者 MERGE_ON_READ

HoodieRecord 字段相关

参数值	必填	默认值	描述
hoodie.datasource.write.recordkey.field	否	uuid	key 字段，flink table 如果有 primary key，则采用 flink table 的 pk
hoodie.datasource.write.partitionpath.field	否	""	分区路径字段，为空表示不分区
write.precombine.field	否	ts	预合并时，相同 key 记录时候，用于比较(Object.compareTo(..))的字段

并行度相关

参数值	必填	默认值	描述
write.tasks	否	4	写算子的并行度
write.index_bootstrap.tasks	否	无	index bootstrap 的并行度，默认为作业的并行度
write.bucket_assign.tasks	否	无	bucket assign 的并行度，默认为作业的并行度
compaction.tasks	否	4	compaction 任务的并行度

compaction相关

参数值	必填	默认值	描述
compaction.schedule.enabled	否	true	是否启动 compaction
compaction.async.enabled	否	true	compaction 是否采用异步

compaction.trigger.strategy	否	num_commits	num_commits / time_elapsed / num_and_time / num_or_time

hive 元数据同步相关

参数值	必填	默认值	描述
hive_sync.enable	否	false	-
hive_sync.db	否	-	-
hive_sync.table	否	-	-
hive_sync.mode	否	jdbc	可选值 hms，jdbc and hiveql
hive_sync.username	否	-	-
hive_sync.password	否	-	-
hive_sync.jdbc_url	否	-	-

更多参数

其他更详细的参数，可参见 Flink Options 。

作为 Source 的参数

参数值	必填	默认值	描述
read.tasks	否	4	读算子的并行度
hoodie.datasource.query.type	否	snapshot	可选值 snapshot / read_optimized / incremental
read.streaming.enabled	否	false	-
read.streaming.check-interval	否	60	单位秒，streaming read的检查时间间隔
read.streaming.skip_compaction	否	false	-
read.start-commit	否	无	格式为 yyyyMMddHHmmss；可设置为 earliest，表示从最早的 commit 开始消费
read.end-commit	否	无	streaming read，无需设置该值

更详细的参数配置

可参见 Flink Options 。

COS 配置

无需做额外配置。path 填写为对应的 cosn 路径即可。

HDFS 配置

获取 HDFS 链接配置 jar

Flink SQL 任务写 Hudi , 使用 HDFS 存储时需要使用包含 HDFS 配置信息的 jar 包来连接到 HDFS 集群。具体获取连接配置 jar 及其使用的步骤如下：

1. ssh 登录到对应 HDFS 集群节点。
2. 获取 hdfs-site.xml，EMR 集群中的配置文件在如下位置。

```
/usr/local/service/hadoop/etc/hadoop/hdfs-site.xml
```

3. 对获取到的配置文件打 jar 包。

```
jar -cvf hdfs-xxx.jar hdfs-site.xml
```

4. 校验 jar 的结构（可以通过 vi 命令查看 ），jar 里面包含如下信息，请确保文件不缺失且结构正确。

```
vi hdfs-xxx.jar
```

```
META-INF/
META-INF/MANIFEST.MF
hdfs-site.xml
```

配置写入 HDFS 的用户

> **说明**
> Flink 作业默认以 flink 用户操作 HDFS，若没有 HDFS 路径的写入权限，可通过 作业高级参数 设置为有权限的用户，或者设置为超级用户 hadoop。

```
containerized.taskmanager.env.HADOOP_USER_NAME: hadoop
containerized.master.env.HADOOP_USER_NAME: hadoop
```

Kerberos 认证授权

1. 登录集群 Master 节点，获取 krb5.conf、emr.keytab、core-site.xml、hdfs-site.xml 文件，路径如下。

```
/etc/krb5.conf
/var/krb5kdc/emr.keytab
/usr/local/service/hadoop/etc/hadoop/core-site.xml
/usr/local/service/hadoop/etc/hadoop/hdfs-site.xml
```

2. 对获取的配置文件打 jar 包。

```
jar cvf hdfs-xxx.jar krb5.conf emr.keytab core-site.xml hdfs-site.xml
```

3. 校验 jar 的结构（可以通过 vim 命令查看 vim hdfs-xxx.jar），jar 里面包含如下信息，请确保文件不缺失且结构正确。

```
META-INF/
META-INF/MANIFEST.MF
emr.keytab
krb5.conf
hdfs-site.xml
core-site.xml
```

4. 在 程序包管理 页面上传 jar 包，并在作业参数配置里引用该程序包。
5. 获取 kerberos principal，用于 作业高级参数 配置。

```
klist -kt /var/krb5kdc/emr.keytab

# 输出如下所示，选取第一个即可：hadoop/172.28.28.51@EMR-OQPO48B9
KVNO Timestamp     Principal
```

```
---- ------------------- -------------------------------------------------------
 2 08/09/2021 15:34:40 hadoop/172.28.28.51@EMR-OQPO48B9
 2 08/09/2021 15:34:40 HTTP/172.28.28.51@EMR-OQPO48B9
 2 08/09/2021 15:34:40 hadoop/VM-28-51-centos@EMR-OQPO48B9
 2 08/09/2021 15:34:40 HTTP/VM-28-51-centos@EMR-OQPO48B9
```

6. 作业高级参数 配置。

```
containerized.taskmanager.env.HADOOP_USER_NAME: hadoop
containerized.master.env.HADOOP_USER_NAME: hadoop
security.kerberos.login.principal: hadoop/172.28.28.51@EMR-OQPO48B9
security.kerberos.login.keytab: emr.keytab
security.kerberos.login.conf: krb5.conf
```

常见问题

hive sync

hive 表同步失败，报错。

```
java.lang.ClassNotFoundException: org.apache.hudi.hadoop.HoodieParquetInputFormat
```

请确认对应的 hive 环境,是否包含 Hudi 需要的 jar 包，详情请参见 Hive 。
hudi-hadoop-mr-bundle-x.y.z.jar 下载链接 。

数据湖 Iceberg

最近更新时间：2024-07-15 14:14:51

版本说明

Flink 版本	说明
1.11	不支持
1.13	支持 Source 和 Sink
1.14	支持 Source 和 Sink
1.16	支持 Source 和 Sink

使用范围

可以作为 Source/Sink 使用。其中，作为 source 使用时，不支持 upsert 写入的源 iceberg。

DDL 定义

用作数据目的：

```
CREATE TABLE `sink` (
  `id` bigint,
  `YCSB_KEY` string,
  `FIELD0` string,
  `FIELD1` string,
  `FIELD2` string,
  `database_name` string,
  `table_name` string,
  `op_ts` timestamp(3),
  `date` string
) PARTITIONED BY (`date`) WITH (
  'connector' = 'iceberg-1.1',
  'write.upsert.enabled'='false', -- 是否开启upsert
  'catalog-type' = 'hive',
  'catalog-name'='xxx',
  'catalog-database'='xxx',
  'catalog-table'='xxx',
  'warehouse' = 'hdfs://HDFS14979/usr/hive/warehouse',
  -- Hive metastore 的 thrift URI，可以从hive-site.xml配置文件中获取，对应的Key为：hive-metastore-uris
  'uri'='thrift://ip:port',
  'engine.hive.enabled' = 'true',
  'format-version' = '2'
);
```

作为数据源：

```
CREATE TABLE `icesource` (
  `id` bigint,
  `YCSB_KEY` string,
  `FIELD0` string,
  `FIELD1` string,
  `FIELD2` string,
  `database_name` string,
  `table_name` string,
  `op_ts` timestamp(3),
```

```
PRIMARY KEY(id) NOT ENFORCED
) WITH (
  'connector' = 'iceberg-1.1',
  'catalog-name' = 'hive_catalog',
  'catalog-type' = 'hive',
  'catalog-database' = 'database_ta',
  'catalog-table' = 't_p1_hive3_avro_3',
  'warehouse'='hdfs://HDFS14979/usr/hive/warehouse',
  'engine.hive.enabled' = 'true',
  'format-version' = '2',
  'streaming'='true',
  'monitor-interval'='10',
   -- Hive metastore 的 thrift URI，可以从hive-site.xml配置文件中获取，对应的Key为：hive-metastore-uris
   'uri'='thrift://ip:port'
);
```

WITH 参数

通用参数

参数值	必填	默认值	描述
connector	是	无	必须填 iceberg-1.1
warehouse	是	无	数据的存储路径（如果存储到 HDFS，格式为 hdfs://；存储为 COS 为 COSN://$bucket/$path）
catalog-name	是	无	自定义的 catalog 名
catalog-type	是	无	catalog 类型，可选值为 hadoop / hive / custom
catalog-database	是	无	iceberg 数据库名称
catalog-table	是	无	iceberg 表名称
catalog-impl	否	无	catalog-type 为 custom时，必填
uri	否	无	Hive metastore 的 thrift URI，可以从 hive-site.xml 配置文件中获取，对应的 Key 为：hive-metastore-uris; Eg. thrift://172.28.1.149:7004
format-version	否	1	Iceberg 格式 请参见 Iceberg Table Spec

更多参数请参见 Configuration 。

COS 配置

无需做额外配置，path 填写为对应的 cosn 路径即可。

HDFS 配置

获取 HDFS 链接配置 jar

Flink SQL 任务写 Iceberg，使用 HDFS 存储时需要使用包含 HDFS 配置信息的 jar 包来连接到 HDFS 集群。具体获取连接配置 jar 及其使用的步骤如下：

1. ssh 登录到对应 HDFS 集群节点。
2. 获取 hdfs-site.xml，EMR 集群中的配置文件在如下位置。

```
/usr/local/service/hadoop/etc/hadoop/hdfs-site.xml
```

3. 对获取到的配置文件 打 jar 包。

```
jar -cvf hdfs-xxx.jar hdfs-site.xml
```

4. 校验 jar 的结构（可以通过 vi 命令查看 ），jar 里面包含如下信息，请确保文件不缺失且结构正确。

```
vi hdfs-xxx.jar
```

```
META-INF/
META-INF/MANIFEST.MF
hdfs-site.xml
```

配置写入 hdfs 的用户

> **说明**
> Flink 作业默认以 flink 用户操作 HDFS，若没有 HDFS 路径的写入权限，可通过 作业高级参数 设置为有权限的用户，或者设置为超级用户 hadoop。

```
containerized.taskmanager.env.HADOOP_USER_NAME: hadoop
containerized.master.env.HADOOP_USER_NAME: hadoop
```

Kerberos 认证授权

1. 登录集群 Master 节点，获取 krb5.conf、emr.keytab、core-site.xml、hdfs-site.xml、hive-site.xml 文件，路径如下：

```
/etc/krb5.conf
/var/krb5kdc/emr.keytab
/usr/local/service/hadoop/etc/hadoop/core-site.xml
/usr/local/service/hadoop/etc/hadoop/hdfs-site.xml
/usr/local/service/hive/conf/hive-site.xml
```

2. 修改 hive-site.xml 文件。在 hive-site.xml 中增加如下配置，IP 的值取配置文件中 `hive.server2.thrift.bind.host` 的 value。

```
<property>
 <name>hive.metastore.uris</name>
 <value>thrift://ip:7004</value>
</property>
```

3. 获取 hivemetastore-site.xml 和 hiveserver2-site.xml，点击文件名下载。
4. 对获取的配置文件打 jar 包。

```
jar cvf hive-xxx.jar krb5.conf emr.keytab core-site.xml hdfs-site.xml hive-site.xml hivemetastore-
site.xml hiveserver2-site.xml
```

5. 校验 jar 的结构（可以通过 vim 命令查看 vim hdfs-xxx.jar），jar 里面包含如下信息，请确保文件不缺失且结构正确。

```
META-INF/
META-INF/MANIFEST.MF
emr.keytab
krb5.conf
hdfs-site.xml
core-site.xml
hive-site.xml
hivemetastore-site.xml
```

```
hiveserver2-site.xml
```

6. 在 程序包管理 页面上传 jar 包，并在作业参数配置里引用该程序包。
7. 获取 kerberos principal，用于 作业高级参数 配置。

```
klist -kt /var/krb5kdc/emr.keytab

# 输出如下所示，选取第一个即可：hadoop/172.28.28.51@EMR-OQPO48B9
KVNO Timestamp         Principal
---- ------------------- ------------------------------------------------------
   2 08/09/2021 15:34:40 hadoop/172.28.28.51@EMR-OQPO48B9
   2 08/09/2021 15:34:40 HTTP/172.28.28.51@EMR-OQPO48B9
   2 08/09/2021 15:34:40 hadoop/VM-28-51-centos@EMR-OQPO48B9
   2 08/09/2021 15:34:40 HTTP/VM-28-51-centos@EMR-OQPO48B9
```

8. 作业高级参数 配置。

```
containerized.taskmanager.env.HADOOP_USER_NAME: hadoop
containerized.master.env.HADOOP_USER_NAME: hadoop
security.kerberos.login.principal: hadoop/172.28.28.51@EMR-OQPO48B9
security.kerberos.login.keytab: emr.keytab
security.kerberos.login.conf: krb5.conf
```

> **说明**
> security.kerberos.login.keytab 和 security.kerberos.login.conf 的值为对应的文件名。

数据湖计算 DLC

最近更新时间：2024-12-11 10:13:02

版本说明

Flink 版本	说明
1.11	不支持
1.13	支持
1.14	不支持
1.16	支持

使用范围

可以作为 Sink 使用。目前支持写入 DLC 托管的原生表。

DDL 定义

```
CREATE TABLE `eason_internal_test`(
    `name` STRING,
    `age` INT
) WITH (
    'connector' = 'iceberg-inlong',
    'catalog-database' = 'test',
    'catalog-table' = 'eason_internal_test',
    'default-database' = 'test',
    'catalog-name' = 'HYBRIS',
    'qcloud.dlc.secret-id' = 'xxx',
    'qcloud.dlc.secret-key' = 'xxx',
    'qcloud.dlc.region' = 'ap-guangzhou',
    'qcloud.dlc.jdbc.url' = 'jdbc:dlc:dlc.internal.tencentcloudapi.com?
task_type=SparkSQLTask&database_name=test&datasource_connection_name=DataLakeCatalog&region=ap-
guangzhou&data_engine_name=dailai_test',
    'user.appid' = '1257058945'
);
```

WITH 参数

通用参数

参数值	必填	默认值	描述
connector	是	无	connector 类型，必须填 iceberg-inlong
catalog-database	是	无	DLC 内表所在的数据库名称
catalog-table	是	无	DLC 内表名称
default-database	是	无	DLC 内表所在的数据库名称
catalog-name	是	无	catalog 名称，必须填 HYBRIS
qcloud.dlc.secret-id	是	无	DLC 用户的 secretId，从 https://console.cloud.tencent.com/cam/capi 中获取
qcloud.dlc.secret-key	是	无	DLC 用户的 secretKey，从 https://console.cloud.tencent.com/cam/capi 中获取

qcloud.dlc.region	是	无	DLC 所在地域，必须填 ap-地域 格式
qcloud.dlc.jdbc.url	是	无	DLC jdbc 接入 url，格式见 JDBC 访问
user.appid	是	无	DLC 用户的 appid
sink.ignore.changelog	否	是	是否忽略 delete 数据，默认为 false，设为 true 则进入 append mode

DLC 表配置

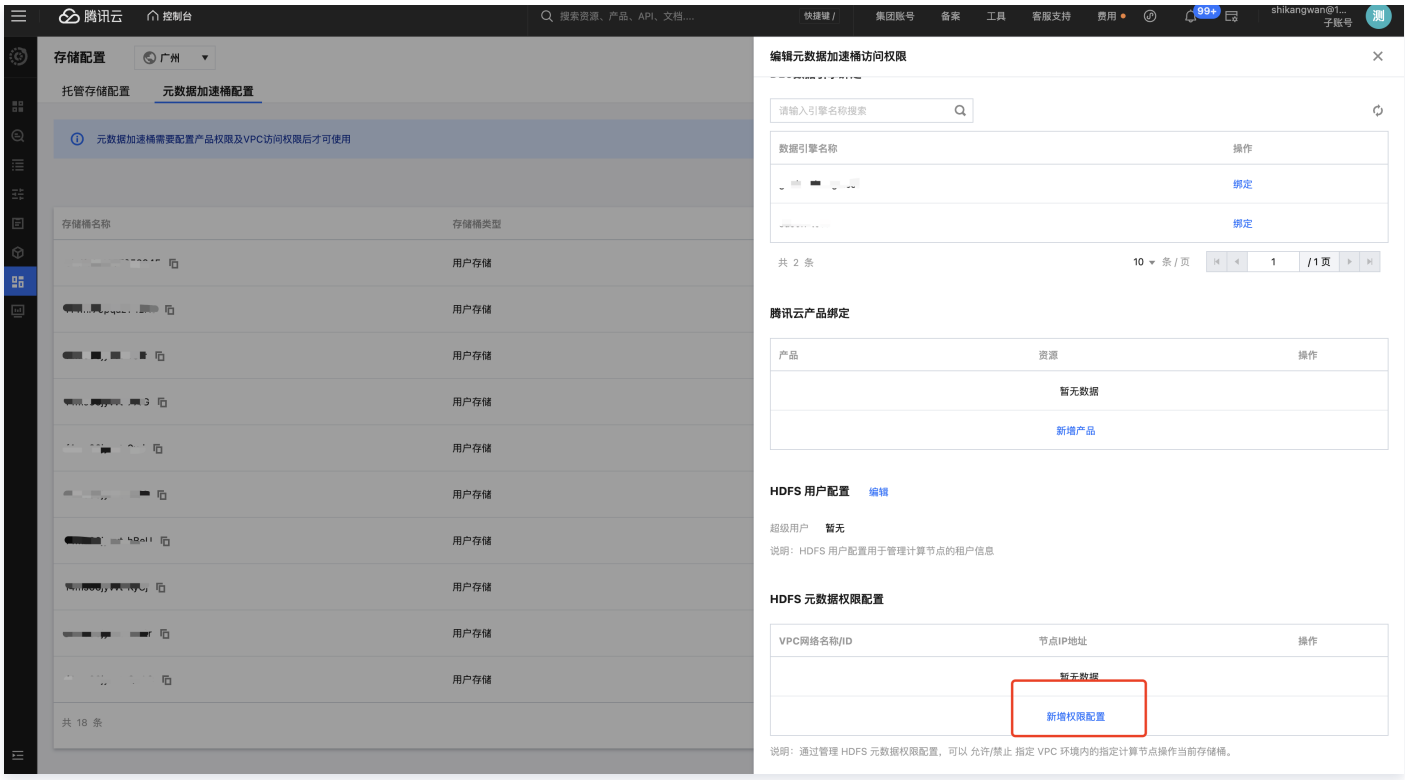
```
Upsert 模式
-- DLC 建表语句
CREATE TABLE `bi_sensor`(
    `uuid` string,
    `id` string,
    `type` string,
    `project` string,
    `properties` string,
    `sensors_id` string,
    `time` int,
    `hour` int) PARTITIONED BY (`time`);
-- 将目标表设为 v2 表，允许 upsert
ALTER TABLE `bi_sensor` SET TBLPROPERTIES ('format-version'='2','write.metadata.delete-after-
commit.enabled' = 'true', 'write.metadata.previous-versions-max' = '100',
'write.metadata.metrics.default' = 'full', 'write.upsert.enabled'='true', 'write.distribution-
mode'='hash');

-- oceanus sink DDL，dlc 的主键和分区字段必须在 flink 定义的主键字段中
create table bi_sensors (
    `uuid` STRING,
    `id` STRING,
    `type` STRING,
    `project` STRING,
    `properties` STRING,
    `sensors_id` STRING,
    `time` int,
    `hour` int,
    PRIMARY KEY (`uuid`, `time`) NOT ENFORCED
) with (...)
```

注意：
若您写入的 DLC 是基于元数据加速桶的，需要参考如下操作流程。

基于元数据加速桶的 DLC 操作流程

客户为 Oceanus 集群设置元数据加速桶的 DLC 权限

1. 进入 数据湖计算 DLC 控制台，单击**全局配置** > **存储配置** > **元数据加速桶配置**。
2. 选择需要写入的托管桶，单击**配置**。
3. 单击**新增权限配置**，网络选择创建 Oceanus 集群时绑定的子网。Oceanus 集群子网可以在 集群信息 中查看。



4. 为作业添加配置高级参数，详情请参见 高级参数 。

```
flink.hadoop.fs.AbstractFileSystem.ofs.impl: com.qcloud.chdfs.fs.CHDFSDelegateFSAdapter
flink.hadoop.fs.ofs.impl: com.qcloud.chdfs.fs.CHDFSHadoopFileSystemAdapter
flink.hadoop.fs.ofs.tmp.cache.dir: /tmp/chdfs/
flink.hadoop.fs.ofs.upload.flush.flag: true
flink.hadoop.fs.ofs.user.appid: 客户的 appid
flink.hadoop.fs.cosn.trsf.fs.ofs.user.appid: 客户的 appid
flink.hadoop.fs.cosn.trsf.fs.ofs.bucket.region: cos 所在地域（例如：弗吉尼亚对应 na-ashburn，广州对应 ap-guangzhou）
flink.hadoop.fs.cosn.trsf.fs.ofs.tmp.cache.dir: /tmp/chdfs

# 注意：使用 hadoop 用户权限则添加下面参数，使用 flink 用户无需添加
containerized.taskmanager.env.HADOOP_USER_NAME: hadoop
containerized.master.env.HADOOP_USER_NAME: hadoop
```

数据仓库 Hive

最近更新时间：2024-08-23 14:38:32

介绍

Hive Connector 支持数据流的目的表，但只支持 append only，不支持 Upsert 数据流。数据格式支持包括 Text、SequenceFile、ORC 和 Parquet 等。

版本说明

Flink 版本	说明
1.11	• 支持 hive 版本 1.1.0、2.3.2、2.3.5、3.1.1 • 配置项 'connector.type' = 'hive'
1.13	• 支持 hive 版本 1.0.0 ~ 1.2.2、2.0.0 ~ 2.2.0、2.3.0 ~ 2.3.6、3.0.0 ~ 3.1.2 • 配置项 'connector' = 'hive'
1.14	不支持
1.16	• 支持 hive 版本 2.3.0 ~ 2.3.6、3.0.0 ~ 3.1.2 • 配置项 'connector' = 'hive'

DDL 定义

用作数据目的（Sink）

```
CREATE TABLE hive_table (
  `id` INT,
  `name` STRING,
  `dt` STRING,
  `hr` STRING
) PARTITIONED BY (dt, hr)
with (
    'connector' = 'hive',  -- Flink 1.13 请使用 'connector' = 'hive'
    'hive-version' = '3.1.1',
    'hive-database' = 'testdb',
    'partition.time-extractor.timestamp-pattern'='$dt $hr:00:00',
    'sink.partition-commit.trigger'='partition-time',
    'sink.partition-commit.delay'='1 h',
    'sink.partition-commit.policy.kind'='metastore,success-file'
);
```

作业配置

在 Hive 数据库中创建 Hive 表。

```
# 在 Hive 的 testdb 数据库创建 hive_table 数据表
USE testdb;
CREATE TABLE `hive_table` (
  `id` int,
  `name` string)
PARTITIONED BY (`dt` string, `hr` string)
STORED AS ORC;
```

对 Hive 表的 HDFS 路径开启写权限。

- 方式一：可登录 EMR Hive 集群节点（具体可参见 Hive 基础操作），对目的库 testdb 库的 hive_table 表执行 chmod 操作。

```
hdfs dfs -chmod 777 /usr/hive/warehouse/testdb.db/hive_table
```

- 方式二：在**作业管理 > 作业参数**中添加以下高级参数，可以让 hadoop 用户角色获取 HDFS 路径权限。

```
containerized.taskmanager.env.HADOOP_USER_NAME: hadoop
containerized.master.env.HADOOP_USER_NAME: hadoop
```

> **说明**
> Flink SQL 中使用 Hive 表 testdb.hive_table，这里 CREATE TABLE 的表名对应 Hive 库的表名（Flink 1.13支持通过hive-table参数配置覆盖该值），库名通过 hive-database 参数指定。

WITH 参数

参数值	必填	默认值	描述
connector.type	是	无	Flink-1.11支持，填 'hive' 选择使用 hive connector。
connector	是	无	Flink-1.13支持，填 'hive' 选择使用 hive connector。
hive-version	是	无	EMR 创建的 Hive 集群对应的版本。
hive-database	是	无	数据要写入的 Hive database。
hive-table	否	无	Flink-1.13支持，填写后该值会作为Hive库的对应表名
sink.partition-commit.trigger	否	process-time	分区关闭策略。可选值包括： • process-time：当分区创建超过一定时间之后将这个分区关闭，分区创建时间为分区创建时的物理时间。 • partition-time：当分区创建超过一定时间之后将这个分区关闭，分区创建时间从分区中抽取出来。partition-time 依赖于 watermark 生成，需要配合 watermark 才能支持自动分区发现。当 watermark 时间超过了 `从分区抽取的时间` 与 `delay` `参数配置时间` 之和后会提交分区。
sink.partition-commit.delay	否	0s	分区关闭延迟。当分区在创建超过一定时间之后将被关闭。
sink.partition-commit.policy.kind	是	无	用于提交分区的策略。可选值可以组合使用，可选值包括： • success-file：当分区关闭时将在分区对应的目录下生成一个 _success 的文件。 • metastore：向 Hive Metastore 更新分区信息。 • custom：用户实现的自定义分区提交策略。
partition.time-extractor.timestamp-pattern	否	无	分区时间戳的抽取格式。需要写成 yyyy-MM-dd HH:mm:ss 的形式，并用 Hive 表中相应的分区字段做占位符替换。默认支持第一个字段为 yyyy-mm-dd hh:mm:ss。 • 如果时间戳应该从单个分区字段 'dt' 提取，可以配置 '$dt'。 • 如果时间戳应该从多个分区字段中提取，例如 'year'、'month'、'day' 和 'hour'，可以配置 '$year-$month-$day $hour:00:00'。 • 如果时间戳应该从两个分区字段 'dt' 和 'hour' 提取，可以配置 '$dt $hour:00:00'。
sink.partition-commit.policy.class	否	无	分区提交类，配合 sink.partition-commit.policy.kind = 'custom' 使用，类必须实现 PartitionCommitPolicy。

partition.time-extractor.kind	否	default	分区时间抽取方式。这个配置仅当 sink.partition-commit.trigger 配置为 partition-time 时生效。如果用户有自定义的分区时间抽取方法，配置为 custom。
partition.time-extractor.class	否	无	分区时间抽取类，这个类必须实现 PartitionTimeExtractor 接口。

代码示例1

```
CREATE TABLE datagen_source_table (
  id INT,
  name STRING,
  log_ts TIMESTAMP(3),
  WATERMARK FOR log_ts AS log_ts - INTERVAL '5' SECOND
) WITH (
  'connector' = 'datagen',
  'rows-per-second' = '10'
);

CREATE TABLE hive_table (
  `id` INT,
  `name` STRING,
  `dt` STRING,
  `hr` STRING
) PARTITIONED BY (dt, hr)
with (
    'connector' = 'hive',  -- Flink 1.13 请使用 'connector' = 'hive'
    'hive-version' = '3.1.1',
    'hive-database' = 'testdb',
    'partition.time-extractor.timestamp-pattern'='$dt $hr:00:00',
    'sink.partition-commit.trigger'='partition-time',
    'sink.partition-commit.delay'='1 h',
    'sink.partition-commit.policy.kind'='metastore,success-file'
);

-- streaming sql, insert into hive table
INSERT INTO hive_table
SELECT id, name, DATE_FORMAT(log_ts, 'yyyy-MM-dd'), DATE_FORMAT(log_ts, 'HH')
FROM datagen_source_table;
```

代码示例2

```
-- hive table 作为source 需要显示指定hive中的database，否则会用默认的数据库default_database
CREATE database testdb;

CREATE TABLE `testdb`.hive_table (
  `user_id` INT,
  `first_name` STRING,
  `dt` STRING,
  `hr` STRING
) PARTITIONED BY (dt, hr) with (
  'connector' = 'hive',
  'hive-version' = '3.1.1',
  'hive-database' = 'testdb',
  'partition.time-extractor.timestamp-pattern' = '$dt $hr:00:00',
```

```
  'streaming-source.enable' = 'true',
  'streaming-source.partition.include' = 'all' -- all 、 latest
,
  'streaming-source.monitor-interval' = '1 min',
  'streaming-source.partition-order' = 'partition-name'
);


CREATE TABLE logger_sink_table (
  `user_id` INT,
  first_name STRING,
  `dt` STRING,
  `hr` STRING
) WITH (
  'connector' = 'logger',
  'print-identifier' = 'DebugData'
);

INSERT into
  logger_sink_table
SELECT
  *
from
  `testdb`.hive_table;
```

Hive 配置

获取 Hive 连接配置 jar 包

Flink SQL 任务写 Hive 时需要使用包含 Hive 及 HDFS 配置信息的 jar 包来连接到 Hive 集群。具体获取连接配置 jar 及其使用的步骤如下：

1. ssh 登录到对应 Hive 集群节点。
2. 获取 hive-site.xml 和 hdfs-site.xml，EMR 集群中的配置文件在如下位置。

```
/usr/local/service/hive/conf/hive-site.xml
/usr/local/service/hadoop/etc/hadoop/hdfs-site.xml
```

3. 修改 hive-site.xml 文件

```
在hive-site增加如下配置，ip的值取配置文件里 hive.server2.thrift.bind.host 的 value
<property>
 <name>hive.metastore.uris</name>
 <value>thrift://ip:7004</value>
</property>
```

4. 获取 hivemetastore-site.xml 和 hiveserver2-site.xml，点击文件名下载。
5. 对获取到的配置文件 打 jar 包。

```
jar -cvf hive-xxx.jar hive-site.xml hdfs-site.xml hivemetastore-site.xml hiveserver2-site.xml
```

6. 校验 jar 的结构（可以通过 vi 命令查看 `vi hive-xxx.jar`），jar 里面包含如下信息，请确保文件不缺失且结构正确。

```
META-INF/
META-INF/MANIFEST.MF
hive-site.xml
hdfs-site.xml
hivemetastore-site.xml
```

```
hiveserver2-site.xml
```

在任务中使用配置 jar

引用程序包中选择 Hive 连接配置 jar 包（该 jar 包为在 获取 Hive 连接配置 jar 包 中得到的 hive-xxx.jar，必须在依赖管理上传后才使用）。

Kerberos 认证授权

1. 登录集群 Master 节点，获取 krb5.conf、emr.keytab、core-site.xml、hdfs-site.xml、hive-site.xml 文件，路径如下。

```
/etc/krb5.conf
/var/krb5kdc/emr.keytab
/usr/local/service/hadoop/etc/hadoop/core-site.xml
/usr/local/service/hadoop/etc/hadoop/hdfs-site.xml
/usr/local/service/hive/conf/hive-site.xml
```

2. 修改 hive-site.xml 文件。在 hive-site.xml 中增加如下配置，IP 的值取配置文件中 `hive.server2.thrift.bind.host` 的 value。

```
<property>
 <name>hive.metastore.uris</name>
 <value>thrift://ip:7004</value>
</property>
```

3. 获取 hivemetastore-site.xml 和 hiveserver2-site.xml，点击文件名下载。
4. 对获取的配置文件打 jar 包。

```
jar cvf hive-xxx.jar krb5.conf emr.keytab core-site.xml hdfs-site.xml hive-site.xml hivemetastore-
site.xml hiveserver2-site.xml
```

5. 校验 jar 的结构（可以通过 vim 命令查看 vim hive-xxx.jar），jar 里面包含如下信息，请确保文件不缺失且结构正确。

```
META-INF/
META-INF/MANIFEST.MF
emr.keytab
krb5.conf
hdfs-site.xml
core-site.xml
hive-site.xml
hivemetastore-site.xml
hiveserver2-site.xml
```

6. 在 程序包管理 页面上传 jar 包，并在作业参数配置里引用该程序包。
7. 获取 kerberos principal，用于作业 高级参数 配置。

```
klist -kt /var/krb5kdc/emr.keytab

# 输出如下所示，选取第一个即可：hadoop/172.28.28.51@EMR-OQPO48B9
KVNO Timestamp         Principal
---- ------------------- ------------------------------------------------------
   2 08/09/2021 15:34:40 hadoop/172.28.28.51@EMR-OQPO48B9
   2 08/09/2021 15:34:40 HTTP/172.28.28.51@EMR-OQPO48B9
   2 08/09/2021 15:34:40 hadoop/VM-28-51-centos@EMR-OQPO48B9
   2 08/09/2021 15:34:40 HTTP/VM-28-51-centos@EMR-OQPO48B9
```

8. 作业 高级参数 配置。

```
containerized.taskmanager.env.HADOOP_USER_NAME: hadoop
containerized.master.env.HADOOP_USER_NAME: hadoop
security.kerberos.login.principal: hadoop/172.28.28.51@EMR-OQPO48B9
security.kerberos.login.keytab: emr.keytab
security.kerberos.login.conf: ${krb5.conf.fileName}
```

> **注意：**
> 历史 Oceanus 集群可能不支持该功能，您可通过 在线客服 联系我们升级集群管控服务，以支持 Kerberos 访问。

注意事项

如果 Flink 作业正常运行，日志中没有报错，但是客户端查不到这个 Hive 表，可以使用如下命令对 Hive 表进行修复（需要将 hive_table_xxx 替换为要修复的表名）。

```
msck repair table hive_table_xxx;
```

数据仓库 Doris

最近更新时间：2025-05-08 19:47:22

介绍

Flink Connector Doris 目前支持通过 Flink 将数据写入 Doris，基于 开源版本 实现。

版本说明

Flink 版本	说明
1.11	支持
1.13	支持
1.14	支持，基于社区 release-1.1.1
1.16	支持，基于社区 release-24.0.1
1.18	支持，基于社区 release-24.0.1

> **注意：**
> Flink 1.14+（包含 1.14）默认开启两阶段（2PC）提交。

使用范围

Flink Connector Doris 目前仅支持 Doris sink。支持的 Doris 版本为0.14.0及以上版本，并且要求开启配置 `enable_http_server_v2 = true` 。

DDL 定义

> **注意：**
> Flink 1.13 跟其他 Flink 版本 的 DDL 参数不同，请选择对应的版本使用。

作为数据目的地 Sink（Flink 1.13）

```
CREATE TABLE doris_sink_table (
  id INT,
  name VARCHAR
) WITH (
  'connector' = 'doris',                    -- 固定值 'doris'
  'fenodes' = 'FE_IP:FE_HTTP_PORT',        -- Doris FE HTTP 地址
  'table.identifier' = 'test.sales_order',  -- Doris 表名 格式：db.tbl
  'username' = 'root',                      -- 访问Doris的用户名，拥有库的写权限
  'password' = 'password',                  -- 访问Doris的密码
  'sink.batch.size' = '500',                -- 单次写BE的最大行数
  'sink.batch.interval' = '1s'              -- flush 间隔时间，超过该时间后异步线程将 缓存中数据写入BE。 默认值为
1秒，支持时间单位ms、s、min、h和d。设置为0表示关闭定期写入。
);
```

作为数据目的地 Sink

```
CREATE TABLE doris_sink_table (
  id INT,
  name VARCHAR
) WITH (
  'connector' = 'doris',                    -- 固定值 'doris'
  'fenodes' = 'FE_IP:FE_HTTP_PORT',         -- Doris FE HTTP 地址
```

```
  'table.identifier' = 'test.sales_order',  -- Doris 表名 格式：db.tbl
  'username' = 'root',                      -- 访问Doris的用户名，拥有库的写权限
  'password' = 'password'                   -- 访问Doris的密码
);

-- 注意：默认打开 2PC 两阶段提交写入
```

作为 Catalog

```
CREATE CATALOG doris_catalog WITH (
  'type' = 'doris',
  'fenodes' = 'FE_IP:FE_HTTP_PORT',        -- Doris FE HTTP 地址
  'username' = 'root',                      -- 访问Doris的用户名，拥有库的写权限
  'password' = 'password',                  -- 访问Doris的密码
  'default-database' = 'default'
  )
```

WITH 参数 (Flink 1.13 版本)

Sink

参数	说明	是否必填	备注
connector	源表类型	是	固定值 `doris`
fenodes	Doris FE HTTP 地址	是	–
table.identifier	Doris 表名，格式：db1.tbl1	是	–
username	访问 Doris 的用户名	是	–
password	访问 Doris 的密码	是	–
sink.batch.size	单次写 BE 的最大行数	否	默认100
sink.max-retries	写 BE 失败之后的重试次数	否	默认1
sink.batch.interval	flush 间隔时间，超过该时间后异步线程将缓存中数据写入 BE。默认值为1秒，支持时间单位 ms、s、min、h 和 d。设置为0，表示关闭定期写入	否	默认1s
sink.properties.*	Stream load 的导入 参数 。例如 `sink.properties.column_separator' = ','` 等	否	–
sink.enable-2pc	是否采用事务写入	否	false

Catalog

参数	说明	是否必填	备注
type	–	是	固定值 `doris`
fenodes	Doris FE HTTP 地址	是	–
username	访问 Doris 的用户名	是	–
password	访问 Doris 的密码	是	–
default-database	默认的database	是	–

WITH 参数

Sink

参数	说明	是否必填	默认值
connector	源表类型	是	固定值 `doris`
fenodes	Doris FE HTTP 地址	是	-
table.identifier	Doris 表名，格式：db1.tbl1	是	-
username	访问 Doris 的用户名	是	-
password	访问 Doris 的密码	是	-
sink.properties.*	• Stream Load 的导入参数 例如： 'sink.properties.column_separator' = ', ' 定义列分隔符，'sink.properties.escape_delimiters' = 'true' 特殊字符作为分隔符，\x01 会被转换为二进制的 0x01。 • JSON 格式导入 'sink.properties.format' = 'json' 'sink.properties.read_json_by_line' = 'true'，详细参数请参考 Stream load > 导入配置参数 。 • Group Commit 模式 例如：'sink.properties.group_commit' = 'sync_mode' 设置 group commit 为同步模式。flink connector 从 1.6.2 开始支持导入配置 group commit，详细使用和限制请参考 group commit 。	否	-
sink.enable-2pc	是否开启两阶段提交 (2pc)，默认为 true，保证 Exactly-Once 语义	否	TRUE
sink.buffer-size	写数据缓存 buffer 大小，单位字节。不建议修改，默认配置即可	否	1MB
sink.buffer-count	写数据缓存 buffer 个数。不建议修改，默认配置即可	否	3
sink.max-retries	Commit 失败后的最大重试次数，默认 3 次	否	3
sink.use-cache	异常时，是否使用内存缓存进行恢复，开启后缓存中会保留 Checkpoint 期间的数据	否	false
sink.enable.batch-mode	• 是否使用攒批模式写入 Doris，开启后写入时机不依赖 Checkpoint，通过 sink.buffer-flush.max-rows/sink.buffer-flush.max-bytes/sink.buffer-flush.interval 参数来控制写入时机 • 同时开启后将不保证 Exactly-once 语义，可借助 Uniq 模型做到幂等	否	false
sink.flush.queue-size	攒批模式下，缓存的队列大小。	否	2
sink.buffer-flush.max-rows	攒批模式下，单个批次最多写入的数据行数。	否	500000
sink.buffer-flush.max-bytes	攒批模式下，单个批次最多写入的字节数。	否	100MB
sink.buffer-flush.interval	攒批模式下，异步刷新缓存的间隔	否	10s
sink.ignore.update-before	是否忽略 update-before 事件，默认忽略。	否	true

更多参数说明请参见 Flink Doris Connector 。

类型映射

Doris 字段类型	Flink 字段类型
NULL_TYPE	NULL
BOOLEAN	BOOLEAN
TINYINT	TINYINT
SMALLINT	SMALLINT
INT	INT
BIGINT	BIGINT
FLOAT	FLOAT
DOUBLE	DOUBLE
TIME	
DATE	DATE
DATETIME	TIMESTAMP
CHAR	STRING
LARGEINT	
VARCHAR	
DECIMAL	DECIMAL
DECIMALV2	
HLL	Unsupported datatype

代码示例

Flink 1.13

```
----- Flink 1.13 -----
CREATE TABLE datagen_source_table (
    id INT,
    name STRING
) WITH (
  'connector' = 'datagen',
  'rows-per-second'='1'  -- 每秒产生的数据条数
);

CREATE TABLE doris_sink_table (
  id INT,
  name STRING
) WITH (
  'connector' = 'doris',                    -- 固定值 'doris'
  'fenodes' = 'FE_IP:FE_RESFUL_PORT',       -- Doris FE HTTP 地址
  'table.identifier' = 'test.sales_order',  -- Doris 表名 格式: db.tbl
  'username' = 'root',                      -- 访问Doris的用户名，拥有库的写权限
  'password' = 'password',                  -- 访问Doris的密码
  'sink.batch.size' = '500',                -- 单次写BE的最大行数
```

```
  'sink.batch.interval' = '1s'              -- flush 间隔时间，超过该时间后异步线程将 缓存中数据写入BE。 默认值为
1秒，支持时间单位ms、s、min、h和d。设置为0表示关闭定期写入。
);

INSERT INTO doris_sink_table select * from datagen_source_table;
```

Flink 1.14、Flink 1.16、Flink 1.18

```
CREATE TABLE datagen_source_table (
    id INT,
    name STRING
) WITH (
  'connector' = 'datagen',
  'rows-per-second'='1'  -- 每秒产生的数据条数
);

CREATE TABLE doris_sink_table (
  id INT,
  name STRING
) WITH (
  'connector' = 'doris',
  'fenodes' = 'FE_IP:HTTP_PORT',
  'table.identifier' = 'db.table',
  'username' = 'root',
  'password' = 'password',
  'sink.label-prefix' = 'doris_label'
);

INSERT INTO doris_sink_table select * from datagen_source_table;
```

```
CREATE CATALOG doris_catalog WITH (
  'fenodes' = 'FE_IP:FE_RESFUL_PORT',        -- Doris FE HTTP 地址
  'username' = 'root',                       -- 访问Doris的用户名，拥有库的写权限
  'password' = 'password',                   -- 访问Doris的密码
  'default-database' = 'default'
);

CREATE TABLE datagen_source_table (
    id INT,
    name STRING
) WITH (
  'connector' = 'datagen',
  'rows-per-second'='1'  -- 每秒产生的数据条数
);

INSERT INTO `doris_catalog`.`my_database`.`my_table` SELECT * FROM.datagen_source_table;
```

MySQL-CDC 对接 Doris 代码示例：

```
--mysql cdc 源表
CREATE TABLE `mysql_cdc_source_table` (
  `id`      INT NOT NULL,
  `name`    VARCHAR,
  PRIMARY KEY (`id`) NOT ENFORCED
) WITH (
  'connector' = 'mysql-cdc',            -- 固定值 'mysql-cdc'
```

```
  'hostname' = 'YourHostName',          -- 数据库的 IP
  'port' = '3306',                      -- 数据库的访问端口
  'username' = 'YourUserName',          -- 数据库访问的用户名（需要提供 SHOW DATABASES、REPLICATION SLAVE、
REPLICATION CLIENT、SELECT 和 RELOAD 权限）
  'password' = 'YourPassword',          -- 数据库访问的密码
  'database-name' = 'YourDatabase',     -- 需要同步的数据库
  'table-name' = 'YourTable'            -- 需要同步的数据表名
);

--写入doris表
CREATE TABLE `print_table` (
  `id`       INT,
  `name`     STRING,
  PRIMARY KEY (`id`) NOT ENFORCED
) WITH (
  'connector' = 'doris',                    -- 固定值 'doris'
  'fenodes' = 'FE_IP:FE_RESFUL_PORT',       -- Doris FE HTTP 地址
  'table.identifier' = 'dbName.tableName',  -- Doris 表名 格式：db.tbl
  'username' = 'YourUserName',              -- 访问Doris的用户名，拥有库的写权限
  'password' = 'YourPassword',              -- 访问Doris的密码
);

insert into print_table
select id,name from mysql_cdc_source_table;
```

注意事项

Upsert

若需要 Upsert，则要求 Doris 表必须是 Uniqe 模型或者 Aggregate 模型。建表示例如下：

```
-- Uniqe 模型建表语句
CREATE TABLE `doris_sink_table` (
    `id`   int(11),
    `name` varchar(1024)
)
UNIQUE KEY(`id`)
DISTRIBUTED BY HASH(`id`) BUCKETS 10
PROPERTIES("replication_num" = "3");

-- Aggregate 模型建表语句
CREATE TABLE `doris_sink_table` (
    `id`   int(11),
    `name` varchar(1024) REPLACE DEFAULT '0'
)
AGGREGATE KEY('id')
DISTRIBUTED BY HASH(`id`) BUCKETS 10
PROPERTIES("replication_num" = "3");  -- 注意若 BE 节点不够，会报 `Failed to find enough host in all
backends` 错误，可适当减少该值。
```

用户权限

用户必须拥有对应的库的写权限。

```
CREATE USER 'test' IDENTIFIED BY 'test_passwd';
GRANT ALL ON test TO test;
```

更多常见问题

请参见 Doris Flink Connector 官网文档 。

数据仓库 StarRocks

最近更新时间：2025-02-27 15:48:35

介绍

flink-connector-starrocks 基于开源的 starrocks-connector-for-apache-flink v1.2.4 实现，支持通过 Flink 读写 StarRocks。

版本说明

Flink 版本	说明
1.11	不支持
1.13	支持（目的表、维表、批数据源）
1.14	支持（目的表、维表、批数据源）
1.16	支持（目的表、维表）
1.18	支持（目的表、维表）

作为数据目的（Sink）

flink-connector-starrocks 作为数据目的，用于导入数据至 StarRocks，相比于 Flink 官方提供的 flink-connector-jdbc，导入性能更佳，flink-connector-starrocks 的内部实现是通过缓存并批量由 Stream Load 导入。支持 cvs、json 两种数据格式。
以下为 MySQL-CDC 数据实时导入 StarRocks 的示例。

```
CREATE TABLE `mysql_cdc` (
  `user_id` bigint,
  `item_id` bigint,
  `behavior` STRING,
  PRIMARY KEY (`user_id`) NOT ENFORCED
) WITH (
  'connector' = 'mysql-cdc',       -- 固定值 'mysql-cdc'
  'hostname' = '9.134.34.15',    -- 数据库的 IP
  'port' = '3306',                 -- 数据库的访问端口
  'username' = 'root',         -- 数据库访问的用户名（需要提供 SHOW DATABASES、REPLICATION SLAVE、REPLICATION
CLIENT、SELECT 和 RELOAD 权限）
  'password' = 'xxx',     -- 数据库访问的密码
  'database-name' = 'test',    -- 需要同步的数据库
  'table-name' = 'user_behavior'       -- 需要同步的数据表名
);

CREATE TABLE `pk_starrocks`(
  `user_id` bigint,
  `item_id` bigint,
  `behavior` STRING,
  PRIMARY KEY (`user_id`) NOT ENFORCED
) WITH (
  'connector' = 'starrocks',
  'jdbc-url' = 'jdbc:mysql://172.28.28.98:9030',
  'load-url' = '172.28.28.98:8030',
  'database-name' = 'oceanus',
  'table-name' = 'pk_user_behavior',
  'username' = 'root',
  'password' = 'xxx',
  'sink.buffer-flush.interval-ms' = '15000',
  'sink.properties.format' = 'json',
  'sink.properties.strip_outer_array' = 'true', -- 需要设置为 true
  -- 'sink.parallelism' = '1',
  'sink.max-retries' = '3',
  'sink.semantic' = 'exactly-once'
);
```

```
INSERT INTO `pk_starrocks` SELECT * FROM `mysql_cdc`;
```

说明：
- StarRocks 表须使用 主键模型，否则源表数据删除时，无法同步到 StarRocks。
- 可用 StarRocks 的 SMT 工具，同步库表结构 到 StarRocks。

WITH 参数

参数	必填	默认值	数据类型	描述
connector	YES	NONE	String	固定值 `starrocks`
jdbc-url	YES	NONE	String	fe 节点 query_port: `jdbc:mysql://fe_ip1:query_port,fe_ip2:query_port...` ，例如 `jdbc:mysql://172.28.28.98:9030`
load-url	YES	NONE	String	fe 节点 http_port: `fe_ip1:http_port;fe_ip2:http_port....` ，例如 `172.28.28.98:8030`
database-name	YES	NONE	String	starrocks 数据库名
table-name	YES	NONE	String	starrocks 表名
username	YES	NONE	String	starrocks 用户名
password	YES	NONE	String	starrocks 用户密码
sink.semantic	NO	at-least-once	String	可选值： • `at-least-once` • `exactly-once` ，只在 checkpoint 时写数据。注意此时 `sink.buffer-flush.*` 相关参数无效
sink.version	NO	AUTO	String	可选值： • `V1` ：使用 stream-load 实现 exactly-once 语义，性能比较低，适用所有的 starrocks 版本 • `V2` ：使用 transaction-load 实现 exactly-once 语义，适用 starrocks 2.4 及之后的版本 • `AUTO` ：根据 starrocks 是否支持 transaction-load 自动选择
sink.buffer-flush.max-bytes	NO	94371840(90M)	String	批量写入参数， `sink.semantic` 为 `at-least-once` 时有效。当 buffer 中数据大小超过设置值后，触发写入 StarRocks。取值范围 `[64MB, 10GB]`
sink.buffer-flush.max-rows	NO	500000	String	批量写入参数， `sink.semantic` 为 `at-least-once` 时有效。当 buffer 中数据条数超过设置值后，触发写入 StarRocks。取值范围 `[64,000, 5000,000]`
sink.buffer-flush.interval-ms	NO	300000	String	批量写入参数， `sink.semantic` 为 `at-least-once` 时有效。间隔固定的周期（毫秒）触发写入 StarRocks。取值范围 `[1000, 3600000]`
sink.max-retries	NO	3	String	写 StarRocks 的 stream load 请求的重试次数，取值范

				围 `[0, 1000]`
sink.parallelism	NO	NULL	String	单独指定 sink 的并行度，不设置则使用全局并行度
sink.connect.timeout-ms	NO	1000	String	连接 load-url 的超时时间（毫秒），取值范围 `[100, 60000]`
sink.label-prefix	NO	NO	String	stream load label 的前缀，合法字符 `[-_A-Za-z0-9]` 。关于 lable，参考 stream load 可选参数
sink.properties.format	NO	CSV	String	导入 StarRocks 的数据格式，可选值 `CSV` 和 `JSON` ，默认为 `CSV`
sink.properties.column_separator	NO	\t	String	用于指定 CSV 格式的列分隔符，参考 CSV 适用参数
sink.properties.row_delimiter	NO	\n	String	用于指定 CSV 格式的行分隔符，参考 CSV 适用参数
sink.properties.strip_outer_array	NO	false	String	用于 JSON 格式，指定是否裁剪最外层的数组结构。取值范围： `true` 和 `false` 。默认值： `false` 。Flink 批量导入 JSON 数据在最外层有一对表示数组结构的中括号 []。**因此，需要您指定该参数取值为 `true`** ，这样 StarRocks 会剪裁掉外层的中括号 []，并把中括号 [] 里的每个内层数组都作为一行单独的数据导入。如果您指定该参数取值为 `false` ，则 StarRocks 会把整个 JSON 数据文件解析成一个数组，并作为一行数据导入。例如，待导入的 JSON 数据为 `[ {"category" : 1, "author" : 2}, {"category" : 3, "author" : 4} ]` ，如果指定该参数取值为 `true` ，则 StarRocks 会把 `{"category" : 1, "author" : 2}` 和 `{"category" : 3, "author" : 4}` 解析成两行数据，并导入到目标 StarRocks 表中对应的数据行。参考 JSON 适用参数
sink.properties.*	NO	NONE	String	stream load 属性，例如 'sink.properties.columns' = 'k1, v1'。自 StarRocks 2.4 开始，主键模型 支持部分列更新。更多参数请参考 stream load 文档

说明：

- `sink.semantic` 为 `at-least-once` 时，sink.buffer-flush.max-bytes、sink.buffer-flush.max-rows、sink.buffer-flush.interval-ms 任意条件满足时触发 StarRocks 写操作。
- `sink.semantic` 为 `exactly-once` 时，依赖 flink 的 checkpoint，在每次 checkpoint 时保存批数据以及其 label，在checkpoint 完成后的第一次 invoke 中阻塞 flush 所有缓存在 state 当中的数据，以此达到精准一次。此时 sink.buffer-flush.* 相关参数无效。

类型映射（Sink）

Flink type	StarRocks type
BOOLEAN	BOOLEAN
TINYINT	TINYINT
SMALLINT	SMALLINT
INTEGER	INTEGER
BIGINT	BIGINT

FLOAT	FLOAT
DOUBLE	DOUBLE
DECIMAL	DECIMAL
BINARY	INT
CHAR	STRING
VARCHAR	STRING
STRING	STRING
DATE	DATE
TIMESTAMP_WITHOUT_TIME_ZONE(N)	DATETIME
TIMESTAMP_WITH_LOCAL_TIME_ZONE(N)	DATETIME
ARRAY<T>	ARRAY<T>
MAP<KT,VT>	JSON STRING
ROW<arg T...>	JSON STRING

注意：当前不支持 Flink 的 BYTES、VARBINARY、TIME、INTERVAL、MULTISET、RAW，具体可参考 Flink 数据类型 。

注意事项

StarRocks 数据模型

StarRocks 支持四种 数据模型：明细模型、聚合模型、更新模型、主键模型。主键模型支持谓词和索引下推，能够在支持实时和频繁更新等场景的同时，提供高效查询。如果没有特殊需求，推荐使用主键模型。

- 对于 Upsert 流，需要使用主键模型，否则 DELETE 消息无法写入 StarRocks。
- 相对于 Merge-On-Read 策略的更新模型，主键模型的查询性能能够提升 3~10 倍。
- 主键模型可利用部分列更新实现多流 JOIN。

作为数据源（Source）

WITH 参数

参数	必填	默认值	数据类型	描述
connector	YES	NONE	String	固定值 `starrocks`
scan-url	YES	NONE	String	fe 节点 http_port: `fe_ip1:http_port,fe_ip2:http_port....` ，例如 `172.28.28.98:8030`
jdbc-url	YES	NONE	String	fe 节点 query_port: `jdbc:mysql://fe_ip1:query_port,fe_ip2:query_port...` ，例如 `jdbc:mysql://172.28.28.98:9030`
username	YES	NONE	String	StarRocks 用户名
password	YES	NONE	String	StarRocks 用户密码
database-name	YES	NONE	String	StarRocks 数据库名

table-name	YES	NONE	String	StarRocks 表名
scan.connect.timeout-ms	NO	1000	String	网络连接超时时间（毫秒）
scan.params.keep-alive-min	NO	10	String	最大 keep alive 时间（分钟）
scan.params.query-timeout-s	NO	600(5min)	String	单次查询超时时间（秒）
scan.params.mem-limit-byte	NO	102410241024(1G)	String	单次查询内存限制
scan.max-retries	NO	1	String	重试次数
lookup.cache.ttl-ms	NO	5000	Long	维表查询的 cache 超时时间

批数据源

```
CREATE TABLE `starrocks` (
  `user_id` bigint,
  `item_id` bigint,
  `behavior` STRING,
  PRIMARY KEY (`user_id`) NOT ENFORCED
) WITH (
  'connector' = 'starrocks' ,
  'jdbc-url' = 'jdbc:mysql://172.28.28.98:9030', -- query_port, FE mysql server port
  'scan-url' = '172.28.28.98:8030', -- http_port
  'database-name' = 'oceanus',
  'table-name' = 'pk_user_behavior',
  'username' = 'root',
  'password' = 'xxx'
);

CREATE TABLE `print_sink` (
  `user_id` BIGINT,
  `item_id` BIGINT,
  `behavior` STRING,
  PRIMARY KEY (`user_id`) NOT ENFORCED
) WITH (
  'connector' = 'logger'
);

INSERT INTO `print_sink`
SELECT * FROM starrocks;
```

维表

```
CREATE TABLE `starrocks` (
  `user_id` bigint,
  `item_id` bigint,
  `behavior` STRING,
  PRIMARY KEY (`user_id`) NOT ENFORCED
) WITH (
    'connector' = 'jdbc',
    'url' = 'jdbc:mysql://172.28.28.98:9030/CDB?
rewriteBatchedStatements=true&serverTimezone=Asia/Shanghai', -- query_port, FE mysql server port
    'table-name' = 'my-table', -- 需要写入的数据表
```

```
    'username' = 'admin',       -- 数据库访问的用户名（需要提供 INSERT 权限）
    'password' = 'MyPa$$w0rd',  -- 数据库访问的密码
    'lookup.cache.max-rows' = '100',    -- 读缓存大小
    'lookup.cache.ttl' = '5000'        -- 读缓存的 TTL
);

CREATE TABLE `datagen` (
  `user_id` BIGINT,
  `ts` as PROCTIME(),
  PRIMARY KEY (`user_id`) NOT ENFORCED
) WITH (
  'connector' = 'datagen',
  'rows-per-second' = '1',
  'fields.user_id.min' = '1',
  'fields.user_id.max' = '20'
);

CREATE TABLE `print_sink` (
  `user_id` BIGINT,
  `item_id` BIGINT,
  `behavior` STRING,
  `ts` TIMESTAMP,
  PRIMARY KEY (`user_id`) NOT ENFORCED
) WITH (
  'connector' = 'logger'
);

INSERT INTO `print_sink`
SELECT a.user_id,b.item_id,b.behavior,a.ts
FROM `datagen` a LEFT JOIN `starrocks` FOR SYSTEM_TIME AS OF a.ts as b
ON a.user_id = b.user_id;
```

类型映射（source）

StarRocks	Flink
NULL	NULL
BOOLEAN	BOOLEAN
TINYINT	TINYINT
SMALLINT	SMALLINT
INT	INT
BIGINT	BIGINT
LARGEINT	STRING
FLOAT	FLOAT
DOUBLE	DOUBLE
DATE	DATE
DATETIME	TIMESTAMP
DECIMAL	DECIMAL
DECIMALV2	DECIMAL

DECIMAL32	DECIMAL
DECIMAL64	DECIMAL
DECIMAL128	DECIMAL
CHAR	CHAR
VARCHAR	STRING

版权所有：腾讯云计算（北京）有限责任公司

数据仓库 TCHouse-P

最近更新时间：2024-01-12 18:21:41

介绍

JDBC-PG Connector 提供了对 云数据仓库 TCHouse-P 写支持。

版本说明

Flink 版本	说明
1.11	暂不支持
1.13	支持
1.14	不支持
1.16	支持

使用范围

支持用作数据目的表（sink），用于 Tuple 数据流表和用于 Upsert 数据流表（需要指定主键）。

DDL 定义

用作数据目的（Tuple Sink）

```
CREATE TABLE `jdbc_sink_table` (
    `id` INT,
    `name` STRING
) WITH (
    -- 指定数据库连接参数
    'connector' = 'jdbcPG',
    'url' = 'jdbc:postgresql://10.1.28.93:3306/CDB?
reWriteBatchedInserts=true&serverTimezone=Asia/Shanghai', -- 请替换为您的实际 PG 连接参数
    'table-name' = 'my-table',  -- 需要写入的数据表
    'username' = 'admin',       -- 数据库访问的用户名（需要提供 INSERT 权限）
    'password' = 'MyPa$$w0rd',  -- 数据库访问的密码
    'sink.buffer-flush.max-rows' = '200',  -- 批量输出的条数
    'sink.buffer-flush.interval' = '2s'    -- 批量输出的间隔
);
```

用作数据目的（Upsert Sink）

```
CREATE TABLE `jdbc_upsert_sink_table` (
    `id` INT PRIMARY KEY NOT ENFORCED,
    `name` STRING
) WITH (
    -- 指定数据库连接参数
    'connector' = 'jdbcPG',
    'url' = 'jdbc:postgresql://10.1.28.93:3306/CDB?
reWriteBatchedInserts=true&serverTimezone=Asia/Shanghai', -- 请替换为您的实际 PG 连接参数
    'table-name' = 'my-upsert-table', -- 需要写入的数据表
    'username' = 'admin',             -- 数据库访问的用户名（需要提供 INSERT 权限）
    'password' = 'MyPa$$w0rd',        -- 数据库访问的密码
    'sink.buffer-flush.max-rows' = '200',  -- 批量输出的条数
    'sink.buffer-flush.interval' = '2s'    -- 批量输出的间隔
);
```

> **注意**
> Upsert 需要定义 PRIMARY KEY。

WITH 参数

参数值	必填	默认值	描述
connector	是	无	连接数据库时，需要填写 `'jdbcPG'` 。
url	是	无	JDBC 数据库的连接 URL。
table-name	是	无	数据库表名。
driver	否	无	JDBC Driver 的类名。如果不输入，则自动从 url 中推断。
username	否	无	数据库用户名。 `'username'` 和 `'password'` 必须同时使用。
password	否	无	数据库密码。
scan.partition.column	否	无	指定对输入分区扫描（Partitioned Scan）的列名，该列必须是数值类型、日期类型、时间戳类型等。关于分区扫描的细节，可参见 分区扫描 。
scan.partition.num	否	无	分区扫描启用后，指定分区数。
scan.partition.lower-bound	否	无	分区扫描启用后，指定首个分区的最小值。
scan.partition.upper-bound	否	无	分区扫描启用后，指定最后一个分区的最大值。
scan.fetch-size	否	0	每次从数据库读取时，批量获取的行数。默认为0，表示一行一行读取，效率较低（吞吐量不高）。
lookup.cache.max-rows	否	无	查询缓存（Lookup Cache）中最多缓存的数据条数。
lookup.cache.ttl	否	无	查询缓存中每条记录最长的缓存时间。
lookup.max-retries	否	3	数据库查询失败时，最多重试的次数。
sink.buffer-flush.max-rows	否	100	批量输出时，缓存中最多缓存多少数据。如果设置为0，表示禁止输出缓存。
sink.buffer-flush.interval	否	1s	批量输出时，每批次最大的间隔（毫秒）。**如果 `'sink.buffer-flush.max-rows'` 设为 `'0'` ，而这个选项不为零，则说明启用纯异步输出功能，即数据输出到算子、从算子最终写入数据库这两部分线程完全解耦。**
sink.max-retries	否	3	数据库写入失败时，最多重试的次数。
write-mode	否	insert	以 copy 模式写入，可选 insert/copy
copy-delimiter	否	&#x399	copy 模式下面，字段分割符

代码示例

```
CREATE TABLE jdbc_upsert_sink_table (
  id INT ,
  age INT,
  name STRING,
  PRIMARY KEY (id) NOT ENFORCED -- 如果要同步的数据库表定义了主键，则这里也需要定义
) WITH (
    -- 指定数据库连接参数
```

```
    'connector' = 'jdbcPG',
    'url' = 'jdbc:postgresql://10.0.0.2:5436/postgres',
    'table-name' = 'my-upsert-table', -- 需要写入的数据表
    'username' = 'admin',             -- 数据库访问的用户名（需要提供 INSERT 权限）
    'password' = 'password',         -- 数据库访问的密码
    'sink.buffer-flush.max-rows' = '300',  -- 批量输出的条数
    'sink.buffer-flush.interval' = '100s'    -- 批量输出的间隔
);


-- MySQL CDC Source。配合 flink-connector-mysql-cdc 使用。要求 MySQL 版本 >= 5.7
-- 参见 https://cloud.tencent.com/document/product/849/52698
CREATE TABLE mysql_cdc_source_table (
  id INT,
  age INT,
  name STRING,
  PRIMARY KEY (id) NOT ENFORCED -- 如果要同步的数据库表定义了主键，则这里也需要定义
) WITH (
  'connector' = 'mysql-cdc',       -- 固定值 'mysql-cdc'
  'hostname' = '10.0.0.6',    -- 数据库的 IP
  'port' = '3360',                 -- 数据库的访问端口
  'username' = 'root',         -- 数据库访问的用户名（需要提供 SHOW DATABASES、REPLICATION SLAVE、REPLICATION
CLIENT、SELECT 和 RELOAD 权限）
  'password' = 'password',     -- 数据库访问的密码
  -- 'scan.incremental.snapshot.enabled' = 'false' -- 如果 source 表没有设置 PRIMARY Key，需要启用该设置
  'database-name' = 'database',    -- 需要同步的数据库
  'table-name' = 'table'       -- 需要同步的数据表名
);


INSERT INTO jdbc_upsert_sink_table select * from mysql_cdc_source_table;
```

主键说明

- 对于 Append（Tuple）数据，作业 DDL 不需要设置主键，JDBC 数据库的表也不需要定义主键，也不建议定义主键（否则可能因为重复数据而造成写入失败）。
- 对于 Upsert 数据，JDBC 数据库的表**必须**定义主键，并需要在 DDL 语句的 CREATE TABLE 中，将相应列名也加上 `PRIMARY KEY NOT ENFORCED` 约束。

> **注意**
> Upsert 功能实现不依赖于 `INSERT .. ON CONFLICT .. DO UPDATE SET ..` 语法，而是依赖于攒批 upsert +攒批 delete。该语法兼容 PostgreSQL 9.5以下版本。

批量写入优化

通过设置 sink.buffer-flush 开头的两个参数，可以实现批量写入数据库。建议配合相应底层数据库的参数，以达到更好的批量写入效果，否则底层仍然会一条一条写入，效率不高。

- 对于 PostgreSQL，建议在 url 连接参数后加入 reWriteBatchedInserts=true 参数。

```
jdbc:postgresql://10.1.28.93:3306/PG?reWriteBatchedInserts=true&currentSchema=数据库的Schema
```

数据仓库 ClickHouse

最近更新时间：2024-12-23 15:34:12

介绍

ClickHouse Sink Connector 提供了对 ClickHouse 数据仓库的写入支持。ClickHouse Source Connector 提供了 ClickHouse 作为批数据源和维表的功能。

版本说明

Flink 版本	说明
1.11	支持 Sink
1.13	支持 Source 和 Sink
1.14	支持 Source 和 Sink
1.16	支持 Source 和 Sink

使用范围

ClickHouse 不支持标准的 update 和 delete 操作。作为 Sink 时，若您的任务有 update 和 delete 操作，可以通过 CollapsingMergeTree 来实现。
对于 JAR 作业，Java/Scala 编写的作业可以参考 JDBC 方式写入 ClickHouse，这里不做阐述。

DDL 定义

用作数据目的（不包含更新）（Sink with insert only）

```
CREATE TABLE clickhouse_sink_table (
   `id` INT,
   `name` STRING
) WITH (
   -- 指定数据库连接参数
   'connector' = 'clickhouse',                        -- 指定使用clickhouse连接器
   'url' = 'clickhouse://172.28.28.160:8123',         -- 指定集群地址，可以通过ClickHouse集群界面查看
   -- 如果ClickHouse集群未配置账号密码可以不指定
   --'username' = 'root',                             -- ClickHouse集群用户名
   --'password' = 'root',                             -- ClickHouse集群的密码
   'database-name' = 'db',                            -- 数据写入目的数据库
   'table-name' = 'table',                            -- 数据写入目的数据表
   'sink.batch-size' = '1000'                         -- 触发批量写的条数
);
```

包含 update 和 delete 操作的数据目的（Sink with upsert）

```
CREATE TABLE clickhouse_upsert_sink_table (
   `id` INT,
   `name` STRING,
  PRIMARY KEY (`id`) NOT ENFORCED -- 如果要同步的数据库表定义了主键，则这里也需要定义
) WITH (
   -- 指定数据库连接参数
   'connector' = 'clickhouse',                        -- 指定使用clickhouse连接器
   'url' = 'clickhouse://172.28.28.160:8123',         -- 指定集群地址，可以通过ClickHouse集群界面查看
   -- 如果ClickHouse集群未配置账号密码可以不指定
   --'username' = 'root',                             -- ClickHouse集群用户名
   --'password' = 'root',                             -- ClickHouse集群的密码
```

```
    'database-name' = 'db',                          -- 数据写入目的数据库
    'table-name' = 'table',                          -- 数据写入目的数据表
    'table.collapsing.field' = 'Sign',               -- CollapsingMergeTree 类型列字段的名称
    'sink.batch-size' = '1000'                       -- 触发批量写的条数
);
```

> **注意**
> 一定要定义 pk 和申明 table.collapsing.field 字段，才会启动该功能。对应的 ClickHouse 建表语句，请参考 常见问题 。

作为批数据源

```
CREATE TABLE `clickhosue_batch_source` (
    `when`      TIMESTAMP,
    `userid`    BIGINT,
    `bytes`     FLOAT
) WITH (
    'connector' = 'clickhouse',
    'url' = 'clickhouse://172.28.1.21:8123',
    'database-name' = 'dts',
    'table-name' = 'download_dist'
--     'scan.by-part.enabled' = 'false', -- 是否启用读 ClickHouse 表 part。若启用，必须先在所有节点上使用命令
'STOP MERGES' 和 'STOP TTL MERGES' 停止表的后台 merge 和基于 TTL 的数据删除操作，否则读取的数据会不正确。
--     'scan.part.modification-time.lower-bound' = '2021-09-24 16:00:00', -- 用于根据 modification_time 过
滤 ClickHouse 表 part 的最小时间（包含），格式 yyyy-MM-dd HH:mm:ss。
--      'scan.part.modification-time.upper-bound' = '2021-09-17 19:16:26', -- 用于根据 modification_time
过滤 ClickHouse 表 part 的最大时间（不包含），格式 yyyy-MM-dd HH:mm:ss。
--     'local.read-write' = 'false', -- 是否读本地表，默认 false 。
--     'table.local-nodes' = '172.28.1.24:8123,172.28.1.226:8123,172.28.1.109:8123,172.28.1.36:8123' --
local node 列表，需要使用 http port。注意一个 shard 只能配置一个 replica 节点地址，否则会读取到重复数据。
);
```

> **注意**
> MergeTree 系列引擎才支持按 part 读取，且一定要停止待读取表所有节点的后台 merge 和基于 TTL 的数据删除操作，避免 part 变化导致数据读取不准确。本地读时一个 shard 只能配置一个 replica 地址，避免重复数据。

作为维表

```
CREATE TABLE `clickhouse_dimension` (
    `userid` BIGINT,
    `comment` STRING
) WITH (
    'connector' = 'clickhouse',
    'url' = 'clickhouse://172.28.1.21:8123',
    'database-name' = 'dimension',
    'table-name' = 'download_dist',
    'lookup.cache.max-rows' = '500', -- 查询缓存（Lookup Cache）中最多缓存的数据条数。
    'lookup.cache.ttl' = '10min', -- 查询缓存中每条记录最长的缓存时间。
    'lookup.max-retries' = '10' -- 数据库查询失败时，最多重试的次数。
);
```

WITH 参数

参数值	必填	默认值	描述

connector	是	-	当要使用 ClickHouse 作为数据目的（Sink）需要填写，取值 `clickhouse` 。
url	是	-	ClickHouse 集群连接 url，可以通过集群界面查看，举例 'clickhouse://127.1.1.1:8123'。
username	否	-	ClickHouse 集群用户名。
password	否	-	ClickHouse 集群密码。
database-name	是	-	ClickHouse 集群数据库。
table-name	是	-	ClickHouse 集群数据表。
sink.batch-size	否	1000	Connector batch 写入的条数。
sink.flush-interval	否	1000（单位 ms）	Connector 异步线程刷新写入 ClickHouse 间隔。
table.collapsing.field	否	-	CollapsingMergeTree 类型列字段的名称。
sink.max-retries	否	3	写入失败时的重试次数。
local.read-write	否	false	是否启用直写本地表功能。默认不开启。 注意：该功能仅供高级用户使用，需配合下面几个参数使用。详见本文后续的“写入本地表”章节。
table.local-nodes	否	-	启用写本地表（`local.read-write` 为 `true` ）时，local node 列表，举例 '127.1.1.10:8123,127.1.2.13:8123'（**需要使用 http port**）。
sink.partition-strategy	否	balanced	启用写本地表（`local.read-write` 为 `true` ）时，需要设置数据分发策略，支持 balanced/shuffle/hash。如果希望实现数据的动态更新，且表引擎使用 CollapsingMergeTree，则取值必须为 `hash` ，且需要配合 `sink.partition-key` 一同使用。 取值说明：`balanced` 轮询模式写入，`shuffle` 随机挑选节点写入，`hash` 根据 `sink.partition-key` 的 hash 值，选择节点写入。
sink.partition-key	否	-	启用写本地表（`local.read-write` 为 `true` ），且 `sink.partition-strategy` 为 `hash` 时需要设置，值为所定义表中的主键。如果主键包含多个字段，则需要指定为第一个字段。
sink.ignore-delete	否	false	启用该参数后，会过滤所有向 ClickHouse 写入的 DELETE（删除）消息。该选项适用于使用 ReplacingMergeTree 表引擎，并期望实现数据的动态更新的场景。
sink.backpressure-aware	否	false	当 Flink 日志频繁出现 "Too many parts" 报错，且作业因此崩溃时，启用该参数可以大幅减轻服务端负载，提升整体的吞吐量和稳定性。
sink.reduce-batch-by-key	否	false	开启该参数后，对于定义了主键的 ClickHouse Sink 表，在给定的刷新周期内，同 Key 的数据会做归并，只取最后一条。
sink.max-partitions-per-insert	否	20	当clickhouse是分区表，且分区函数CK内置为intHash32、toYYYYMM 或 toYYYYMMDD 之一时，Flink写入Clickhouse会通过预先在sink端按分区攒数据buffer，当攒的分区数目到达设定值时会触发往下游clickhouse写入（如果 `sink.flush-interval` 和 `sink.batch-size` 先到的话也会先触发写入），极大的提高写入clickhouse的吞吐效率。设置为-1时会关闭分区聚合写入功能。
scan.fetch-size	否	100	每次从数据库读取时，批量获取的行数。
scan.by-part.enabled	否	false	是否启用读 ClickHouse 表 part。若启用，必须先在所有节点上使用命令'STOP MERGES'和'STOP TTL MERGES'停止表的后台 merge 和基于 TTL 的数据删除操作，否则读取的数据会不正确。

scan.part.modification-time.lower-bound	否	-	用于根据 modification_time 过滤 ClickHouse 表 part 的最小时间（包含），格式 yyyy-MM-dd HH:mm:ss。
scan.part.modification-time.upper-bound	否	-	用于根据 modification_time 过滤 ClickHouse 表 part 的最大时间（不包含），格式 yyyy-MM-dd HH:mm:ss。
lookup.cache.max-rows	否	无	查询缓存（Lookup Cache）中最多缓存的数据条数。
lookup.cache.ttl	否	无	查询缓存中每条记录最长的缓存时间。
lookup.max-retries	否	3	数据库查询失败时，最多重试的次数。

注意

定义 WITH 参数时，通常只需要填写必填参数即可。当您启用非必须参数时，请您一定要明确参数含义以及可能对数据写入产生的影响。

类型映射

关于 ClickHouse 支持的数据类型定义及其使用，可参考 ClickHouse data-types，这里列举了常用的数据类型，及其与 Flink 类型的对应关系。无特殊情况，建议按下表映射 Flink 数据类型，以避免意想不到的结果。特别注意：

- DateTime：由于在 ClickHouse 中时间精度为 1 秒，且 ClickHouse 默认的配置项 date_time_input_format 为 `basic`，只能解析 `YYYY-MM-DD HH:MM:SS` 或者 `YYYY-MM-DD` 格式的时间，如果您的作业出现时间解析异常（例如 `java.sql.SQLException: Code: 6. DB::Exception: Cannot parse string '2023-05-24 14:34:55.166' as DateTime`），请参考下表调整 Flink 中对应的数据类型为 `TIMESTAMP(0)`，或者调整 ClickHouse 集群的 date_time_input_format 值为 `best_effort`。此外 ClickHouse 支持以整数格式插入 DateTime 数据，因此您也可以在 Flink 中映射类型为 INTEGER，但不推荐。
- DateTime64：ClickHouse 支持以整数和 DECIMAL 格式插入 DateTime64 数据，因此您可能会在 Flink 中以 BIGINT 或者 DECIMAL 映射。如果您映射为 BIGINT 的 Flink 数据类型，请特别注意写入的 BIGINT 值要与 DateTime64 的精度匹配。例如 ClickHouse 中创建表时 DateTime64 默认的精度为毫秒，即 DateTime64(3)，写入的 BIGINT 值也需要为毫秒级，示例 1672542671000（`2023-01-01 11:11:11.000`）。为了避免相关问题，建议参考下表映射为 TIMESTAMP 数据类型。

ClickHouse 数据类型	Flink 数据类型	Java 数据类型
String	VARCHAR/STRING	String
FixedString(N)	VARCHAR/STRING	String
Bool	BOOLEAN	Byte
Int8	TINYINT	Byte
UInt8	SMALLINT	Short
Int16	SMALLINT	Short
UInt16	INTEGER	Integer
Int32	INTEGER	Integer
UInt32	BIGINT	Long
Int64	BIGINT	Long
UInt64	BIGINT	Long
Int128	DECIMAL	BigInteger
UInt128	DECIMAL	BigInteger

Int256	DECIMAL	BigInteger
UInt256	DECIMAL	BigInteger
Float32	FLOAT	Float
Float64	DOUBLE	Double
Decimal(P,S)/Decimal32(S)/Decimal64(S)/Decimal128(S)/Decimal256(S)	DECIMAL	BigDecimal
Date	DATE	LocalDateTime
DateTime([timezone])	TIMESTAMP(0)	LocalDateTime
DateTime64(precision, [timezone])	TIMESTAMP(precision)	LocalDateTime
Array(T)	ARRAY<T>	T[]
Map(K, V)	MAP<K, V>	Map<?, ?>
Tuple(T1, T2, ...)	ROW<f1 T1, f2 T2, ...>	List<Object>

代码示例

```
CREATE TABLE datagen_source_table (
    id INT,
    name STRING
) WITH (
   'connector' = 'datagen',
   'rows-per-second'='1'  -- 每秒产生的数据条数
);
CREATE TABLE clickhouse_sink_table (
   `id` INT,
   `name` STRING
) WITH (
   -- 指定数据库连接参数
   'connector' = 'clickhouse',                        -- 指定使用clickhouse连接器
   'url' = 'clickhouse://172.28.28.160:8123',         -- 指定集群地址，可以通过ClickHouse集群界面查看
   -- 如果ClickHouse集群未配置账号密码可以不指定
   --'username' = 'root',                            -- ClickHouse集群用户名
   --'password' = 'root',                            -- ClickHouse集群的密码
   'database-name' = 'db',                            -- 数据写入目的数据库
   'table-name' = 'table',                            -- 数据写入目的数据表
   'sink.batch-size' = '1000'                         -- 触发批量写的条数
);
insert into clickhouse_sink_table select * from datagen_source_table;
```

常见问题

关于数据更新（Upsert）和删除（Delete）语义

对于需要动态更新和删除数据的场景，由于 ClickHouse 并不完全支持 Upsert 语义，我们通常使用 ReplacingMergeTree 或 CollapsingMergeTree 来模拟更新或删除操作，它们有各自的适用场景。

ReplacingMergeTree 表引擎

如果 ClickHouse 底层表引擎使用 ReplacingMergeTree，则可以设置 `sink.ignore-delete` 参数为 `true` 。此后 Flink 会自动过滤所有的删除（DELETE）消息，并将插入（INSERT）和更新（UPDATE_AFTER）消息则统一转为插入（INSERT）消息。随后，ClickHouse 底层会自动使用最新写入的记录来覆盖之前同主键的旧记录，从而实现数据的更新。

需要注意的是，此模式只支持插入、更新，而**不支持删除**数据。因此如果对于 CDC 等需要精确同步的简单 ETL 场景，则使用下面介绍的 CollapsingMergeTree 表引擎会有更好的效果。

CollapsingMergeTree 表引擎

如果 ClickHouse 底层表引擎使用 CollapsingMergeTree，则可以通过 `table.collapsing.field` 参数来指定 Sign 字段。它的原理是通过发送内容相同，但符号（Sign）相反的消息，来实现旧数据的删除（抵消），以及新数据的插入。
此外，在生产环境中，一般会使用 ReplicatedCollapsingMergeTree，而 `ReplicatedMergeTree` 的自动去重可能会使得短期内多次写入到 ClickHouse 的数据被判断为重复数据，导致数据丢失。此时，可在建表（或者修改表）时，指定 `replicated_deduplication_window=0`，以关闭自动去重功能。
例如：

```
CREATE TABLE testdb.testtable on cluster default_cluster (`id` Int32,`name` Nullable(String),`age`
Nullable(Int32),`weight` Nullable(Float64),`Sign` Int8) ENGINE =
ReplicatedCollapsingMergeTree('/clickhouse/tables/{layer}-{shard}/testdb/testtable', '{replica}', Sign)
ORDER BY id SETTINGS  replicated_deduplication_window = 0;
```

自动去重更多详情可参见 Data Replication 。

ClickHouse 分布式表的 sharding_key

创建分布式表时，语句中的 `ENGINE = Distributed(cluster_name, database_name, table_name[, sharding_key]);` 中 sharding_key 需为 Flink SQL 的 sink 表中的主键（Primary Key），**以保证同一个 Primary Key 的记录写入到同一个节点中。**
各个参数的含义分别如下：

- cluster_name：集群名称，与集群配置中的自定义名称相对应。
- database_name：数据库名称。
- table_name：表名称。
- sharding_key：选填，用于分片的 key 值，在数据写入的过程中，分布式表会依据分片 key 的规则，将数据分布到各个节点的本地表。

写入本地表（高级内容）

如果将 `local.read-write` 参数设置为 `true`，则 Flink 可以直接写入本地表。

> **注意**
> 启用写本地表功能后，吞吐量可能会大幅提升。但是如果后续 ClickHouse 集群发生扩容、缩容、节点替换时，可能造成数据分布不均匀甚至写入失败、数据无法更新。因此仅供高级用户使用。

如果表含有主键，且使用了 UPDATE 和 DELETE 语义，建议建表时使用 CollapsingMergeTree 表引擎。我们可以通过 `table.collapsing.field` 参数来指定 Sign 字段，并设置 `sink.partition-strategy` 为 `hash` 以令相同主键的数据落在同一个 shard 上，并将 `sink.partition-key` 参数设置为主键字段（对于多字段的混合主键，则设置为第一个字段）。

关于 null 数据

若数据的某些字段可能为空，则需要在 ClickHouse 的建表语句（DDL）中，把字段声明改为 Nullable，否则会导致数据写入异常。

```
CREATE TABLE testdb.testtable on cluster default_cluster (`id` Int32,`name` Nullable(String),`age`
Nullable(Int32),`weight` Nullable(Float64),`Sign` Int8) ENGINE =
ReplicatedCollapsingMergeTree('/clickhouse/tables/{layer}-{shard}/testdb/testtable', '{replica}', Sign)
ORDER BY id ;
```

分区表写入优化

Oceanus 在写入到 Clickhouse 分区表时，如果 clickhouse 的分区定义中使用的函数在 Oceanus 的支持范围内（intHash32、toYYYYMM、toYYYYMMDD），写入时默认会开启按分区预先攒 buffer 数据再写入的功能。启用后，Oceanus 单批次写入数据会包含尽量少的分区数量（业务分区数据吞吐足够大时，每个批次仅包含一个分区），从而提高 ClickHouse 的 merge 性能。如果单个分区数据不够批大小时，多个分区数据合并成一个批次写入 ClickHouse。详细的配置项可以参见上文中 `sink.max-partitions-per-insert` 参数。

监控指标说明

Oceanus 为 ClickHouse Connector 增加了很多实用的统计指标。单击 Flink UI 的运行图中的 ClickHouse Sink 算子，即可搜索并查看指标：
numberOfInsertRecords：获取输出的 +I 消息数。
numberOfDeleteRecords：获取输出的 -D 消息数。
numberOfUpdateBeforeRecords：获取输出的 -U 消息数。
numberOfUpdateAfterRecords：获取输出的 +U 消息数。

示例：ClickHouse 建表语句

支持 UPDATE、DELETE 的 CollapsingMergeTree 建表语句

```
-- 创建数据库
CREATE DATABASE test ON cluster default_cluster;

-- 创建本地表
CREATE TABLE test.datagen ON cluster default_cluster (`id` Int32,`name` Nullable(String),`age`
Nullable(Int32),`weight` Nullable(Float64),`Sign` Int8) ENGINE =
ReplicatedCollapsingMergeTree('/clickhouse/tables/{layer}-{shard}/test/datagen', '{replica}', Sign)
ORDER BY id SETTINGS replicated_deduplication_window = 0;

-- 基于本地表，创建分布式表
CREATE TABLE test.datagen_all ON CLUSTER default_cluster AS test.datagen ENGINE =
Distributed(default_cluster, test, datagen, id);
```

仅包含 INSERT 的普通 MergeTree 建表语句

```
-- 创建数据库
CREATE DATABASE test ON cluster default_cluster;

-- 创建本地表
CREATE TABLE test.datagen ON cluster default_cluster (`id` Int32,`name` Nullable(String),`age`
Nullable(Int32),`weight` Nullable(Float64) ) ENGINE = ReplicatedMergeTree('/clickhouse/tables/{layer}-
{shard}/test/datagen', '{replica}') ORDER BY id SETTINGS replicated_deduplication_window = 0;

-- 基于本地表，创建分布式表
CREATE TABLE test.datagen_all ON CLUSTER default_cluster AS test.datagen ENGINE =
Distributed(default_cluster, test, datagen, id);
```

版权所有：腾讯云计算（北京）有限责任公司

数据仓库 Kudu

最近更新时间：2025-05-08 19:47:22

介绍

Kudu Connector 提供了对 Kudu 的读写支持。

版本说明

Flink 版本	说明
1.11	支持
1.13	支持
1.14	不支持
1.16	支持

使用范围

Kudu Connector 支持用作数据源表（Source，仅限于普通和维表 JOIN 的右表），也可以作为 Tuple 数据流的目的表（Sink），还可以作为 Upsert 数据流的目的表（Sink，需要包含主键）。

DDL 定义

用作数据源（Source）

```
CREATE TABLE `kudu_source_table` (
    `id` INT,
    `name` STRING
) WITH (
    -- 指定Kudu连接参数
    'connector' = 'kudu',
    'kudu.masters' = 'master-01:7051,master-02:7051,master-03:7051', -- 连接地址
    'kudu.table' = 'TableName1', -- 替换为 Kudu 中对应的表，如 default.TestTable1
    'kudu.hash-columns' = 'id', -- 可选参数，Hash 键
    'kudu.primary-key-columns' = 'id', -- 可选参数，主键
    'kudu.operation-timeout' = '10000', -- 可选参数，插入超时时间
    'kudu.max-buffer-size' = '2000', -- 可选参数，buffer 大小
    'kudu.flush-interval' = '1000' -- 可选参数，刷新数据到 kudu 的时间间隔
);
```

用作数据目的（Tuple Sink）

```
CREATE TABLE `kudu_sink_table` (
    `id` INT,
    `name` STRING
) WITH (
    -- 指定Kudu连接参数
    'connector' = 'kudu',
    'kudu.masters' = 'master-01:7051,master-02:7051,master-03:7051', -- 连接地址
    'kudu.table' = 'TableName1', -- 替换为Kudu中对应的表，如 default.TestTable1
    'kudu.ignore-duplicate' = 'true' --可选参数，为 true 时会忽略主键重复的数据
);
```

用作数据目的（Upsert Sink）

```sql
CREATE TABLE `kudu_upsert_sink_table` (
    `id` INT,
    `name` STRING
) WITH (
    -- 指定 Kudu 连接参数
    'connector' = 'kudu',
    'kudu.masters' = 'master-01:7051,master-02:7051,master-03:7051', -- 连接地址
    'kudu.table' = 'TableName1', -- 替换为 Kudu 中对应的表，如default.TestTable1
    'kudu.hash-columns' = 'id', -- 可选参数，Hash 键
    'kudu.primary-key-columns' = 'id' -- 必选参数，主键。Upsert Sink 需要包含主键。
);
```

WITH 参数

参数值	必填	默认值	描述
connector	是	无	连接 Kudu 数据库时，需要填写 `'kudu'`
kudu.masters	是	无	Kudu 数据库 MasterServer 的连接地址。端口默认为7051。若使用腾讯云的 Kudu 组件，master 地址和端口可以在 弹性 MapReduce 控制台 的集群列表，单击集群 ID/名称，进入集群详情页，然后在**集群服务 > Kudu > 操作 > 查看端口**中找到对应的 master server IP 和端口
kudu.table	是	无	数据库表名。例如 Impala 创建的 kudu 内表一般为 `impala::db_name.table_name`，Java API 创建的 Kudu 表 `db_name.tablename`
kudu.hash-columns	否	无	Hash 键
kudu.primary-key-columns	否	无	主键
kudu.replicas	否	无	副本数量
kudu.operation-timeout	否	30000	插入超时时间，单位为毫秒
kudu.max-buffer-size	否	1000	默认为1000
kudu.flush-interval	否	1000	默认为1000
kudu.ignore-not-found	否	false	是否忽略未找到的数据
kudu.ignore-duplicate	否	false	如果下游表有主键约束，需要设置为 true，此时新增数据通过 upsert 模式写入

类型映射

Flink 类型	Kudu
STRING	STRING
BOOLEAN	BOOL
TINYINT	INT8
SMALLINT	INT16
INT	INT32
BIGINT	INT64

FLOAT	FLOAT
DOUBLE	DOUBLE
BYTES	BINARY
TIMESTAMP(3)	UNIXTIME_MICROS

代码示例

```
CREATE TABLE `kudu_source_table` (
    `id` INT,
    `name` STRING
) WITH (
    -- 指定Kudu连接参数
    'connector' = 'kudu',
    'kudu.masters' = 'master-01:7051,master-02:7051,master-03:7051', -- 连接地址
    'kudu.table' = 'TableName1', -- 替换为 Kudu 中对应的表，如 default.TestTable1
    'kudu.hash-columns' = 'id', -- 可选参数，Hash 键
    'kudu.primary-key-columns' = 'id', -- 可选参数，主键
    'kudu.operation-timeout' = '10000', -- 可选参数，插入超时时间
    'kudu.max-buffer-size' = '2000', -- 可选参数，buffer 大小
    'kudu.flush-interval' = '1000' -- 可选参数，刷新数据到 kudu 的时间间隔
);

CREATE TABLE `kudu_upsert_sink_table` (
    `id` INT,
    `name` STRING
) WITH (
    -- 指定 Kudu 连接参数
    'connector' = 'kudu',
    'kudu.masters' = 'master-01:7051,master-02:7051,master-03:7051', -- 连接地址
    'kudu.table' = 'TableName1', -- 替换为 Kudu 中对应的表，如default.TestTable1
    'kudu.hash-columns' = 'id', -- 可选参数，Hash 键
    'kudu.primary-key-columns' = 'id' -- 必选参数，主键。Upsert Sink 需要包含主键。
);

insert into kudu_upsert_sink_table select * from kudu_source_table;
```

注意事项

1. 若需要使用 Impala 查询 Kudu 数据库的表时，需确认是否已经创建了对应的外表。
2. 非 Impala-shell 创建的表，默认在 Impala 中没有对应的外表，需创建对应的 Kudu 外表才能查到记录。
3. Kudu 作为 Oceanus 的 Sink 端时，若 Kudu 中该表不存在，则会在 Kudu 中创建对应的内表。

Kudu Kerberos 认证授权

1. 登录集群 Master 节点，获取 krb5.conf、emr.keytab文件，路径如下。

```
/etc/krb5.conf
/var/krb5kdc/emr.keytab
```

2. 对步骤1中获取的文件打包成 jar 包。

```
jar cvf kudu-xxx.jar krb5.conf emr.keytab
```

3. 校验 jar 的结构（可以通过 vim 命令查看 vim kudu-xxx.jar），jar 里面包含如下信息，请确保文件不缺失且结构正确。

```
META-INF/
META-INF/MANIFEST.MF
emr.keytab
krb5.conf
```

4. 在 程序包管理 页面上传 jar 包，并在作业参数配置里引用该程序包。
5. 获取 kerberos principal，用于作业 高级参数 配置。

```
klist -kt /var/krb5kdc/emr.keytab

# 输出如下所示，选取第一个即可：hadoop/172.28.22.43@EMR-E4331BF2
KVNO Timestamp           Principal
---- ------------------- ------------------------------------------------------
   2 07/06/2023 18:50:41 hadoop/172.28.22.43@EMR-E4331BF2
   2 07/06/2023 18:50:41 HTTP/172.28.22.43@EMR-E4331BF2
   2 07/06/2023 18:50:41 kudu/172.28.22.43@EMR-E4331BF2
```

6. 作业 高级参数 配置。

```
containerized.taskmanager.env.HADOOP_USER_NAME: hadoop
containerized.master.env.HADOOP_USER_NAME: hadoop
security.kerberos.login.principal: hadoop/172.28.22.43@EMR-E4331BF2
security.kerberos.login.keytab: emr.keytab
security.kerberos.login.conf: krb5.conf
fs.hdfs.hadoop.security.authentication: kerberos
```

> **注意：**
> 历史 Oceanus 集群可能不支持该功能，您可通过 在线客服 联系我们升级集群管控服务，以支持 Kerberos 访问。

Legacy Sink（历史版本）

由于 Flink SQL Connector 语法曾经作过变更（https://issues.apache.org/jira/browse/FLINK-16987），Kudu Sink 目前也包括历史版本的 Sink，通过 'connector.type'='kudu' 进行指定。

```
CREATE TABLE `kudu_upsert_sink_table` (
    `id` INT,
    `name` STRING
) WITH (
    -- 指定 Kudu 连接参数
    'connector.type' = 'kudu',
    'kudu.masters' = 'master-01:7051,master-02:7051,master-03:7051', -- 连接地址
    'kudu.table' = 'TableName1', -- 替换为 Kudu 中对应的表，如default.TestTable1
    'kudu.hash-columns' = 'id', -- 可选参数，Hash 键
    'kudu.primary-key-columns' = 'id' -- 必选参数，主键。Upsert Sink 需要包含主键。
);
```

WITH 参数

参数值	必填	默认值	描述
connector.type	是	无	连接 Kudu 数据库时，需要填写 `'kudu'`
kudu.masters	是	无	Kudu 数据库 MasterServer 的连接地址。端口默认为7051。若使用腾讯云的 Kudu 组件，master 地址和端口可以在 弹性 MapReduce 控制台 的集群列表，单击集群 ID/名称，进入集

			群详情页，然后在**集群服务 > Kudu > 操作 > 查看端口**中找到对应的 master server IP 和端口
kudu.table	是	无	数据库表名。例如 Impala 创建的 kudu 内表一般为 `impala::db_name.table_name`，Java API 创建的 Kudu 表 `db_name.tablename`
kudu.hash-columns	否	无	Hash 键
kudu.primary-key-columns	否	无	主键
kudu.replicas	否	无	副本数量
kudu.operation-timeout	否	30000	插入超时时间，单位为毫秒
kudu.max-buffer-size	否	1000	默认为1000
kudu.flush-interval	否	1000	默认为1000
kudu.ignore-not-found	否	false	是否忽略未找到的数据
kudu.ignore-duplicate	**否**	**false**	**当配置为false时，将通过Upsert模式写入 当配置为true是，将通过Insert模式写入并忽略写入冲突，最终效果将对每一个主键保留每一次写入的记录**

数据库 Redis

最近更新时间：2025-02-17 16:25:52

介绍

Redis Connector 提供了对 Redis 写入和维表的支持。

版本说明

Flink 版本	说明
1.11	支持（写入，维表）
1.13	支持（写入，维表）
1.14	支持（写入、维表）
1.16	支持（写入、维表）

使用范围

- 可以作为维表的使用。
- 可以作为 Tuple、Upsert 数据流的目的表。

DDL 定义

set 命令（字符串键）

```
-- 第1列为 key，第2列为 value。Redis 命令为 set key value
CREATE TABLE `redis_set_sink_table` (
 `key` STRING,
 `value` STRING
) WITH (
  'connector' = 'redis',      -- 输出到 Redis
  'command' = 'set',          -- set 命令
  'nodes' = '<host>:<port>', -- redis 连接地址
  'password' = '<password>',
  'database' = '<database>'
);
```

lpush 命令（列表键）

```
-- 第1列为 key，第2列为 value。Redis 命令为 lpush key value
CREATE TABLE `redis_lpush_sink_table` (
 `key` STRING,
 `value` STRING
) WITH (
  'connector' = 'redis',      -- 输出到 Redis
  'command' = 'lpush',        -- lpush 命令
  'nodes' = '<host>:<port>', -- redis 连接地址
  'password' = '<password>',
  'database' = '<database>'
);
```

sadd 命令（集合键）

```
-- 第1列为 key，第2列为 value。Redis 命令为 sadd key value
CREATE TABLE `redis_sadd_sink_table` (
 `key`   STRING,
 `value` STRING
) WITH (
  'connector' = 'redis',      -- 输出到 Redis
  'command' = 'sadd',         -- sadd 命令
  'nodes' = '<host>:<port>', -- redis 连接地址
  'password' = '<password>',
  'database' = '<database>'
);
```

hset 命令（哈希键）

```
-- 第1列为 hash_key，第2列为 hash_value。Redis 命令为 hset key hash_key hash_value。
CREATE TABLE `redis_hset_sink_table` (
 `hash_key`   STRING,
 `hash_value` STRING
) WITH (
  'connector' = 'redis',      -- 输出到 Redis
  'command' = 'hset',         -- hset 命令
  'nodes' = '<host>:<port>', -- redis 连接地址
  'password' = '<password>',
  'database' = '<database>',
  'additional-key' = '<key>' -- 哈希键
);
```

HSET_WITH_KEY 命令（哈希键）

```
-- 第1列为 key, 第2列为 hash_key，第3列为 hash_value。Redis 命令为 hset key hash_key hash_value。
CREATE TABLE `redis_hset_sink_table` (
`key`        STRING,
`hash_key`   STRING,
`hash_value` STRING
) WITH (
'connector' = 'redis',      -- 输出到 Redis
'command' = 'hset_with_key',         -- hset 命令
'nodes' = '<host>:<port>', -- redis 连接地址
'password' = '<password>',
'database' = '<database>'
);
```

hmset 命令

```
-- 第1列为key，第2列为hash_key，第3列为hash_key对应的hash_value。Redis插入数据的命令为hmset key hash_key
hash_value
CREATE TABLE `redis_hmset_sink_table` (
  `key`        STRING,
  `fieldKey`   STRING,
  `fieldValue` STRING
) WITH (
  'connector' = 'redis',      -- 输出到 Redis
  'command' = 'hmset',        -- hmset 命令
```

```
  'nodes' = '<host>:<port>', -- redis 连接地址
  'password' = '<password>'
);
```

zadd 命令（有序集合键）

```
-- 第1列为 hash_key，第2列为 hash_value。zadd key score value。
CREATE TABLE `redis_zadd_sink_table` (
 `value` STRING,
 `score` DOUBLE
) WITH (
  'connector' = 'redis',      -- 输出到 Redis
  'command' = 'zadd',         -- zadd 命令
  'nodes' = '<host>:<port>', -- redis 连接地址
  'password' = '<password>',
  'database' = '<database>',
  'additional-key' = '<key>' -- 有序集合键
);
```

ZADD_WITH_KEY 命令（有序集合键）

```
  -- 第1列为 key, 第2列为 value，第3列为 score。zadd key score value。
CREATE TABLE `redis_zadd_sink_table` (
`key` STRING,
`value` STRING,
`score` DOUBLE
) WITH (
'connector' = 'redis',      -- 输出到 Redis
'command' = 'zadd_with_key',         -- zadd 命令
'nodes' = '<host>:<port>', -- redis 连接地址
'password' = '<password>',
'database' = '<database>',
);
```

get 命令（维表）

```
CREATE TABLE `redis_dimension_table` (
 `key`   STRING,
 `value` STRING
) WITH (
  'connector' = 'redis',
  'command' = 'get',               -- get命令
  'nodes' = '<host>:<port>',      -- redis连接地址,集群模式多个节点使用'','分隔。
  'lookup.cache.max-rows' = '500', -- 查询缓存中最多缓存的数据条数。
  'lookup.cache.ttl' = '10 min', -- 每条记录的最长缓存时间。
  'lookup.max-retries' = '5', -- 查询失败，最多重试次数。
  -- 'password' = '<password>',    -- 可选参数，密码
  -- 'database' = '<database>',    -- 可选参数，数据库：默认0
  -- 'redis-mode' = 'standalone' -- 可选参数，redis 部署模式，默认standalone。(可选：standalone 单机模式，
cluster 集群模式)
);
```

WITH 参数

参数值	必填	默认值	描述
connector	是	-	固定值为 redis
command	是	-	操作命令。取值与对应的键类型如下： • set：字符串键 • lpush：类别键 • sadd：集合键 • hset：哈希键 • hset_with_key：哈希键 • zadd：有序集合键 • zadd_with_key：有序集合键
nodes	是	-	redis server 连接地址，示例：127.0.0.1:6379。集群架构下多个节点使用','分隔
password	否	空	redis 密码，默认值为空，不进行权限验证
database	否	0	要操作的数据库的 DB number，默认值0
redis-mode	否	standalone	redis 部署模式 • standalone：标准架构，单机 • cluster：集群架构，分布式
ignore-delete	否	false	是否忽略 Retraction 消息
additional-ttl	否	-	过期时间，单位：秒。示例：60，设置为60秒。
additional-key	-	-	用于指定 hset 和 zadd 的 key。执行 hset 和 zadd 命令时必须设置
lookup.cache.max-rows	否	-	查询缓存中最多缓存的数据条数
lookup.cache.ttl	否	10s	每条记录的最长缓存时间
lookup.max-retries	否	1	查询失败最多重试次数
lookup.cache.caching-missing-key	否	true	是否缓存空结果

代码示例

```
CREATE TABLE datagen_source_table (
    id INT,
    name STRING
) WITH (
   'connector' = 'datagen',
   'rows-per-second'='1'  -- 每秒产生的数据条数
);

CREATE TABLE `redis_set_sink_table` (
 `key` STRING,
 `value` STRING
) WITH (
  'connector' = 'redis',      -- 输出到 Redis
  'command' = 'set',          -- set 命令
  'nodes' = '127.0.0.1:8121', -- redis 连接地址
  'password' = '<password>',
  'database' = '0'
);
```

```
insert into redis_set_sink_table select cast(id as string), name from datagen_source_table;
```

注意事项

1. 自建 Redis 的集群模式不支持多数据库，集群模式下 database 参数无效。
2. 腾讯云数据库 Redis 的集群架构支持多数据库，可以使用 standalone 模式指定写入云数据库 Redis 集群架构中的非0数据库，例如：

```
CREATE TABLE `redis_set_sink_table` (
 `key` STRING,
 `value` STRING
) WITH (
  'connector' = 'redis',      -- 输出到Redis
  'command' = 'set',          -- set命令
  'nodes' = '<host>:<port>', -- redis连接地址
  'password' = '<password>',
  'redis-mode' = 'standalone',
  'database' = '1'
);
```

数据库 MongoDB

最近更新时间：2025-04-15 14:16:12

介绍

Flink connector mongodb 目前支持通过 Flink 将数据批量写入到 mongodb 中，目前支持 append 流和 upsert 流。

版本说明

Flink 版本	说明
1.11	不支持
1.13	支持
1.14	不支持
1.16	支持

DDL定义

```
CREATE TABLE mongodb (
  user_id STRING,
  item_id INT,
  category_id INT,
  behavior VARCHAR
) WITH (
  'connector' = 'mongodb', -- 固定值 'mongodb'
  'database' = 'test', --数据库名
  'collection' = 'table1',--数据集合
  'uri' = 'mongodb://$username:$password@$IP:$PORT,$IP:$PORT,$IP:$PORT/test?authSource=admin', --
MongoDB连接串
  'sink.buffer-flush.max-rows' = '1024', -- 每次批量写入的条数
  'sink.buffer-flush.interval' = '1s' -- 每次批量写入的间隔
);
```

使用范围

Flink connector mongodb 目前仅支持 mongodb sink。支持将腾讯云数据库 MongoDB 作为结果表使用。

WITH参数

参数	说明	是否必填	备注
connector	结果表类型	是	固定值 `mongodb`
database	数据库名称	是	-
collection	数据集合	是	-
uri	MongoDB 连接串	是	-
sink.buffer-flush.max-rows	每次批量写入的条数	否	默认1000
maxConnectionIdleTime	连接超时时长	否	默认值为60000，单位为毫秒
sink.buffer-flush.interval	每次批量写入的间隔	否	默认值为1，单位为秒

版权所有：腾讯云计算（北京）有限责任公司

sink.max-retries	写入失败的最大重试次数	否	默认为3
sink.retry.interval	写入失败的重试间隔	否	默认值为1000，单位为ms

代码示例

```
CREATE TABLE random_source (
  user_id STRING,
  item_id INT,
  category_id INT,
  behavior VARCHAR
) WITH (
  'connector' = 'datagen',
  'rows-per-second' = '100',  -- 每秒产生的数据条数
  'fields.user_id.kind' = 'sequence',  -- 有界序列（结束后自动停止输出）
  'fields.user_id.start' = '1',  -- 序列的起始值
  'fields.user_id.end' = '10000',  -- 序列的终止值
  'fields.item_id.kind' = 'random',  -- 无界的随机数
  'fields.item_id.min' = '1',  -- 随机数的最小值
  'fields.item_id.max' = '1000',  -- 随机数的最大值
  'fields.category_id.kind' = 'random',  -- 无界的随机数
  'fields.category_id.min' = '1',  -- 随机数的最小值
  'fields.category_id.max' = '1000',  -- 随机数的最大值
  'fields.behavior.length' = '5' -- 随机字符串的长度
);

CREATE TABLE mongodb (
  user_id STRING,
  item_id INT,
  category_id INT,
  behavior VARCHAR
) WITH (
  'connector' = 'mongodb', -- 固定值 'mongodb'
  'database' = 'test', --数据库名
  'collection' = 'table1',--数据集合
  'uri' = 'mongodb://$username:$password@$IP:$PORT,$IP:$PORT,$IP:$PORT/test?authSource=admin', --
MongoDB连接串
  'sink.buffer-flush.max-rows' = '1024', -- 每次批量写入的条数
  'sink.buffer-flush.interval' = '1s' -- 每次批量写入的间隔
);

insert into mongodb select * from random_source;
```

Upsert

MongoDB sink 支持 upsert，但是需要在创建 table 时指定 primary key。分为以下两种情况：

1. 只指定 _id 作为 key。

```
CREATE TABLE mongodb (
  _id STRING,
  item_id INT,
  category_id INT,
  behavior VARCHAR,
  PRIMARY KEY (`_id`) NOT ENFORCED
) WITH (
  ...
```

```
);
```

2. 指定除 _id 以外的其他自定义字段作为 key。**注意：**此时会将自定义的字段转成 _id。

   举例：字段 a 和字段 b 共同作为主键，那么 {a = 1, b = '2'，c=3} 同步到 mongodb 端会变成{_id : {a:1, b:'2'}, a: 1, b: '2', c: 3}。

```
CREATE TABLE mongodb (
  user_id STRING,
  item_id INT,
  category_id INT,
  behavior VARCHAR,
  PRIMARY KEY (`user_id`, `item_id`) NOT ENFORCED
) WITH (
  ...
);
```

注意事项

用户权限

MongoDB 的 User 必须拥有 database 的写权限。

数据库 HBase

最近更新时间：2025-02-27 15:48:35

介绍

HBase Connector 提供了对 HBase 集群的读写支持。Oceanus 已经提供了内置的 `flink-connector-hbase` Connector 组件，具体使用可参考 使用 MySQL 关联 HBase 维表数据到 ClickHouse 进一步了解。

版本说明

Flink 版本	说明
1.11	支持 Hbase 版本为：1.4.x
1.13	支持 Hbase 版本为：1.4.x、2.2.x、2.3.x
1.14	支持 Hbase 版本为：1.4.x、2.2.x
1.16	支持 Hbase 版本为：1.4.x、2.2.x
1.18	支持 Hbase 版本为：1.4.x、2.2.x

适用范围

可以作为源表，维表，以及Tuple、Upsert 数据流的目的表。

DDL 定义

```
CREATE TABLE hbase_table (
  rowkey INT,
  cf ROW < school_name STRING >,
  PRIMARY KEY (rowkey) NOT ENFORCED
) WITH (
  'connector' = 'hbase-1.4',                          -- Flink 1.13 支持 hbase-2.2
  'table-name' = 'hbase_sink_table',                          -- Hbase 表名
  'zookeeper.quorum' = 'ip:port,ip:port,ip:port'    -- Hbase 的 zookeeper 地址
);
```

WITH 参数

参数	说明	是否必填	备注
connector	表类型	是	`hbase-1.4` 或者 `hbase-2.2` 如果您用了 hbase 2.3.x 版本，那么，connector 参数值需要替换为 hbase-2.2
table-name	HBase 表名	是	-
zookeeper.quorum	HBase 的 zookeeper 地址	是	查看 hbase-site.xml 确定参数值
zookeeper.znode.parent	HBase 在 zookeeper 中的根目录	否	查看 hbase-site.xml 确定参数值
null-string-literal	HBase 字段类型为字符串时，如果 Flink 字段数据为 null，则将该字段赋值为 null-string-literal，并写入 HBase	否	默认为 null
sink.buffer-flush.max-size	写入 HBase 前，内存中缓存的数据量（字节）大小。调大该值有利于提高 HBase 写入性能，但会增加写入延迟	否	默认值为2MB，支持字节单位 B、KB、MB 和 GB，不区分大小写。设置为0表示

	和内存使用。**仅作为 Sink 时使用**		不进行缓存
sink.buffer-flush.max-rows	写入 HBase 前，内存中缓存的数据条数。调大该值有利于提高 HBase 写入性能，但会增加写入延迟和内存使用。**仅作为 Sink 时使用**	否	默认值为1000，设置为0表示不进行缓存
sink.buffer-flush.interval	将缓存数据周期性写入到 HBase 的间隔，可以控制写入 HBase 的延迟。**仅作为 Sink 时使用**。	否	默认值为1秒，支持时间单位 ms、s、min、h 和 d。设置为0表示关闭定期写入

类型映射

HBase 将所有的数据存为字节数组。读写操作时需要将数据进行序列化和反序列化。Flink 与 HBase 的数据转换关系如下：

Flink 字段类型	HBase 转换
CHAR / VARCHAR / STRING	byte[] toBytes(String s) String toString(byte[] b)
BOOLEAN	byte[] toBytes(boolean b)boolean toBoolean(byte[] b)
BINARY / VARBINARY	byte[]
DECIMAL	byte[] toBytes(BigDecimal v)BigDecimal toBigDecimal(byte[] b)
TINYINT	new byte[] { val } bytes[0]
SMALLINT	byte[] toBytes(short val)short toShort(byte[] bytes)
INT	byte[] toBytes(int val)int toInt(byte[] bytes)
BIGINT	byte[] toBytes(long val)long toLong(byte[] bytes)
FLOAT	byte[] toBytes(float val)float toFloat(byte[] bytes)
DOUBLE	byte[] toBytes(double val)double toDouble(byte[] bytes)
DATE	将日期转换成自1970.01.01以来的天数，用 int 表示，并通过 byte[] toBytes(int val) 转换成字节数组
TIME	将时间转换成自00:00:00以来的毫秒数，用 int 表示，并通过 byte[] toBytes(int val) 转换成字节数组
TIMESTAMP	将时间戳转换成自1970-01-01 00:00:00以来的毫秒数，用 long 表示，并通过 byte[] toBytes(long val) 转换成字节数组
ARRAY	不支持
MAP / MULTISET	不支持
ROW	不支持

代码示例

包含 HBase 维表的实时计算作业代码，示例如下：

```
CREATE TABLE datagen_source_table (
  id INT,
  name STRING,
  `proc_time` AS PROCTIME()
) with (
  'connector'='datagen',
  'rows-per-second'='1'
);

CREATE TABLE hbase_table (
  rowkey INT,
```

```
  cf ROW < school_name STRING >,
  PRIMARY KEY (rowkey) NOT ENFORCED
) WITH (
  'connector' = 'hbase-1.4',                          -- Flink 1.13 支持 hbase-2.2
  'table-name' = 'hbase_sink_table',                        -- Hbase 表名
  'zookeeper.quorum' = 'ip:port,ip:port,ip:port'    -- Hbase 的 zookeeper 地址
);

CREATE TABLE blackhole_sink(
  id INT,
  name STRING
) with (
  'connector' = 'blackhole'
);

INSERT INTO blackhole_sink
     SELECT id, cf.school_name as name FROM datagen_source_table src
JOIN hbase_table FOR SYSTEM_TIME AS OF src.`proc_time` as h ON src.id = h.rowkey;
```

注意事项

HBase Connector 一般会使用 DDL 语句中定义的主键，以 `upsert` 模式工作，与外部系统交换变更日志信息。因此，必须在 HBase 的 rowkey 字段上定义主键（必须声明 rowkey 字段）。如果未声明 PRIMARY KEY 子句，则 HBase 连接器默认将 rowkey 作为主键。

Kerberos 认证授权

1. 登录集群 Master 节点，获取 krb5.conf、emr.keytab、core-site.xml、hdfs-site.xml、hbase-site.xml 文件，路径如下。

```
/etc/krb5.conf
/var/krb5kdc/emr.keytab
/usr/local/service/hadoop/etc/hadoop/core-site.xml
/usr/local/service/hadoop/etc/hadoop/hdfs-site.xml
/usr/local/service/hbase/conf/hbase-site.xml
```

2. 对获取的配置文件构建 jar 包。

```
jar cvf hbase-xxx.jar krb5.conf emr.keytab core-site.xml hdfs-site.xml hbase-site.xml
```

3. 校验 jar 的结构（可以通过 vim 命令查看），jar 里面包含如下信息，请确保文件不缺失且结构正确。

```
META-INF/
META-INF/MANIFEST.MF
krb5.conf
emr.keytab
core-site.xml
hdfs-site.xml
hbase-site.xml
```

4. 在 程序包管理 页面上传 jar 包，并在作业参数配置里引用该程序包。
5. 获取 kerberos principal，用于 作业高级参数 配置。

```
klist -kt /var/krb5kdc/emr.keytab

# 输出如下所示，选取第一个即可：hadoop/172.28.28.51@EMR-OQPO48B9
KVNO Timestamp         Principal
---- ------------------- ------------------------------------------------------
   2 08/09/2021 15:34:40 hadoop/172.28.28.51@EMR-OQPO48B9
```

```
  2 08/09/2021 15:34:40 HTTP/172.28.28.51@EMR-OQPO48B9
  2 08/09/2021 15:34:40 hadoop/VM-28-51-centos@EMR-OQPO48B9
  2 08/09/2021 15:34:40 HTTP/VM-28-51-centos@EMR-OQPO48B9
```

6. **作业高级参数** 配置。

```
containerized.taskmanager.env.HADOOP_USER_NAME: hadoop
containerized.master.env.HADOOP_USER_NAME: hadoop
security.kerberos.login.principal: hadoop/172.28.28.51@EMR-OQPO48B9
security.kerberos.login.keytab: emr.keytab
security.kerberos.login.conf: krb5.conf
```

> **注意：**
> 历史 Oceanus 集群可能不支持该功能，您可以联系我们升级集群管控服务，以支持 Kerberos 访问。

数据库 Oceanbase

最近更新时间：2025-02-25 16:23:42

介绍

Oceanbase Connector 提供了对 oceanbase 数据库读写支持。
目前 Oceanus 提供的 `flink-connector-oceanbase-jdbc` Connector 组件支持oceanbase的 `mysql` 模式和 `oracle` 模式。

版本说明

Flink 版本	说明
1.11	不支持
1.13	支持
1.14	支持
1.16	支持

使用范围

jdbc-oceanbase 支持用作数据源表（Source），用于按固定列扫描表和用于 JOIN 的右表（维表）；也支持用作数据目的表（sink），用于 Tuple 数据流表和用于 Upsert 数据流表（需要指定主键）。

DDL 定义

用作按列扫描的数据源（Source）

```
CREATE TABLE `jdbc_source_table` (
    `id` INT,
    `name` STRING
) WITH (
    -- 指定数据库连接参数
    'connector' = 'jdbc-oceabase',
    'compatible-mode'= 'mysql',
    'url' = 'jdbc:oceanbase://10.1.28.93:3306/CDB?
rewriteBatchedStatements=true&serverTimezone=Asia/Shanghai', -- 请替换为您的实际 MySQL 连接参数
    'table-name' = 'my-table', -- 需要写入的数据表
    'username' = 'admin',      -- 数据库访问的用户名（需要提供 INSERT 权限）
    'password' = 'MyPa$$w0rd',  -- 数据库访问的密码
    'scan.partition.column' = 'id',   -- 分区扫描列名
    'scan.partition.num' = '2',       -- 分区数量
    'scan.partition.lower-bound' = '0', -- 首个分区的最小值
    'scan.partition.upper-bound' = '100', -- 最后一个分区的最大值
    'scan.fetch-size' = '1' -- 每次从数据库读取时，批量获取的行数
);
```

用作维表数据源（Source）

```
CREATE TABLE `jdbc_dim_table` (
    `id` INT,
    `name` STRING
) WITH (
    -- 指定数据库连接参数
    'connector' = 'jdbc-oceanbase',
    'compatible-mode'= 'mysql',
```

```
    'url' = 'jdbc:oceanbase://10.1.28.93:3306/CDB?
rewriteBatchedStatements=true&serverTimezone=Asia/Shanghai', -- 请替换为您的实际 MySQL 连接参数
    'table-name' = 'my-table', -- 需要写入的数据表
    'username' = 'admin',      -- 数据库访问的用户名（需要提供 INSERT 权限）
    'password' = 'MyPa$$w0rd',  -- 数据库访问的密码
    'lookup.cache.max-rows' = '100',   -- 读缓存大小
    'lookup.cache.ttl' = '5000'       -- 读缓存的 TTL
);
```

用作数据目的（Tuple Sink）

```
CREATE TABLE `jdbc_sink_table` (
    `id` INT,
    `name` STRING
) WITH (
    -- 指定数据库连接参数
    'connector' = 'jdbc-oceanbase',
    'compatible-mode'= 'mysql',
    'url' = 'jdbc:oceanbase://10.1.28.93:3306/CDB?
rewriteBatchedStatements=true&serverTimezone=Asia/Shanghai', -- 请替换为您的实际 MySQL 连接参数
    'table-name' = 'my-table',  -- 需要写入的数据表
    'username' = 'admin',       -- 数据库访问的用户名（需要提供 INSERT 权限）
    'password' = 'MyPa$$w0rd',  -- 数据库访问的密码
    'sink.buffer-flush.max-rows' = '200',  -- 批量输出的条数
    'sink.buffer-flush.interval' = '2s'    -- 批量输出的间隔
);
```

用作数据目的（Upsert Sink）

```
CREATE TABLE `jdbc_upsert_sink_table` (
    `id` INT PRIMARY KEY NOT ENFORCED,
    `name` STRING
) WITH (
    -- 指定数据库连接参数
    'connector' = 'jdbc-oceanbase',
    'compatible-mode'= 'mysql',
    'url' = 'jdbc:oceanbase://10.1.28.93:3306/CDB?
rewriteBatchedStatements=true&serverTimezone=Asia/Shanghai', -- 请替换为您的实际 MySQL 连接参数
    'table-name' = 'my-upsert-table', -- 需要写入的数据表
    'username' = 'admin',             -- 数据库访问的用户名（需要提供 INSERT 权限）
    'password' = 'MyPa$$w0rd',        -- 数据库访问的密码
    'sink.buffer-flush.max-rows' = '200',  -- 批量输出的条数
    'sink.buffer-flush.interval' = '2s'    -- 批量输出的间隔
);
```

> **注意：**
> Upsert 需要定义 PRIMARY KEY。

WITH 参数

参数值	必填	默认值	描述
connector	是	无	连接数据库时，需要填写 `'jdbc-oceanbase'`。
compatible-mode	是	无	连接数据时，根据实际情况填写，`mysql` 或者 `oracle`

url	是	无	JDBC 数据库的连接 URL。
table-name	是	无	数据库表名。
driver	否	无	JDBC Driver 的类名。如果不输入，则自动使用各个模式默认的驱动类名。
username	否	无	数据库用户名。`'username'` 和 `'password'` 必须同时使用。
password	否	无	数据库密码。
scan.partition.column	否	无	指定对输入分区扫描（Partitioned Scan）的列名，该列必须是数值类型、日期类型、时间戳类型等。
scan.partition.num	否	无	分区扫描启用后，指定分区数。
scan.partition.lower-bound	否	无	分区扫描启用后，指定首个分区的最小值。
scan.partition.upper-bound	否	无	分区扫描启用后，指定最后一个分区的最大值。
scan.fetch-size	否	0	每次循环读取时应该从数据库中获取的行数。如果指定的值为 '0'，则该配置项会被忽略。
lookup.cache.max-rows	否	无	lookup cache 的最大行数，若超过该值，则最老的行记录将会过期。 默认情况下，lookup cache 是未开启的。
lookup.cache.ttl	否	无	lookup cache 中每一行记录的最大存活时间，若超过该时间，则最老的行记录将会过期。 默认情况下，lookup cache 是未开启的。
lookup.max-retries	否	3	数据库查询失败时，最多重试的次数。
sink.buffer-flush.max-rows	否	100	flush 前缓存记录的最大值，可以设置为 '0' 来禁用它。
sink.buffer-flush.interval	否	1s	批量输出时，每批次最大的间隔（毫秒）。**如果 `'sink.buffer-flush.max-rows'` 设为 `'0'`，而这个选项不为零，则说明启用纯异步输出功能，即数据输出到算子、从算子最终写入数据库这两部分线程完全解耦。**
sink.max-retries	否	3	数据库写入失败时，最多重试的次数。

代码示例

```
CREATE TABLE `oceanbase_source_table` (
    `id` INT,
    `name` STRING
) WITH (
    -- 指定数据库连接参数
    'connector' = 'jdbc-oceanbase',
    'url' = 'jdbc:oceanbase://10.1.28.93:3306/CDB?
rewriteBatchedStatements=true&serverTimezone=Asia/Shanghai', -- 请替换为您的实际 MySQL 连接参数
    'table-name' = 'my-table', -- 需要写入的数据表
    'username' = 'admin',      -- 数据库访问的用户名（需要提供 INSERT 权限）
    'password' = 'MyPa$$w0rd',  -- 数据库访问的密码
    'scan.partition.column' = 'id',    -- 分区扫描列名
    'scan.partition.num' = '2',        -- 分区数量
    'scan.partition.lower-bound' = '0', -- 首个分区的最小值
    'scan.partition.upper-bound' = '100', -- 最后一个分区的最大值
    'scan.fetch-size' = '1' -- 每次从数据库读取时，批量获取的行数
);
CREATE TABLE `oceanbase_upsert_sink_table` (
    `id` INT PRIMARY KEY NOT ENFORCED,
    `name` STRING
```

```
) WITH (
    -- 指定数据库连接参数
    'connector' = 'jdbc-oceanbase',
    'url' = 'jdbc:oceanbase://10.1.28.93:3306/CDB?
rewriteBatchedStatements=true&serverTimezone=Asia/Shanghai', -- 请替换为您的实际 MySQL 连接参数
    'table-name' = 'my-upsert-table', -- 需要写入的数据表
    'username' = 'admin',             -- 数据库访问的用户名（需要提供 INSERT 权限）
    'password' = 'MyPa$$w0rd',        -- 数据库访问的密码
    'sink.buffer-flush.max-rows' = '200',  -- 批量输出的条数
    'sink.buffer-flush.interval' = '2s'    -- 批量输出的间隔
);
insert into oceanbase_upsert_sink_table select * from oceanbase_source_table;
```

主键说明

- 对于 Append（Tuple）数据，作业 DDL 不需要设置主键，Oceanbase 数据库的表也不需要定义主键，也不建议定义主键（否则可能因为重复数据而造成写入失败）。
- 对于 Upsert 数据，Oceanbase 数据库的表**必须**定义主键，并需要在 DDL 语句的 CREATE TABLE 中，将相应列名也加上 `PRIMARY KEY NOT ENFORCED` 约束。

分区扫描

分区扫描（Partitioned Scan）可以加速多个并行度的 Source 算子读取 Oceanbase 数据表，每个子任务可以读取自己的专属分区。使用该功能时，所有四个 `scan.partition` 开头的参数都必须指定，否则会报错。

> **注意**
> 这里的 `scan.partition.upper-bound` 指定的最大值和 `scan.partition.lower-bound` 指定的最小值，指的是每个分区的步长，不会影响最终读取的数据条数和精确性。

读取缓存

Flink 提供了读取缓存（Lookup Cache）功能，可以提升维表读取的性能。目前该缓存的实现是同步的，默认未启用（每次请求都会读取数据库，吞吐量很低），必须手动设置 `lookup.cache.max-rows` 和 `lookup.cache.ttl` 两个参数来启用该功能。

> **注意**
> 如果缓存的 TTL 太长，或者缓存的条数太多，可能会造成数据库中数据更新后，Flink 作业仍然读取的是缓存中的旧数据。因此对于数据库变动敏感的作业，请谨慎使用缓存功能。

批量写入优化

通过设置 sink.buffer-flush 开头的两个参数，可以实现批量写入数据库。建议配合相应底层数据库的参数，以达到更好的批量写入效果，否则底层仍然会一条一条写入，效率不高。

- 建议在 url 连接参数后加入 rewriteBatchedStatements=true 参数。

```
jdbc:oceanbase://10.1.28.93:3306/CDB?rewriteBatchedStatements=true
```

数据库连接 JDBC

最近更新时间：2023-07-07 16:35:32

介绍

JDBC Connector 提供了对 MySQL、PostgreSQL、Oracle 等常见的数据库读写支持。
目前 Oceanus 提供的 `flink-connector-jdbc` Connector 组件已经内置了 MySQL 和 PostgreSQL 的驱动程序。若需要连接 Oracle 等其他的数据库，可通过附加**自定义程序包**的方式，上传相应的 JDBC Driver 的 JAR 包。

> **注意**
> CDW PostgreSQL 只能使用 PostgreSQL connector。

版本说明

Flink 版本	说明
1.11	支持
1.13	支持
1.14	支持
1.16	支持

使用范围

JDBC 支持用作数据源表（Source），用于按固定列扫描表和用于 JOIN 的右表（维表）；也支持用作数据目的表（sink），用于 Tuple 数据流表和用于 Upsert 数据流表（需要指定主键）。
若需要将 JDBC 数据库的变动记录作为流式源表消费，请使用内置的 CDC 数据源。若内置的 CDC 数据源无法满足需求，还可使用 Debezium 、Canal 等，对 JDBC 数据库的变更进行捕获和订阅，然后 Oceanus 即可对这些变更事件进行进一步的处理。详情可参见 消息队列 Kafka 。

DDL 定义

用作按列扫描的数据源（Source）

```
CREATE TABLE `jdbc_source_table` (
    `id` INT,
    `name` STRING
) WITH (
    -- 指定数据库连接参数
    'connector' = 'jdbc',
    'url' = 'jdbc:mysql://10.1.28.93:3306/CDB?
rewriteBatchedStatements=true&serverTimezone=Asia/Shanghai', -- 请替换为您的实际 MySQL 连接参数
    'table-name' = 'my-table', -- 需要写入的数据表
    'username' = 'admin',      -- 数据库访问的用户名（需要提供 INSERT 权限）
    'password' = 'MyPa$$w0rd',  -- 数据库访问的密码
    'scan.partition.column' = 'id',    -- 分区扫描列名
    'scan.partition.num' = '2',        -- 分区数量
    'scan.partition.lower-bound' = '0', -- 首个分区的最小值
    'scan.partition.upper-bound' = '100', -- 最后一个分区的最大值
    'scan.fetch-size' = '1' -- 每次从数据库读取时，批量获取的行数
);
```

用作维表数据源（Source）

```
CREATE TABLE `jdbc_dim_table` (
    `id` INT,
```

```
    `name` STRING
) WITH (
    -- 指定数据库连接参数
    'connector' = 'jdbc',
    'url' = 'jdbc:mysql://10.1.28.93:3306/CDB?
rewriteBatchedStatements=true&serverTimezone=Asia/Shanghai', -- 请替换为您的实际 MySQL 连接参数
    'table-name' = 'my-table', -- 需要写入的数据表
    'username' = 'admin',      -- 数据库访问的用户名（需要提供 INSERT 权限）
    'password' = 'MyPa$$w0rd',  -- 数据库访问的密码
    'lookup.cache.max-rows' = '100',   -- 读缓存大小
    'lookup.cache.ttl' = '5000'       -- 读缓存的 TTL
);
```

用作数据目的（Tuple Sink）

```
CREATE TABLE `jdbc_sink_table` (
    `id` INT,
    `name` STRING
) WITH (
    -- 指定数据库连接参数
    'connector' = 'jdbc',
    'url' = 'jdbc:mysql://10.1.28.93:3306/CDB?
rewriteBatchedStatements=true&serverTimezone=Asia/Shanghai', -- 请替换为您的实际 MySQL 连接参数
    'table-name' = 'my-table',  -- 需要写入的数据表
    'username' = 'admin',       -- 数据库访问的用户名（需要提供 INSERT 权限）
    'password' = 'MyPa$$w0rd',  -- 数据库访问的密码
    'sink.buffer-flush.max-rows' = '200',  -- 批量输出的条数
    'sink.buffer-flush.interval' = '2s'    -- 批量输出的间隔
);
```

用作数据目的（Upsert Sink）

```
CREATE TABLE `jdbc_upsert_sink_table` (
    `id` INT PRIMARY KEY NOT ENFORCED,
    `name` STRING
) WITH (
    -- 指定数据库连接参数
    'connector' = 'jdbc',
    'url' = 'jdbc:mysql://10.1.28.93:3306/CDB?
rewriteBatchedStatements=true&serverTimezone=Asia/Shanghai', -- 请替换为您的实际 MySQL 连接参数
    'table-name' = 'my-upsert-table', -- 需要写入的数据表
    'username' = 'admin',             -- 数据库访问的用户名（需要提供 INSERT 权限）
    'password' = 'MyPa$$w0rd',        -- 数据库访问的密码
    'sink.buffer-flush.max-rows' = '200',  -- 批量输出的条数
    'sink.buffer-flush.interval' = '2s'    -- 批量输出的间隔
);
```

> **注意：**
> Upsert 需要定义 PRIMARY KEY。

WITH 参数

参数值	必填	默认值	描述
connector	是	无	连接数据库时，需要填写 `'jdbc'`。

url	是	无	JDBC 数据库的连接 URL。
table-name	是	无	数据库表名。
driver	否	无	JDBC Driver 的类名。如果不输入，则自动从 url 中推断。
username	否	无	数据库用户名。`'username'` 和 `'password'` 必须同时使用。
password	否	无	数据库密码。
scan.partition.column	否	无	指定对输入分区扫描（Partitioned Scan）的列名，该列必须是数值类型、日期类型、时间戳类型等。关于分区扫描的细节，可参见 分区扫描 。
scan.partition.num	否	无	分区扫描启用后，指定分区数。
scan.partition.lower-bound	否	无	分区扫描启用后，指定首个分区的最小值。
scan.partition.upper-bound	否	无	分区扫描启用后，指定最后一个分区的最大值。
scan.fetch-size	否	0	每次从数据库读取时，批量获取的行数。默认为0，表示一行一行读取，效率较低（吞吐量不高）。
lookup.cache.max-rows	否	无	查询缓存（Lookup Cache）中最多缓存的数据条数。
lookup.cache.ttl	否	无	查询缓存中每条记录最长的缓存时间。
lookup.max-retries	否	3	数据库查询失败时，最多重试的次数。
sink.buffer-flush.max-rows	否	100	批量输出时，缓存中最多缓存多少数据。如果设置为0，表示禁止输出缓存。
sink.buffer-flush.interval	否	1s	批量输出时，每批次最大的间隔（毫秒）。**如果 `'sink.buffer-flush.max-rows'` 设为 `'0'`，而这个选项不为零，则说明启用纯异步输出功能，即数据输出到算子、从算子最终写入数据库这两部分线程完全解耦。**
sink.max-retries	否	3	数据库写入失败时，最多重试的次数。
sink.ignore-delete	否	false	是否忽略 delete 操作。

代码示例

```
CREATE TABLE `jdbc_source_table` (
    `id` INT,
    `name` STRING
) WITH (
    -- 指定数据库连接参数
    'connector' = 'jdbc',
    'url' = 'jdbc:mysql://10.1.28.93:3306/CDB?
rewriteBatchedStatements=true&serverTimezone=Asia/Shanghai', -- 请替换为您的实际 MySQL 连接参数
    'table-name' = 'my-table', -- 需要写入的数据表
    'username' = 'admin',       -- 数据库访问的用户名（需要提供 INSERT 权限）
    'password' = 'MyPa$$w0rd',  -- 数据库访问的密码
    'scan.partition.column' = 'id',    -- 分区扫描列名
    'scan.partition.num' = '2',        -- 分区数量
    'scan.partition.lower-bound' = '0', -- 首个分区的最小值
    'scan.partition.upper-bound' = '100', -- 最后一个分区的最大值
    'scan.fetch-size' = '1' -- 每次从数据库读取时，批量获取的行数
);
CREATE TABLE `jdbc_upsert_sink_table` (
    `id` INT PRIMARY KEY NOT ENFORCED,
```

```
    `name` STRING
) WITH (
    -- 指定数据库连接参数
    'connector' = 'jdbc',
    'url' = 'jdbc:mysql://10.1.28.93:3306/CDB?
rewriteBatchedStatements=true&serverTimezone=Asia/Shanghai', -- 请替换为您的实际 MySQL 连接参数
    'table-name' = 'my-upsert-table', -- 需要写入的数据表
    'username' = 'admin',             -- 数据库访问的用户名（需要提供 INSERT 权限）
    'password' = 'MyPa$$w0rd',        -- 数据库访问的密码
    'sink.buffer-flush.max-rows' = '200',  -- 批量输出的条数
    'sink.buffer-flush.interval' = '2s'    -- 批量输出的间隔
);
insert into jdbc_upsert_sink_table select * from jdbc_source_table;
```

主键说明

- 对于 Append（Tuple）数据，作业 DDL 不需要设置主键，JDBC 数据库的表也不需要定义主键，也不建议定义主键（否则可能因为重复数据而造成写入失败）。
- 对于 Upsert 数据，JDBC 数据库的表**必须**定义主键，并需要在 DDL 语句的 CREATE TABLE 中，将相应列名也加上 `PRIMARY KEY NOT ENFORCED` 约束。

> **注意**
> - 对于 MySQL 表，Upsert 功能的实现依赖于 `INSERT .. ON DUPLICATE KEY UPDATE ..` 语法，常见版本的 MySQL 均支持该语法。
> - 对于 PostgreSQL 表，Upsert 功能实现依赖于 `INSERT .. ON CONFLICT .. DO UPDATE SET ..` 语法，该语法需要 PostgreSQL 9.5 及以上版本才可支持。
> - 对于 Oracle 表，Upsert 功能实现依赖 `MERGE .. INTO .. USING ON .. WHEN UPDATE .. WHEN INSERT ..` 语法，该语法需要 Oracle 9i 及以上版本才可以支持。

分区扫描

分区扫描（Partitioned Scan）可以加速多个并行度的 Source 算子读取 JDBC 数据表，每个子任务可以读取自己的专属分区。使用该功能时，所有四个 `scan.partition` 开头的参数都必须指定，否则会报错。

> **注意**
> 这里的 `scan.partition.upper-bound` 指定的最大值和 `scan.partition.lower-bound` 指定的最小值，指的是每个分区的步长，不会影响最终读取的数据条数和精确性。

读取缓存

Flink 提供了读取缓存（Lookup Cache）功能，可以提升维表读取的性能。目前该缓存的实现是同步的，默认未启用（每次请求都会读取数据库，吞吐量很低），必须手动设置 `lookup.cache.max-rows` 和 `lookup.cache.ttl` 两个参数来启用该功能。

> **注意**
> 如果缓存的 TTL 太长，或者缓存的条数太多，可能会造成数据库中数据更新后，Flink 作业仍然读取的是缓存中的旧数据。因此对于数据库变动敏感的作业，请谨慎使用缓存功能。

批量写入优化

通过设置 sink.buffer-flush 开头的两个参数，可以实现批量写入数据库。建议配合相应底层数据库的参数，以达到更好的批量写入效果，否则底层仍然会一条一条写入，效率不高。

- 对于 MySQL，建议在 url 连接参数后加入 rewriteBatchedStatements=true 参数。

```
jdbc:mysql://10.1.28.93:3306/CDB?rewriteBatchedStatements=true
```

- 对于 PostgreSQL，建议在 url 连接参数后加入 reWriteBatchedInserts=true 参数。

```
jdbc:postgresql://10.1.28.93:3306/PG?reWriteBatchedInserts=true&currentSchema=数据库的Schema
```

监控指标说明

Oceanus 为 JDBC Connector 也增加了很多实用的统计指标。单击 Flink UI 的运行图中的 JDBC Sink 算子，即可搜索并查看指标：
numberOfInsertRecords：获取输出的 +I 消息数。
numberOfDeleteRecords：获取输出的 −D 消息数。
numberOfUpdateBeforeRecords：获取输出的 −U 消息数。
numberOfUpdateAfterRecords：获取输出的 +U 消息数。

数据分析引擎 Elasticsearch Service

最近更新时间：2023-10-08 17:53:22

介绍

Elasticsearch Connector 提供了对 Elasticsearch 的写入和读取支持。目前 Oceanus 支持 Elasticsearch 6.x 和 7.x 版本。

版本说明

Flink 版本	说明
1.11	支持
1.13	支持（写入、批数据源）
1.14	支持
1.16	支持

使用范围

Elasticsearch 支持写入，可以作为 Tuple 数据流的目的表（Sink），也可以作为 Upsert 数据流的目的表（Sink，自动以文档 `_id` 字段生成主键，并更新之前的文档版本）。
如果希望将 JDBC 数据库的变动记录，将其作为流式源表消费，可以使用 Debezium 、Canal 等，对 JDBC 数据库的变更进行捕获和订阅，然后 Flink 即可对这些变更事件进行进一步的处理。可参见 Kafka 。
Oceanus 支持 Elasticsearch 的批模式读，目前只支持 Elasticsearch 7。

DDL 定义

用作 Elasticsearch 6 数据目的（Sink）

```
CREATE TABLE elasticsearch6_sink_table (
    `id` INT,
    `name` STRING,
    PRIMARY KEY (`id`) NOT ENFORCED -- 对应 Elasticsearch 中的 _id
) WITH (
    'connector' = 'elasticsearch-6', -- 输出到 Elasticsearch 6
    'username' = '$username', -- 选填 用户名
    'password' = '$password', -- 选填 密码
    'hosts' = 'http://10.28.28.94:9200', -- Elasticsearch 的连接地址
    'index' = 'my-index', -- Elasticsearch 的 Index 名
    'document-type' = '_doc', -- Elasticsearch 的 Document 类型
    'format' = 'json' -- 输出数据格式，目前只支持 'json'
);
```

用作 Elasticsearch 7 数据目的（Sink）

```
CREATE TABLE elasticsearch7_sink_table (
    `id` INT,
    `name` STRING,
    PRIMARY KEY (`id`) NOT ENFORCED -- 对应 Elasticsearch 中的 _id
) WITH (
    'connector' = 'elasticsearch-7', -- 输出到 Elasticsearch 7
    'username' = '$username', -- 选填 用户名
    'password' = '$password', -- 选填 密码
    'hosts' = 'http://10.28.28.94:9200', -- Elasticsearch 的连接地址
    'index' = 'my-index', -- Elasticsearch 的 Index 名
    'format' = 'json' -- 输出数据格式，目前只支持 'json'
```

```
);
```

作为 Elasticsearch 7 批数据源（Source）

```
CREATE TABLE elasticsearch7_source_table (
  `id` bigint,
  `event_date` int,
  `app` int,
  primary key (`id`) not enforced
) with (
  -- 必填参数
  'connector' = 'es-source',
  'endPoint' = '127.0.0.1', -- Elasticsearch 的连接 ip
  'accessId' = 'elastic', -- 用户名
  'accessKey' = 'PASSWORD', -- 密码
  'indexName' = 'my-index', -- Elasticsearch 的 Index 名
  'format' = 'json', -- 数据格式，只支持 'json'
  -- 可选参数
  'scheme' = 'http', -- 连接协议
  'port' = '9200', -- 端口
  'batchSize' = '2000', -- 每个 scroll 请求从 Elasticsearch 集群获取的最大文档数
  'keepScrollAliveSecs' = '60' -- scroll 上下文保留的最长时间，单位为分钟
);
```

WITH 参数

作为数据目的

参数值	必填	默认值	描述
connector	是	无	当写入 Elasticsearch 6.x 版本时，取值 `elasticsearch-6` 。当写入 Elasticsearch 7.x 及以上版本时，取值 `elasticsearch-7` 。
username	否	无	用户名。
password	否	无	密码。
hosts	是	无	Elasticsearch 的连接地址。
index	是	无	数据要写入的 Index。支持固定 Index（例如 `'myIndex'` ），也支持动态 Index（例如 `'index-{log_ts|yyyy-MM-dd}'` ）。
document-type	6.x 版本：必填 7.x 版本：不需要	无	Elasticsearch 文档的 Type 信息。当选择 `elasticsearch-7` 时，不能填写这个字段，否则会报错。
document-id.key-delimiter	否	_	为复合键生成 _id 时的分隔符 (默认是 "_")。例如有 a、b、c 三个主键，某条数据的 a 字段为 "1"，b 字段为 "2"，c 字段为 "3"，使用默认分隔符，则最终写入 Elasticsearch 的 _id 是 "1_2_3"。
drop-delete	否	false	是否过滤上游传来的 DELETE（删除）消息。 此外，在多表 LEFT JOIN 且 JOIN Key 非主键的场景下，启用该选项后，可以解决 Elasticsearch 收到较多临时 null 值数据的问题。需要注意的是，JOIN 左右表的字段不能含有 null 值，否则可能会丢失部分数据。
failure-handler	否	fail	指定请求 Elasticsearch 失败时，错误处理策略。选项为： • `fail` ：抛出一个异常。 • `ignore` ：忽略错误，直接继续。 • `retry-rejected` ：重试写入该条记录。

			• 另外也支持自定义错误处理器，这里可以填写用户自己编写的 Handler 的类全名（需要上传自定义程序包）。
sink.flush-on-checkpoint	否	true	Flink 进行快照时，是否等待现有记录完全写入 Elasticsearch 。如果设置为 false，则可能造成恢复时部分数据丢失或者重复等异常情况，但快照速度会提升。
sink.bulk-flush.max-actions	否	1000	批量写入的最大条数。设置为 `0` 则禁用批量功能。
sink.bulk-flush.max-size	否	2mb	批量写入缓存的最大容量，必须以 `mb` 为单位。设置为 `0` 则禁用批量功能。
sink.bulk-flush.interval	否	1s	批量写入的刷新周期。设置为 `0` 则禁用批量功能。
sink.bulk-flush.backoff.strategy	否	DISABLED	批量写入时，失败重试的策略。 • `DISABLED`：不重试。 • `CONSTANT`：等待 `sink.bulk-flush.backoff.delay` 选项设置的毫秒后重试。 • `EXPONENTIAL`：一开始等待 `sink.bulk-flush.backoff.delay` 选项设置的毫秒后重试，每次失败后将指数增加下次的等待时间。
sink.bulk-flush.backoff.max-retries	否	8	批量写入时，最多失败重试的次数。
sink.bulk-flush.backoff.delay	否	50ms	批量写入失败时，每次重试之间的等待间隔（对于 CONSTANT 策略而言）或间隔的初始基数（对于 EXPONENTIAL 策略而言）。
connection.max-retry-timeout	否	无	重试请求的最大超时时间，例如："20 s"。
connection.path-prefix	否	无	指定每个 REST 请求的前缀，例如 `'/v1'` 。通常不需要设置该选项。
format	否	json	指定输出的格式，默认是内置的 `json` 格式，可以使用 前文（Kafka）描述过的 JSON 格式选项，例如 `json.fail-on-missing-field` 、`json.ignore-parse-errors` 、`json.timestamp-format.standard` 等。
retry-on-conflict	否	无	更新操作中，允许因版本冲突异常而重试的最大次数。超过该次数后将抛出异常导致作业失败。 **说明：暂时只支持 Flink-1.13。**

作为数据源

参数值	必填	默认值	描述
connector	是	无	固定值 `es-source`
endPoint	是	无	Elasticsearch 的连接 IP，示例 `127.0.0.1`
accessId	是	无	用户名
accessKey	是	无	密码
indexName	是	无	要读取的 Index
format	是	无	指定读取的格式，只支持内置的 `json` 格式，可以使用 前文（Kafka）描述过的 JSON 格式选项，例如 `json.fail-on-missing-field` 、`json.ignore-parse-errors` 、`json.timestamp-format.standard` 等。

scheme	否	http	Elasticsearch 连接模式，例如 `http` 、 `https`
port	否	9200	Elasticsearch 连接端口
batchSize	否	2000	每个 scroll 请求从 Elasticsearch 集群获取的最大文档数
keepScrollAliveSecs	否	60	scroll上下文保留的最长时间，单位为分钟

代码示例

作为数据目的

```
CREATE TABLE datagen_source_table (
    id INT,
    name STRING
) WITH (
    'connector' = 'datagen',
    'rows-per-second'='1'  -- 每秒产生的数据条数
);

CREATE TABLE elasticsearch7_sink_table (
    `id` INT,
    `name` STRING
) WITH (
    'connector' = 'elasticsearch-7',        -- 输出到 Elasticsearch 7
    'username' = '$username',               -- 选填 用户名
    'password' = '$password',               -- 选填 密码
    'hosts' = 'http://10.28.28.94:9200',    -- Elasticsearch 的连接地址
    'index' = 'my-index',                   -- Elasticsearch 的 Index 名
    'sink.bulk-flush.max-actions' = '1000',    -- 数据刷新频率
    'sink.bulk-flush.interval' = '1s'       -- 数据刷新周期
    'format' = 'json'                       -- 输出数据格式，目前只支持 'json'
);

INSERT INTO elasticsearch7_sink_table select * from datagen_source_table;
```

作为批数据源

```
CREATE TABLE elasticsearch7_source_table (
  `id` bigint,
  `event_date` int,
  `app` int,
  primary key (`id`) not enforced
) with (
  -- 必填参数
  'connector' = 'es-source',
  'endPoint' = '127.0.0.1', -- Elasticsearch 的连接 ip
  'accessId' = 'elastic', -- 用户名
  'accessKey' = 'PASSWORD', -- 密码
  'indexName' = 'my-index', -- Elasticsearch 的 Index 名
  'format' = 'json', -- 数据格式，只支持 'json'
  -- 可选参数
  'scheme' = 'http', -- 连接协议
  'port' = '9200', -- 端口
  'batchSize' = '2000', -- 每个 scroll 请求从 Elasticsearch 集群获取的最大文档数
  'keepScrollAliveSecs' = '60' -- scroll 上下文保留的最长时间，单位为分钟
```

```
);

CREATE TABLE logger_sink (
  `id` bigint,
  `event_date` int,
  `app` int,
  primary key (`id`) not enforced
) with (
  'connector' = 'logger'
);

INSERT INTO logger_sink SELECT * from elasticsearch7_source_table;
```

注意事项

如果您希望连接其他版本的 Elasticsearch，请通过附加**自定义程序包**的方式，上传相应的 Elasticsearch Sink 的 JAR 包。

监控指标说明

Oceanus 为 ES Connector 增加了很多实用的统计指标。单击 Flink UI 的运行图中的 ES Sink 算子，即可搜索并查看指标：
numberOfInsertRecords：获取输出的 +I 消息数。
numberOfDeleteRecords：获取输出的 -D 消息数。
numberOfUpdateBeforeRecords：获取输出的 -U 消息数。
numberOfUpdateAfterRecords：获取输出的 +U 消息数。

模拟上下游 Datagen Logger Blackhole

最近更新时间：2023-08-15 17:47:41

调试 Source 和 Sink 介绍

当需要检验作业是否可以正常运行、逻辑是否正确时，为了减少外部系统的部署开销，以及避免干扰因素，我们可以使用一些调试专用的 Connector。

版本说明

Flink 版本	说明
1.11	支持
1.13	支持
1.14	支持
1.16	支持

Datagen Source

Datagen 是 Flink 自带的随机数据生成器，它可以作为数据源直接引用。详细的使用方式可参考 Flink 官方文档 。
下面是 Datagen 数据源的一个示例，它生成的数据含有两个字段：第一个字段 `id` 是一个随机数，第二个字段 `name` 是一个随机字符串。

DDL 定义

```
CREATE TABLE datagen_source_table (
    id INT,
    name STRING
) WITH (
  'connector' = 'datagen',
  'rows-per-second'='1'  -- 每秒产生的数据条数
);
```

WITH 参数

参数	是否必选	默认参数	数据类型	描述
connector	必须	(none)	String	指定要使用的连接器，这里是 'datagen'。
rows-per-second	可选	10000	Long	每秒生成的行数，用以控制数据发出速率。
fields.#.kind	可选	random	String	指定 '#' 字段的生成器。可以是 'sequence' 或 'random'。
fields.#.min	可选	(Minimum value of type)	(Type of field)	随机生成器的最小值，适用于数字类型。
fields.#.max	可选	(Maximum value of type)	(Type of field)	随机生成器的最大值，适用于数字类型。
fields.#.length	可选	100	Integer	随机生成器生成字符的长度，适用于 char、varchar、string。
fields.#.start	可选	(none)	(Type of field)	序列生成器的起始值。
fields.#.end	可选	(none)	(Type of field)	序列生成器的结束值。

Logger Sink

Logger Sink 是腾讯云 Oceanus 提供的一个自定义 Logger 示例，它可以将最终的结果数据写入 TaskManager 的日志文件中，后续可以通过 Flink UI 或者控制台的日志面板查看这些日志的输出。Oceanus 平台已内置Logger Sink。

DDL 定义

```
CREATE TABLE logger_sink_table (
    id INT,
    name STRING
) WITH (
    'connector' = 'logger',
    'print-identifier' = 'DebugData'
);
```

WITH 参数

参数	是否必选	数据类型	描述
connector	必须	String	指定要使用的连接器，这里是 'logger'。
print-identifier	可选	String	日志打印的前缀信息。
all-changelog-mode	可选	Boolean	启用后，不会过滤 -U 数据，可用来模拟 ClickHouse Collapsing 模式的数据流。
records-per-second	可选	Integer	可指定每秒输出多少条数据，起到限流的作用。
mute-output	可选	Boolean	丢弃所有输出，只做条数统计（类似增强版的 Blackhole Sink）。

监控指标说明

Oceanus 为 Logger Sink 增加了很多实用的统计指标。单击 Flink UI 的运行图中的 Logger Sink 算子，即可搜索并查看指标：
numberOfInsertRecords：获取输出的 +I 消息数。
numberOfDeleteRecords：获取输出的 -D 消息数。
numberOfUpdateBeforeRecords：获取输出的 -U 消息数。
numberOfUpdateAfterRecords：获取输出的 +U 消息数。

Print Sink（不建议使用）

Flink 内置了输出到 STDOUT（标准输出）的 Print Sink，但是由于打印的格式不符合 Oceanus 日志采集器的规则，目前不能很好地展示在界面上。我们建议使用上述 Logger Sink 来代替 Print Sink。

DDL 定义

```
CREATE TABLE `print_table` (
  `id` INT,
  `name` STRING
) WITH (
 'connector' = 'print'
);
```

WITH 参数

参数	是否必选	默认值	数据类型	描述
connector	必选	(none)	String	指定要使用的连接器，此处应为 'print'。
print-identifier	可选	(none)	String	配置一个标识符作为输出数据的前缀。

standard-error	可选	false	Boolean	如果 format 需要打印为标准错误而不是标准输出，则为 True。

自定义 Connector

最近更新时间：2023-06-21 15:21:53

介绍

若内置的 Connector 无法满足需求，可以考虑**自定义 Connector** 功能，即用户可以自行上传实现了相应 Source 和 Sink 接口的类实现，然后作业在运行时会动态加载并调用。

版本说明

Flink 版本	说明
1.11	支持
1.13	支持
1.14	支持
1.16	支持

选择合适的 Connector

用户可以选择第三方提供的 Connector 实现包（例如下面介绍的 Bahir），或者自行通过编程的方式实现。

Apache Bahir 第三方包

Apache Bahir 为 Flink 提供了常见的数据源和数据目的的扩展包。
目前 Bahir 支持如下的第三方组件：

- ActiveMQ
- Akka
- Flume
- InfluxDB
- Kudu
- Redis
- Netty

自行编程实现

参见 Flink API。

构建并上传 Connector 包

步骤一：源码构建

建议参考现有的 Connector 的项目，修改其 `pom.xml` 配置文件，引入相关的依赖包，然后通过 Maven 构建一个 JAR 包。

> **说明**
> 尽量使用 `maven-shade-plugin` 将常见的依赖（例如 Apache Commons、Guava 等相关的包）进行 shade 化，以避免引入的库与流计算平台本身的类发生冲突。

步骤二：上传程序包

可以在流计算的 依赖管理 界面，上传 Connector 的程序包。首次上传是 V1 版本，以此类推。

步骤三：作业参数引用程序包

在作业的详情页，作业参数选择引用之前上传的程序包和版本。

> **注意**

> 请务必确认程序包的版本是否符合预期，避免出现各种不可预知的错误。

步骤四：保存并发布

选择程序包后，可以单击**保存**，也可以选择直接**发布草稿**。

SET 控制语句
Flink 配置项

最近更新时间：2025-05-23 14:32:12

简介

SET 语句可以调整作业的关键运行参数。目前大多数参数都可以通过 作业高级参数 功能来配置，只有极少数参数需要使用 SET 语句。

> **注意：**
> - SET 命令属于高级用法，请谨慎使用，避免参数配置不当而引起的运行时异常。
> - SET 命令不支持注释，请不要在其后增加 `--` 注释信息。
> - SET 语句行尾需加上分号。

语法

SET 语句中字符串类型的参数值必须用半角单引号括起来，布尔值和数值型的可以不加单引号。

```
SET 配置项 = '参数值';
```

示例

配置 SQL 作业状态保留时间

针对 GROUP BY 和 JOIN 等大状态的语句，Flink 提供了 Idle State Retention Time 机制，用户可以通过设置状态的最短保留时间和最长保留时间，避免状态的无限制增长造成作业崩溃（OOM）。例如，下面的语句指定了状态的最短保留时间是3小时。

> **注意：**
> - 状态的最短保留时间和最长保留时间之间的差值必须大于5分钟，否则 Flink 会报错并忽略该设置。
> - 时间单位的写法遵循 Flink 配置的规范，例如可以写10min、也可以写3h、3hour、7day、7d等。数值和单位之间的空格为可选项。

```
set table.exec.state.ttl = '3h';
```

启用 Table 的 Mini-Batch 支持

Flink SQL 对聚合提供了 Mini-Batch 支持，可以显著提升吞吐量。默认没有开启，因为会增加处理时延。如果希望使用 Mini-Batch，需要在 SQL 作业的编辑页面中添加以下语句，通过下面的设置项启用此功能（批次大小和延迟参数可以自行设置，但不可省略）：

```
set table.exec.mini-batch.enabled = true;
set table.exec.mini-batch.size = 5000;
set table.exec.mini-batch.allow-latency = '200 ms';
```

避免使用 SinkUpsertMaterializer

SinkUpsertMaterializer 在状态中维护了全量数量，可能会导致状态过大。
引入原因： Flink 非确定性更新（NDU）问题探索和规避 。
可以通过 table.exec.sink.upsert-materialize 进行配置：

- auto（默认值）：Flink 自行推断出乱序是否存在，如果可能存在乱序，则会添加 SinkUpsertMaterializer。
- none：不使用。
- force：强制使用。

需要注意，Flink 自行推断添加的 SinkUpsertMaterializer 不一定是有意义的，如果作业并行度为1或者用户明确知道 join 的 Key 不会发生变更，可以通过下面的配置关闭 SinkUpsertMaterializer。

```
table.exec.sink.upsert-materialize = 'none';
```

运算符和内置函数

概览

最近更新时间：2023-06-21 15:21:55

本节介绍流计算 Oceanus 系统内置的运算符和函数。流计算 Oceanus 在主流社区的基础上，补充并优化了部分函数。详情可参见 内置函数差异对比 。

表格中函数名的符号含义如下：

- `{ }` 表示可选且任选一项，例如 `{BOTH|LEADING|TRAILING}` 表示可以不填（使用默认行为），也可以使用 BOTH、LEADING 或 TRAILING 的任意一项。不同项的功能不同。
- `[ ]` 表示可选项，而 `*` 表示重复0或多次，例如 `[, value]*` 表示后续有0个或多个 `value` ，这种用法常用来表示不定数目的参数，例如 `value, value, value … value` 。
- 小写字母构成的字符串表示某个变量，例如 value1、value、boolean、numeric 等。
- 大写字母（例如 IS NULL）或符号（例如 =、<）表示运算符或内置函数。

内置函数分类：

- 比较函数
- 逻辑和位运算函数
- 算术函数
- 字符串操作函数
- 条件函数
- 类型转换函数
- 时间相关函数
- 聚合函数
- 时间窗口函数
- 其他函数

内置函数差异对比

最近更新时间：2023-06-21 15:21:55

流计算 Oceanus 在主流社区的基础上，补充并优化了部分函数。已注册函数如下表所示。

内置函数	流计算 Oceanus 函数定义	Flink 社区函数定义	注意点
unbase64	unbase64(string)	-	流计算 Oceanus：对 base64的字符串解码，输入 string， 输出 Array[Byte]。
decode	DECODE(binary,charset), DECODE(binary)	DECODE(binary, string)	• 流计算 Oceanus：DECODE(binary)默认使用 charset="UTF-8"。 • Flink 社区：如果任一参数为空，则返回为空。
split	SPLIT(string, separator)	-	流计算 Oceanus：将 string 表示的字符串以 separator 指定的分隔符拆分，并返回一个 Row 类型的对象。
get_row_field_str	GET_ROW_FIELD_STR(row, index)	-	流计算 Oceanus：获取某个 Row 类型对象 row 的第 index 列的值，index 从0开始计数。返回值为字符串 VARCHAR 类型。
get_row_arity	GET_ROW_ARITY(row)	-	流计算 Oceanus：获取某个 Row 类型对象 row 的列数。
can_cast_to	CAN_CAST_TO(str, type)	-	流计算 Oceanus：判断 str 字符串是否可以被转换为 type 指定的类型，返回值为布尔型。返回值可以在 CASE 语句中作为条件使用。例如 CAN_CAST_TO('123456', 'INTEGER') 则返回 True，而 CAN_CAST_TO('a145', 'DOUBLE') 则返回 False。
date_format_simple	DATE_FORMAT_SIMPLE(timestamp, simple_format)	-	计算 Oceanus：将 BIGINT(Long)类型的字段（毫秒为单位的 Unix 时间戳）以 Java 的 SimpleDateFormat 支持的时间格式化模板转为字符串形式。例如 DATE_FORMAT_SIMPLE(unix_ts, 'yyyy-MM-dd HH:mm:ss') 会返回 "2020-01-01 12:13:14" 这样格式的字符串。传入 Long，返回 String。
to_timestamp	TO_TIMESTAMP(string, simple_format)	TO_TIMESTAMP(string1[, string2])	• 流计算 Oceanus：以 Java 的 SimpleDateFormat 支持的时间格式化模板 simple_format，将 string 字符串格式化为 Timestamp 类型的时间戳。默认以东八区为准。 例如 TO_TIMESTAMP(ts, 'yyyy-MM-dd HH:mm:ss')。传入 string，返回 Timestamp。 • Flink 社区：一致
timestamp_to_long	TIMESTAMP_TO_LONG(timestamp), TIMESTAMP_TO_LONG(timestamp, mode)	-	流计算 Oceanus：将某个 TIMESTAMP 类型的参数转为 BIGINT (Long) 类型的值。若 mode 为 'SECOND'，则转为以秒来计数的 Unix 时间戳，例如1548403425。若 mode 为其他值或者省略，则转为以毫秒计数的 Unix 时间戳，例如 1548403425512。
get_tumble_start	get_tumble_start(timestamp,windowSize), get_tumble_start(timestamp,offset,windowSize)	-	流计算 Oceanus：获取窗口的起始时间，默认 offset = 0。
if_null_str	IF_NULL_STR(str, defaultValue)	-	流计算 Oceanus：如果 str 不为 NULL，则返回 str 本身；如果 str 为 NULL，则返回第二项参数 defaultValue。
get_json_object	GET_JSON_OBJECT (json_str, path_str)	-	流计算 Oceanus：获取嵌套JSON，示例可参见 文档 。

split_index	SPLIT_INDEX(string, separator, index)	SPLIT_INDEX(string1, string2, integer1)	• 将 string 表示的字符串以 separator 指定的分隔符拆分，并获取第 index 项，返回值为字符串 VARCHAR 类型。其中 index 从0开始计数。 • 流计算 Oceanus：如果 string == null 或者 separator == null 则返回 string。 • Flink 社区： 其中三项如果有任一项参数为空则返回 null。
first_str	FIRST_VALUE(expression)	FIRST_VALUE(expression)	• 流计算 Oceanus：聚合函数，返回一系列数据中，第一个数据。 • Flink 社区：一致。
last_str	LAST_VALUE(expression)	LAST_VALUE(expression)	• 流计算 Oceanus：聚合函数，返回一系列数据中，最后一个数据。 • Flink 社区：一致。
explode	EXPLODE(inputStr, separator)	–	• 流计算 Oceanus：将某个字符串分割为一张有多行的临时表。这个函数属于 Table Function，需要使用 LATERAL TABLE ( ) 关键字来引用此动态生成的临时表并作为 JOIN 条件的右表。

比较函数

最近更新时间：2023-10-08 18:04:42

比较函数的函数名和功能描述如下：

函数名	功能描述
value1 = value2	比较 value1 和 value2 是否相等，如果相等则返回 TRUE，如果不相等则返回 FALSE。 NULL 与任何值比较的结果均为 NULL，在 WHERE 条件中会被当作 FALSE。因此请使用 IS NULL 而不是 = NULL 来与 NULL 进行比较。 = 和 IS NOT DISTINCT FROM 的区别主要在于对 NULL 值的处理方式不同。
value1 <> value2	比较 value1 和 value2 是否不相等。如果不相等则返回 TRUE，否则返回 FALSE。
value1 > value2	比较 value1 是否大于 value2。如果大于返回 TRUE，否则返回 FALSE。
value1 >= value2	比较 value1 是否大于等于 value2。如果大于等于则返回 TRUE，否则返回 FALSE。
value1 < value2	比较 value1 是否小于 value2。如果小于则返回 TRUE，否则返回 FALSE。
value1 <= value2	比较 value1 是否小于等于 value2。如果小于等于则返回 TRUE，否则返回 FALSE。
value IS NULL	如果 value 为 NULL 则返回 TRUE，否则返回 FALSE。
value IS NOT NULL	如果 value 不为 NULL 则返回 TRUE，否则返回 FALSE。
value1 IS DISTINCT FROM value2	如果两个值不等（所有 NULL 值视为彼此相等），则返回 TRUE，否则返回 FALSE。
value1 IS NOT DISTINCT FROM value2	如果两个值相等（所有 NULL 值视为彼此相等），则返回 TRUE，否则返回 FALSE。
value1 BETWEEN [ASYMMETRIC | SYMMETRIC] value2 AND value3	如果 value1 大于等于 value2 且小于等于 value3，则返回 TRUE，否则返回 FALSE。 [ ] 内为可选参数，默认行为是 ASYMMETRIC。如果显式声明 SYMMETRIC，则 value2 和 value3 的顺序可以任意对调，不影响结果。
value1 NOT BETWEEN [ASYMMETRIC | SYMMETRIC] value2 AND value3	如果 value1 小于 value2 或者大于 value3，则返回 TRUE，否则返回 FALSE。 [ ] 内为可选参数，默认行为是 ASYMMETRIC。如果显式声明 SYMMETRIC，则 value2 和 value3 的顺序可以任意对调，不影响结果。
string1 LIKE string2	如果 string1 符合 string2 表示的 pattern 则返回 TRUE，否则返回 FALSE。
string1 NOT LIKE string2	如果 string1 不符合 string2 表示的 pattern 则返回 TRUE，否则返回 FALSE。
string1 SIMILAR TO string2	如果 string1 符合 string2 表示的正则表达式，则返回 TRUE，否则返回 FALSE。
string1 NOT SIMILAR TO string2	如果 string1 不符合 string2 表示的正则表达式，则返回 TRUE，否则返回 FALSE。
value IN (listItem [, listItem]*)	如果 value 处于 IN 后面的值列表中，则返回 TRUE。该语句等价于多个 OR 表达式的连接。 如果值列表中包含 NULL，则找不到时会返回 NULL，否则找不到时返回 FALSE。 如果 value 是 NULL，则结果永远为 NULL。
value NOT IN (listItem, [, listItem]*)	如果 value 不在 IN 后的值列表中，则返回 TRUE，否则返回 FALSE。
EXISTS (某个子查询)	如果子查询返回至少一行，则返回 TRUE，否则返回 FALSE。 该查询对内存压力较大，请谨慎使用。
value IN (某个子查询)	如果子查询返回的多行结果中，有一条等于 value 的值，则返回 TRUE，否则返回 FALSE。 该查询对内存压力较大，请谨慎使用。
value NOT IN (某个子查询)	如果子查询返回的多个结果中，没有一条等于 value，则返回 TRUE，否则返回 FALSE。 该查询对内存压力较大，请谨慎使用。

版权所有：腾讯云计算（北京）有限责任公司

逻辑和位运算函数

最近更新时间：2023-06-21 15:21:56

版权所有：腾讯云计算（北京）有限责任公司

逻辑函数用来执行逻辑运算，结果是布尔值（Boolean）。
逻辑状态有 TRUE、FALSE、UNKNOWN 三种（NULL 值的逻辑状态是 UNKNOWN），因而 NOT TRUE 不一定是 FALSE，还可能是 UNKNOWN。

函数名	功能描述
boolean1 OR boolean2	如果 boolean1 或者 boolean2 任意一个为 TRUE，则返回 TRUE，否则返回 FALSE。
boolean1 AND boolean2	当且仅当 boolean1 和 boolean2 均为 TRUE 时才返回 TRUE，否则返回 FALSE。
NOT boolean	如果 boolean 为 TRUE 则返回 FALSE；如果为 FALSE 则返回 TRUE；如果为 UNKNOWN 则返回 UNKNOWN。
boolean IS FALSE	如果 boolean 为 FALSE 则返回 TRUE；如果 boolean 为 UNKNOWN 则返回 FALSE。
boolean IS NOT FALSE	如果 boolean 不为 FALSE 则返回 TRUE；如果为 UNKNOWN 则返回 TRUE。
boolean IS UNKNOWN	如果 boolean 为 UNKNOWN 则返回 TRUE，否则返回 FALSE。
boolean IS NOT UNKNOWN	如果 boolean 不为 UNKNOWN 则返回 TRUE，否则返回 FALSE。

版权所有：腾讯云计算（北京）有限责任公司

算术函数

最近更新时间：2023-06-21 15:21:56

算术函数的函数名和功能描述如下：

函数名	功能描述
+numeric	返回 numeric 本身。
-numeric	返回 0-numeric 的值，即反转符号。
numeric1 + numeric2	计算 numeric1 加 numeric2 的结果。
numeric1 - numeric2	计算 numeric1 减 numeric2 的结果。
numeric1 * numeric2	计算 numeric1 乘 numeric2 的结果。
numeric1 / numeric2	计算 numeric1 除以 numeric2 的结果。
POWER(numeric1, numeric2)	计算 numeric1 的 numeric2 次方。
ABS(numeric)	返回 numeric 的绝对值。
MOD(numeric1, numeric2)	返回 numeric1 除以 numeric2 的余数。如果 numeric1 是负数，那么余数也为负数。
SQRT(numeric)	计算 numeric 的平方根。
LN(numeric)	计算 numeric 的自然对数（以 e 为底）。
LOG10(numeric)	计算 numeric 以10为底的对数。
LOG2(numeric)	计算 numeric 以2为底的对数。
LOG(numeric2)LOG(numeric1, numeric2)	如果提供一个参数，计算 numeric2 的自然对数（等价于 LN）。如果提供两个参数，计算 numeric2 以 numeric1 为底的对数。
EXP(numeric)	计算 e 的 numeric 次方。
CEIL(numeric)CEILING(numeric)	返回 numeric 向上取整的值。
FLOOR(numeric)	返回 numeric 向下取整的值。
TRUNCATE(numeric1, numeric2)	对 numeric1 的小数部分以截断的方式取整，取整的位数由 numeric2 决定。 如果参数为 NULL，则结果也是 NULL。 如果 numeric2 为0或不填，则结果没有小数部分。numeric2 可为负数，此时对整数部分取整。例如 TRUNCATE(42.345, 2) 返回42.34；TRUNCATE(42.345) 返回42.0；TRUNCATE(42.345, -1) 返回40.0。
SIN(numeric)	计算 numeric 的正弦值。
SINH(numeric)	计算 numeric 的双曲正弦值（返回值为 DOUBLE 类型）。
COS(numeric)	计算 numeric 的余弦值。
COSH(numeric)	计算 numeric 的双曲余弦值（返回值为 DOUBLE 类型）。
TAN(numeric)	计算 numeric 的正切值。
TANH(numeric)	计算 numeric 的双曲正切值（返回值为 DOUBLE 类型）。
COT(numeric)	计算 numeric 的余切值。
ASIN(numeric)	计算 numeric 的反正弦值。
ACOS(numeric)	计算 numeric 的反余弦值。

版权所有：腾讯云计算（北京）有限责任公司

ATAN(numeric)	计算 numeric 的反正切值。
ATAN2(numeric1, numeric2)	计算 (numeric1、numeric2) 坐标点的四象限反正切值。
DEGREES(numeric)	将 numeric 从弧度转为角度。
RADIANS(numeric)	将 numeric 从角度转为弧度。
SIGN(numeric)	得到 numeric 的符号，负数是-1，0返回0，正数是1。
ROUND(numeric, int)	对 numeric 取整，位数由 int 值给定，可正也可负。
PI()	返回一个可以代表 π 的值。
E()	返回一个可以代表自然对数的底数 e 的值。
RAND()RAND(种子值)	返回一个0.0 – 1.0（不包含）的伪随机数，可以指定一个整数作为种子值。
RAND_INTEGER(上限值) RAND_INTEGER(种子值, 上限值)	返回一个0.0 – 指定上限值（不包含）的伪随机数。
UUID()	返回一个随机生成的 Type-4 UUID 字符串。
LOG(numeric)LOG(base, numeric)	返回 numeric 的自然对数，或者返回以指定 base 为底的对数。
BIN(numeric)	获取 numeric 的二进制表示的字符串，例如输入4，则返回"100"。
HEX(numeric)HEX(string)	获取 numeric 或 string 的十六进制表示的字符串，例如输入15或"15" 则返回"F"。

条件函数

最近更新时间：2024-10-11 15:19:01

条件函数的函数名和功能描述如下：

函数名	功能描述
CASE valueWHEN value1 [, value11]* THEN result1[WHEN valueN [, valueN1]* THEN resultN]*[ELSE resultZ]END	当满足 value1 ~ value11 的任意值时，返回 result1。 当满足 valueN ~ valueN1 的任意值时，返回 resultN。 否则返回 resultZ。
CASEWHEN condition1 THEN result1[WHEN conditionN THEN resultN]*[ELSE resultZ]END	当满足 condition1 时返回 result1。 当满足 conditionN 时返回 resultN。 否则返回 resultZ。
NULLIF(value1, value2)	如果 value1 与 value2 相同，则返回 NULL，否则返回第一个值。例如 NULLIF(5, 5) 返回 NULL，而 NULLIF(5, 0) 返回5。
COALESCE(value, value [, value]*)	如果前值是 NULL，则提供一个后续的值，例如 COALESCE(NULL, 5) 则返回5。
IF(condition, true_value, false_value)	如果 condition 的条件满足，返回 true_value，否则返回 false_value。例如，IF(2 > 1, 2, 1) 返回2，而 IF (1 > 2, 99, 100) 返回100。
IS_ALPHA(string)	判断字符串是不是仅由纯字母组成。如果是，则返回 true，否则返回 false。
IS_DECIMAL(string)	判断字符串是不是一个合法的数字（整数、小数、负数均可）。如果是，则返回 true，否则返回 false。
IS_DIGIT(string)	判断字符串是不是仅由纯数字组成（即无符号整数）。如果是，则返回 true，否则返回 false。
IF_NULL_STR(str, defaultValue)	如果 str 不为 NULL，则返回 str 本身；如果 str 为 NULL，则返回第二项参数 defaultValue。

版权所有：腾讯云计算（北京）有限责任公司

字符串操作函数

最近更新时间：2025-05-26 15:55:21

函数

函数名	功能描述
string1 || string2	连接两个字符串，返回两个字符串拼接后的结果，等同于 CONCAT(string1, string2)。
CHAR_LENGTH(string)	返回字符串的长度。
CHARACTER_LENGTH(string)	与 CHAR_LENGTH(string) 相同。
UPPER(string)	返回 string 的全大写字母形式。
LOWER(string)	返回 string 的全小写字母形式。
POSITION(string1 IN string2)	获取 string1 在 string2 中第一次出现的位置（**位置从1开始计数**）。当 string1 在 string2 中找不到时，返回0。
TRIM({BOTH |LEADING |TRAILING }string1 FROM string2)	从 string2 中除去字符串首尾/首位/末尾的 string1。默认情况下，首尾的空格都被删除。
LTRIM(string)	去掉 string 字符串最左边的所有空格。例如 LTRIM(' Hello') 会返回 'Hello'。
RTRIM(string)	去掉 string 字符串最右边的所有空格。例如 RTRIM(' World ') 会返回 ' World'。
REPEAT(string, integer)	将 string 字符串重复 integer 次。例如 REPEAT('Meow', 3) 会返回 'MeowMeowMeow'。
REGEXP_REPLACE(string1, string2, string3)	对 string1 字符串以 string2 表示的正则表达式进行替换，替换内容是 string3。例如 REGEXP_REPLACE('banana', 'a|n', 'A') 返回 'bAAAAA'。
REPLACE(string1, string2, string3)	将 string1 字符串中所有的 string2 替换为 string3。例如 REPLACE('banana', 'a', 'A') 返回 'bAnAnA'。
OVERLAY(string1 PLACING string2 FROM start_pos [FOR length])	将 string1 从第 start_pos 位（**start_pos 从1开始计数**）开始的子串替换为 string2。可以指定替换的长度。
SUBSTRING(string from pos [FOR length])	获取从 pos 位开始的子串，默认行为是直到源字符串的最后，可以使用 FOR 来指定子串的长度。其中字符串**起始 pos 从1开始计数**，而不是0。
REGEXP_EXTRACT(string1, string2[, integer])	从 string1 中提取正则分组，正则表达式为 string2，第一个括号为第一组，以此类推。可通过第三个参数 integer 来指定所需的分组号（从1开始）。如果不指定分组号或者分组号为0，则表示返回整个正则表达式匹配到的字符串。例如，`REGEXP_EXTRACT('foothebar', 'foo(.*?)(bar)', 2)` 返回 'bar'。
INITCAP(string)	将 string 中的单词，转为以大写字母开头，其他是小写字母（首字母大写）的形式。例如 INITCAP('i have a dream') 返回 'I Have A Dream'。
CONCAT(string1, string2 …)	连接多个字符串。若任意字符串为 NULL，则结果为 NULL。
CONCAT_WS(separator, string1, string2, …)	使用指定的分隔符 separator 连接多个字符串。如果 separator 为 NULL，则结果为 NULL。如果某个字符串为 NULL，则跳过它；但是不会跳过空字符串。例如 `CONCAT_WS('~', 'AA','BB', '', 'CC')` 会返回 `AA~BB~~CC`。
LPAD(text, length, padding)	使用 padding 指定的字符串从左侧填充 text 字符串到指定长度 length。如果 text 比 length 更长，则会截断到 length 的长度。
RPAD(text, length, padding)	使用 padding 指定的字符串从右侧填充 text 字符串到指定长度 length。如果 text 比 length 更长，则会截断到 length 的长度。
FROM_BASE64(string)	将 Base64 编码的 string 字符串解码为字符串。如果 string 为 NULL，则返回 NULL。

TO_BASE64(string)	将 string 表示的字符串编码为 Base64 字符串。
ASCII(string)	返回 string 字符串中第一个字符的 ASCII 码。如果 string 为 NULL，则返回 NULL。例如 ASCII('an apple') 返回97，因为首字母 'a' 的 ASCII 编码是97。
CHR(integer)	返回编码为 integer 的 ASCII 字符。例如 CHR(97) 返回 'a'。
ENCODE(string, charset)	将 string 字符串转码为 charset 指定的字符集编码的 BINARY 类型，例如 ENCODE(hello, 'GBK')。
DECODE(binary, charset)	将 binary 表示的 BINARY 类型以 charset 指定的字符集解码，例如 DECODE(binary_field, 'UTF-16LE')。
INSTR(string1, string2)	返回 string2 在 string1 字符串中首次出现的位置。如果任意参数为 NULL，结果为 NULL。
LEFT(string, n)	返回 string 从左起前 n 个字符。如果 n 为负数，则返回空字符串。如果任意参数为 NULL，结果为 NULL。
RIGHT(string, n)	返回 string 从右起后 n 个字符。如果 n 为负数，则返回空字符串。如果任意参数为 NULL，结果为 NULL。
LOCATE(string1, string2[, integer])	返回跳过 integer 个字符后，string1 在 string2 中首次出现的位置（**参数顺序与 INSTR 函数相反**）。如果未找到，则返回0。如果任意参数为 NULL，结果为 NULL。
REGEXP(string, regex)	如果 regex 表示的正则表达式可以匹配 string 中的字符串的任意子串，那么返回 TRUE，否则返回 FALSE。如果任意参数为 NULL，结果为 NULL。
REVERSE(string)	反转 string 字符串。如果任意参数为 NULL，结果为 NULL。
SPLIT_INDEX(string, separator, index)	将 string 表示的字符串以 separator 指定的分隔符拆分，并获取第 index 项，返回值为字符串 VARCHAR 类型。其中 **index 从0开始计数**。
SPLIT(string, separator)	将 string 表示的字符串以 separator 指定的分隔符拆分，并返回一个 Row 类型的对象。
STR_TO_MAP(string1[, string2, string3])	将 string1 字符串用 string2 提供的数据分隔符（默认为半角逗号 `,` ）和 string3 提供的键值间分隔符（默认为半角等号 `=` ）进行拆分，结果为键值对 `MAP<string, string>` 类型。例如 STR_TO_MAP('k1=v1,k2=v2,k3=v3') 返回键值对（非字符串）{'k1': 'v1', 'k2': 'v2', 'k3': 'v3'}。
SUBSTR(string[, pos[, length]])	返回 string 字符串从 pos 位置开始，长度为 length 的子串。如果不提供 length，则默认到该字符串尾部。
EXPLODE(inputStr, separator)	将某个字符串分割为一张有多行的临时表。这个函数属于 Table Function，需要使用 LATERAL TABLE ( ) 关键字来引用此动态生成的临时表并作为 JOIN 条件的右表。
GET_ROW_ARITY(row)	获取某个 Row 类型对象 row 的列数。
GET_ROW_FIELD_STR(row, index)	获取某个 Row 类型对象 row 的第 index 列的值，**index 从0开始计数**。返回值为字符串 VARCHAR 类型。
GET_JSON_OBJECT(json_str, path_str)	按 path_str 指定的 JSONPath 路径，获取某个 JSON 字符串 json_str 中的元素，可以任意嵌套。支持的 JSONPath 语法：`$` 表示根对象，`.` 表示子元素，`[]` 表示数组索引，`*` 为数组索引 [] 的通配符。
IS_ALPHA(content)	判断字符串是否只包含字母。
IS_DIGIT(content)	判断字符串是否只包含数字。
MD5(string)	返回字符串的 MD5 值。
POSITION(string1 IN string2)	返回目标字符串 x 在被查询字符串y里第一次出现的位置。如果目标字符串 x 在被查询字符串 y 中不存在，返回值为0。
SHA1	返回字符串 expr 的 SHA1 值。
SHA256	返回字符串 expr 的 SHA256 值。

示例

||

功能描述：连接两个字符串，返回两个字符串拼接后的结果，等同于 CONCAT(string1, string2)
语法：string1 || string2
示例测试语句：SELECT string1 || string2 FROM Test ;
测试数据和结果：

测试数据（VARCHAR string1）	测试数据（VARCHAR string2）	测试结果（VARCHAR）
Oce	anus	Oceanus

CHAR_LENGTH

功能描述：返回字符串的长度。
语法：CHAR_LENGTH(string)
示例测试语句：SELECT CHAR_LENGTH(var1) AS length FROM Test;
测试数据和结果：

测试数据（VARCHAR var1）	测试结果（INT length）
Oceanus	7

CHARACTER_LENGTH

功能描述：与 CHAR_LENGTH(string) 相同。
语法：CHARACTER_LENGTH(string)
示例测试语句：SELECT CHAR_LENGTH(var1) AS length FROM Test;
测试数据和结果：

测试数据（VARCHAR var1）	测试结果（INT length）
Oceanus	7

LOWER

功能描述：返回小写字符的字符串。
语法：LOWER(string)
示例测试语句：SELECT LOWER(var1) AS lower FROM Test;
测试数据和结果：

测试数据（VARCHAR var1）	测试结果（VARCHAR lower）
OCeanus	oceanus

UPPER

功能描述：返回大写字符的字符串。
语法：UPPER(string)
示例测试语句：SELECT UPPER(var1) AS upper FROM Test;
测试数据和结果：

测试数据（VARCHAR var1）	测试结果（VARCHAR upper）
OCeanus	OCEANUS

TRIM

功能描述：从 string2 中除去字符串首尾/首位/末尾的 string1。默认情况下，首尾的空格都被删除。
语法：TRIM({BOTH | LEADING | TRAILING } string1 FROM string2)

示例测试语句： SELECT TRIM(BOTH string1 FROM string2) AS res FROM Test;
测试数据和结果：

测试数据（VARCHAR string1）	测试数据（VARCHAR string2）	测试结果（VARCHAR res）
a	aoceanusa	oceanus

CONCAT

功能描述： 拼接两个或多个字符串值从而组成一个新的字符串。如果任一参数为 NULL 时，则跳过该参数。
语法： CONCAT(string1, string2 ···)
示例测试语句： SELECT CONCAT('123', '456', 'abc', 'def') AS res FROM Test;
测试数据和结果： '123456abcdef'

测试数据（VARCHAR string1）	测试数据（VARCHAR string2）	测试数据（VARCHAR string3）	测试数据（VARCHAR string4）	测试结果（VARCHAR res）
123	456	abc	def	123456abcdef

CONCAT_WS

功能描述： 使用指定的分隔符 separator 连接多个字符串。如果 separator 为 NULL，则结果为 NULL。如果某个字符串为 NULL，则跳过它。但是不会跳过空字符串。
语法： CONCAT_WS(separator,string1,string2, ···)
示例测试语句： SELECT CONCAT_WS(separator, string1,string2, string3) AS res FROM Test;
测试数据和结果：

测试数据（VARCHAR separator）	测试数据（VARCHAR string1）	测试数据（VARCHAR string2）	测试数据（VARCHAR string3）	测试结果（VARCHAR res）
-	AA	BB	CC	AA-BB-CC

INITCAP

功能描述： 将 string 中的单词，转为以大写字母开头，其他是小写字母（首字母大写）的形式。
语法： INITCAP(string)
示例测试语句： SELECT INITCAP(var1) AS str FROM Test;
数据和结果：

测试数据（VARCHAR var1）	测试结果（VARCHAR str）
i have a dream	I Have A Dream

IS_ALPHA

功能描述： 判断字符串是否只包含字母。
语法： IS_ALPHA(content)
示例测试语句： SELECT IS_ALPHA(content) AS result FROM Test;
测试数据和结果：

测试数据（VARCHAR content）	测试结果（BOOLEAN result）
Oceanus	true
oceanus123	false
''	false
null	false

IS_DIGIT

功能描述：判断字符串是否只包含数字。
语法：IS_DIGIT(content)
示例测试语句：SELECT IS_DIGIT(content) AS result FROM Test;
测试数据和结果：

测试数据（VARCHAR content）	测试结果（BOOLEAN case_result）
58.0	true
58	true
58pl	false
''	false
null	false

LPAD

功能描述：使用 padding 指定的字符串从左侧填充 text 字符串到指定长度 length。如果 text 比 length 更长，则会截断到 length 的长度。
语法：LPAD(text , length , padding)
示例测试语句：SELECT LPAD(test, length, padding) AS res FROM Test;
测试数据和结果：

测试数据（VARCHAR text）	测试数据（INT length）	测试数据（VARCHAR padding）	测试结果（VARCHAR res）
oceanus	3	hello	hel
oceanus	-1	hello	''
oceanus	12	hello	hellooceanus

RPAD

功能描述：使用 padding 指定的字符串从右侧填充 text 字符串到指定长度 length。如果 text 比 length 更长，则会截断到 length 的长度。
语法：RPAD(text , length , padding)
示例测试语句：SELECT RPAD(text, length, padding) AS res FROM Test;
测试数据和结果：

测试数据（VARCHAR text）	测试数据（INT length）	测试数据（VARCHAR padding）	测试结果（VARCHAR res）
oceanus	3	hello	oce
oceanus	-1	hello	''
oceanus	12	hello	oceanushello

MD5

功能描述：返回字符串的 MD5 值。
语法：MD5(string)
示例测试语句：SELECT MD5(content) AS res FROM Test;
测试数据和结果：

测试数据（VARCHAR content）	测试结果（VARCHAR res）
abc	900150983cd24fb0d6963f7d28e17f72

OVERLAY

功能描述：将 string1 从第 start_pos 位（**start_pos 从1开始计数**）开始的子串替换为 string2。可以指定替换的长度。
语法：SELECT OVERLAY(string1 PLACING string2 FROM start_pos [FOR length])
示例测试语句：SELECT OVERLAY(string1 PLACING string2 FROM start_pos FOR length) AS res FROM Test;
测试数据和结果：

测试数据（VARCHAR string1）	测试数据（VARCHAR string2）	测试数据（INT start_pos）	测试数据（INT length）	测试结果（VARCHAR res）
oceanus	abc	2	2	oabcanus

POSITION

功能描述：返回目标字符串 x 在被查询字符串 y 里第一次出现的位置。如果目标字符串 x 在被查询字符串 y 中不存在，返回值为0。
语法：POSITION(string1 IN string2)
示例测试语句： SELECT POSITION(string1 IN string2) AS res FROM Test;
测试数据和结果：

测试数据（VARCHAR string1）	测试数据（VARCHAR string2）	测试结果（VARCHAR res）
nu	oceanus	5

GET_JSON_OBJECT(json_str, path_str)

功能描述：按 path_str 指定的 JSONPath 路径，获取某个 JSON 字符串 json_str 中的元素，可以任意嵌套。支持的 JSONPath 语法：`$` 表示根对象，`.` 表示子元素，`[]` 表示数组索引，`*` 为数组索引 [] 的通配符。
语法：GET_JSON_OBJECT(json_str, path_str)
示例测试语句：SELECT GET_JSON_OBJECT(json_str, path_str) AS res FROM Test;
测试数据和结果：

测试数据（VARCHAR json_str）
{"school": {"student":[{"num":8,"type":"A"},{"num":9,"type":"B"}],"teacher":{"num":200,"type":"A"} },"headmaster":"mark" }

测试数据（VARCHAR path_str）	测试结果（VARCHAR res）
$.school	{\"student\":[{\"num\":8,\"type\":\"A\"},{\"num\":9,\"type\":\"B\"}], \"teacher\":{\"num\":200,\"type\":\"A\"}}
$.school.student[1]	{\"num\":9,\"type\":\"B\"}
$.school.teacher	{\"num\":200,\"type\":\"A\"}
$.headmaster	mark

REPLACE

功能描述：将 string1 字符串中所有的 string2 替换为 string3。
语法：REPLACE(string1, string2, string3)
示例测试语句：SELECT REPLACE(string1, string2, string3) AS res FROM Test;
测试数据和结果：

测试数据（VARCHAR string1）	测试数据（VARCHAR string2）	测试数据（VARCHAR string3）	测试结果（VARCHAR res）
banana	a	A	bAnAnA

SHA1

功能描述：返回字符串 expr 的 SHA1 值。
语法：SHA1(expr)

示例测试语句：SELECT SHA1(expr) AS res FROM Test;
测试数据和结果：

测试数据（VARCHAR expr）	测试结果（VARCHAR res）
abc	a9993e364706816aba3e25717850c26c9cd0d89d

SHA256

功能描述：返回字符串 expr 的 SHA256 值。
语法：SHA256（expr）
示例测试语句：SELECT SHA256(expr) FROM Test;
测试数据和结果：

测试数据（VARCHAR expr）	测试结果（VARCHAR res）
abc	ba7816bf8f01cfea414140de5dae2223b00361a396177a9cb410ff61f20015ad

类型转换函数

最近更新时间：2023-06-21 15:21:57

函数

> **注意**
> 如果使用 CAST() 函数，将时间段 INTERVAL 转为数字，则结果会是字面值（可能不符合预期）。例如 `CAST(INTERVAL '1234' MINUTE AS BIGINT)`，则结果会是字面值1234，而非时间段表示的毫秒值。

函数名	功能描述
CAST(value AS type)	将某个值转为 type 类型，例如 CAST(`hello` AS VARCHAR) 会将 `hello` 字段转为 VARCHAR 类型。
CAN_CAST_TO(str, type)	判断 str 字符串是否可以被转换为 type 指定的类型，返回值为布尔型。返回值可以在 CASE 语句中作为条件使用。例如 `CAN_CAST_TO('123456', 'INTEGER')` 则返回 True，而 `CAN_CAST_TO('a145', 'DOUBLE')` 则返回 False。
TO_TIMESTAMP(string, simple_format)	以 Java 的 SimpleDateFormat 支持的时间格式化模板 simple_format，将 string 字符串格式化为 Timestamp 类型的时间戳。默认以东八区为准。 例如 `TO_TIMESTAMP(ts, 'yyyy-MM-dd HH:mm:ss')` 。
DATE_FORMAT_SIMPLE(timestamp, simple_format)	将 BIGINT（Long）类型的字段（毫秒为单位的 Unix 时间戳）以 Java 的 SimpleDateFormat 支持的时间格式化模板转为字符串形式。例如 `DATE_FORMAT_SIMPLE(unix_ts, 'yyyy-MM-dd HH:mm:ss')` 会返回 "2020-01-01 12:13:14" 这样格式的字符串。
DATE_FORMAT(timestamp, format)	将 Timestamp 类型的字段以 Java 的 SimpleDateFormat 支持的时间格式化模板转为字符串形式。例如 `DATE_FORMAT(ts, 'yyyy-MM-dd HH:mm:ss')` 会返回 "2020-01-01 12:13:14"。
TIMESTAMP_TO_LONG(timestamp) 或 TIMESTAMP_TO_LONG(timestamp, mode)	将某个 TIMESTAMP 类型的参数转为 BIGINT (Long) 类型的值。若 mode 为 'SECOND'，则转为以秒来计数的 Unix 时间戳，例如1548403425。若 mode 为其他值或者省略，则转为以毫秒计数的 Unix 时间戳，例如1548403425512。

示例

CAST

功能描述：将某个值转为 type 类型。
语法：CAST(value AS type)
示例测试语句：SELECT CAST(var1 AS VARCHAR) FROM TEST;
测试数据和结果：

测试数据（INT var1）	测试结果 VARCHAR
58	'58'

CAN_CAST_TO

功能描述：判断 str 字符串是否可以被转换为 type 指定的类型，返回值为布尔型。返回值可以在 CASE 语句中作为条件使用。
语法：CAN_CAST_TO(str, type)
示例测试语句：SELECT CAN_CAST_TO(var1,type) FROM Test;
测试数据和结果：True

测试数据（VARCHAR var1）	测试数据（VARCHAR type）	测试结果（BOOLEAN）
123456	INTEGER	true

DATE_FORMAT_SIMPLE

功能描述：将 **BIGINT（Long）**类型的字段（必须是以毫秒为单位的 Unix 时间戳）以 Java 的 SimpleDateFormat 支持的时间格式化模板转为字符串形式（输出时区为：GMT+8）。
语法：DATE_FORMAT_SIMPLE(timestamp, simple_format)
示例测试语句：SELECT DATE_FORMAT_SIMPLE(unix_ts,'yyyy-MM-dd HH:mm:ss') FROM Test;
测试数据和结果：

测试数据（unix_ts）	测试结果 VARCHAR
1627997937000	2021-08-03 21:38:57

DATE_FORMAT

功能描述：将 **Timestamp** 类型的字段以 Java 的 SimpleDateFormat 支持的时间格式化模板转为字符串形式。
语法：DATE_FORMAT(timestamp, format)
示例测试语句：SELECT DATE_FORMAT(timestamp,format) FROM Test;
测试数据和结果：

测试数据（timestamp）	测试数据（format）	测试结果 VARCHAR
2021-01-01 12:13:14	yyMMdd	210101
2021-01-01 12:13:14	yyyyMMdd	20210101

时间相关函数

最近更新时间：2023-06-21 15:21:57

时间相关函数的函数名和功能描述如下：

函数名	功能描述
DATE string	将 "yyyy-MM-dd" 形式表示的字符串转为 SQL 日期（DATE）类型。
TIME string	将 "HH:mm:ss" 形式表示的字符串转为 SQL 时间（TIME）类型。
TIMESTAMP string	将 "yyyy-MM-dd HH:mm:ss[.SSS]" 形式的字符串转为 SQL 时间戳（TIMESTAMP）类型。
INTERVAL string range	接受 "dd hh:mm:ss.fff" 形式的字符串（毫秒），或者 "yyyy-MM" 形式的字符串（月）。 对于毫秒，可以接受 DAY、MINUTE、DAY TO HOUR、DAY TO SECOND 作为 range。 对于月，接受 YEAR 或 YEAR TO MONTH 作为 range。例如 INTERVAL '10 00:00:00.004' DAY TO SECOND、INTERVAL '10' DAY、INTERVAL '2-10' YEAR TO MONTH。
CURRENT_DATE	返回当前的 SQL DATE 格式表示的日期（UTC）。
CURRENT_TIME	返回当前的 SQL TIME 格式表示的时间（UTC）。
CURRENT_TIMESTAMP	返回当前 SQL TIMESTAMP 格式表示的时间戳（UTC）。
LOCALTIME	返回本地时区表示的 SQL 时间。
LOCALTIMESTAMP	返回本地时区表示的 SQL 时间戳。
EXTRACT(timeintervalunit FROM temporal)	获取时间点或时间段字符串中的某项。例如 EXTRACT(DAY FROM DATE '2006-06-05') 返回5，而 EXTRACT(YEAR FROM DATE '2018-06-12') 则返回2018。
YEAR(date)	返回指定日期中的年份，等价于 EXTRACT(YEAR FROM date)。例如 YEAR(DATE '2020-08-12') 返回2020。
QUARTER(date)	返回指定日期的季度，等价于 EXTRACT(QUARTER FROM date)。例如 QUARTER(DATE '2012-09-10')返回3。
MONTH(date)	返回指定日期的月份，等价于 EXTRACT(MONTH FROM date)。例如 MONTH(DATE '2012-09-10') 返回9。
WEEK(date)	返回指定日期的周数，即当年的第几周，等价于 EXTRACT(WEEK FROM date)。例如 WEEK(DATE '1994-09-27') 返回39。
DAYOFYEAR(date)	返回指定日期在当年的天数，即当年的第几天（范围是[1，366]），等价于 EXTRACT(DOY FROM date)。例如 DAYOFYEAR(DATE '1994-09-27') 返回270。
DAYOFMONTH(date)	返回指定日期在当月的天数，即当月的第几天（范围是[1，31]），等价于 EXTRACT(DAY FROM date)。例如 DAYOFMONTH(DATE '1994-09-27') 返回27。
DAYOFWEEK(date)	返回指定日期在本周的天数，即本周的第几天（范围是[1，7]），等价于 EXTRACT(DOW FROM date)。例如 DAYOFWEEK(DATE '1994-09-27') 返回3。
HOUR(timestamp)	返回指定时间戳的小时部分（范围是[0，23]）。等价于 EXTRACT(HOUR FROM timestamp)。例如 HOUR('2017-10-02 12:25:44') 返回12。
MINUTE(timestamp)	返回指定时间戳的分钟部分（范围是[0，59]）。等价于 EXTRACT(MINUTE FROM timestamp)。例如 MINUTE('2017-10-02 12:25:44') 返回25。
SECOND(timestamp)	返回指定时间戳的秒部分（范围是[0，59]）。等价于 EXTRACT(SECOND FROM timestamp)。例如 SECOND('2017-10-02 12:25:44') 返回44。
FLOOR(timepoint TO timeintervalunit)	将一个时间点向下取整，例如 FLOOR(TIME '12:44:31' TO MINUTE) 返回12:44:00。

版权所有：腾讯云计算（北京）有限责任公司

CEIL(timepoint TO timeintervalunit)	将一个时间点向上取整，例如 CEIL(TIME '12:44:31' TO MINUTE) 返回12:45:00。
(timepoint, temporal) OVERLAPS (timepoint, temporal)	判断两个时间段是否重叠。例如 (TIME'2:55:00', INTERVAL '1' HOUR) OVERLAPS (TIME'3:30:00', INTERVAL '2' HOUR) 返回 TRUE；而 (TIME'9:00:00', TIME '10:00:00') OVERLAPS (TIME'10:15:00', INTERVAL '3' HOUR) 返回 FALSE。
TO_TIMESTAMP(string, simple_format)	将字符串格式的时间戳转为 Timestamp 类型。详见 类型转换函数 。
DATE_FORMAT_SIMPLE (timestamp, simple_format)	将 BIGINT 类型以毫秒为单位的 Unix 时间戳格式化为字符串。详见 类型转换函数 。
DATE_FORMAT(timestamp, format)	将 Timestamp 类型的时间戳格式化为字符串。详见 类型转换函数 。
TIMESTAMP_TO_LONG(timestamp)	将 Timestamp 类型的时间戳转为 BIGINT 类型以毫秒或秒（可选）为单位的 Unix 时间戳。详见 类型转换函数 。
TIMESTAMPADD(unit, interval, timestamp)	对指定 timestamp 增加一个时间段（允许为负数），单位区间必须为 SECOND、MINUTE、HOUR、DAY、WEEK、MONTH、QUARTER、YEAR 中的一种。例如 TIMESTAMPADD(WEEK, 1, '2013-01-02')返回'2013-01-09'。
TIMESTAMPDIFF(timepointunit, timepoint1, timepoint2)	计算指定单位区间下，timepoint1 和 timepoint2 之间的时间差（允许为负数）。timepointunit 区间必须为 SECOND、MINUTE、HOUR、DAY、WEEK、MONTH、QUARTER、YEAR 中的一种。例如 TIMESTAMPDIFF(DAY, TIMESTAMP '2003-01-02 10:00:00', TIMESTAMP '2003-01-03 10:00:00') 返回1，即相差正1天。
CONVERT_TZ(string1, string2, string3)	将 string1 表示的时间戳（必须是 'yyyy-MM-dd HH:mm:ss' 格式）从时区（允许缩写例如 "PST"，全称例如 "Asia/Shanghai"，或者任意时区例如 "GMT-8:00"）string2 转换到时区 string3。例如 CONVERT('1970-01-01 00:00:00', 'UTC', 'America/Los_Angeles') 返回 '1969-12-31 16:00:00'。
FROM_UNIXTIME(numeric[, string])	返回 numeric 代表的 Unix 时间戳（从 1970-01-01 00:00:00 UTC 到现今的秒数）转为 'YYYY-MM-DD hh:mm:ss' 格式的字符串。默认使用 UTC+8 时区，即北京时间（Asia/Shanghai）。
UNIX_TIMESTAMP()	返回以秒为单位的 Unix 时间戳（从 1970-01-01 00:00:00 UTC 到现今的秒数），类型为 BIGINT。默认使用 UTC+8 时区，即北京时间（Asia/Shanghai）。
UNIX_TIMESTAMP(string1[, string2])	将 string1 字符串以 string2 的格式（可选，默认是 'yyyy-MM-dd HH:mm:ss'）转为 Unix 时间戳，类型为 BIGINT。
TO_DATE(string1[, string2])	将 string1 字符串以 string2 的格式（可选，默认是 'yyyy-MM-dd HH:mm:ss'）转为 DATE 格式。
TO_TIMESTAMP(string1[, string2])	将 string1 字符串以 string2 的格式（可选，默认是 'yyyy-MM-dd HH:mm:ss'）转为 TIMESTAMP 格式。默认使用 UTC+8 时区，即北京时间（Asia/Shanghai）。
NOW()	返回当前的 SQL TIMESTAMP 时间戳，时区为 UTC。

聚合函数

最近更新时间：2023-06-21 15:21:58

函数

函数名	功能描述
COUNT([ALL] expression | DISTINCT expression1 [, expression2]*)	默认情况和 ALL 时，返回 expression 表达式筛选后，非 NULL 值的输入行数。如果是 DISTINCT，则会先对数据进行去重，然后再进行统计总行数。
COUNT(*) COUNT(1)	返回输入的总行数，含 NULL 值。
AVG([ALL | DISTINCT] expression)	默认情况和 ALL 时，返回 expression 表达式筛选后，所有输入的算术平均值。如果是 DISTINCT，则会先对数据进行去重，然后再进行统计求平均。
SUM([ALL | DISTINCT] expression)	默认情况和 ALL 时，返回 expression 表达式筛选后，所有输入和。如果是 DISTINCT，则会先对数据进行去重，然后再进行统计求和。
MAX([ALL | DISTINCT] expression)	默认情况和 ALL 时，返回 expression 表达式筛选后，所有输入的最大值（不可用于 TIMESTAMP 类型）。如果是 DISTINCT，则会先对数据进行去重，然后再进行统计求最大值。
MIN([ALL | DISTINCT] expression)	默认情况和 ALL 时，返回 expression 表达式筛选后，所有输入的最小值（不可用于 TIMESTAMP 类型）。如果是 DISTINCT，则会先对数据进行去重，然后再进行统计求最小值。
STDDEV_POP([ALL | DISTINCT] expression)	默认情况和 ALL 时，返回 expression 表达式筛选后，所有输入的总体标准差。如果是 DISTINCT，则会先对数据进行去重，然后再进行统计求总体标准差。
STDDEV_SAMP([ALL | DISTINCT] expression)	默认情况和 ALL 时，返回 expression 表达式筛选后，所有输入的样本标准差。如果是 DISTINCT，则会先对数据进行去重，然后再进行统计求样本标准差。
VAR_POP([ALL | DISTINCT] expression)	默认情况和 ALL 时，返回 expression 表达式筛选后，所有输入的总体方差。如果是 DISTINCT，则会先对数据进行去重，然后再进行统计求总体方差。
VAR_SAMP([ALL | DISTINCT] expression) VARIANCE([ALL | DISTINCT] expression)	默认情况和 ALL 时，返回 expression 表达式筛选后，所有输入的样本方差。如果是 DISTINCT，则会先对数据进行去重，然后再进行统计求样本方差。两种写法等价。
COLLECT([ALL | DISTINCT] expression)	默认情况和 ALL 时，返回 expression 表达式筛选后，所有输入的非 NULL 输入的 MULTISET 集合（允许重复值）。如果所有值都是 NULL，则返回一个空集。
RANK()	返回某个数据在一组数据中的排名，前后调用的结果可能不连续。例如有五个数据，其中两个并列第二，那么 RANK() 的结果是1、2、2、4、5。
DENSE_RANK()	返回某个数据在一组数据中的排名，前后调用的结果保证连续。例如有五个数据，其中两个并列第二，那么 RANK() 的结果是1、2、2、3、4。
ROW_NUMBER()	为一组数据的每行分配一个递增且连续的值，从1开始，不会重复。例如有五个数据，其中两个并列第二，那么 RANK() 的结果是1、2、3、4、5。
LEAD(expression [, offset] [, default])	在窗口计算中，访问当前行之后 offset 行的数据，默认 offset 为1，即访问下一行的数据。default 表示无数据时的默认值，如果不提供，默认为 NULL。
LAG(expression [, offset] [, default])	在窗口计算中，访问当前行之前 offset 行的数据，默认 offset 为1，即访问上一行的数据。default 表示无数据时的默认值，如果不提供，默认为 NULL。
FIRST_VALUE(expression)	返回一系列数据中，第一个数据。
LAST_VALUE(expression)	返回一系列数据中，最后一个数据。
LISTAGG(expression [, separator])	将一组数据使用给定的分隔符进行连接，最终返回一个连接后的字符串。默认分隔符是半角逗号 `,` 。类似于其他语言的 String.join() 方法。

示例

为方便演示，建立示例测试数据表 Test：

id	site_id	count	date
1	1	45	2021-07-10
2	3	100	2021-07-13
3	1	230	2021-07-14
4	2	10	2021-07-14
5	5	205	2021-07-14
6	4	13	2021-07-15
7	3	220	2021-07-15
8	5	545	2021-07-16
9	3	201	2021-07-17

COUNT

功能描述：统计总行数。
语法：COUNT([ALL] expression | DISTINCT expression1 [, expression2] *)
示例测试语句： `SELECT  COUNT(*) FROM Test;` **测试结果：**

测试语句	测试结果（nums）
SELECT COUNT(*) AS nums FROM Test;	9
SELECT COUNT(DISTINCT site_id) AS nums FROM Test;	5

AVG

功能描述： 统计求平均。
语法：AVG([ALL | DISTINCT] expression)
示例测试语句： SELECT AVG(count) FROM Test;
测试数据和结果：

测试语句	测试结果（nums）
SELECT AVG(count) AS nums FROM Test;	176.3

SUM

功能描述：统计求和。
语法： SUM([ALL | DISTINCT] expression)
示例测试语句： SELECT SUM(count) FROM Test;
测试数据和结果：

测试语句	测试结果（nums）
SELECT SUM(count) AS nums FROM Test;	1569

MAX

功能描述：统计求最大值。
语法： MAX([ALL | DISTINCT] expression)
示例测试语句： SELECT MAX(count) FROM Test;
测试数据和结果：

测试语句	测试结果（nums）
SELECT MAX(count) AS nums FROM Test;	545

MIN

功能描述：统计求最小值。
语法： MIN([ALL | DISTINCT] expression)
示例测试语句： SELECT MIN(count) FROM Test;
测试数据和结果：

测试语句	测试结果（nums）
SELECT MIN(count) AS nums FROM Test;	13

STDDEV_POP

功能描述：统计求总体标准差。
语法： STDDEV_POP([ALL | DISTINCT] expression)
示例测试语句： SELECT STDDEV_POP(count) FROM Test;
测试数据和结果：

测试语句	测试结果（pop）
SELECT STDDEV_POP(count) AS pop FROM Test;	156.18

VAR_POP

功能描述：统计求总体方差。
语法： VAR_POP([ALL | DISTINCT] expression)
示例测试语句： SELECT VAR_POP(count) FROM Test;
测试数据和结果：

测试语句	测试结果（pop）
SELECT VAR_POP(count) AS pop FROM Test;	24390.7

FIRST_VALUE

功能描述：返回一系列数据中，第一个数据。
语法： FIRST_VALUE(expression)
示例测试语句： SELECT FIRST_VALUE(count) FROM Test;
测试数据和结果：

测试语句	测试结果（first）
SELECT FIRST_VALUE(count) AS first FROM Test;	45

LAST_VALUE

功能描述：返回一系列数据中，最后一个数据。
语法： LAST_VALUE(expression)
示例测试语句： SELECT LAST_VALUE(count) FROM Test;
测试数据和结果：

测试语句	测试结果（last）
SELECT LAST_VALUE(count) AS last FROM Test;	201

版权所有：腾讯云计算（北京）有限责任公司

时间窗口函数

最近更新时间：2024-10-11 15:19:01

在流式计算中，流通常是无穷无尽的，我们无法知道什么时候数据源会继续/停止发送数据，所以在流上处理聚合事件（count、sum 等）的处理方式与批处理中的处理方式会有所差异。在流上一般用窗口（Windows）来限定聚合的范围，例如“过去2分钟网站点击量的计数”、“在最近100个人中点赞这个视频的总人数”。窗口的概念相当于帮我们收集了一张有限数据的动态表，我们可以对表中的数据进行聚合计算。
窗口函数是一种特殊的函数，它并不在 SELECT 的投影列表中使用，而是在 GROUP BY 子句中使用。流计算 Oceanus 支持三种类型的窗口函数 TUMBLE、HOP 和 SESSION。
Flink 流处理介绍详见 及时流处理 。

TUMBLE WINDOW

TUMBLE WINDOW（滚动窗口）将每个进入的数据分配到一个指定窗口大小的窗口中。滚动窗口可以自定义固定的大小，并且不会出现重叠。我们可以对窗口内的数据进行计算。

语法

```
TUMBLE(time_attr, interval)
```

- time_attr 参数表示时间戳字段，表示每条记录被处理的时间戳。**如果指定为 PROCTIME 是自动生成的时间戳，记录了数据被 flink 处理的时刻，一般用在 Processing Time 模式下。**
- interval 参数用来设置窗口大小。例如，设置为1天：`INTERVAL '1' DAY`；设置为2小时：`INTERVAL '2' HOUR`，其他用法可参见 时间相关函数 。

> **注意**
> - 如果在 Event Time 时间模式下（使用 WATERMARK FOR 语句定义了时间戳字段），那么 TUMBLE、HOP、SESSION 窗口函数的第一个参数必须为该字段。
> - 如果在 Processing Time 时间模式下，则 TUMBLE、HOP、SESSION 窗口函数的第一个参数必须为 proctime() 函数生成的计算列，下文用 PROCTIME 举例，请在实际作业中替换为实际的列名。

标识函数

标识函数用来表示窗口的起始与结束时间。

函数名	功能描述
TUMBLE_START(time-attr, size-interval)	返回该窗口的起始时间
TUMBLE_END(time-attr, size-interval)	返回该窗口的结束时间

模拟用例

下文以 TUMBLE WINDOW 为例，帮助您更容易地理解 TUMBLE WINDOW。使用 Event Time 模拟统计每小时各用户收入金额。
示例数据：

username（VARCHAR）	income(BIGINT)	times(TIMESTAMP)
Tom	20	2021-11-11 10:30:00.0
Jack	10	2021-11-11 10:35:00.0
Tom	10	2021-11-11 10:35:00.0
Tom	10	2021-11-11 10:40:00.0
Tom	15	2021-11-11 11:30:00.0
Jack	10	2021-11-11 11:30:00.0

Jack	15	2021-11-11 11:40:00.0

SQL 语句：

```
CREATE TABLE user_income (
      username VARCHAR,
      Income INT,
      times TIMESTAMP(3),
      WATERMARK FOR times AS times - INTERVAL '3' SECOND
) WITH (
      ...
  );

CREATE TABLE output (
            win_start TIMESTAMP,
            win_end TIMESTAMP,
            username VARCHAR,
            hour_income BIGINT
)WITH(
      ...
);

INSERT INTO output
SELECT
TUMBLE_START(times,INTERVAL '1' HOUR),
TUMBLE_END(times,INTERVAL '1' HOUR),
username,
SUM(Income)
FROM user_income
GROUP BY TUMBLE(times,INTERVAL '1' HOUR),username;
```

输出结果：

win_start(TIMESTAMP)	win_end(TIMESTAMP)	username(VARCHAR)	hour_income(BIGINT)
2021-11-11 10:00:00.0	2021-11-11 11:00:00.0	Tom	40
2021-11-11 10:00:00.0	2021-11-11 11:00:00.0	Jack	10
2021-11-11 11:00:00.0	2021-11-11 12:00:00.0	Tom	15
2021-11-11 11:00:00.0	2021-11-11 12:00:00.0	Jack	25

HOP WINDOW

HOP WINDOW（滑动窗口）将元素分配到固定长度的窗口中，与滚动窗口类似，窗口的大小由窗口大小参数来配置，另一个窗口滑动参数控制滑动窗口开始的频率。
HOP WINDOW（滑动窗口）保持窗口大小（Size）不变，每次滑动指定的时间周期（Slide），因而允许窗口之间的相互重叠。
Slide 的大小决定了 Flink 创建新窗口的频率。

- 当 Slide 小于 Size 时（如图 window1 与 window2），相邻窗口会重叠，一个时间会被分配到多个窗口。
- 当 Slide 大于 Size 时（如图 window1 与 window4），可能会导致有些事件被丢弃。
- 当 Slide 等于 Size 时（如图 window1 与 window3），等于是 TUMBLE WINDOW。

语法

```
HOP(time_attr, sliding_interval, window_size_interval)
```

- time_attr 参数表示时间戳字段，表示每条记录被处理的时间戳。如果指定为 PROCTIME 是自动生成的时间戳，记录了数据被 flink 处理的时刻，一般用在 Processing Time 模式下。
- window_size_interval 参数用来设置窗口大小。例如，设置为1天：`INTERVAL '1' DAY`；设置为2小时：`INTERVAL '2' HOUR`，其他用法可参见 时间相关函数 。
- sliding_interval 参数用来设置滑动时间周期大小。例如，设置为1天：`INTERVAL '1' DAY`；设置为2小时：`INTERVAL '2' HOUR`，其他用法可参见 时间相关函数 。

标识函数

标识函数用来表示窗口的起始与结束时间。

函数名	功能描述
HOP_START(time-attr, slide-interval,size-interval)	返回该窗口的起始时间
HOP_END(time-attr, slide-interval,size-interval)	返回该窗口的结束时间

模拟用例

下文以 HOP WINDOW 为例，帮助您更容易地理解 HOP WINDOW。使用 Event Time 模拟统计每小时各用户收入金额，每30分钟更新一次。**1小时的窗口，10分钟滑动一次。**
样例数据：

username (VARCHAR)	income(BIGINT)	times(TIMESTAMP)
Tom	20	2021-11-11 10:30:00.0
Jack	10	2021-11-11 10:35:00.0
Tom	10	2021-11-11 10:35:00.0
Tom	10	2021-11-11 10:40:00.0
Tom	15	2021-11-11 11:35:00.0
Jack	10	2021-11-11 11:30:00.0
Jack	15	2021-11-11 11:40:00.0

SQL 语句：

```
CREATE TABLE user_income (
      username VARCHAR,
      Income INT,
      times TIMESTAMP(3),
      WATERMARK FOR times AS times - INTERVAL '3' MINUTE
)WITH(
      ...
  );

CREATE TABLE output (
            win_start TIMESTAMP,
            win_end TIMESTAMP,
            username VARCHAR,
            hour_income BIGINT
)WITH(
      ...
  );

INSERT INTO output
SELECT
```

```
HOP_START(times,INTERVAL '30' MINUTE,INTERVAL '1' HOUR),
HOP_END(times,INTERVAL '30' MINUTE,INTERVAL '1' HOUR),
username,
SUM(income)
FROM user_income
GROUP BY HOP(times,INTERVAL '30' MINUTE,INTERVAL '1' HOUR),username;
```

输出结果：

win_start(TIMESTAMP)	win_end(TIMESTAMP)	username(VARCHAR)	hour_income(BIGINT)
2021-11-11 10:00:00.0	2021-11-11 11:00:00.0	Tom	40
2021-11-11 10:00:00.0	2021-11-11 11:00:00.0	Jack	10
2021-11-11 10:30:00.0	2021-11-11 11:30:00.0	Jack	10
2021-11-11 10:30:00.0	2021-11-11 11:30:00.0	Tom	40
2021-11-11 11:00:00.0	2021-11-11 12:00:00.0	Tom	15
2021-11-11 11:00:00.0	2021-11-11 12:00:00.0	Jack	25
2021-11-11 11:30:00.0	2021-11-11 12:30:00.0	Jack	25
2021-11-11 11:30:00.0	2021-11-11 12:30:00.0	Tom	15

SESSION WINDOW

SESSION WINDOW（会话窗口）通过 session 活动对元素进行分组，session 窗口与滚动窗口和滑动窗口相比，不会有重叠和固定的开始时间和结束时间的情况，相反，当它在一个固定的时间周期内不再收到元素，即非活动间隔产生，那么这个窗口就会关闭。一个 session 窗口通过一个 session 间隔来配置。这个 session 间隔定义了非活跃周期的长度，当这个非活跃周期产生，那么当前的 session 将关闭并且后续的元素将被分配到新的 session 窗口中。
Session Window 并非以长度来划分窗口，而是以非活跃时间来划分。例如超过30分钟不活跃（没有新数据），则之前的窗口结束，下一个来到的数据将会形成一个新窗口。

语法

```
SESSION(time_attr, interval)
```

- time_attr 参数表示时间戳字段，表示每条记录被处理的时间戳。如果指定为 PROCTIME 是自动生成的时间戳，记录了数据被 flink 处理的时刻，一般用在 Processing Time 模式下。
- interval 参数用来设置窗口大小。例如，设置为1天：`INTERVAL '1' DAY`；设置为2小时：INTERVAL '2' HOUR，其他用法可参见 时间相关函数 。

标识函数

标识函数用来表示窗口的起始与结束时间。

函数名	功能描述
SESSION_START(time-attr, size-interval)	返回该窗口的起始时间
SESSION_END(time-attr, size-interval)	返回该窗口的结束时间

模拟用例

下文以 SESSION WINDOW 为例，帮助您更容易地理解 SESSION WINDOW。使用 Event Time 模拟统计每小时各用户收入金额，会话超时时长为30分钟。
样例数据：

username（VARCHAR）	income(BIGINT)	times(TIMESTAMP)
Tom	20	2021-11-11 10:30:00.0
Jack	10	2021-11-11 10:35:00.0
Tom	10	2021-11-11 10:35:00.0
Tom	10	2021-11-11 10:40:00.0
Tom	15	2021-11-11 11:50:00.0
Jack	10	2021-11-11 11:40:00.0
Jack	15	2021-11-11 11:45:00.0

SQL 语句：

```
CREATE TABLE user_income (
      username VARCHAR,
      Income INT,
      times TIMESTAMP(3),
      WATERMARK FOR times AS times - INTERVAL '3' MINUTE
)WITH(
      ...
  );

CREATE TABLE output (
            win_start TIMESTAMP,
            win_end TIMESTAMP,
            username VARCHAR,
            hour_income BIGINT
)WITH(
      ...
  );

INSERT INTO output
SELECT
SESSION_START(times,INTERVAL '30' MINUTE),
SESSION_END(times,INTERVAL '30' MINUTE),
username,
SUM(Income)
FROM user_income
GROUP BY SESSION(times,INTERVAL '30' MINUTE),username;
```

输出结果：

win_start(TIMESTAMP)	win_end(TIMESTAMP)	username(VARCHAR)	hour_income(BIGINT)
2021-11-11 10:30:00.0	2021-11-11 11:10:00.0	Tom	40
2021-11-11 10:35:00.0	2021-11-11 11:05:00.0	Jack	10
2021-11-11 11:30:00.0	2021-11-11 12:00:00.0	Tom	15
2021-11-11 11:30:00.0	2021-11-11 12:10:00.0	Jack	25

更多说明

以上三种窗口都有对应的辅助函数。以 TUMBLE 窗口为例（HOP、SESSION 也一样，只是前缀不同），辅助函数如下：

- **TUMBLE_ROWTIME：**表示 TUMBLE 窗口的末端界限（包含，可用作 JOIN 或 GROUP 以及 OVER 条件，Event Time 时间模式下使用）。示例如下：

```
SELECT user,
TUMBLE_START(rowtime, INTERVAL '12' HOUR) AS sStart,
TUMBLE_ROWTIME(rowtime, INTERVAL '12' HOUR) AS snd,
SUM(amount)
FROM Orders
GROUP BY TUMBLE(rowtime, INTERVAL '12' HOUR), user
```

- **TUMBLE_PROCTIME：**表示 TUMBLE 窗口的末端界限（包含，可用作 JOIN 或 GROUP 以及 OVER 条件，Processing Time 时间模式下使用）。示例如下：

```
SELECT user,
TUMBLE_START(PROCTIME, INTERVAL '12' HOUR) AS sStart,
TUMBLE_PROCTIME(PROCTIME, INTERVAL '12' HOUR) AS snd,
SUM(amount)
FROM Orders
GROUP BY TUMBLE(PROCTIME, INTERVAL '12' HOUR), user
```

版权所有：腾讯云计算（北京）有限责任公司

其他函数

最近更新时间：2023-06-21 15:21:58

散列函数

散列函数的函数名和功能描述如下：

函数名	功能描述
MD5(string)	返回字符串的 MD5 值（32位十六进制数字组成的字符串）。如果输入为 NULL，则返回 NULL。
SHA1(string)	返回字符串的 SHA1 值（40位十六进制数字组成的字符串）。如果输入为 NULL，则返回 NULL。
SHA256(string)	返回字符串的 SHA256 值（64位十六进制数字组成的字符串）。如果输入为 NULL，则返回 NULL。
SHA224(string)	返回字符串的 SHA224 值（56位十六进制数字组成的字符串）。如果输入为 NULL，则返回 NULL。
SHA384(string)	返回字符串的 SHA384 值（96位十六进制数字组成的字符串）。如果输入为 NULL，则返回 NULL。
SHA512(string)	返回字符串的 SHA512 值（128位十六进制数字组成的字符串）。如果输入为 NULL，则返回 NULL。
SHA2(string, hashLength)	通用 SHA-2 系列函数，返回 string 字符串的哈希值，hashLength 为256时，等价于 SHA256(string)，其他的以此类推。

值访问函数

值访问函数的函数名和功能描述如下：

函数名	功能描述
tableName.compositeType.field	访问复合类型（Tuple、POJO）等的字段。
tableName.compositeType.*	访问 Tuple 或 POJO 的所有字段。

值构造函数

值构造函数的函数名和功能描述如下：

函数名	功能描述
(value, [, value]*) ROW(value, [, value]*)	创建一个包含若干值的行。两种写法完全等价。
ARRAY '[' value [, value]* ']'	创建一个包含若干值的数组。
MAP '[' key, value [, key, value]* ']'	创建一个包含若干键值对的映射。

集合函数

集合函数包括数组（Array）和键值对映射（Map）的操作，函数名和功能描述如下：

函数名	功能描述
CARDINALITY(array)	返回某个数组的长度。
array '[' index ']'	返回某个数组的指定位置的项（下标从1开始）。
ELEMENT(array)	返回单元素数组的内容（如果数组为空则返回 NULL；如果数组存放的元素大于一个，则抛出异常）。
CARDINALITY(map)	返回 map 中键值对的总个数。
map '[' key ']'	返回 map 中指定 key 所对应的值。

版权所有：腾讯云计算（北京）有限责任公司

分组函数

分组函数用来做 GROUP BY 分组后的表示，函数名和功能描述如下：

函数名	功能描述
GROUP_ID()	返回一个唯一标识该 GROUP 所有键的整数。
GROUPING(expression1 [, expression2]*) GROUPING_ID(expression1 [, expression2]*)	返回指定分组表达式的分组 ID（二进制矢量转为十进制数）。

标识符与保留字
命名规则

最近更新时间：2023-06-21 15:21:59

标识符用来唯一地表示表名或列名。当前流计算 Oceanus 中标识符的命名规则如下：

- 不能使用 数据类型 中所指定的保留字。若必须使用这些保留字作为表名或列名，请使用反引号（``）将其包括，例如 `time`。
- 请不要使用 `PROCTIME` 和 `SOURCETIME` 作为列名，以避免与系统自动生成的时间戳发生冲突。
- 不能以 `_DataStreamTable_` 开头。
- 如果表名或列名含有空格或者特殊字符，需使用反引号将其括起来，例如 `HELLO WORLD`。
- 长度必须小于等于128个英文（半角）字符（一个汉字或全角字符等价于两个英文字符）。

保留字

最近更新时间：2023-09-26 15:04:54

流计算 Oceanus 内部的保留字如下所示。在表名或列名中使用这些保留字时，必须使用反引号（``）括起来，否则语法检查时会报错。

> **注意**
> 在 Processing Time 时间模式下（即未使用 WATERMARK FOR 来定义数据源的时间戳字段），请不要使用 PROCTIME 作为列名，以免与系统自动生成的时间戳发生命名冲突。

A

```
A, ABS, ABSOLUTE, ACTION, ADA, ADD, ADMIN, AFTER, ALL, ALLOCATE, ALLOW,
ALTER, ALWAYS, AND, ANY, ARE, ARRAY, AS, ASC, ASENSITIVE, ASSERTION,
ASSIGNMENT, ASYMMETRIC, AT, ATOMIC, ATTRIBUTE, ATTRIBUTES, AUTHORIZATION, AVG
```

B

```
BEFORE, BEGIN, BERNOULLI, BETWEEN, BIGINT, BINARY, BIT, BLOB, BOOLEAN, BOTH BREADTH, BY
```

C

```
C, CALL, CALLED, CARDINALITY, CASCADE, CASCADED, CASE, CAST, CATALOG, CATALOG_NAME,
CEIL, CEILING, CENTURY, CHAIN, CHAR, CHARACTER, CHARACTERISTICTS, CHARACTERS,
CHARACTER_LENGTH, CHARACTER_SET_CATALOG, CHARACTER_SET_NAME, CHARACTER_SET_SCHEMA,
CHAR_LENGTH, CHECK, CLASS_ORIGIN, CLOB, CLOSE, COALESCE, COBOL, COLLATE, COLLATION,
COLLATION_CATALOG, COLLATION_NAME, COLLATION_SCHEMA, COLLECT, COLUMN, COLUMN_NAME,
COMMAND_FUNCTION, COMMAND_FUNCTION_CODE, COMMIT, COMMITTED,CONDITION, CONDITION_NUMBER,
CONNECT, CONNECTION, CONNECTION_NAME, CONSTRAINT, CONSTRAINTS, CONSTRAINT_CATALOG,
CONSTRAINT_NAME, CONSTRAINT_SCHEMA, CONSTRUCTOR, CONTAINS, CONTINUE, CONVERT, CORR,
CORRESPONDING, COUNT, COVAR_POP, COVAR_SAMP, CREATE, CROSS, CUBE, CUME_DIST, CURRENT,
CURRENT_CATALOG, CURRENT_DATE, CURRENT_DEFAULT_TRANSFORM_GROUP, CURRENT_PATH, CURRENT_ROLE,
CURRENT_SCHEMA, CURRENT_TIME, CURRENT_TIMESTAMP, CURRENT_TRANSFORM_GROUP_FOR_TYPE,
CURRENT_USER, CURSOR, CURSOR_NAME, CYCLE, DATA, DATABASE, DATE, DATETIME_INTERVAL_CODE
```

D

```
DATETIME_INTERVAL_PRECISION, DAY, DEALLOCATE, DEC, DECADE, DECIMAL, DECLARE, DEFAULT,
DEFAULTS, DEFERRABLE, DEFERRED, DEFINED, DEFINER, DEGREE, DELETE, DENSE_RANK, DEPTH, DEREF,
DERIVED, DESC, DESCRIBE, DESCRIPTION, DESCRIPTOR, DETERMINISTIC, DIAGNOSTICS, DISALLOW, DISCONNECT,
DISPATCH, DISTINCT, DOMAIN, DOUBLE, DOW, DOY, DROP, DYNAMIC, DYNAMIC_FUNCTION, DYNAMIC_FUNCTION_CODE
```

E

```
EACH, ELEMENT, ELSE, END, END-EXEC, EPOCH, EQUALS, ESCAPE, EVERY, EXCEPT, EXCEPTION,
EXCLUDE, EXCLUDING, EXEC, EXECUTE, EXISTS, EXP, EXPLAIN, EXTEND, EXTERNAL, EXTRACT
```

F

```
FALSE, FETCH, FILTER, FINAL, FIRST, FIRST_VALUE, FLOAT, FLOOR, FOLLOWING, FOR, FOREIGN,
FORTRAN, FOUND, FRAC_SECOND, FREE, FROM, FULL, FUNCTION, FUSION
```

G

```
G, GENERAL, GENERATED, GET, GLOBAL, GO, GOTO, GRANT, GRANTED, GROUP, GROUPING
```

H

```
HAVING, HIERARCHY, HOLD, HOUR
```

I

```
IDENTITY, IMMEDIATE, IMPLEMENTATION, IMPORT, IN, INCLUDING, INCREMENT, INDICATOR,
INITIALLY, INNER, INOUT, INPUT, INSENSITIVE, INSERT, INSTANCE, INSTANTIABLE, INT,
INTEGER, INTERSECT, INTERSECTION, INTERVAL, INTO, INVOKER, IS, ISOLATION
```

J

```
JAVA, JOIN
```

K

```
K, KEY, KEY_MEMBER, KEY_TYPE
```

L

```
LABEL, LANGUAGE, LARGE, LAST, LAST_VALUE, LATERAL, LEADING, LEFT, LENGTH, LEVEL,
LIBRARY, LIKE, LIMIT, LN, LOCAL, LOCALTIME, LOCALTIMESTAMP, LOCATOR, LOWER
```

M

```
M, MAP, MATCH, MATCHED, MAX, MAXVALUE, MEMBER, MERGE, MESSAGE_LENGTH, MESSAGE_OCTET_LENGTH,
MESSAGE_TEXT, METHOD, MICROSECOND, MILLENNIUM, MIN, MINUTE, MINVALUE, MOD, MODIFIES, MODULE,
MONTH, MORE, MULTISET, MUMPS
```

N

```
NAME, NAMES, NATIONAL, NATURAL, NCHAR, NCLOB, NESTING, NEW, NEXT, NO, NONE, NORMALIZE,
NORMALIZED, NOT, NULL, NULLABLE, NULLIF, NULLS, NUMBER, NUMERIC
```

O

```
OBJECT, OCTETS, OCTET_LENGTH, OF, OFFSET, OLD, ON, ONLY, OPEN, OPTION, OPTIONS, OR,
ORDER, ORDERING, ORDINALITY, OTHERS, OUT, OUTER, OUTPUT, OVER, OVERLAPS, OVERLAY, OVERRIDING
```

P

```
PAD, PARAMETER, PARAMETER_MODE, PARAMETER_NAME, PARAMETER_ORDINAL_POSITION,
PARAMETER_SPECIFIC_CATALOG, PARAMETER_SPECIFIC_NAME, PARAMETER_SPECIFIC_SCHEMA, PARTIAL,
PARTITION, PASCAL, PASSTHROUGH, PATH, PERCENTILE_CONT, PERCENTILE_DISC, PERCENT_RANK, PLACING,
PLAN, PLI, POSITION, POWER, PRECEDING, PRECISION, PREPARE, PRESERVE, PRIMARY, PRIOR, PRIVILEGES,
PROCEDURE, PUBLIC
```

Q

```
QUARTER
```

R

```
RANGE, RANK, READ, READS, REAL, RECURSIVE, REF, REFERENCES, REFERENCING, REGR_AVGX,
REGR_AVGY, REGR_COUNT, REGR_INTERCEPT, REGR_R2, REGR_SLOPE, REGR_SXX, REGR_SXY, REGR_SYY,
RELATIVE, RELEASE, REPEATABLE, RESET, RESTART, RESTRICT, RESULT, RETURN, RETURNED_CARDINALITY,
RETURNED_LENGTH, RETURNED_OCTET_LENGTH, RETURNED_SQLSTATE, RETURNS, REVOKE, RIGHT, ROLE, ROLLBACK,
ROLLUP, ROUTINE, ROUTINE_CATALOG, ROUTINE_NAME, ROUTINE_SCHEMA, ROW, ROWS, ROW_COUNT, ROW_NUMBER
```

S

```
SAVEPOINT, SCALE, SCHEMA, SCHEMA_NAME, SCOPE, SCOPE_CATALOGS, SCOPE_NAME, SCOPE_SCHEMA,
SCROLL, SEARCH, SECOND, SECTION, SECURITY, SELECT, SELF, SENSITIVE, SEQUENCE, SERIALIZABLE,
SERVER, SERVER_NAME, SESSION, SESSION_USER, SET, SETS, SIMILAR, SIMPLE, SIZE, SMALLINT,
SOME, SOURCE, SPACE, SPECIFIC, SPECIFICTYPE, SPECIFIC_NAME, SQL, SQLEXCEPTION, SQLSTATE,
SQLWARNING, SQL_TSI_DAY, SQL_TSI_FRAC_SECOND, SQL_TSI_HOUR, SQL_TSI_MICROSECOND, SQL_TSI_MINUTE,
SQL_TSI_MONTH, SQL_TSI_QUARTER, SQL_TSI_SECOND, SQL_TSI_WEEK, SQL_TSI_YEAR, SQRT, START,
STATE, STATEMENT, STATIC, STDDEV_POP, STDDEV_SAMP, STREAM, STRUCTURE, STYLE, SUBCLASS_ORIGIN,
SUBMULTISET, SUBSTITUTE, SUBSTRING, SUM, SYMMETRIC, SYSTEM, SYSTEM_USER
```

T

```
TABLE, TABLESAMPLE, TABLE_NAME, TEMPORARY, THEN, TIES, TIME, TIMESTAMP, TIMESTAMPADD,
TIMESTAMPDIFF, TIMEZONE_HOUR, TIMEZONE_MINUTE, TINYINT, TO, TOP_LEVEL_COUNT, TRAILING,
TRANSACTION, TRANSACTIONS_ACTIVE, TRANSACTIONS_COMMITTED, TRANSACTIONS_ROLLED_BACK,
TRANSFORM, TRANSFORMS, TRANSLATE, TRANSLATION, TREAT, TRIGGER, TRIGGER_CATALOG,
TRIGGER_NAME, TRIGGER_SCHEMA, TRIM, TRUE, TYPE
```

U

```
UESCAPE, UNBOUNDED, UNCOMMITTED, UNDER, UNION, UNIQUE, UNKNOWN, UNNAMED, UNNEST,
UPDATE, UPPER, UPSERT, USAGE, USER, USER_DEFINED_TYPE_CATALOG, USER_DEFINED_TYPE_CODE,
USER_DEFINED_TYPE_NAME, USER_DEFINED_TYPE_SCHEMA, USING
```

V

```
VALUE, VALUES, VARBINARY, VARCHAR, VARYING, VAR_POP, VAR_SAMP, VERSION, VIEW
```

W

```
WATERMARK, WEEK, WHEN, WHENEVER, WHERE, WIDTH_BUCKET, WINDOW, WITH, WITHIN, WITHOUT,
WORK, WRAPPER, WRITE
```

X

```
XML
```

Y

```
YEAR
```

Z

```
ZONE
```