阿里云
实时计算（流计算）

Flink SQL开发指南

文档版本：20190918

法律声明

阿里云提醒您在阅读或使用本文档之前仔细阅读、充分理解本法律声明各条款的内容。如果您阅读或使用本文档，您的阅读或使用行为将被视为对本声明全部内容的认可。

1. 您应当通过阿里云网站或阿里云提供的其他授权通道下载、获取本文档，且仅能用于自身的合法合规的业务活动。本文档的内容视为阿里云的保密信息，您应当严格遵守保密义务；未经阿里云事先书面同意，您不得向任何第三方披露本手册内容或提供给任何第三方使用。
2. 未经阿里云事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。
3. 由于产品版本升级、调整或其他原因，本文档内容有可能变更。阿里云保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在阿里云授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过阿里云授权渠道下载、获取最新版的用户文档。
4. 本文档仅作为用户使用阿里云产品及服务的参考性指引，阿里云以产品及服务的”现状“、“有缺陷”和“当前功能”的状态提供本文档。阿里云在现有技术的基础上尽最大努力提供相应的介绍及操作指引，但阿里云在此明确声明对本文档内容的准确性、完整性、适用性、可靠性等不作任何明示或暗示的保证。任何单位、公司或个人因为下载、使用或信赖本文档而发生任何差错或经济损失的，阿里云不承担任何法律责任。在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性或刑罚性的损害，包括用户使用或信赖本文档而遭受的利润损失，承担责任（即使阿里云已被告知该等损失的可能性）。
5. 阿里云网站上所有内容，包括但不限于著作、产品、图片、档案、资讯、资料、网站架构、网站画面的安排、网页设计，均由阿里云和/或其关联公司依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经阿里云和/或其关联公司书面同意，任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表阿里云网站、产品程序或内容。此外，未经阿里云事先书面同意，任何人不得为了任何营销、广告、促销或其他目的使用、公布或复制阿里云的名称（包括但不限于单独为或以组合形式包含”阿里云”、Aliyun”、“万网”等阿里云和/或其关联公司品牌，上述品牌的附属标志及图案或任何类似公司名称、商号、商标、产品或服务名称、域名、图案标示、标志、标识或通过特定描述使第三方能够识别阿里云和/或其关联公司）。
6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

通用约定

格式	说明	样例
	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	 **禁止:** 重置操作将丢失用户配置数据。
	该类警示信息可能导致系统重大变更甚至故障，或者导致人身伤害等结果。	 **警告:** 重启操作将导致业务中断，恢复业务所需时间约10分钟。
	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	 **说明:** 您也可以通过按Ctrl + A选中全部文件。
>	多级菜单递进。	设置 > 网络 > 设置网络类型
粗体	表示按键、菜单、页面名称等UI元素。	单击 确定。
`courier` 字体	命令。	执行 `cd /d C:/windows` 命令，进入Windows系统文件夹。
*## *	表示参数、变量。	`bae log list --instanceid` *`Instance_ID`*
[]或者[a|b]	表示可选项，至多选择一个。	`ipconfig` *`[-all|-t]`*
{}或者{a|b}	表示必选项，至多选择一个。	`swich` *`{stand | slave}`*

目录

1 Flink SQL开发指南概述

阿里实时计算开发平台为实时计算Flink SQL作业提供了存储管理、作业开发、作业调试、运维管理、监控报警和配置调优功能。

Flink SQL开发指南主要包含以下内容：

- 数据存储

  实时计算开发平台提供了实时计算上下游存储设备（例如，RDS、DataHub、OTS等）管理功能。通过存储注册方式引入的上下存储设备，可以实现数据预览、数据抽样以及DDL生成功能。数据存储详情请参见#unique_4。

  > 说明：
  > - 若共享模式集群需访问阿里云VPC网络下的存储资源，请参见#unique_5。
  > - 若实时计算访问的上下游存储存在白名单机制，请参见#unique_6。

- 作业开发

  作业开发章节为您介绍Flink SQL作业的#unique_7、#unique_8 和#unique_9流程。

- 作业调试

  作业调试章节为您介绍Flink SQL作业的调试功能，包括#unique_10和#unique_11。

- 作业运维

  作业运维章节为您介绍#unique_12、#unique_13 和#unique_14等实时计算作业运维内容。

- 监控报警

  监控报警章节为您介绍如何创建和启动报警规则，详情请参见#unique_15。

- 配置调优

  配置调优章节为您介绍Flink SQL作业的调优功能，主要包括#unique_16、#unique_17、#unique_18和#unique_19。

- Flink SQL

  Flink SQL章节为您介绍Flink SQL语法，详情请参见Flink SQL概述。

2 数据存储

2.1 数据存储概述

阿里云实时计算集成了RDS、DataHub、OTS等数据存储系统的管理界面，为您提供一站式云上数据存储管理服务。

数据存储的含义

数据存储有2层含义：

- 实时计算产品上下游生态对应的数据存储系统/数据库表（以下简称为存储资源，详情请参见产品生态）。
- 实时计算产品对上下游存储资源的管理功能（以下简称为数据存储功能）。

说明：

使用注册存储功能前，请先完成实时计算对存储设备的访问授权，授权方法请您参见#unique_23和#unique_24。

实时计算产品支持2种引用上下游存储资源的方式：明文方式和存储注册方式。详细介绍如下。

明文方式

明文方式是通过在作业的DDL语句WITH参数中配置accessId和accessKey的方法，引用上下游存储资源，详情请参见#unique_25。明文方式不仅支持同账号（包括主子账号）授权，同时还支持跨账号授权。例如，当前实时计算A用户(包括A下所属的子账户)需要使用用户B的存储资源，可以通过如下明文方式定义DDL。

```
CREATE TABLE in_stream(
  a varchar,
  b varchar,
  c timestamp
)with(
type='datahub',
  endPoint='http://dh-cn-hangzhou.aliyuncs.com',
  project='<dataHubProjectName>',
  topic='<dataHubTopicName>',
  accessId='<accessIdOfUserB>',
  accessKey='<accessKeyOfUserB>'
```

```
);
```

存储注册方式

存储注册方式是将上下游存储资源预先注册至实时计算开发平台，然后通过实时计算控制台的数据存储管理功能，对上下游存储资源进行引用。使用存储注册方式后，实时计算控制台能够为您提供数据预览、数据抽样、DDL自动生成等功能，便于您一站式管理云上存储资源。

 说明:

实时计算数据存储功能当前仅支持同账号属主下的存储资源，即当前使用实时计算的A用户（包括A用户的子账户）所注册的存储资源，必须是A购买的存储资源。不支持跨账号授权，如果您需要跨账号授权使用存储资源，请使用明文方式。

- 注册数据存储

  使用存储注册方式需要将上下游存储资源预先注册至实时计算开发平台。注册步骤如下：

  1. 登录实时计算控制台。
  2. 单击页面顶部的开发。
  3. 在开发界面，单击左侧导航栏中的数据存储。
  4. 在数据存储页签的右上角，单击+注册与网络。
  5. 在注册数据存储与网络探测窗口，完成对应储存设备的参数配置。

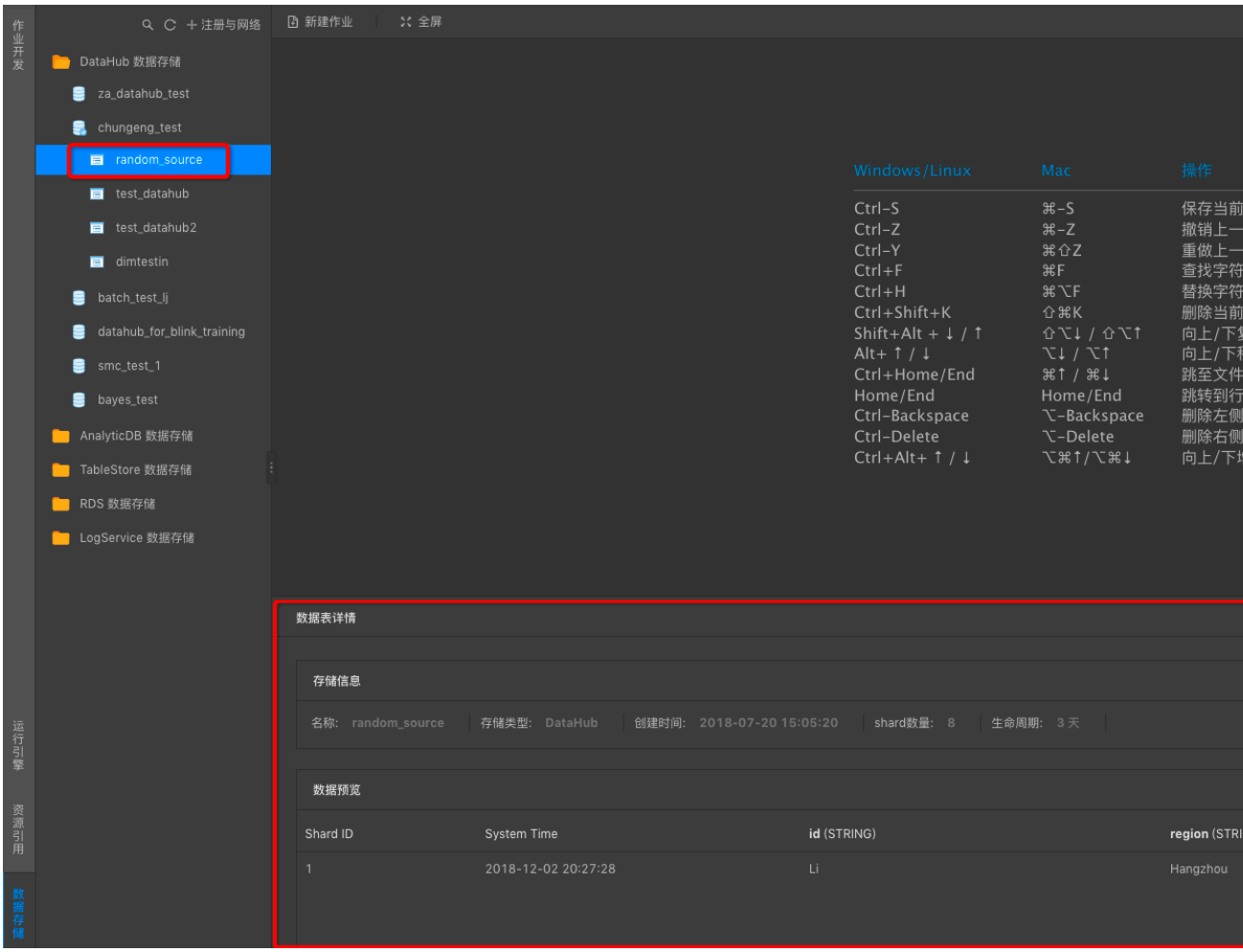
     目前实时计算产品仅支持注册如下五种存储资源，具体方法请点击以下产品链接：

     - #unique_26
     - #unique_27
     - #unique_28
     - #unique_29
     - #unique_30

- 数据预览

  对于已经注册的存储资源，实时计算提供数据预览功能，功能实现步骤如下：

  1. 在开发，单击左侧导航栏底部的数据存储。
  2. 在数据存储区域，双击数据存储类型文件夹以及文件夹下的各节点，直至目标数据表。
  3. 在数据表详情 > 数据预览，查看存储资源的数据。



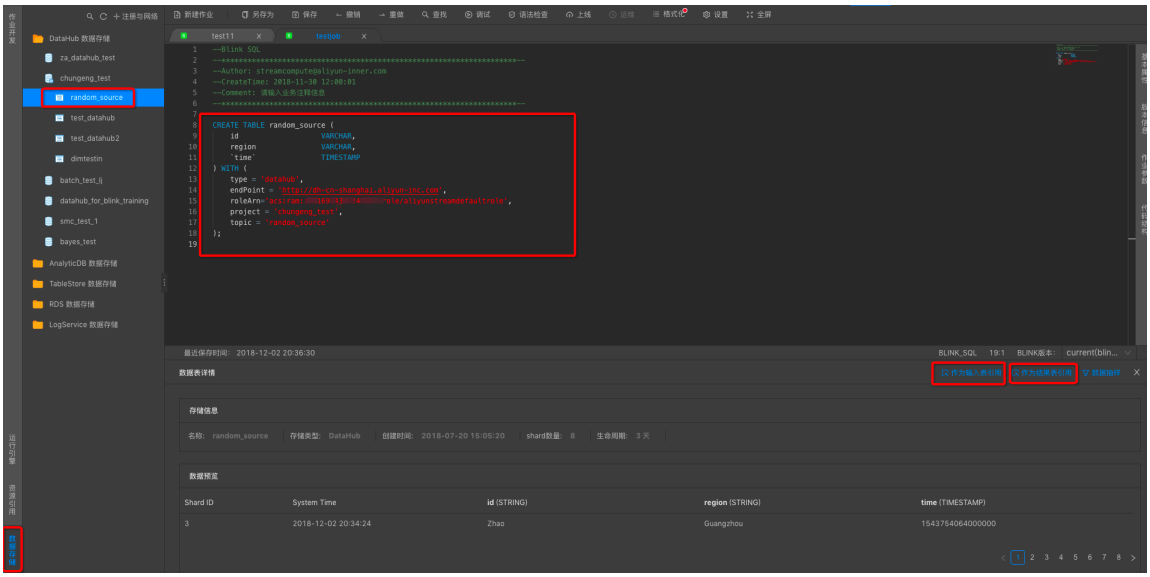
- 自动生成DDL

  对于已经注册的存储资源，实时计算提供自动生成DDL的功能，功能实现步骤如下：

  1. 在开发，单击左侧导航栏底部的数据存储。
  2. 在数据存储区域，双击数据存储类型文件夹以及文件夹下的各节点，直至目标数据表。
  3. 在数据表详情，单击作为输入表引用、作为结果表引用或作为维表引用，即可自动生成DDL。

  说明：

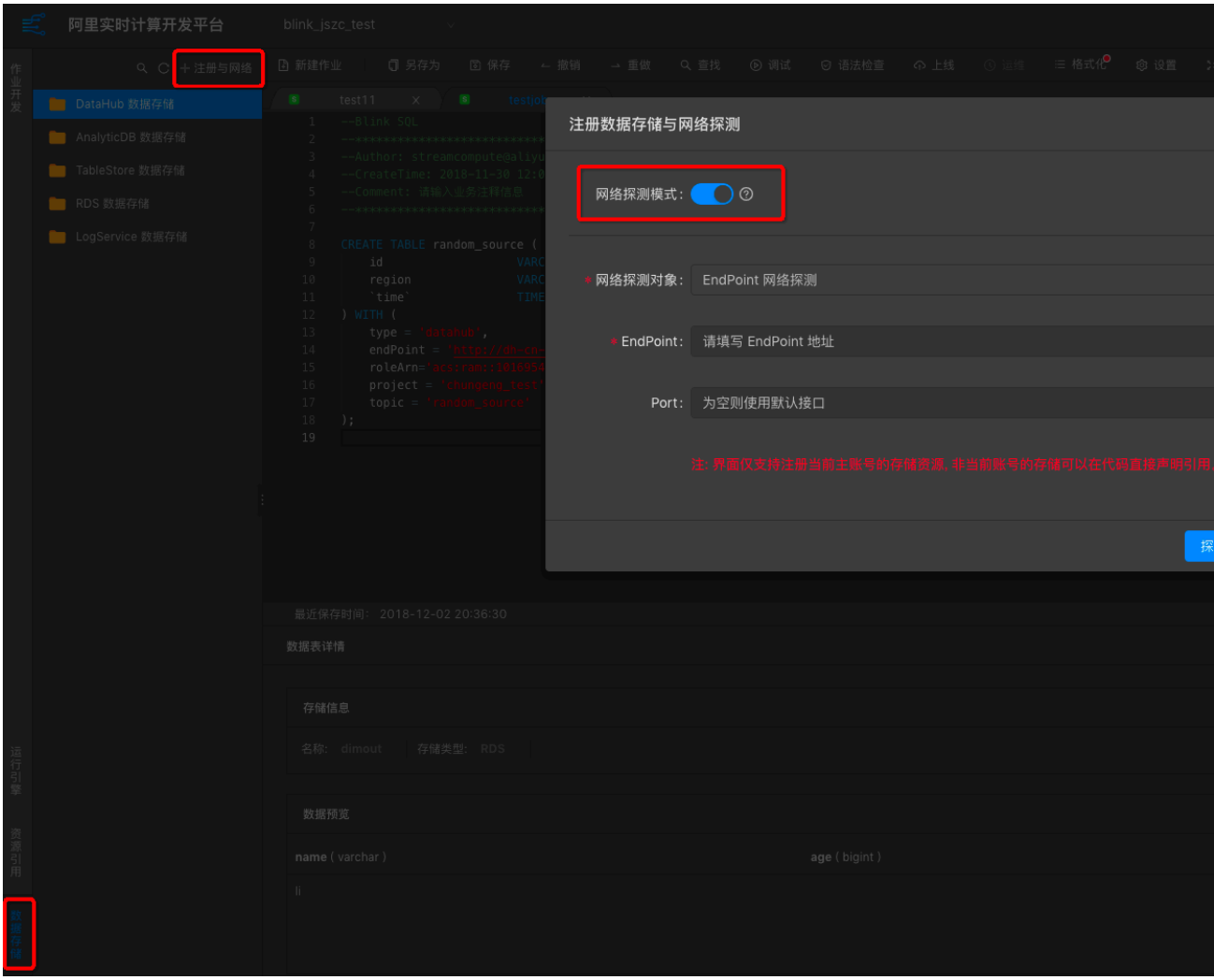
自动生成的DDL仅包含基本的WITH参数，以保证实时计算与存储资源的连通性。您可在此基础上增加其他WITH参数。



- 网络探测

实时计算的数据存储功能提供网络探测功能，用于探测实时计算产品与被探测的存储资源的网络连通性。网络探测功能开启方式如下：

1. 在开发，单击左侧导航栏底部的数据存储。
2. 在数据存储页签的右上角，单击+注册与网络。
3. 在注册数据存储与网络探测页面，打开网络探测模式开关。



2.2 注册数据存储

2.2.1 注册数据总线 DataHub

本文为您介绍注册数据总线 DataHub数据存储所需的参数信息，以及注册存储过程中的常见问题。

> **说明:**
> DataHub未正式商用。

数据总线 DataHub为阿里云数加平台提供了大数据的入口服务。结合阿里云众多云产品，DataHub可以构建一站式的数据处理平台。实时计算通常使用DataHub作为流式数据存储的源头和目的端。

登录注册页面

> **说明:**

使用注册存储功能前，请先完成实时计算对存储设备的访问授权，授权方法请您参见#unique_23和#unique_24。

登录步骤请参见#unique_33/unique_33_Connect_42_section_sx2_4tz_zfb。

参数说明

- EndPoint

  填写DataHub的EndPoint。不同地域下DataHub有不同的Endpoint，具体请参见DataHub访问控制。

  说明:
  EndPoint地址格式为`http://****.aliyun-inc.com`，不能以/结尾。

  - 共享模式：使用经典网络ECS Endpoint。例如，华东1（杭州）使用`http://dh-cn-hangzhou.aliyun-inc.com`。

    说明:
    共享模式项目所在的区域与DataHub所在区域不要求一致，但不一致可能增加网络延时。

  - 独享模式：使用VPC ECS Endpoint。例如，华东1（杭州）使用`http://dh-cn-hangzhou.aliyun-inc.com`。

    说明:
    独享模式集群所在的区域与DataHub所在区域要求一致。

- Project

  填写DataHub的Project。

  说明:
  跨属主的数据存储不能注册。例如，A用户拥有DataHub的ProjectA，但B用户希望在实时计算使用ProjectA，目前实时计算暂不支持这类使用场景下注册，若需使用可使用明文方式，具体参见考数据存储概述-明文#unique_34/unique_34_Connect_42_section_qx2_4tz_zfb。

常见问题

Q: 注册数据存储失败的原因有哪些?

A: 实时计算的数据存储页面能够协助您完成数据管理，注册数据存储功能是使用相关存储SDK代为访问各类存储。如果注册数据存储失败，请从以下方面进行排查：

- 是否已经开通并拥有DataHub的Project。请登录DataHub控制台，查看您是否具备访问对应Project的权限。
- 您是否为DataHub Project的属主。跨属主的数据存储不能注册。
- 您填写的DataHub的Endpoint和Project是否完全正确。DataHub的Endpoint必须以`http`开头，且不能以/结尾。
- 您填写的DataHub Endpoint必须为经典网络地址，而非VPC地址。目前实时计算暂不支持VPC内部地址。
- 请不要重复注册，实时计算提供注册检测机制，重复注册会导致注册失败。

Q: 为什么数据抽样仅支持时间抽样?

A: DataHub定位是流数据存储，对外提供的接口也仅有时间参数。因此，实时计算也仅能提供基于时间的抽样。

2.2.2 注册分析型数据库MySQL版

本文为您介绍注册分析型数据库MySQL版数据存储所需的参数信息。

登录注册页面

登录步骤请参见#unique_33/unique_33_Connect_42_section_sx2_4tz_zfb。

参数说明

参数名称	说明
URL	AnalyticDB for MySQL控制台链接信息串。 · 共享模式：请使用AnalyticDB for MySQL的经典网络的连接点 · 独享模式：请使用AnalyticDB for MySQL的VPC网络的连接点。
Database	AnalyticDB for MySQL数据库名称，即AnalyticDB for MySQL实例名称。
AccessKey ID	阿里云账号安全信息页面的AccessID。
AccessKey Secret	阿里云账号安全信息页面的AccessKey。

 说明:

- URL地址查询方法

  1. 登录AnalyticDB 控制台。
  2. 单击对应的实例名称，进入基本信息页面。
  3. 在连接信息 > 连接地址中查看相应的连接地址。

- AccessId、AccessKey信息查询方法参见如何查看AccessID、AccessKey信息。

2.2.3 注册表格存储 TableStore

本文为您介绍注册表格存储 TableStore数据存储所需的参数信息。

表格存储 Table Store是构建在阿里云飞天分布式系统之上的NoSQL数据存储服务。表格存储能够提供海量结构化数据的存储服务和实时访问服务。实时计算要求访问低延迟，同时对运算的复杂程度要求较低，因此，实时计算适合使用TableStore作为数据存储维表和结果表。

 说明:
使用注册存储功能前，请先完成实时计算对存储设备的访问授权，授权方法请参见#unique_37。

登录注册页面

登录步骤请参见#unique_33/unique_33_Connect_42_section_sx2_4tz_zfb。

参数说明

- Endpoint
  - 填写TableStore的Endpoint。请在表格存储控制台查看TableStore的Endpoint信息，填写TableStore私网地址。具体步骤请参见#unique_38/unique_38_Connect_42_section_c7o_v0k_41k。
  - TableStore访问网络类型设置为允许任意网络访问。登录表格存储控制台，选择实例详情 > 实例网络类型 > 更改 > 允许任意网络访问。
- 实例名称

  填写TableStore的实例名称。

2.2.4 注册云数据库RDS版

本文为您介绍注册云数据库RDS版数据存储所需的参数信息，以及连接过程中的常见问题。

云数据库RDS版是一种稳定可靠、可弹性伸缩的在线数据库服务。目前实时计算支持包括MySQL在内的部分RDS数据库引擎。

 说明:

- 使用注册存储功能前，请先完成实时计算对存储设备的访问授权，授权方法请参见#unique_23和#unique_24。
- 高频或高并发写入场景，一般不建议使用RDS作为实时计算作业的结果表，存在死锁风险。建议使用表格存储作为结果表（详情请参见#unique_40）。
- 云数据库RDS8.0版本不支持注册存储功能，请使用#unique_41/unique_41_Connect_42_section_qx2_4tz_zfb。

登录注册页面

登录步骤请参见#unique_33/unique_33_Connect_42_section_sx2_4tz_zfb。

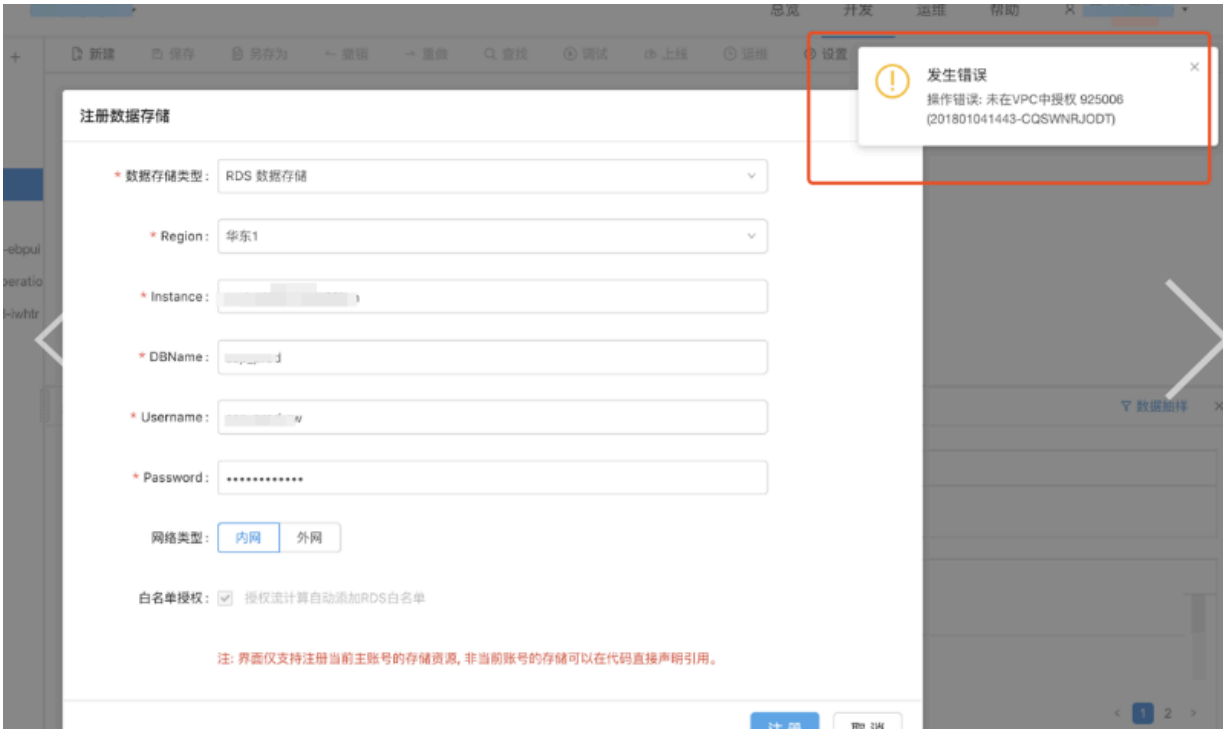
参数说明

 说明:
存储注册过程中系统会为RDS自动配置IP白名单。

参数名称	说明
Region	选择RDS的地域，请您选择RDS所在地域。
Instance	填写RDS实例ID。  说明: 请填写实例ID，不是实例名称。
DBName	填写RDS中DataBase的名称。  说明: DataBase是RDS的数据库名称，不是Instance的名称。
User Name	RDS数据库登录名称。
Password	RDS数据库登录密码。
网络类型	默认为内网。
白名单授权	默认设置。实时计算存储注册方式为RDS自动添加IP白名单。

常见问题

Q：存储注册过程中出现操作错误：未在VPC中授权?



A：注册的RDS实例为专有网络（VPC）实例。对于VPC实例，请先完成实时计算对VPC的访问授权，授权步骤请参见#unique_5。

2.2.5 注册日志服务 LOG

本文为您介绍注册日志服务 LOG数据存储所需的参数信息，以及存储注册过程中的常见问题。

日志服务是针对日志场景的一站式解决方案。日志服务提供海量日志数据采集、订阅、转储与查询功能。在您使用日志服务完成了对ECS日志的管理的前提下，实时计算无需进行数据迁移，即可对接日志服务的LogHub存储。

> 说明:
> 使用注册存储功能前，请先完成实时计算对存储设备的访问授权，授权方法请参见#unique_37。

登录注册页面

登录步骤请参见#unique_33/unique_33_Connect_42_section_sx2_4tz_zfb。

参数说明

- Endpoint

  填写日志服务的Endpoint。不同的地域下日志服务有不同的Endpoint，请参见日志服务服务入口#unique_43。

  > 说明:

- Endpoint需增加http://开头，且不能使用/结尾，如http://cn-hangzhou-intranet.log.aliyuncs.com。
- 实时计算和日志服务均处于阿里云内网，建议您填写经典网络或VPC网络服务入口。不建议填写公网服务入口，填写公网服务入口可能导致性能问题和外网宽带的消耗。

· Project

填写日志服务的Project。

 说明:
跨属主的数据存储不能注册。例如，A用户拥有日志服务的Project A，但B用户希望在实时计算使用Project A。目前，实时计算暂不支持这类场景下的数据存储注册方式，请使用#unique_34/unique_34_Connect_42_section_qx2_4tz_zfb。

常见问题

Q：注册数据存储失败的原因有哪些?

A：实时计算的数据存储页面能够协助您完成数据管理，注册数据存储功能是使用相关存储SDK代为访问各类存储。如果注册数据存储失败，请从以下方面进行排查：

· 是否已经开通并拥有LogService的Project。请登录日志服务控制台，查看您是否具备访问对应Project的权限。
· 您是否为日志服务Project的属主。跨属主的数据存储不能注册。
· 您填写的日志服务的Endpoint和Project是否完全正确。日志服务的Endpoint必须以http开头，且不能以/结尾。
· 您填写的日志服务的Endpoint必须为经典网络地址，而非VPC地址。目前实时计算暂不支持VPC内部地址。
· 请不要重复注册，实时计算提供注册检测机制，重复注册会导致注册失败。

Q：为什么数据抽样仅支持时间抽样?

A：日志服务定位是流数据存储，对外提供的接口也仅有时间参数。因此，实时计算也仅能提供基于时间的抽样。

2.3 VPC访问授权

实时计算共享模式访问阿里云专有网络（VPC）中的存储资源前，需要进行VPC访问授权。本文为您介绍VPC访问授权的流程。

背景

实时计算共享模式属于阿里云经典网络，若需要访问阿里云VPC中的存储资源（例如，VPC中的RDS、MySQL、HybridDB数据库等），则需要完成VPC访问授权。

> 说明:
>
> - 独享模式集群处于阿里云VPC，无需进行VPC访问授权。
> - 完成VPC访问授权后，实时计算访问对应的存储资源可能存在带宽受限等性能问题。通常不建议在实时计算共享模式下访问VPC中的存储资源。

VPC访问授权步骤

1. 登录实时计算控制台。
2. 将鼠标悬停至页面右上角账号名称。
3. 在下拉菜单中，单击项目管理。
4. 在左侧导航栏中单击VPC访问授权。
5. 在VPC访问授权页面的右上角，单击新增授权。
6. 在授权流计算访问VPC页面，输入相应的配置参数。

参数	说明
名称	VPC的名称。
地域	VPC中存储设备所在的区域。
VPC ID	VPC中存储设备的VPC网络ID。 说明: RDS VPC网络ID查看步骤如下: a. 登录RDS管理控制台。 b. 在页面左上角，选择实例所在地域。 c. 单击目标实例ID。 d. 单击左侧导航栏中的数据库连接。 e. 在实例连接 > 数据库连接 > 网络类型中，查看RDS的VPC ID。例如，`vpc-bp1lysht98wrvl9n3****`。

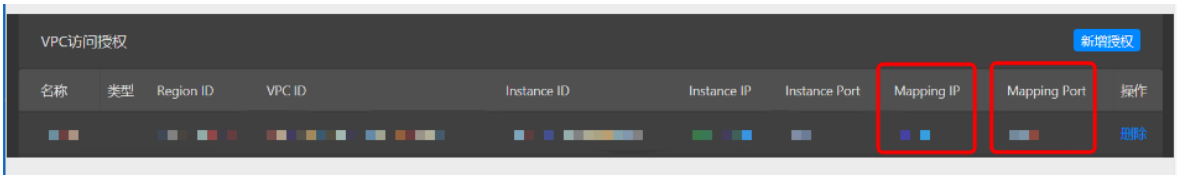
参数	说明
Instance ID	VPC中存储设设备的实例ID。 说明: RDS中实例ID查看步骤如下: a. 登录RDS管理控制台。 b. 在页面左上角，选择实例所在地域。 c. 单击目标实例ID，进入基本信息页面。 d. 在基本信息 > 实例ID中，查看RDS实例的ID。
Instance Port	VPC中存储设备的端口ID。 说明: RDS端口查询方法请参见#unique_45。

常见问题

Q：明文方式中，专有网络存储设备URL参数如何添加?

A：使用明文方式引用VPC中的存储时，DDL WITH参数中的URL参数值需填写VPC访问授权页面中的Mapping IP和Mapping Port参数，例如，`url='jdbc:mysql://<mappingIP>:<mappingPort>/<databaseName>'`。Mapping IP和Mapping Port信息查看步骤如下:

1. 登录实时计算控制台。
2. 将鼠标悬停至页面右上角账号名称。
3. 在下拉菜单中，单击项目管理。
4. 在左侧导航栏中单击VPC访问授权。
5. 在VPC访问授权页面查看Mapping IP和Mapping Port信息。



2.4 数据存储白名单配置

新建的数据库通常默认拒绝外部设备的访问，只有配置在数据存储白名单中的IP地址才被允许访问。本文以RDS为例，为您介绍如何配置数据存储白名单。

实时计算IP地址

实时计算分为共享模式和独享模式，2种模式的IP地址有所不同。

- 共享模式IP地址
  - 访问经典网络下的数据存储

    可根据项目所在的区域，配置对应区域的IP地址。

Region	白名单
华东2（上海）	11.53.0.0/16,11.50.0.0/16,10.152.0.0/16,10.154.0.0/16,11.132.0.0/16,11.178.0.0/16,11.200.210.74,11.200.215.195,11.217.0.0/16,11.219.0.0/16,11.222.0.0/16,11.223.116.79,11.223.69.0/24,11.223.70.0/24,11.223.70.173,11.223.70.48
华北2（北京）	11.223.0.0/16,11.220.0.0/16,11.204.0.0/16
华南1（深圳）	11.200.0.0/16

  - 访问VPC下的数据存储

    共享模式集群位于阿里云的经典网络，若需要访问阿里云VPC网络下的存储资源，需要通过VPC访问授权。VPC授权白名单网段查询步骤如下：#unique_47/unique_47_Connect_42_section_c3c_swz_zfb

    1. 登录实时计算控制台。
    2. 将鼠标悬停至页面右上角账号名称。
    3. 在下拉菜单中，单击项目管理。
    4. 在左侧导航栏中单击VPC访问授权。
    5. 在VPC访问授权页面中，单击对应VPC访问授权的Region ID字段下的链接，进入白名单网段页面。
    6. 在白名单网段窗口查询VPC授权白名单网段。

- 独享模式IP地址

  独享模式中仅需要配置独享集群对应的弹性网卡（ENI）地址。ENI地址的查看步骤如下：

  1. 登录实时计算控制台。
  2. 将鼠标悬停至页面右上角账号名称。
  3. 在下拉菜单中，单击项目管理。
  4. 单击左侧导航栏中的集群列表。
  5. 在集群列表页面，单击名称字段下目标集群名称。
  6. 在集群信息窗口，查看集群的ENI信息。

配置RDS白名单

实时计算将RDS作为数据存储使用时，需要多次读写RDS数据库，必须将实时计算的IP地址配置进入RDS白名单。RDS白名单配置方法，请参见#unique_48。

3 作业开发

3.1 开发

本文为您介绍实时计算作业开发流程以及开发页面中的语法检查、SQL辅助和SQL版本管理功能。

> 说明:
>
> - 实时计算主要使用Flink SQL进行作业开发，Flink SQL开发手册参见Flink SQL开发指南概述。
> - 实时计算独享模式不支持归档保存已停止（含暂停）的作业的运行日志。若需要查询已停止（含暂停）的作业运行日志，请将日志输出至用户自定义的SLS或OSS中。日志输出步骤参见#unique_51。

开发流程

1. 登录实时计算控制台。
2. 单击页面顶部的开发。
3. 在开发页面，单击页面顶部的新建作业。
4. 在新建作业界面，输入作业配置信息。

作业参数	说明
文件名称	作业的名称。 说明: 作业名称在当前项目中需保持唯一。
作业类型	· 共享模式：仅支持FLINK_STREAM/SQL作业类型。 · 独享模式：支持FLINK_STREAM/DATASTREAM和FLINK_STREAM/SQL作业类型。
存储位置	在文件夹目录中，指定该作业的代码文件所属的文件夹。您还可以单击现有文件夹右侧的图标，新建子文件夹。

5. 单击确认，进入编辑界面，编写SQL代码。

> 说明:

- 您可以在作业编辑页面右侧的代码结构查看SQL代码结构。
- 建议使用作业编辑左侧数据存储管理上下游存储，具体参见#unique_34。

语法检查

单击作业开发页面顶部的语法检查可检测SQL语句，并显示出相应的错误信息。

说明:

- 保存作业可以触发SQL语法检查功能。
- 请编写完整的SQL逻辑后再进行语法检查，否则语法检查不生效。

作业参数

您可以在开发界面右侧的作业参数，配置作业所需参数。作业参数配置详情，参见#unique_52/unique_52_Connect_42_section_sxj_yjm_bgb。

SQL辅助

- Flink SQL语法检查

  您在修改SQL后即可进行自动保存。保存操作可以触发SQL语法检查功能。语法校验出错误后，将在作业编辑界面提示出错行数、列数以及错误原因。
- Flink SQL智能提示

  您在输入Flink SQL过程中，作业编辑页面提供包括关键字、内置函数、表、字段智能记忆等提示功能。
- Flink SQL语法高亮显示

  高亮显示Flink SQL中关键字，使用不同的颜色区分Flink SQL语法中不同的结构。

SQL版本管理

数据开发为您提供代码版本管理功能。每提交一次作业即可生成一个代码版本。代码版本用于版本追踪、版本修改以及后期版本回滚。

在版本信息页面，单击操作 > 更多，可以选择相应的版本管理功能：

- 对比：查看最新代码和指定版本的差异。
- 回滚：使用回滚功能回滚到指定版本。
- 删除：实时计算公共云默认版本数上限是20。当版本数小于20时，您可以提交作业。如果当前的版本数为20，系统将不允许该作业的提交请求，并提示您删除部分旧版本作业。

说明:

当前版本数低于版本上限数后可再次提交作业。

- 锁定：锁定当前作业版本。

> 说明:
> 解锁前无法提交新版本。

3.2 上线

完成作业开发、作业调试，并且通过语法检查后，上线作业，即可将数据发布至生产环境。

上线步骤

1. 资源配置

   选择对应的资源配置方式。第1次启动建议使用系统默认配置。

   > 说明:
   > 实时计算支持手动资源配置和自动资源配置2种资源配置方式：
   >
   > - 手动资源配置方法参见#unique_54。
   > - 自动资源配置方法根据实时计算版本，分为以下2种方式：
   >   - 实时计算3.0以上版本：AutoScale自动配置，详情请参见#unique_55。
   >   - 实时计算3.0及以下版本：AutoConf自动配置，详情请参见#unique_17。

2. 数据检查

   通过数据检查后，单击下一步 。

3. 上线作业

   单击上线 。

   > 说明:
   > 作业上线后只是将作业提交至集群，并没有启动作业。启动作业请参见#unique_9。

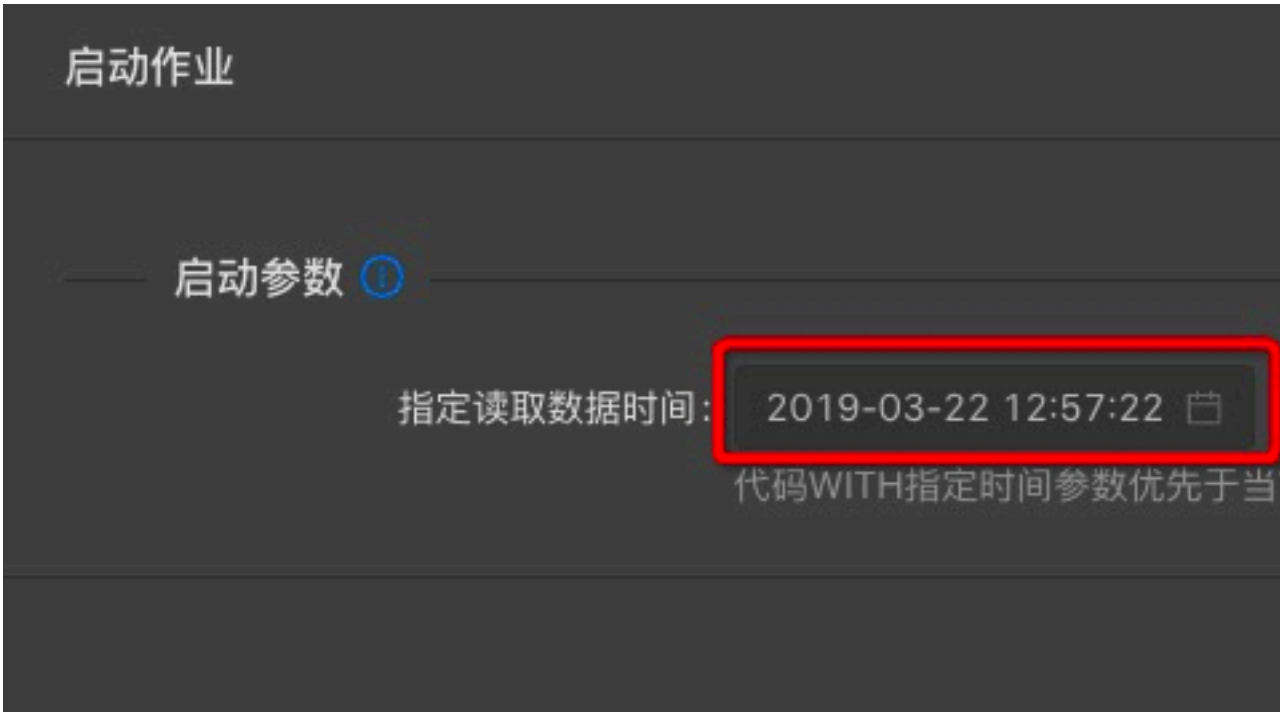
3.3 启动

完成作业开发和作业上线后，您可以在运维界面启动作业。

作业启动步骤如下：

1. 登录实时计算控制台。
2. 单击页面顶部的运维。
3. 在运维，单击目标作业操作列下的启动。

4. 在启动作业页面，单击指定数据读取数据时间（即指定启动位点）文本框。



5. 指定读取数据时间（启动位点），单击确定，完成作业启动。

> 说明：
> 启动位点表示从数据源表中读取数据的时间点：
> - 选择当前时间：表示从当前时间开始读取数据。
> - 选择历史时间：表示从历史时间点开始读取数据，通常用于回追历史数据。

> 说明：
> 作业启动完成后即可进入#unique_57阶段。

3.4 停止

更改SQL逻辑、更改作业版本、增加WITH参数和增加作业参数后，经过停止 > 启动的步骤，才能使变更生效。本文为您介绍如何停止作业。

> 说明：
> 只能对运行状态为运行或启动中的作业进行停止操作。

停止操作会清除任务状态，即如果有COUNT操作，作业停止 > 启动后，COUNT从0开始计算。作业停止操作步骤如下：

1. 登录实时计算控制台。

2. 单击页面顶部的运维。
3. 在运维，单击目标作业操作列下的停止。

4 作业调试

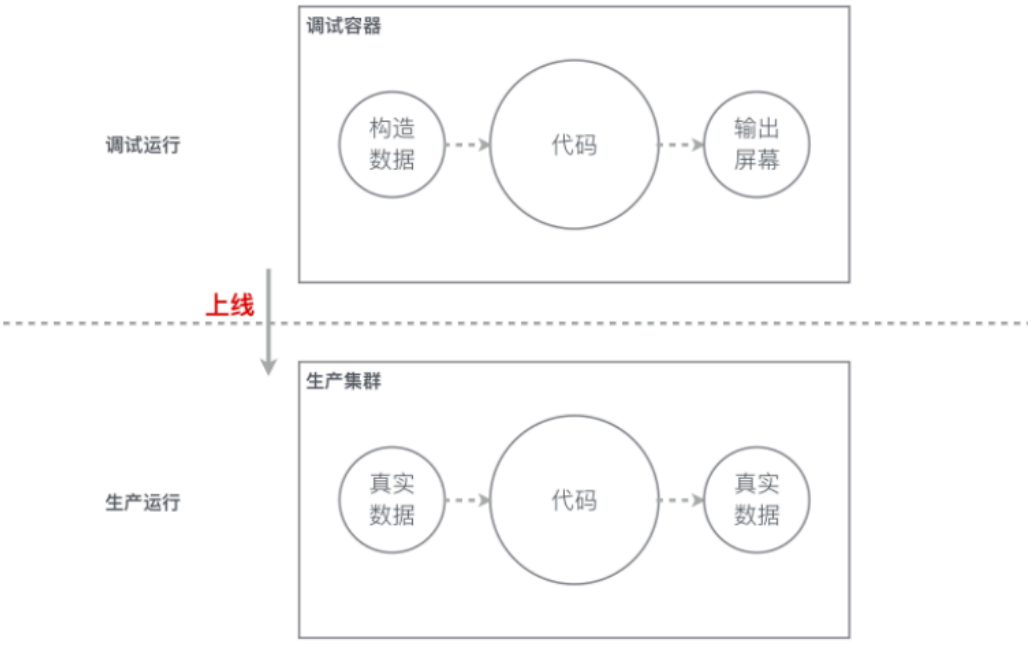
4.1 本地调试

实时计算开发平台为您提供了一套本地调试环境，您可以在本地调试环境中上传自定义数据、模拟作业运行、检查输出结果，最终验证业务逻辑的正确性。

本地调试环境的特点

本地调试环境与生产环境完全隔离。本地调试环境中，所有的Flink SQL在独立的调试容器中运行，调试结果输出至调试环境页面，不会对线上生产流、线上实时计算作业或线上数据存储系统造成影响。

> 说明:
> 实时计算调试模式无法检查出，数据存储中数据格式兼容性问题而导致的运行失败。例如，调试模式无法检查出输出数据长度大于RDS建表最大值的问题。

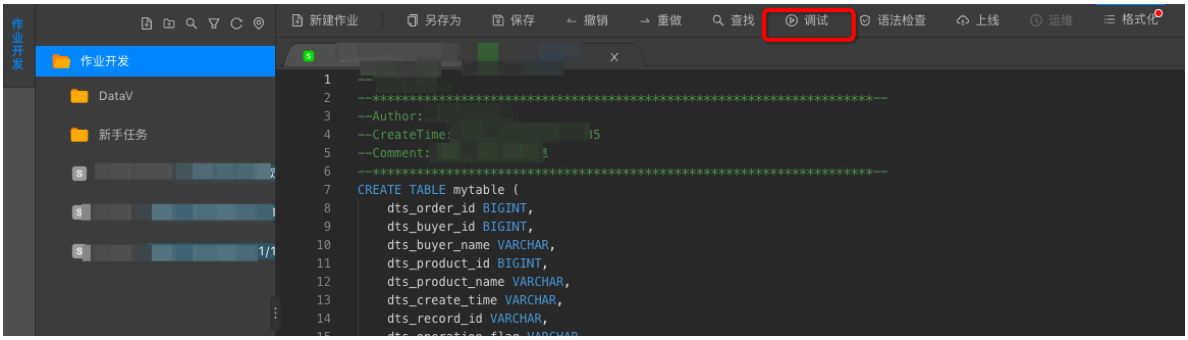


本地调试步骤

> 说明:

进行作业调试前，请您先完成作业的开发。作业开发流程请参见#unique_61/unique_61_Connect_42_section_rcp_bd2_2hb。

1. 在目标作业开发编辑区域，单击页面顶部的调试。



2. 在作业调试页面，输入测试数据。本地调试提供2种调试数据输入方式：

   - 本地上传方式

     a. 在数据预览区域，单击下载模板。

     b. 根据模板，编辑自定义调试数据。

     说明：
     调试数据的默认分隔符为逗号，如果需要自定义分隔符请参见调试数据分隔符。

     c. 在数据预览区域，单击上传，上传自定义调试数据。

   - 线上抽样方式

     说明：
     使用顺序抽样线上数据功能前，请确保数据源在抽取时间段存在数据。

     a. 在数据预览区域，单击随机抽样线上数据或顺序抽样线上数据。

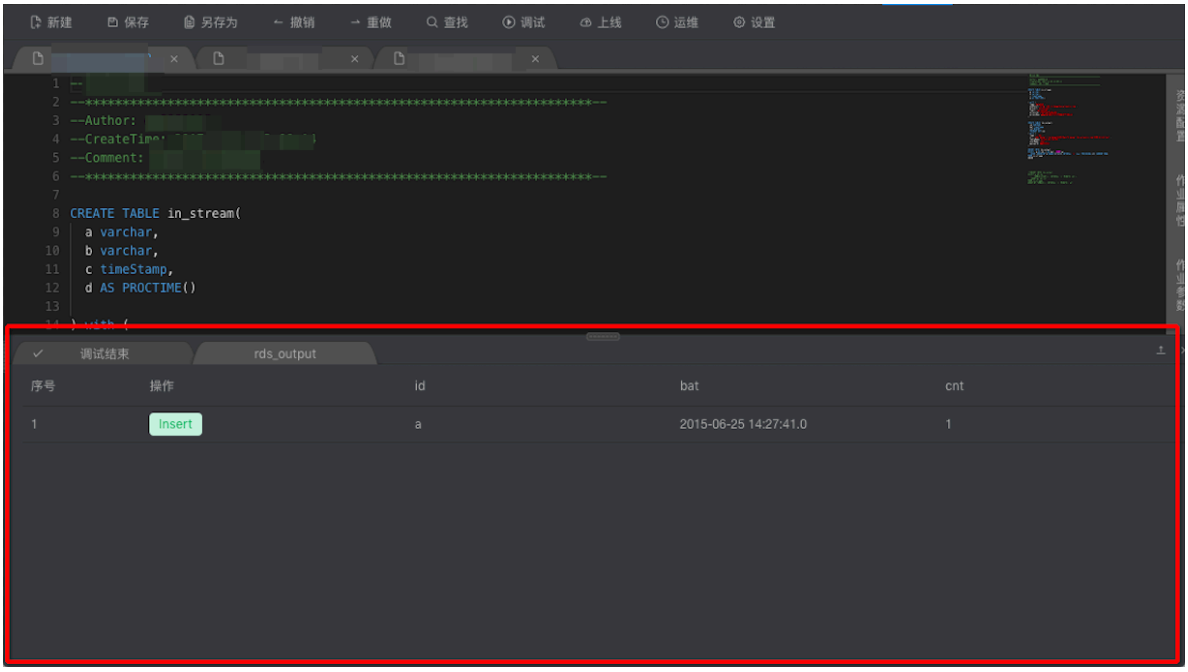
     b. 输入抽样配置信息。

     c. 单击确认。

   说明：
   数据上传成功后，可在数据预览区域查看已上传数据。

3. 单击确定，启动调试。

4. 在作业编辑区域的底部，查看调试输出结果。



调试数据分隔符

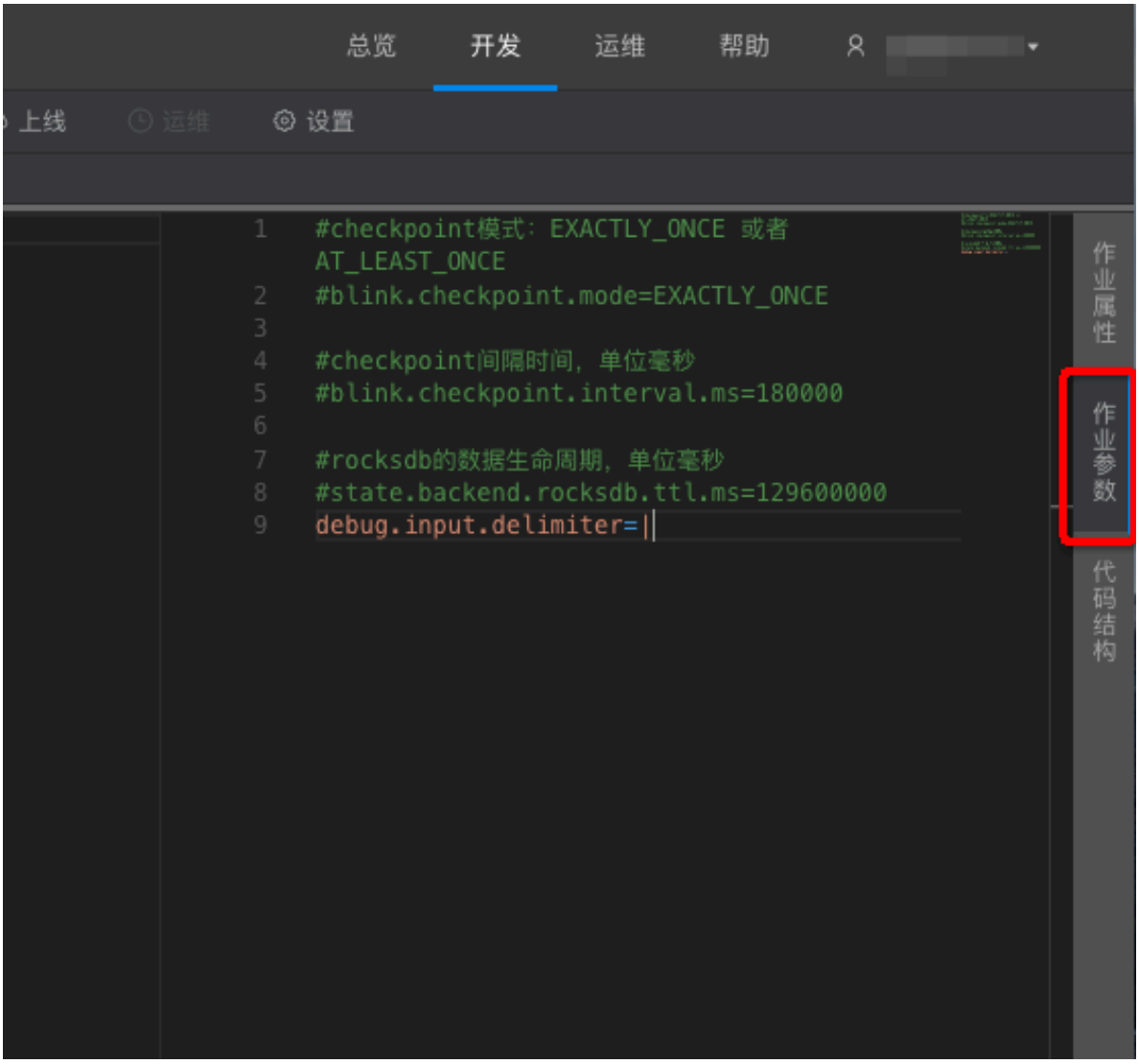
调试数据默认使用逗号（,）作为分隔符，如果输入的数据内容（例如JSON文件）中已存在逗号（,），您需要自定义其它字符（例如竖线（|））作为调试数据的分隔符。

> **说明:**
> 实时计算仅支持使用单个英文字符（例如竖线（|））为分隔符。不支持使用字符串（例如aaa）作为分隔符。

调试数据分隔符的配置步骤如下：

> **说明:**
> 配置数据分隔符前，请您先完成作业的开发。作业开发流程请参见#unique_61/unique_61_Connect_42_section_rcp_bd2_2hb。

1. 在作业编辑区域，单击右侧的作业参数。



2. 在作业参数编辑区域，输入调试数据分隔符的配置参数。配置分隔符为|的示例代码如下。

```
debug.input.delimiter = |
```

本地调试UDX的日志输出

- 本地调式时打印UDX中日志

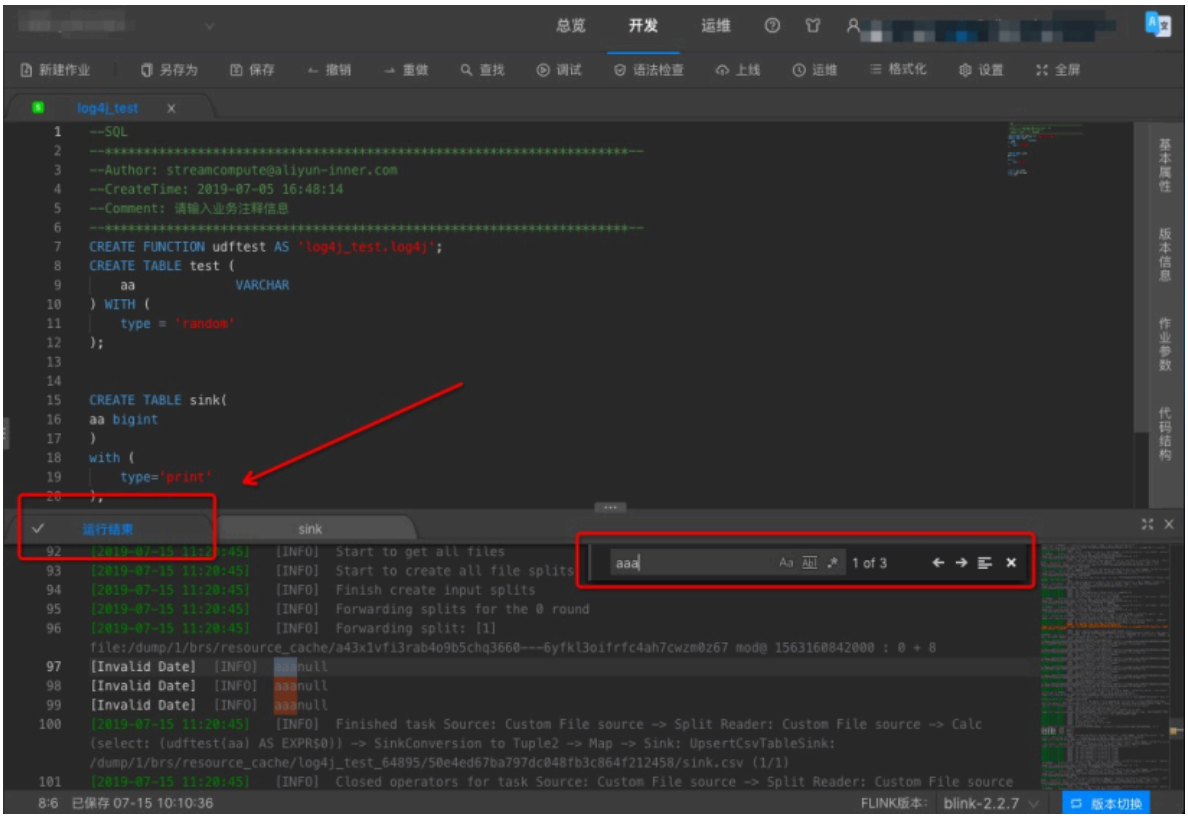
  在Java中通过以下方法，将日志的格式转换为实时计算可解析的格式，并打印本地调试的UDX日志。

```
public static void debugMsgOutput(String msg) {
    System.out.println(
      String.format("{\"type\":\"log\",\"level\": \"INFO\", \"time\": \"%s\", \"message\": \"%s\", \"throwable\": \"null\"}\n",new SimpleDateFormat("yyyy-MM-dd HH:mm:ss").format(new Date()), msg));
```

```
}
```

- 查看UDX的日志输出

  调试结束后，在作业编辑区域底部的运行结束页面，可以查看UDX的日志输出。



说明:

您可以通过快键Ctrl#F的方式搜索相应的日志信息。

4.2 线上调试

阿里实时计算开发平台为您提供了实时计算作业线上调试功能。相对于本地调试功能，线上调试功能需要消耗一定的资源，但是能够更加真实的验证业务逻辑的正确性。

线上调试功能使用真实的数据存储，有效的减少调试输出和生产输出的差异，有助于您在调试阶段发现问题。

线上调试步骤

1. 开发作业。作业开发详情，请参见#unique_61
2. 更新数据存储DDL中的`type`参数。
   - 源表：`type = 'random'`
   - 结果表：`type = 'print'`

3. 上线作业。作业上线步骤请参见#unique_63。
4. 启动作业。作业启动步骤请参见#unique_64。

线上调试Connector

阿里实时计算平台提供以下2种线上调试功能的Connector：

- `random`源表：周期性的生成对应类型的随机数据。
- `print`结果表：输出计算结果。

Connector表参数

- Random表参数

参数	说明
type	必选，取值唯一且为random。
interval	可选，产生数据的时间间隔（单位为毫秒），默认值为500。

- Print表参数

参数	说明
type	必选，取值唯一且为print。
ignoreWrite	可选，默认值为false。可选参数值如下： - false：同时输出结果表和日志。 - true：仅输出无数据的结果表，不输出日志数据。

线上调试示例

- 测试语句

```
CREATE TABLE random_source (
  instr               VARCHAR
) WITH (
  type = 'random'
);

CREATE TABLE print_sink(
  instr VARCHAR,
  substr VARCHAR,
  num INT
)with(
  type = 'print'
);

INSERT INTO print_sink
SELECT
  instr,
  SUBSTRING(instr,0,5) AS substr,
```

```
  CHAR_LENGTH(instr) AS num
FROM random_source
```

- 测试结果

线上调试结果查询步骤

> **说明:**
>
> - 查询线上调试结果前，请先完成作业#unique_8和#unique_9。
> - 线上调试需要消耗一定的CU资源。

线上调试结果查询查看步骤如下：

1. 登录实时计算控制台。
2. 单击顶部导航栏中的运维，进入运维页面。
3. 单击作业名称字段下对应的作业，进入作业运维页面。
4. 在Vertex拓扑区域，点击相应结果表节点。
5. 单击SubTask List > 查看日志，进入日志查看窗口。
6. 查看相应的日志。

   - Print结果表输出

     单击taskmanager.out右侧的查看日志。

   - UDX日志输出

     如果您使用了自定义函数UDX（UDX使用方法请参见#unique_65），可以使用如下2种方式查看日志：

     - system out/err方法

       单击taskmanager.out或taskmanager.err右侧的查看日志。

     - SLF4J的Logger方法

       单击taskmanager.log右侧的查看日志。

5 作业运维

5.1 运行信息

运行信息为您展示作业的实时运行信息。您可以通过作业的状态来分析、判断作业的状态是否健康，是否达到您的预期。

登录运行信息页面

1. 登录作业运维页面。
   a. 登录实时计算控制台。
   b. 单击页面顶部的运维。
   c. 在作业列表区域，单击作业名称下的目标作业名。
2. 在作业运维页面，单击页面顶部的运行信息。

Task状态

Task状态为您显示作业各状态的数量。Task存在以下7种状态：

- 创建
- 运行
- 失败
- 完成
- 调度
- 取消中
- 已取消

作业瞬时值

Task状态下方为您展示作业的瞬时数值。

名称	作用描述
输入TPS	每秒从源端读到的Block数。对于日志服务而言，可以将多条数据打包到一个LogGroup读取。Block数反映的是从源端每秒读取LogGroup数量。
输入RPS	每秒读取数据源表的RPS数，单位为条/秒。
输出RPS	每秒写入数据结果的的RPS数，单位为条/秒。
使用CPU	单个Job当前的CPU的使用状况。

名称	作用描述
启动时间	Job的启动的时间。
运行时长	Job的启动后运行的时间。

Vertex拓扑

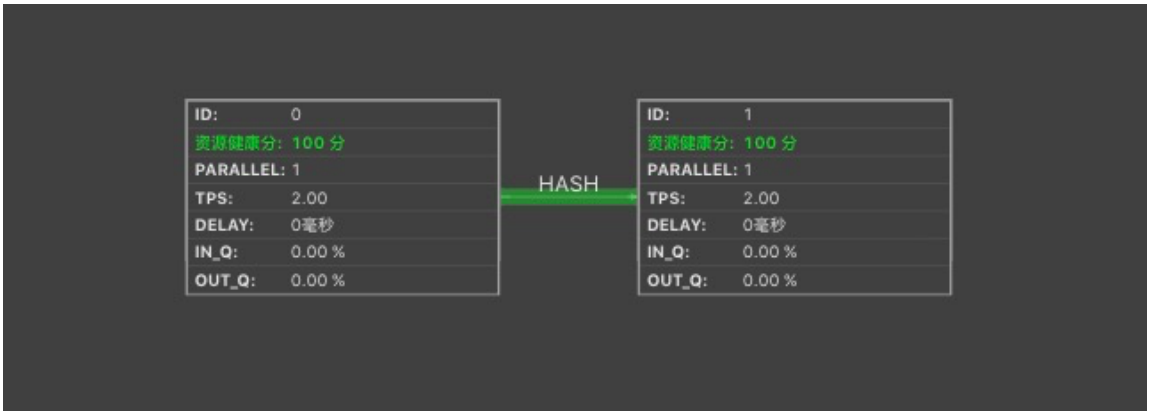
Vertex拓扑描述Realtime Compute底层计算逻辑的执行图。每个组件代表不同的Task，每个数据流从一个或多个数据源，流向另一个或多个数据结果表。数据流类似于任意有向无环图（DAG）。为了更高效地分布式执行，Realtime Compute底层会地将Operator的Subtask链接（Chain）在一起形成Task，每个Task在一个线程中执行。将Operators链接成Task能减少线程之间的切换，减少消息的序列化/反序列化，减少数据在缓冲区的交换，减少了延迟的同时提高整体的吞吐量。Operator代表的是每个计算逻辑的算子，而Task代表是多个Operator的集合。

- 显示模式

  Vertex拓扑默认显示拓扑图，您可以单击右上角的列表模式，完成显示模式的切换。

- Task节点状态信息

  - 视图模式Task参数信息



视图模式的节点框中为您显示了当前Task节点的信息，信息参数说明如下。

名称	信息描述
ID	运行拓扑图的编码。
资源健康分	通过资源健康分检查机制，反映作业的性能状况。资源健康分小于60分表明当前节点存在数据堆积，数据处理性能较差。 说明: 数据处理性能较差时建议使用#unique_68或#unique_69提升性能。
PARALLEL	并发量。

名称	信息描述
TPS	每秒读取上游数据的Block数。
DELAY	Task节点的业务延时。
IN_Q	Task节点的输入队列的百分比。
OUT_Q	Task节点的输出队列的百分比。

- 列表模式Task参数信息

  Vertex拓扑底部，可以通过列表模式查看Task节点的信息，信息参数说明如下。

名称	信息描述
ID	在运行拓扑图的编码。
Name	每个Task的详情信息。
Status	每个Task的运行的状态。
InQ max	Task节点的输入队列，单位是百分比。
OutQ max	Task节点的输出队列，单位是百分比。
RecvCnt sum	Task节点接收到的数据量。
SendCnt sum	Task节点的发送的数据量。
TPS sum	每秒读取上游的信息量。
Delay max	Task节点的业务延时。
StartTime	Task节点的启动时间。
Duration(s)	Task节点的运行时间。
Task	每个Task节点的并发的运行状态。

· Task节点线程信息

  单击Task节点，在Execution Vertex > Subtask List，查看Task的线程列表。

Vertex信息

说明:

仅实时计算3.0.0以上版本支持以下功能。

· Vertex内部Operator信息

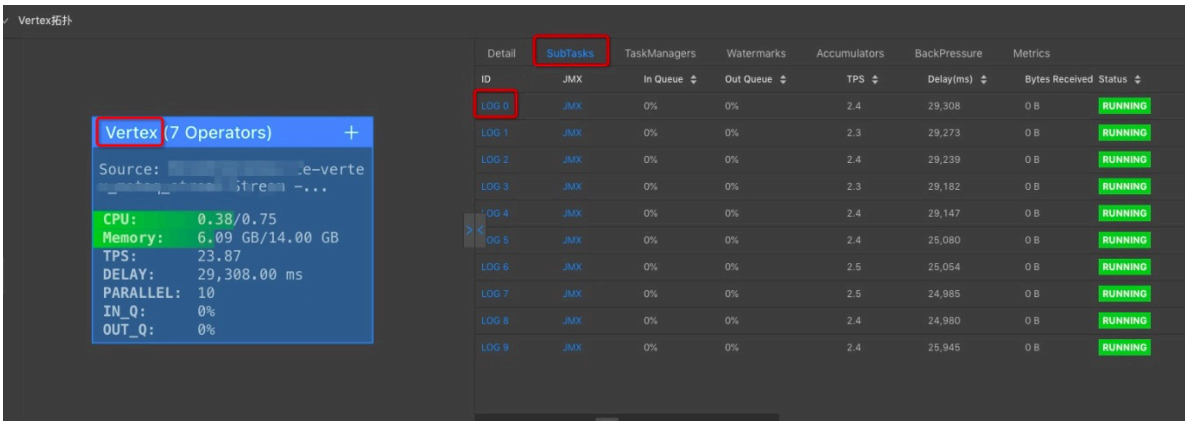
  在Vertex拓扑区域，单击单个Vertex框右上角加号（+），可以显示Vertex内部Operator信息。

- 显示Vertex内部详情

  单击Vertex拓扑区域右上角的Expand All，可以显示所有Vertex内部详情。

- Vertex详情页面

  在Vertex拓扑区域，单击Vertex边框或Vertex列表中的Name即可弹出右侧Vertex的详情页面。在Vertex的详情页面中，单击SubTasks页签中的Task ID（下图中的LOG 0）可以跳转至TaskManager相关日志页面。



5.2 数据曲线

阿里云实时计算提供了当前作业的核心指标概览页面。您可以通过数据曲线对作业的运行情况的进行一键式的诊断。未来实时计算还会提供更多基于作业现状的深度智能分析算法，以辅助您进行智能化和自动化诊断。

数据曲线示例如下。

说明:

- 数据曲线指标在实时计算作业运行状态下才提供显示，暂停以及停止状态均不提供显示。

- 作业指标是实时计算系统异步后台采集，存在一定延迟。作业启动1分钟后，各项指标才能逐步采集，然后在数据曲线显示。

登录数据曲线页面

1. 登录作业运维页面。
   a. 登录实时计算控制台。
   b. 单击页面顶部的运维。
   c. 在作业列表区域，单击作业名称下的目标作业名。
2. 在作业运维页面，单击页面顶部的数据曲线。

OverView

- Failover

  Failover曲线显示当前Job出现Failover（错误或者异常）的频率。计算方法为当前Failover时间点的前1分钟内出现Failover的累计次数除以60。（例如，最近1分钟Failover了一次，Failover的值为1/60=0.01667。）

- Delay

  为了全面的了解实时计算全链路的时效状况和作业的性能，实时计算提供3种延时指标：

  - 业务延时（Processing Delay）：业务延迟 = 当前系统时间 – 当前系统处理的最后一条数据的事件时间（Event time）。如果后续没有数据再进入上游存储，由于当前系统时间在不断往前推进，业务延时也会随之逐渐增大。
  - 数据滞留时间（Data Pending Time）：数据滞留时间 = 数据进入实时计算的时间 - 数据事件时间（Event time）。即使后续没有数据再进入上游存储，数据滞留时间也不会随之逐渐增大。通常用数据滞留时间来评估当前实时计算作业是否存在反压。
  - 数据间隔时间（Data Arrival Interval）：数据间隔时间= 业务延迟– 数据滞留时间。实时计算没有反压（即数据滞留时间较小且平稳）时，数据间隔时间可以反映数据源数据间的稀疏程度；实时计算存在反压（即数据滞留时间较大或不平稳）时，此参数没有实质性参考意义。

  说明：

  - 实时计算是分布式计算框架，以上3类延时指标的Metric首先是针对Source的单个分区（Shard/Partition等）进行计算，然后汇报所有分区中的最大值呈现到前端页面上，因此前端页面上显示的汇聚后的数据间隔时间并不精确等于业务延时 – 数据滞留时间。
  - 如果Source中的某个分区没有新的数据，将会导致业务延迟逐渐增大。

- 目前底层算法实现时，当数据间隔时间小于10秒时，会将数据间隔时间设置为0，进行上报。

- 各Source的TPS数据输入

  各Source的TPS数据输入对实时计算作业所有的流式数据输入进行统计，记录每秒读取数据源表的Block的数，让您直观的了解数据存储TPS（Transactions Per Second）的情况。与RPS（Record Per Second）不同，RPS是读取数据源TPS的Block数解析后的数据，单位是条/秒。（例如，日志服务，1秒读取5个LogGroup，那么TPS=5，如果每个LogGroup解析出来8个日志记录，那么一共解析出40个日志记录，RPS=40。）

- 各Sink的数据输出

  各Sink的数据输出对实时计算作业所有的数据输出（并非是流式数据存储，而是全部数据存储）做出进行统计，让您直观的了解数据存储RPS（Record Per Second）的情况。通常，在系统运维过程中，如果出现没有数据输出的情况，除了检查上游是否存在数据输入，同样要检查下游是否真的存在数据输出。

- 各Source的RPS数据输入

  各Source的RPS数据输入对实时计算作业所有的流式数据输入进行统计，让您直观的了解数据存储RPS（Record Per Second）情况。通常，在系统运维过程中，如果出现没有数据输出的情况，需要查看该值，判断是否数据源输出数据存在异常。

- 各Source的数据流量输入各Source的数据流量输入对实时计算作业所有的流式数据输入进行统计，记录每秒读取输入源表的流量的统计，让您直观的了解数据流量BPS（Byte Per Second）情况。

- 各Source的脏数据各Source的脏数据为您显示实时计算Source各时间段脏数据条数。

- Auto Scaling Successes and Failures

   说明:
  仅实时计算3.0.0以上版本支持此曲线。

  Auto Scaling Successes and Failures 为您显示AutoScale成功执行和未成功执行的次数。

- CPUs Consumed By Auto Scaling

   说明:
  仅实时计算3.0.0以上版本支持此曲线。

  CPUs Consumed By Auto Scaling 为您显示执行AutoScale时消耗的CPU量。

- Memory Consumed By Auto Scaling

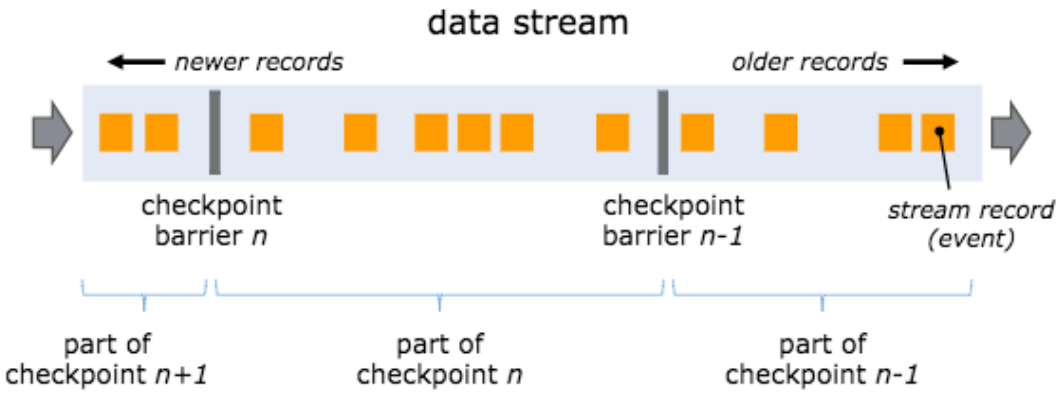
> 说明:
> 仅实时计算3.0.0以上版本支持此曲线。

Memory Consumed By Auto Scaling 为您显示执行AutoScale时消耗的内存量。

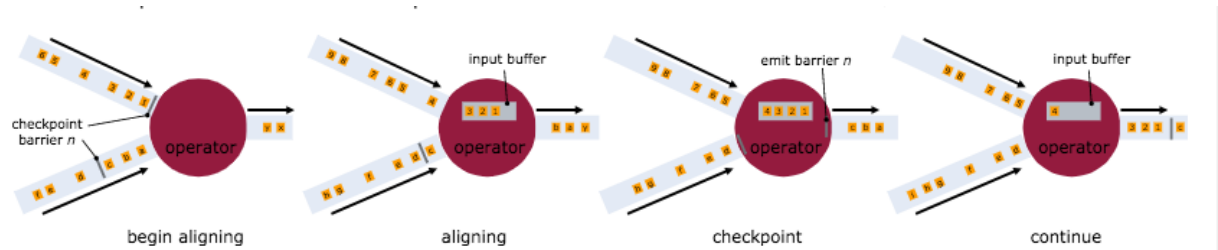
Advanced View

阿里云实时计算提供可以恢复数据流应用到一致状态的容错机制。容错机制的核心就是持续创建分布式数据流及其状态的一致快照。这些快照在系统遇到故障时，充当可以回退的一致性检查点（checkpoint）。

分布式快照的核心概念之一就是数据栅栏（barrier）。这些barrier被插入到数据流中，作为数据流的一部分和数据一起向下流动。barrier不会干扰正常数据，数据流严格有序。一个barrier把数据流分割成两部分：一部分进入到当前快照，另一部分进入下一个快照。每一个barrier都带有快照 ID，并且barrier之前的数据都进入了此快照。barrier不会干扰数据流处理，所以非常轻量。多个不同快照的多个barrier会在流中同时出现，即多个快照可能同时创建。



barrier在数据源端插入，当snapshot n的barrier插入后，系统会记录当前snapshot位置值 n （用Sn表示）。然后barrier继续往下流动，当一个operator从其输入流接收到所有标识snapshot n的barrier时，它会向其所有输出流插入一个标识snapshot n的barrier。当sink operator （DAG 流的终点）从其输入流接收到所有barrier n时，operator向检查点协调器确认snapshot n已完成。当所有sink都确认了这个快照，快照就被标识为完成。



以下就是记录Checkpoint的各种参数配置。

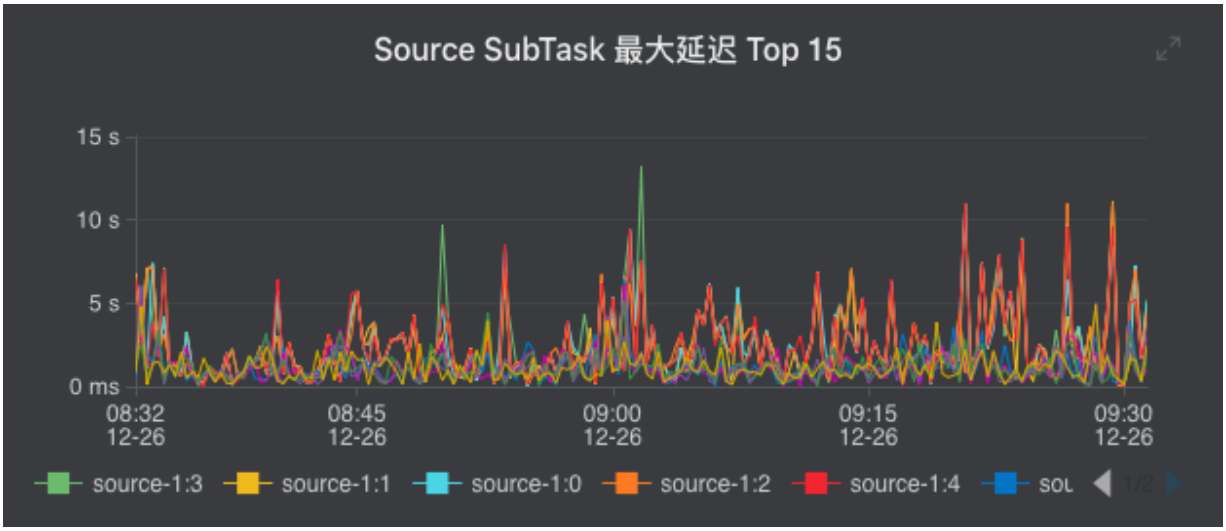
数据曲线名称	说明
Checkpoint Duration	进行Checkpoint保存状态所花费的时间，单位为毫秒。
CheckpointSize	进行Checkpoint所消耗的内存大小。
CheckpointAlignmentTime	当前节点进行checkpoint操作，等待上游所有节点到达当前节点的时间。当sink operator（DAG 流的终点）从其输入流接收到所有barrier n时，它向the checkpoint coordinator确认snapshot n已完成。当所有sink都确认了这个快照，快照就被标识为完成。这个等待的时间就是checkpointAlignmentTime。
CheckpointCount	指定时间段内Checkpoint的数量
Get	指定时间段内每个SubTask对ROCKSDB进行Get操作所花费的时间（最大值）。
Put	指定时间段内每个SubTask对ROCKSDB进行Put操作所花费的时间（最大值）。
Seek	指定时间段内每个SubTask对ROCKSDB进行Seek操作所花费的时间（最大值）。
State Size	指定时间段内Job内部State存储大小（如果增量过快JOB是异常的）。
CMS GC Time	指定时间段内Job内部容器进行GC（Garbage Collection）花费的时间。
CMS GC Rate	指定时间段内Job内部容器进行GC（Garbage Collection）的频率。

WaterMark

数据曲线名称	说明
WaterMark Delay	WaterMark距离系统时间的差值。
数据迟到丢弃TPS	每秒丢弃晚于Watermark时间到达Window的数据的数量。
数据迟到累计丢弃数	累计丢弃晚于Watermark时间到达Window的数据的数量。

Delay

Source SubTask 最大延迟 Top 15



表示每个Source的并发的业务延时的时长。

Throughput

数据曲线名称	说明
Task Input TPS	作业中所有的Task的数据的输入状况。
Task Output TPS	作业中所有的Task的数据的输出状况。

Queue

数据曲线名称	说明
Input Queue Usage	作业中所有的Task的数据的输入队列。
Output Queue Usage	作业中所有的Task的数据的输出队列。

Tracing

数据曲线名称	说明
Time Used In Processing Per Second	处理Task中每秒数据所花费的时长。
Time Used In Waiting Output Per Second	等待Task中每秒数据输出所花费的时长。
TaskLatency Histogram Mean	每个Task的延时时长。
Wait Output Histogram Mean	每个Task的等待输出时长。
Wait Input Histogram Mean	每个Task的等待输入时长。
Partition Latency Mean	每个分区中并发的延时时长。

Process

数据曲线名称	说明
Process Memory RSS	每个进程内存的使用状况。
CPU Usage	每个进程CPU的使用状况。

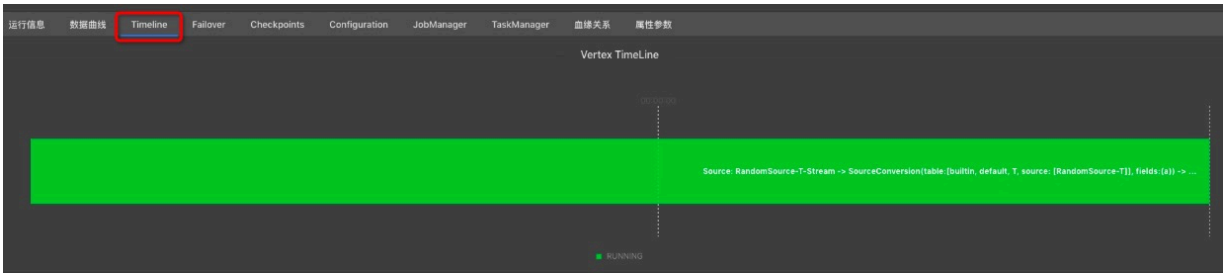
JVM

数据曲线名称	说明
Memory Heap Used	Job使用的JVM Heap存储量。
Memory Non-Heap Used	Job使用的JVM 非Heap存储量。
Thread Count	Job的线程数。
GC(CMS)	Job完成CG（Garbage Collection）的次数。

5.3 Timeline

Timeline页面为您显示各Vertex从启动位点到当前时间的运行状态。

说明:
仅实时计算3.0.0以上版本支持Timeline功能。



5.4 Failover

阿里云实时计算提供了当前作业的Failover页面，您可以在Failover页面了解当前运行作业的运行情况和报错信息。

登录Failover页面

1. 登录作业运维页面。
   a. 登录实时计算控制台。
   b. 单击页面顶部的运维。
   c. 在作业列表区域，单击作业名称下的目标作业名。

2. 在作业运维页面，单击页面顶部的Failover。

Latest FailOver

Latest FailOver为您展示作业当前的报错信息。

> 说明:
> 仅支持实时计算3.0以下版本。

FailOver History

FailOver History为您展示作业的历史报错信息。

> 说明:
> 仅支持实时计算3.0以下版本。

Root Exception

Root Exception为您展示作业当前的报错信息。

> 说明:
> 仅支持实时计算3.0及以上版本。

Exception History

Exception History为您展示作业的历史报错信息。

> 说明:
> 仅支持实时计算3.0及以上版本。

5.5 Checkpoints

阿里云实时计算提供可以恢复数据流，并和应用保持一致状态的容错机制。容错机制的核心是持续创建分布式数据流及其状态的快照。当系统出现故障时，这些快照充当可以回退的一致性检查点（Checkpoint）。

登录Checkpoints页面

1. 登录作业运维页面。
   a. 登录实时计算控制台。
   b. 单击页面顶部的运维。
   c. 在作业列表区域，单击作业名称下的目标作业名。
2. 在作业运维页面，单击页面顶部的Checkpoints。

Overview

> 说明:
> 该功能仅适用于实时计算3.0及以上版本。

Overview为您展示最新的Checkpoint信息，包括各节点Checkpoint的进程、Duration、StateSize等信息。

History

> 说明:
> 该功能仅适用于实时计算3.0及以上版本。

History为您展示近期Checkpoint的信息。单击行首的（+）可展现各节点Checkpoint的进程、Duration和StateSize等信息。

Summary

> 说明:
> 该功能仅适用于实时计算3.0及以上版本。

Summary为您展示已完成的Checkpoint的平均值、最大值和最小值信息。

Configuration

> 说明:
> 该功能仅适用于实时计算3.0及以上版本。

Configuration为您展示Checkpoint的配置信息。

Completed Checkpoints

> 说明:
> 该功能仅适用于实时计算3.0以下版本。

Completed Checkpoints为您展示已完成的Checkpoint信息。

名称	详情描述
ID	Checkpoint的ID编号
Start Time	Checkpoint开始的时间
Durations (ms)	Checkpoint花费的时间

Task Latest Completed Checkpoint

说明:

该功能仅适用于实时计算3.0以下版本。

Task Latest Completed Checkpoint为您展示最新一次Checkpoint的详细信息。

名称	详情描述
SubTask ID	SubTask的ID编号
State Size (Bytes)	Checkpoint的大小
Durations (ms)	Checkpoint的花费的时间

5.6 JobManager

JobManager是实时计算集群启动的重要组成部分，您可以在JobManager页面查看JobManager的详细参数信息。

登录JobManager页面

1. 登录作业运维页面。
   a. 登录实时计算控制台。
   b. 单击页面顶部的运维。
   c. 在作业列表区域，单击作业名称下的目标作业名。
2. 在作业运维页面，单击页面顶部的JobManager。

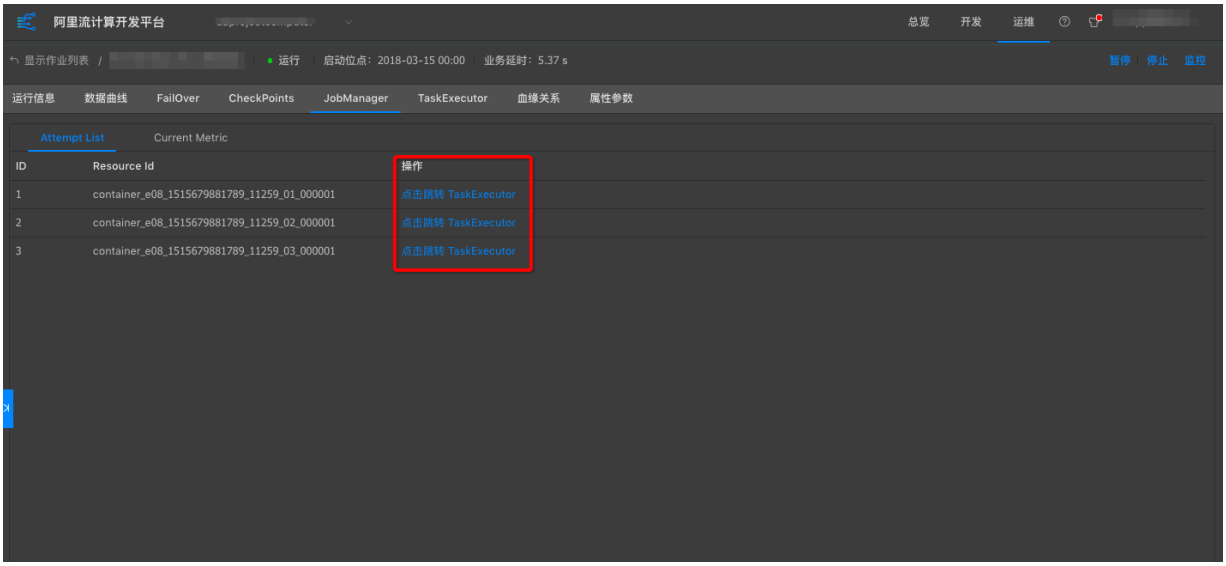
JobManager在集群启动中的作用

JobManager是实时计算集群的启动过程不可或缺的一部分。实时计算集群的启动流程如下：

1. 实时计算集群启动一个JobManger和若干个TaskManager。
2. Client向JobManager提交任务。
3. JobManager向TaskManager分配任务。
4. TaskManager向JobManager汇报心跳和统计信息。

JobManager参数信息

您可以在JobManager > Attempt List 页面，单击操作字段下的查看详情，查看JobManager的详细信息。



5.7 TaskExecutor

本文为您介绍TaskExecutor在实时计算集群启动过程中的作用以及TaskExecutor的界面。

> **说明:**
> 本文档仅适用于实时计算3.0以下版本。

背景

TaskExecutor是实时计算集群的启动过程不可或缺的一部分。TaskExecutor负责接收任务并将信息返还。TaskExecutor在启动时即完成了槽位数（Slot）的设置，每个Slot能启动1个Task线程。TaskExecutor从JobManager处接收需要部署的Task，部署启动后，与上游建立Netty连接，接收数据并处理。

登录TaskExecutor页面

1. 登录作业运维页面。
   a. 登录实时计算控制台。
   b. 单击页面顶部的运维。
   c. 在作业列表区域，单击作业名称下的目标作业名。
2. 在作业运维页面，单击页面顶部的TaskExecutor。

TaskExecutor在集群启动中的作用

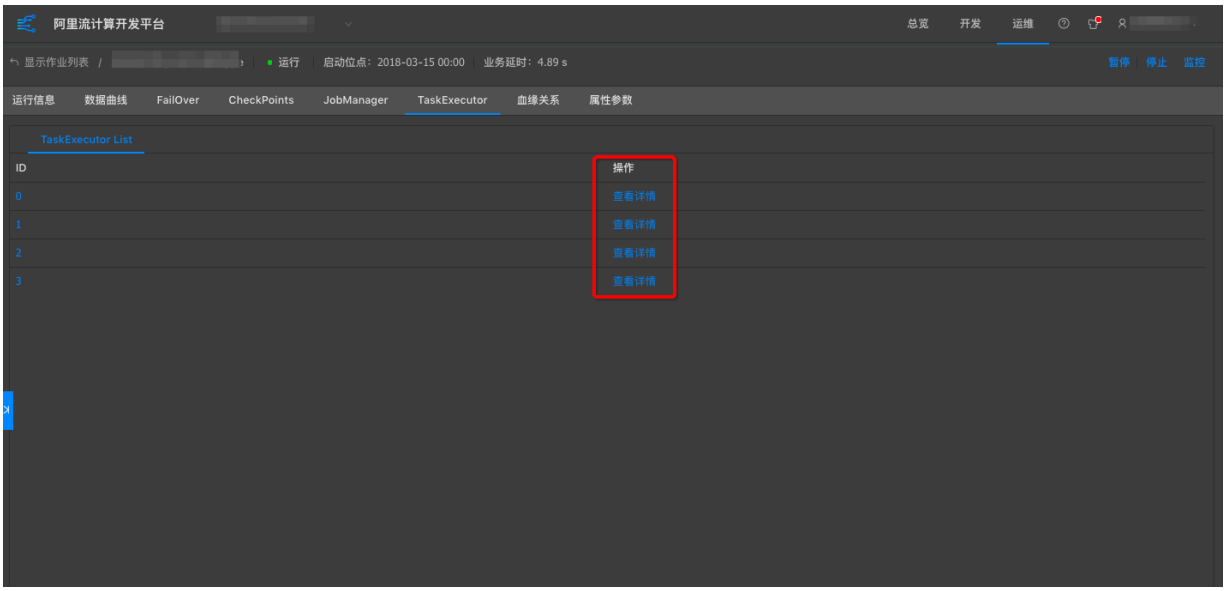
TaskExecutor是实时计算集群的启动过程不可或缺的一部分。实时计算集群的启动流程如下：

1. 实时计算集群启动一个JobManger和若干个TaskExecutor。
2. Client向JobManager提交任务。

3. JobManager向TaskExecutor分配任务。
4. TaskExecutor向JobManager汇报心跳和统计信息。

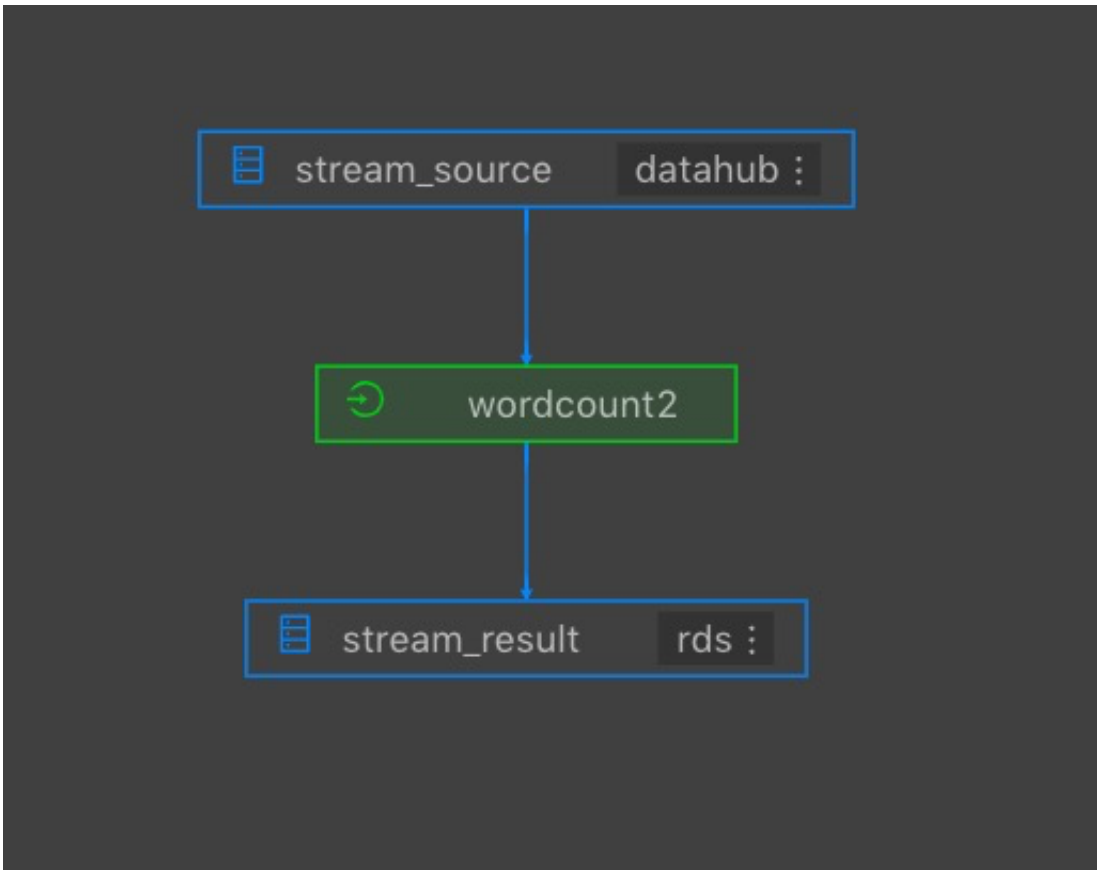
TaskExecutor界面

TaskExecutor界面为您提供Task列表以及Task详情的接口。



5.8 血缘关系

实时计算作业的血缘关系集中反映了一个实时计算作业上下游数据的依赖关系。对于作业较为复杂的上下游业务依赖，血缘关系中的数据拓扑图能够清晰地反映出上下游依赖信息。



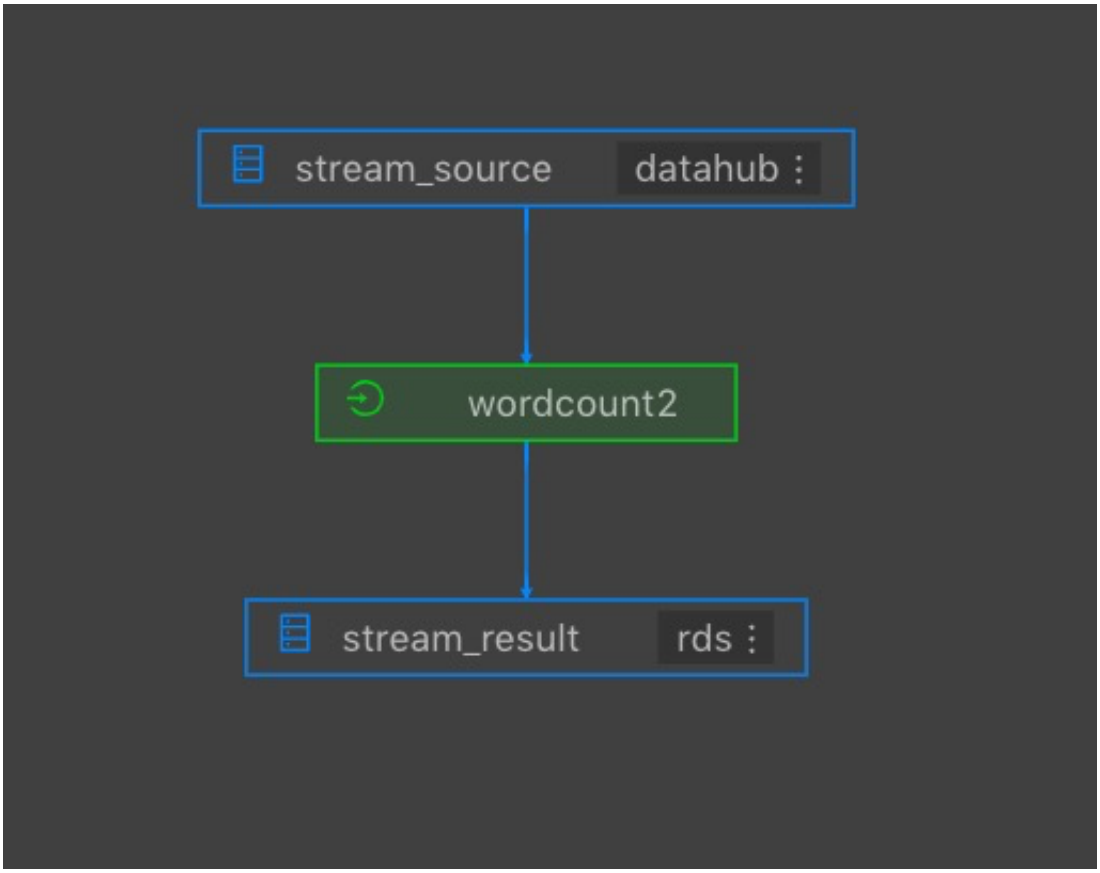
登录血缘关系页面

1. 登录作业运维页面。
   a. 登录实时计算控制台。
   b. 单击页面顶部的运维。
   c. 在作业列表区域，单击作业名称下的目标作业名。
2. 在作业运维页面，单击页面顶部的血缘关系。

数据抽样

血缘关系为作业上下游提供了数据抽样功能，该功能和数据开发页面保持一致，方便您在数据运维页面进行随时数据探测，定位问题。开启数据抽样方法如下：

1. 在作业上下游中单击表名。



2. 数据抽样页面，单击下方的抽样。

5.9 属性参数

属性参数提供了当前作业的详情信息，包括当前运行信息以及历史运行记录。

登录属性参数页面

1. 登录作业运维页面。
   a. 登录实时计算控制台。
   b. 单击页面顶部的运维。
   c. 在作业列表区域，单击作业名称下的目标作业名。
2. 在作业运维页面，单击页面顶部的属性参数。

作业代码

您可以在作业代码页面预览SQL作业代码。可单击右上角编辑作业，跳转至开发页面。

资源配置

- 资源配置页面为您展示Job运行中所涉及资源的配置信息，如CPU、MEM、并发数等。

- 开启AutoScale功能后，可在资源配置页面中查询AutoScale迭代的历史详情。

  说明:
  仅实时计算3.0.0以上版本支持AutoScale迭代的历史查询功能。

作业属性

运行属性为您展示Job的基本信息。

运行参数

运行参数为您展示包含了底层Checkpoint、启动时间、作业运行在内的作业运行参数信息。

历史记录

历史记录为您展示作业操作的所有版本信息，包括操作人员、启动位点、终止时间等。

作业参数

您可以在作业参数页面输入实时计算所支持的作业参数，例如，自定义调试阶段分割符的作业参数。

5.10 作业诊断

实时计算提供作业诊断功能方便您快速的排查作业问题。

说明:
仅运行的作业支持诊断功能。

作业诊断步骤

1. 登录实时计算控制台。
2. 单击顶部菜单栏的运维，进入作业运维界面。
3. 单击作业操作列下的诊断。

诊断指标

- Failover
  - 作业Failover：检查作业在最近30分钟是否出现Failover。
  - 作业整体[AM]Failover：监控AM是否出现Failover。

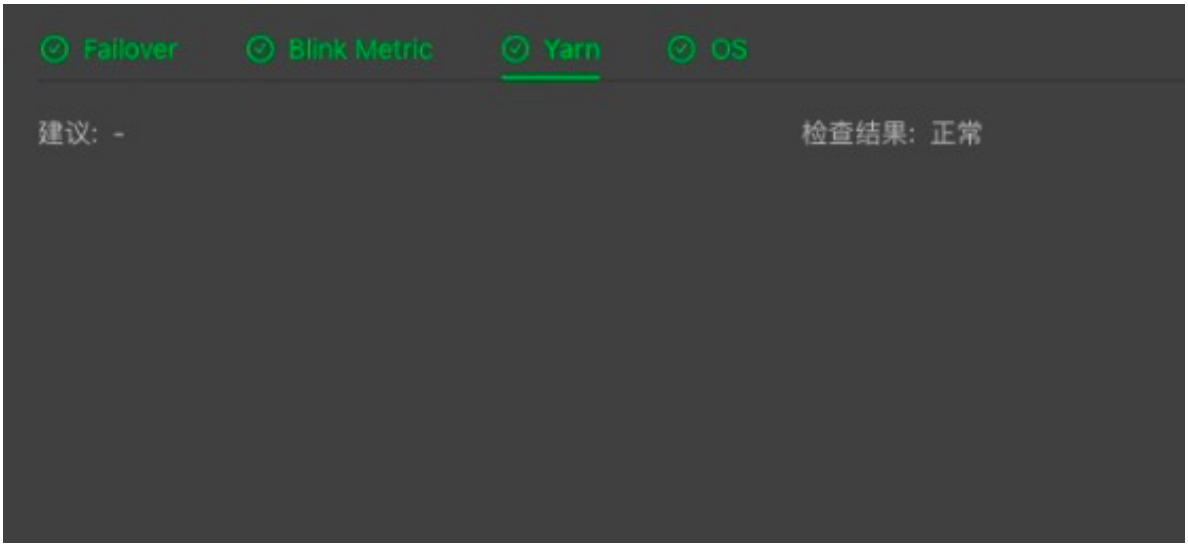
- Blink Metric

  作业延迟：检查并获取作业延时时长。出现延时时，会显示反压的节点。

  - 延时偏高：延时时长大于100秒小于200秒。
  - 延时过高：延时时长大于200秒。

- Yarn

  为您显示Yarn的检查结果。无异常时，返回结果如下图。



- OS

  为您显示OS的检查结果。无异常时，返回结果如下图。

5.11 日志下载

您可以通过编辑实时计算作业参数，自定义实时计算作业日志的下载级别和下载路径。

> **说明:**
> 仅实时计算3.2及以上版本支持日志输出级别和路径的自定义功能。

自定义日志下载级别和下载路径的步骤

1. 登录实时计算控制台。
2. 单击页面顶部的开发。
3. 在左侧导航栏的作业开发区域，双击文件夹，查找目标作业。
4. 双击目标作业，进入作业编辑页面。
5. 在作业编辑页面右侧的作业参数页面，输入log4配置信息。
6. 停止作业。作业停止步骤请参见停止。
7. 启动作业。作业启动步骤请参见#unique_64。

注意事项

- 选择和独享模式集群相同的OSS存储路径。
- 日志下载包含的日志为log4j的输出方式，不包含*system.out*日志。
- 自定义Appender目前只支持SLS和OSS，且对应的类是固定的，如下所示：
  - SLS：`log4j.appender.loghub = com.alibaba.blink.log.loghub.BlinkLogHubAppender`
  - OSS：`log4j.appender.oss=com.alibaba.blink.log.oss.BlinkOssAppender`
- 指定的SLS或OSS存储和作业所在集群可联通。
- 若自定义日志输出方式配置错误，作业通常可以成功启动，但日志打印不能按照自定义配置进行输出。

场景1

- 场景描述

  您需要将日志级别改为DEBUG，并将日志发送至OSS存储空间。

- 作业参数设置方法

  > 注意：
  > 手动配置log4j.rootLogger参数会导致实时计算平台无法查看日志信息以及排查相关问题，请谨慎使用。

```
#将总体日志级别修改为DEBUG，输出到文件及OSS存储。
log4j.rootLogger=DEBUG, file, oss

#固定写法。配置OSS的appender类。
log4j.appender.oss=com.alibaba.blink.log.oss.BlinkOssAppender

#Endpoint地址。
log4j.appender.oss.endpoint=oss-cn-hangzhou****.aliyuncs.com

#accessId。
```

```
log4j.appender.oss.accessId=U****4ZF

#accessKey。
log4j.appender.oss.accessKey=hsf****DeLw

#bucket名称。
log4j.appender.oss.bucket=et****

#定义日志存储的子目录。
log4j.appender.oss.subdir=/luk****/test/
```

说明:

完成作业参数编辑后，请重启作业。新生成的日志即可在OSS存储空间查看。

场景2

- **场景描述**

  您需要排除指定类的日志，并把错误级别日志发送至自定义的SLS存储。

- **作业参数设置方法**

```
#关闭这个包下面的日志输出。
log4j.logger.org.apache.hadoop=OFF

#固定写法。配置Loghub的appender类。

log4j.appender.loghub = com.alibaba.blink.log.loghub.BlinkLogHu
bAppender

#仅将ERROR级别日志发往SLS。
log4j.appender.loghub.Threshold = ERROR
#SLS中的project名称。
log4j.appender.loghub.projectName = blink-errdumpsls-test

#SLS中的logstore名称。
log4j.appender.loghub.logstore = logstore-3

#SLS的Endpoint地址。
log4j.appender.loghub.endpoint = http://cn-shanghai****.sls.aliyuncs
.com
#accessKeyId。
log4j.appender.loghub.accessKeyId = Tq****WR

#accessKey。
log4j.appender.loghub.accessKey = MJ****nfVx
```

6 监控报警

本文为您介绍实时计算监控报警的操作流程以及报警规则的创建步骤。

什么是云监控报警服务

云监控服务能够收集阿里云资源或您自定义的监控指标、探测服务可用性以及针对指标设置警报，让您全面了解阿里云上的资源使用情况、业务的运行状况和健康度，并及时接收异常报警，保证应用程序顺畅运行。

查看监控报警信息

1. 登录实时计算控制台。
2. 单击页面顶部的运维。
3. 在实时计算运维界面，单击目标作业名。
4. 在目标作业运维信息页面的右上角，单击监控。
5. 在监控图标页面，查看作业的监控指标。

创建报警规则

创建报警规则详情，参见#unique_81/unique_81_Connect_42_section_nhz_ccf_zdb。

> 说明:
> Failover Rate表示最近1分钟平均每秒Failover的次数。例如，最近1分钟Failover了1次，则Failover Rate=1/60=0.01667。

7 作业调优

7.1 作业调优概述

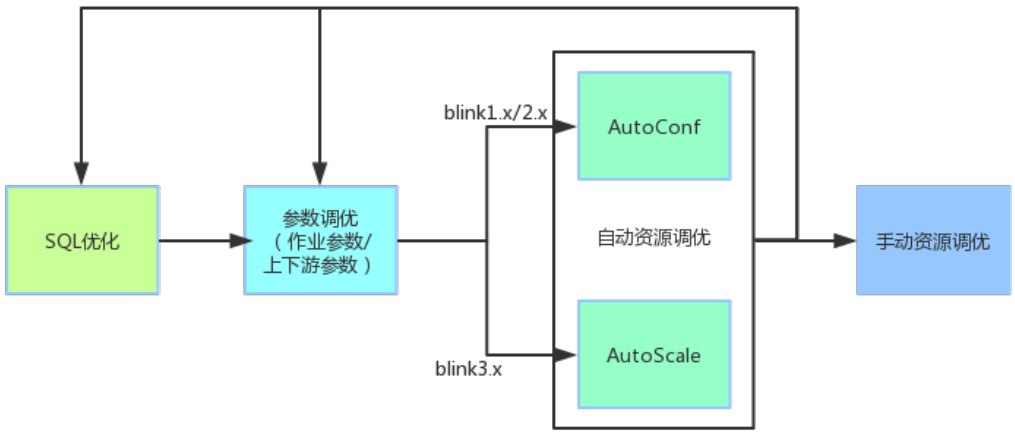
实时计算作业开发过程中，完成了业务逻辑实现、作业上线和启动运行后，为了满足实时计算作业的性能需求，还需对作业进行调优。

作业调优的目的和衡量标准

- 作业正常启动和运行。
- 作业具备合理的延时和吞吐，满足业务性能需求。
- 高效的使用资源，降低成本。

作业调优步骤

建议的作业调优顺序和步骤如下图。



1. SQL优化

SQL优化即根据业务需求选择合适的SQL实现方式，包括但不限于聚合优化、数据热点优化、TOPN优化、使用内置函数、高效去重、慎用正则函数等，具体可参见高性能Flink SQL优化技巧。

2. 参数调优

   - 作业参数调优

     选择底层优化策略，例如设置minibatch优化State访问策略等，具体可参见#unique_85/unique_85_Connect_42_section_sxj_yjm_bgb。

   - 上下游存储参数调优

     优化上下游存储读写，例如，攒批读写提升吞吐和设置Cache策略提升维表JOIN效率等，具体可参考#unique_85/unique_85_Connect_42_section_q6p_kx0_qrm。

3. 自动资源调优

   为了简化作业调优，实时计算开发了自动配置调优功能。建议优先通过自动配置调优功能进行作业调优。

   - blink1.x（自blink 1.6.4开始）和blink 2.x版本请参见#unique_17。
   - blink 3.x版本请参见#unique_55。

4. 手动资源调优或再次进行调优

   - 手动资源调优

     自动配置调优无法满足需求的情况下，可针对性的进行#unique_85。

     > 说明:
     > blink 3.x版本提供反压检测功能协助您判断性能反压点。

   - 再次进行调优

     如果一次调优后的结果不能满足您的业务需求，可按照步骤1~4，再次进行调优。

7.2 高性能Flink SQL优化技巧

本文为您介绍提升性能的Flink SQL推荐写法、推荐配置和推荐函数。

Group Aggregate优化技巧

- 开启MicroBatch或MiniBatch （提升吞吐）

  MicroBatch和MiniBatch都是微批处理，只是微批的触发机制略有不同。原理同样是缓存一定的数据后再触发处理，以减少对State的访问，从而提升吞吐和减少数据的输出量。

  MiniBatch主要依靠在每个Task上注册的Timer线程来触发微批，需要消耗一定的线程调度性能。MicroBatch是MiniBatch的升级版，主要基于事件消息来触发微批，事件消息会按您指

定的时间间隔在源头插入。MicroBatch在元素序列化效率、反压表现、吞吐和延迟性能上都要优于胜于MiniBatch。

- 适用场景

  微批处理是增加延迟来换取高吞吐的策略，如果您有超低延迟的要求，不建议开启微批处理。通常对于聚合的场景，微批处理可以显著的提升系统性能，建议开启。

   说明:
  MicroBatch模式也能解决两级聚合数据抖动问题。

- 开启方式

  MicroBatch和MiniBatch默认关闭，开启方式：

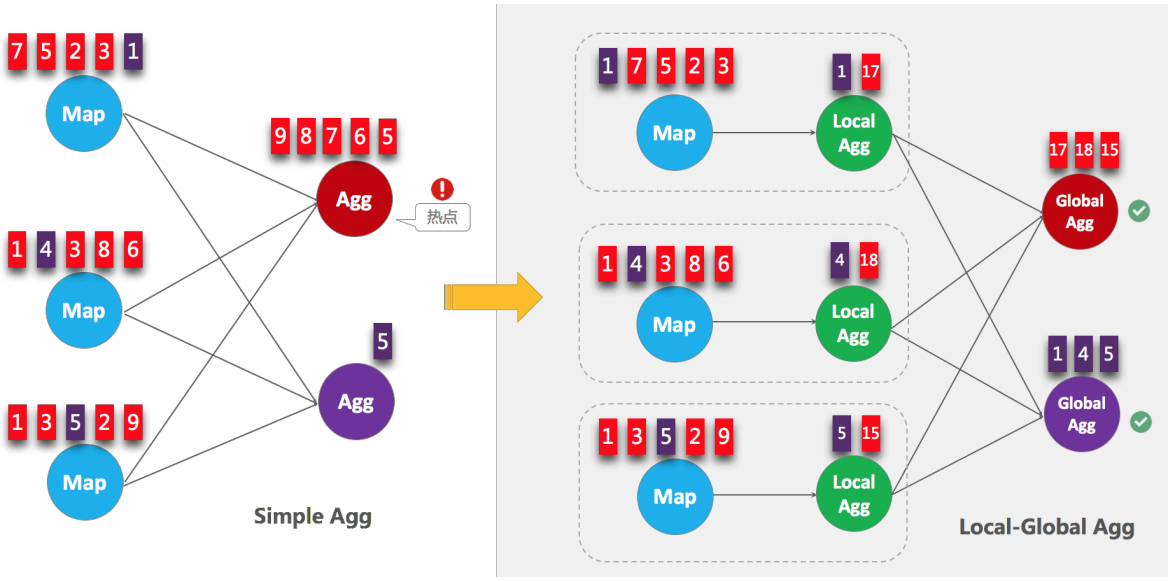
```
#3.2及以上版本开启Window miniBatch方法（3.2及以上版本默认不开启Window
miniBatch。）
sql.exec.mini-batch.window.enabled=true
# 批量输出的间隔时间，使用microBatch策略时需要加上该配置，且建议和blink.
miniBatch.allowLatencyMs保持一致。
blink.microBatch.allowLatencyMs=5000
# 使用microBatch时需要保留以下两个miniBatch配置。
blink.miniBatch.allowLatencyMBs=5000
# 防止OOM设置每个批次最多缓存数据的条数。
blink.miniBatch.size=20000
```

· 开启LocalGlobal（解决常见数据热点问题）

  LocalGlobal优化即将原先的Aggregate分成Local+Global两阶段聚合，也就是在MapReduce模型中熟知的Combine+Reduce处理模式。第一阶段在上游节点本地攒一批数据

进行聚合（localAgg），并输出这次微批的增量值（Accumulator），第二阶段再将收到的Accumulator合并（merge），得到最终的结果（globalAgg）。

LocalGlobal本质上能够靠localAgg的聚合筛除部分倾斜数据，从而降低globalAgg的热点，提升性能。您可以结合下图理解LocalGlobal如何解决数据倾斜的问题。



- 适用场景

  LocalGlobal适用于提升如SUM、COUNT、MAX、MIN和AVG等普通聚合的性能，以及解决这些场景下的数据热点问题。

  > 说明:
  > 开启LocalGlobal需要UDAF实现`merge`方法。

- 开启方式

  在 实时计算`2.0`版本开始，LocalGlobal是默认开启的，参数是：`blink.localAgg.enabled=true`，但是需要在microbatch或minibatch开启的前提下才能生效。

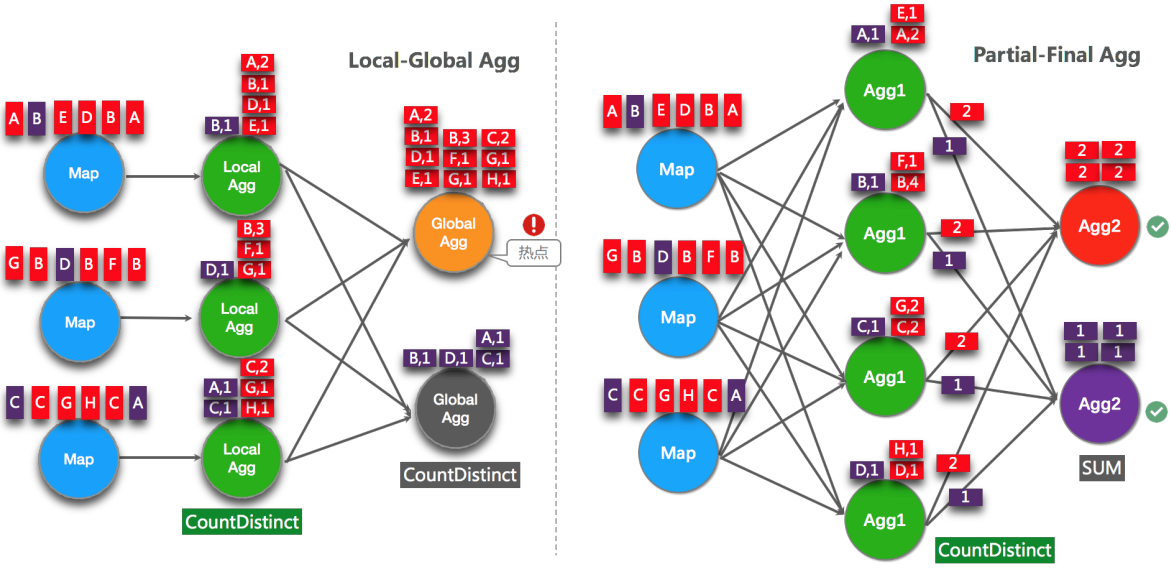
- 如何判断是否生效

  观察最终生成的拓扑图的节点名字中是否包含GlobalGroupAggregate或LocalGroupAggregate。

- 开启PartialFinal（解决COUNT DISTINCT热点问题）

  上述的LocalGlobal优化能针对常见普通聚合有较好的效果（如SUM、COUNT、MAX、MIN和AVG）。但是对于COUNT DISTINCT收效不明显，原因是COUNT DISTINCT在local聚合时，对于DISTINCT KEY的去重率不高，导致在Global节点仍然存在热点。

  实时计算历史版本中，用户为了解决COUNT DISTINCT的热点问题，通常会手动改写成两层聚合（增加按distinct key取模的打散层），自`2.2.0`版本开始，实时计算提供了COUNT DISTINCT自动打散，即PartialFinal优化，您无需自行改写为两层聚合。PartialFinal和LocalGlobal的原理对比参见下图。



  - 适用场景

    使用COUNT DISTINCT且聚合节点性能无法满足时。

    > **说明:**
    >
    > ■ PartialFinal优化方法不能在包含UDAF的Flink SQL中使用。
    >
    > ■ 数据量不大的情况下不建议使用PartialFinal优化方法。PartialFinal优化会自动打散成两层聚合，引入额外的网络Shuffle，在数据量不大的情况下，可能反而会浪费资源。

  - 开启方式

    默认不开启，使用参数显式开启`blink.partialAgg.enabled=true`

  - 如何判断是否生效

    观察最终生成的拓扑图的节点名中是否包含Expand节点，或者原来一层的聚合变成了两层的聚合。

- 改写成AGG WITH FILTER语法（提升大量COUNT DISTINCT场景性能）

  > 说明:
  > 仅支持实时计算2.2.2及以上版本。

  统计作业需要计算各种维度的UV，例如全网UV、来自手机客户端的UV、来自PC的UV等等。建议使用更标准的AGG WITH FILTER语法来代替CASE WHEN实现多维度统计的功能。实时计算目前的SQL优化器能分析出Filter 参数，从而同一个字段上计算不同条件下的COUNT DISTINCT能共享State，减少对State的读写操作。性能测试中，使用AGG WITH FILTER语法来代替CASE WHEN能够能够使性能提高1倍。

  - 适用场景

    建议用户将AGG WITH CASE WHEN的语法都替换成AGG WITH FILTER的语法，尤其是对同一个字段上计算不同条件下的COUNT DISTINCT结果时有极大的性能提升。

  - 原始写法

    ```
    COUNT(distinct visitor_id) as UV1 , COUNT(distinct case when
    is_wireless='y' then visitor_id else null end) as UV2
    ```

  - 优化写法

    ```
    COUNT(distinct visitor_id) as UV1 , COUNT(distinct visitor_id)
    filter (where is_wireless='y') as UV2
    ```

TopN优化技巧

- TopN算法

  当TopN的输入是非更新流（例如Source），TopN只有一种算法AppendRank。当TopN的输入是更新流时（例如经过了AGG/JOIN计算），TopN有3种算法，性能从高到低分别是：UpdateFastRank 、 UnaryUpdateRank和RetractRank。算法名字会显示在拓扑图的节点名字上。

  - UpdateFastRank ： 最优算法，需要具备2个条件：1.输入流有PK信息。2.排序字段的更新是单调的，且单调方向与排序方向相反。例如，order by count/count_distinct/sum（正数） desc。

    > 说明:

order by sum（正数）desc时，要加上正数的过滤条件。且已知sum的参数不可能有负数，那么需要加上过滤条件从而告诉优化器这个信息，才能优化出UpdateFastRank算法（仅支持实时计算2.2.2及以上版本），如下所示。

```
SELECT cate_id, seller_id, stat_date, pay_ord_amt  # 不输出rownum
字段，能减小对结果表的输出量。
FROM (SELECT*
      ROW_NUMBER ()OVER(PARTITIONBY cate_id, stat_date   # 注意要
有时间字段，否则state过期会导致数据错乱。
ORDER BY pay_ord_amt DESC## 根据上游sum结果排序)AS rownum。
  FROM (SELECT cate_id, seller_id, stat_date, # 重点。声明sum的参数
都是正数，所以sum的结果是单调递增的，所以TopN能用优化算法。
sum(total_fee) filter (where total_fee >=0) as pay_ord_amt
    FROM WHERE total_fee >=0 GROUP BY cate_name, seller_id,
stat_date)WHERE rownum <=100))
```

- UnaryUpdateRank：仅次于UpdateFastRank的算法。需要具备1个条件：输入流存在PK信息。例如，order by avg。
- RetractRank：普通算法，性能最差，不建议在生产环境使用该算法。请检查输入流是否存在PK信息，如果存在可进行UnaryUpdateRank或UpdateFastRank优化。

· TopN优化方法

- 无排名优化

  TopN的输出结果无需要显示rownum值，仅需在最终前端显式时进行1次排序，极大地减少输入结果表的数据量。 无排名优化方法详情请参见#unique_86。

- 增加TopN的cache大小

  TopN为了提升性能有一个state cache层，cache层能提升对state的访问效率。TopN的cache命中率的计算公式为：

  ```
  cache_hit = cache_size*parallelism/top_n/partition_key_num
  ```

  例如，Top100配置缓存10000条，并发50，当您的patitionBy的key维度较大时，如10万级别时，cache命中率只有10000*50/100/100000=5%，命中率会很低，导致大量的请求都会击中state（磁盘），性能会大幅下降。因此当partitionKey维度特别大时，可以适当加大TopN的cache size，相对应的也建议适当加大TopN节点的heap memory（参见#unique_69）。

  ```
  ##默认10000条，调整TopN cahce到20万，那么理论命中率能达200000*50/100/
  100000 = 100%。
  blink.topn.cache.size=200000
  ```

- partitionBy的字段中要有时间类字段

  例如每天的排名，要带上day字段。否则TopN的结果到最后会由于state ttl有错乱。

高效去重方案

> **说明:**
> 仅实时计算3.2.1及以上版本支持高效去重方案。

实时计算的源数据在部分场景中存在重复数据，去重成为了用户经常反馈的需求。实时计算有保留第一条（Deduplicate Keep FirstRow）和保留最后一条（Deduplicate Keep LastRow）2种去重方案。

- 语法

  由于SQL上没有直接支持去重的语法，还要灵活的保留第一条或保留最后一条。因此我们使用了SQL的ROW_NUMBER OVER WINDOW功能来实现去重语法。去重本质上是一种特殊的TopN。

```
SELECT *
FROM (
   SELECT *,
    ROW_NUMBER() OVER ([PARTITION BY col1[, col2..]
     ORDER BY timeAttributeCol [asc|desc]) AS rownum
   FROM table_name)
WHERE rownum = 1
```

参数	说明
ROW_NUMBER()	计算行号的OVER窗口函数。行号从1开始计算。
PARTITION BY col1[, col2..]	可选。指定分区的列，即去重的KEYS。
ORDER BY timeAttributeCol [asc|desc])	指定排序的列，必须是一个#unique_87的字段（即 proctime或rowtime）。可以指定顺序（Keep FirstRow）或者倒序 （Keep LastRow）。
rownum	仅支持`rownum=1`或`rownum<=1`。

如上语法所示，去重需要两层Query：

1. 使用`ROW_NUMBER()`窗口函数来对数据根据时间属性列进行排序并标上排名。

   > **说明:**
   > - 当排序字段是proctime列时，Flink就会按照系统时间去重，其每次运行的结果是不确定的。

- 当排序字段是rowtime列时，Flink就会按照业务时间去重，其每次运行的结果是确定的。

2. 对排名进行过滤，只取第一条，达到了去重的目的。

说明:

排序方向可以是按照时间列的顺序，也可以是倒序：

- Deduplicate Keep FirstRow：顺序并取第一条行数据。
- Deduplicate Keep LastRow：倒序并取第一条行数据。

· Deduplicate Keep FirstRow

保留首行的去重策略：保留KEY下第一条出现的数据，之后出现该KEY下的数据会被丢弃掉。因为STATE中只存储了KEY数据，所以性能较优，示例如下。

```
SELECT *
FROM (
  SELECT *,
    ROW_NUMBER() OVER (PARTITION BY b ORDER BY proctime) as rowNum
  FROM T
)
WHERE rowNum = 1
```

说明:

以上示例是将T表按照b字段进行去重，并按照系统时间保留第一条数据。proctime在这里是源表T中的一个具有Processing Time属性的字段。如果您按照系统时间去重，也可以将proctime字段简化 `PROCTIME()` 函数调用，可以省略proctime字段的声明。

· Deduplicate Keep LastRow

保留末行的去重策略：保留KEY下最后一条出现的数据。保留末行的去重策略性能略优于LAST_VALUE函数，示例如下。

```
SELECT *
FROM (
  SELECT *,
    ROW_NUMBER() OVER (PARTITION BY b, d ORDER BY rowtime DESC) as
rowNum
  FROM T
)
WHERE rowNum = 1
```

说明:

以上示例是将T表按照b和d字段进行去重，并按照业务时间保留最后一条数据。rowtime在这里是源表T中的一个具有Event Time属性的字段。

高效的内置函数

- 使用内置函数替换自定义函数

  实时计算的内置函数在持续的优化当中，请尽量使用内部函数提到自定义函数。实时计算2.0版本对内置函数主要进行了如下优化：

  - 优化数据序列化和反序列化的耗时。
  - 新增直接对字节单位进行操作的功能。
- KEY VALUE函数使用单字符的分隔符

  KEY VALUE 的签名是：`KEYVALUE(content, keyValueSplit, keySplit, keyName)`，当keyValueSplit和KeySplit是单字符时，如`:`、`,`会使用优化的算法，会在二进制数据上直接寻找所需的keyName 的值，而不会将整个content做切分。性能约有30%提升。
- 多KEY VALUE场景使用MULTI_KEYVALUE

  说明:

  仅支持实时计算`-2.2.2`及以上版本

  如果在query中有对同一个content做大量KEY VALUE 的操作，例如Content中包含10个key-value对，希望把10个Value的值都取出来作为字段。用户经常会写10个KEY VALUE函数，那么就会对Content 做10次解析。在这种场景建议使用MULTI_KEYVALUE，这是一个表值函数。使用该函数可以只对Content 做一次Split解析。性能约有50%~100%的性能提升。
- LIKE 操作注意事项

  - 如果需要进行startWith操作，使用`LIKE 'xxx%'`。
  - 如果需要进行endWith操作，使用`LIKE '%xxx'`。
  - 如果需要进行contains操作，使用`LIKE '%xxx%'`。
  - 如果需要进行equals操作，使用`LIKE 'xxx'`，等价于`str = 'xxx'`。
  - 如果需要匹配 `_` 字符，请注意要完成转义`LIKE '%seller/id%' ESCAPE '/'`。`_` 在SQL中属于单字符通配符，能匹配任何字符。如果声明为 `LIKE '%seller_id%'`，则不单会匹配`seller_id`还会匹配`seller#id`、`sellerxid`或`seller1id` 等，导致结果错误。

- 慎用正则函数（REGEXP）

  正则表达式是非常耗时的操作，对比加减乘除通常有百倍的性能开销，而且正则表达式在某些极端情况下可能会进入无限循环，导致作业卡住。建议使用LIKE。正则函数包括：

  - #unique_89
  - #unique_90
  - #unique_91

网络传输的优化

目前常见的Partitioner 策略包括：

- KeyGroup/Hash：根据指定的key分配。
- Rebalance：轮询分配给各个channel。
- Dynamic-Rebalance：根据下游负载情况动态选择分配给负载较低的channel。
- Forward：未chain一起时，同Rebalance。chain一起时是一对一分配。
- Rescale：上游与下游一对多或多对一。
- 使用Dynamic-Rebalance替代Rebalance

  Dynamic-Rebalance可以根据当前各subpartition中堆积的buffer的数量，选择负载较轻的subpartition进行写入，从而实现动态的负载均衡。相比于静态的rebalance策略，在下游各任务计算能力不均衡时，可以使各任务相对负载更加均衡，从而提高整个作业的性能。例如，在使用rebalance时，发现下游各个并发负载不均衡时，可以考虑使用 Dynamic-Rebalance。参数：`task.dynamic.rebalance.enabled=true`， 默认关闭。
- 使用Rescale替代Rebalance

  说明:

  仅支持实时计算`2.2.2`及以上版本。

  例如，上游是5个并发，下游是10个并发。当使用Rebalance时，上游每个并发会轮询发给下游10个并发。当使用Rescale时，上游每个并发只需轮询发给下游2个并发。因为 channel个数变少了，subpartition的buffer填充速度能变快，能提高网络效率。当上游的数据比较均匀时，且上下游的并发数成比例时，可以使用Rescale替换Rebalance。参数：`enable.rescale.shuffling=true`，默认关闭。

推荐的优化配置方案

综上所述，作业建议使用如下的推荐配置：

```
# EXACTLY_ONCE语义
blink.checkpoint.mode=EXACTLY_ONCE
```

```
# checkpoint间隔时间，单位毫秒。
blink.checkpoint.interval.ms=180000
blink.checkpoint.timeout.ms=600000
# 2.x使用niagara作为statebackend，以及设定state数据生命周期，单位毫秒。
state.backend.type=niagara
state.backend.niagara.ttl.ms=129600000
# 2.x开启5秒的microbatch（使用窗口函数不能设置该参数）。
blink.microBatch.allowLatencyMs=5000
# 整个Job允许的延迟。
blink.miniBatch.allowLatencyMs=5000
# 单个batch的size
blink.miniBatch.size=20000
# local 优化，2.x默认已经开启，1.6.4需手动开启。
blink.localAgg.enabled=true
# 2.x开启partial优化，解决COUNT DISTINCT热点。
blink.partialAgg.enabled=true
# union all优化。
blink.forbid.unionall.as.breakpoint.in.subsection.optimization=true
# object reuse优化，默认已开启。
#blink.object.reuse=true
# GC优化（SLS做源表不能设置该参数）
blink.job.option=-yD heartbeat.timeout=180000 -yD env.java.opts='-
verbose:gc -XX:NewRatio=3 -XX:+PrintGCDetails -XX:+PrintGCDateStamps -
XX:ParallelGCThreads=4'
# 时区设置
blink.job.timeZone=Asia/Shanghai
```

7.3 AutoConf自动配置调优

为了增加用户的体验度，实时计算团队开发了自动配置调优（AutoConf）功能。

说明:

AutoConf自动配置调优功能支持blink 1.0和2.0版本。

背景及功能范围

在您作业的各个算子和流作业上下游性能达标和稳定的前提下，自动配置调优功能可以帮助您更合理的分配各算子的资源和并发度等配置。 全局优化您的作业，调节作业吞吐量不足、作业全链路的反压等性能调优的问题。

出现下列情况时，自动配置调优功能可以作出优化，但无法彻底解决流作业的性能瓶颈，需要您自行解决或联系实时计算产品支持团队解决性能瓶颈。

- 流作业上下游有性能问题。
  - 流作业上游的数据Source存在性能问题。例如，DataHub分区不足、MQ吞吐不够等需要您扩大相应Source的分区。
  - 流作业下游的数据Sink存在性能问题。例如，RDS死锁等。
- 流作业的自定义函数 （UDF、UDAF和UDTF） 有性能问题。

操作

- 新作业

1. 上线作业

a. 完成SQL开发，通过语法检查后，单击上线，即可出现如下上线新版本界面。

b. 单击智能CU配置。第一次不需要指定CU数直接使用系统默认配置。

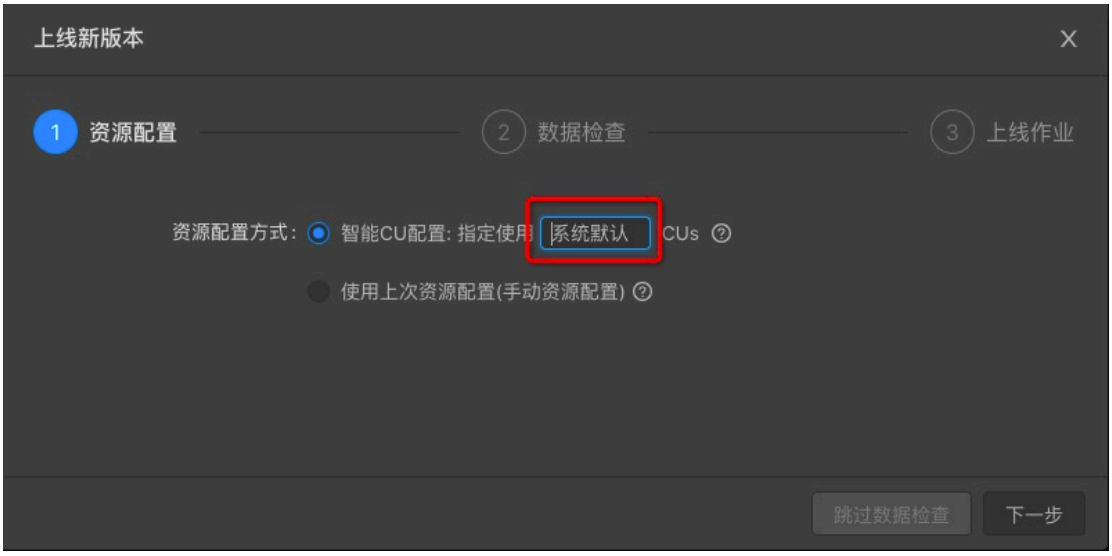
- 智能配置：指定使用CU AutoConf算法会基于系统默认配置，生成CU数，进行优化资源配置。如果是第一次运行，算法会根据经验值生成一份初始配置。建议作业运行

了5~10分钟以上，确认Source RPS等Metrics稳定2~3分钟后，再使用智能配置，重复3~5次才能调优出最佳的配置。

- 使用上次资源配置：即使用最近一次保存的资源配置。如果上一次是智能配置的，就使用上一次智能配置的结果。如果上一次是手工配置的，就使用上次手工配置的结果。

2. 使用默认配置启动作业

a. 使用默认配置启动作业，出现如下的界面。



b. 启动作业。

示例如下。第一次默认配置生成的资源配置为71个CU。

说明:

请您确保作业已经运行10分钟以上，并且Source RPS等数据曲线稳定2-3分钟后，再使用智能CU配置。

3. 使用智能配置启动作业

a. 资源调优

例如，您手动配置40CU，使用智能模式启动。40的CU数是您自行指定调整的，您可以根据具体的作业情况适当的增加或者是减小CU数，来达到资源调优的目的。

- CU的最小配置

  CU的最小配置建议不小于默认配置总数的50%，CU数不能小于1CU。假设智能配置默认CU数为71，则建议最小CU数为36CU。 71*50% = 35.5CU。

- CU增加数量

  假如无法满足作业理想的吞吐量就需要增加适量的CU数。每次增加的CU数，建议是上一次CU总数的30%以上。例如，上一次配置是10CU，下次就需要增加到13CU。

- 可多次调优

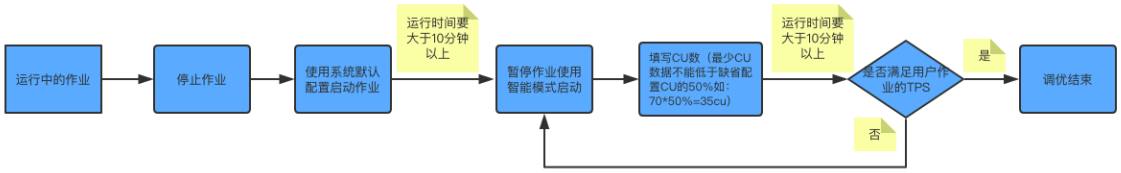
  如果第一次调优不满足您的需求，可以调优多次。可以根据每次调优后Job的状态来增加或减少资源数。

b. 调优后的结果如下图。

说明：

如果是新任务，请不要选择使用上次资源配置，否则会报错。

- **已存在作业**
  - **调优流程示意图**



说明：

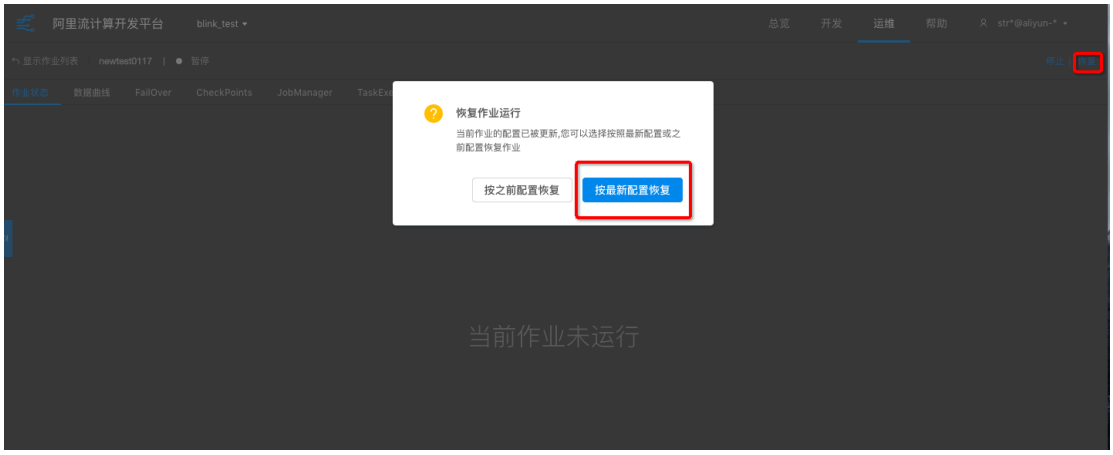
- **已存在的作业在进行调优前，您一定要检查是否是有状态的计算。因为在调优的过程中可能会清除之前作业保存的状态，请您谨慎操作。**
- **当您的作业有改动时（例如，您对作业的SQL有修改，或更改了实时计算版本），自动配置调优功能不保证能按照之前的运行信息进行调优。原因是这些改动会导致拓扑信息变化，造成数据曲线无法对应和状态无法复用等问题。自动配置调优功能无法根据运行历史**

信息作出调优判断。此时再使用自动配置调优会报异常。这时您需要将改动后的任务当做新任务，重新进行操作。

- 调优流程

1. 暂停任务。

2. 重复新作业的调优步骤，使用最新的配置启动作业。



常见问题

以下几点可能会影响自动调优的准确性：

- 任务运行的时间较短，会造成采样得到的有用信息较少，会影响AutoConf算法的效果。建议延长运行时间，确认Source RPS等数据曲线稳定2~3分钟后即可。
- 任务运行有异常（Failover），会影响结果的准确性。建议用户检查和修复Failover的问题。
- 任务的数据量比较少，会影响结果的准确性。建议回追足够多的历史数据。
- 影响的因素有很多，自动调优AutoConf不能保证下一次生成的配置一定比上一次的好。如果还不能满足需求，用户参考手动配置调优，进行手动调优。

调优建议

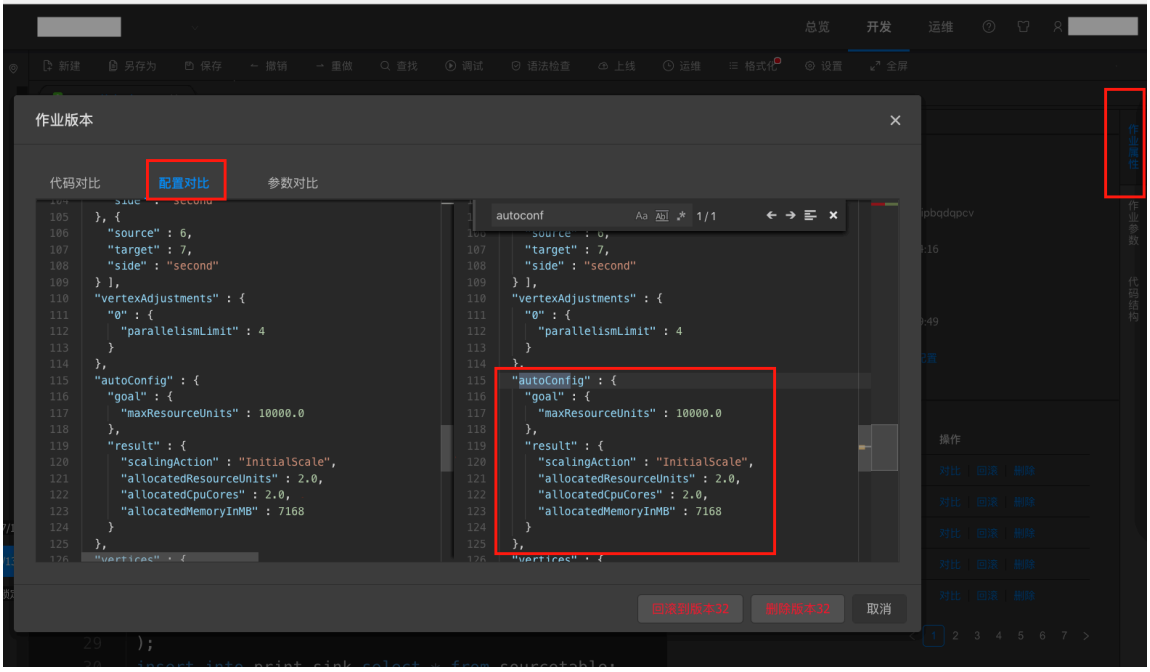
- 每次触发智能配置前任务稳定运行超过10分钟。这样有利于AutoConf准确搜集的任务运行时的指标信息。
- AutoConf可能需要3~5次迭代才能见效。
- 使用AutoConf时，您可以设置让任务回追数据甚至造成反压。这样会更有利于快速体现调优成功。

如何判断自动配置调优功能生效或出现问题?

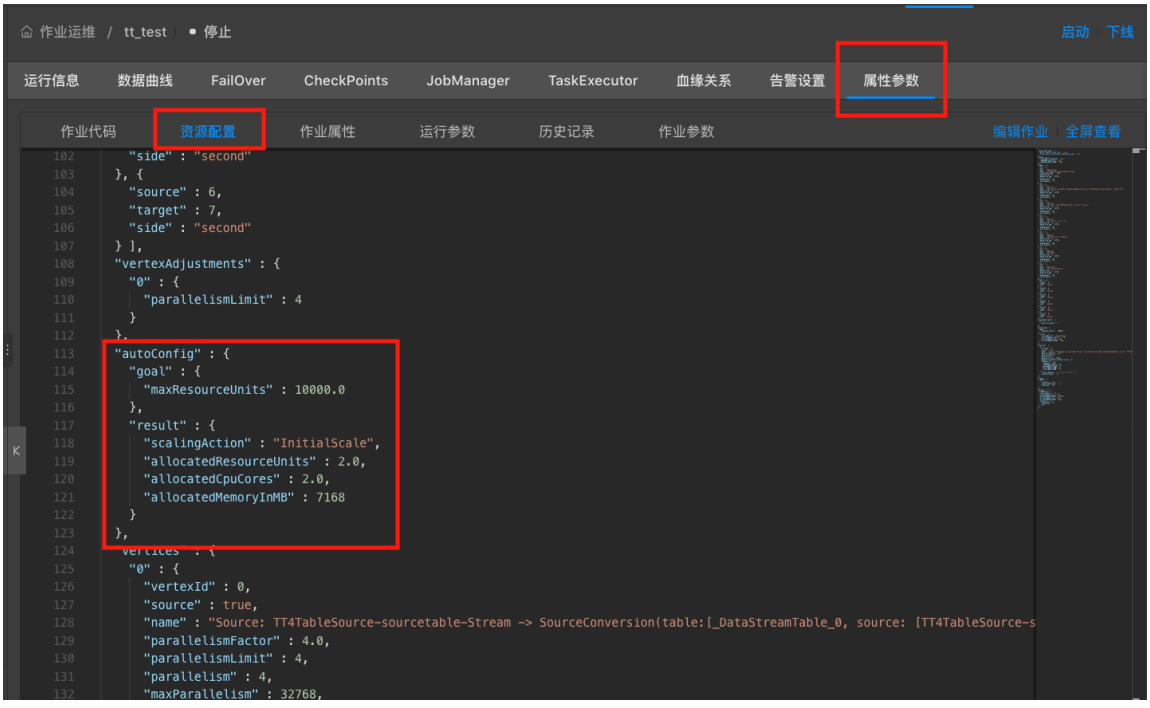
自动配置调优功能通过JSON配置文件与实时计算交互。您在调优后，可以通过查看JSON配置文件了解自动配置调优功能的运行情况。

- 查看JSON配置文件的两种方式

1. 通过作业编辑界面，如下图。



2. 通过作业运维界面，如下图。



- JSON配置解释

```
"autoconfig" : {
    "goal": {  // AutoConf 目标
        "maxResourceUnits": 10000.0,  // 单个Blink作业最大可用CU数，不能修改，查看时可忽略。
        "targetResoureUnits": 20.0  // 用户指定CU数。用户指定为20CU。
    },
    "result" : {  // AutoConf 结果。这里很重要
```

```
      "scalingAction" : "ScaleToTargetResource",  // AutoConf的运行行
动 *
      "allocatedResourceUnits" : 18.5, // AutoConf分配的总资源。
      "allocatedCpuCores" : 18.5,      // AutoConf分配的总CPU。
      "allocatedMemoryInMB" : 40960    // AutoConf分配的总内存。
      "messages" : "xxxx"  // 很重要。 *
    }
}
```

- scalingAction：`InitialScale`代表初次运行，`ScaleToTargetResource`代表非初次运行。
- 如果没有messages，代表运行正常。如果有messages，代表需要分析：messages有两种，如下所示：
  - warning 提示：表示正常运行情况但有潜在问题，需要用户注意，如source的分区不足等。
  - error或者exception提示，常伴有`Previous job statistics and configuration will be used`，代表AutoConf失败。失败也有两种原因：
    - 用户作业或blink版本有修改，AutoConf无法复用以前的信息。
    - 有exception代表AutoConf遇到问题，需要跟据信息、日志等综合分析。如没有足够的信息，请提交工单。

异常信息问题

IllegalStateException异常

出现如下的异常说明内部状态state无法复用，需要停止任务清除状态后重追数据。

如果无法切到备链路，担心对线上业务有影响，可以在开发界面右侧的作业属性里面选择上个版本进行回滚，等到业务低峰期的时候再重追数据。

```
java.lang.IllegalStateException: Could not initialize keyed state
backend.
    at org.apache.flink.streaming.api.operators.AbstractStreamOperator
.initKeyedState(AbstractStreamOperator.java:687)
    at org.apache.flink.streaming.api.operators.AbstractStreamOperator
.initializeState(AbstractStreamOperator.java:275)
    at org.apache.flink.streaming.runtime.tasks.StreamTask.initialize
Operators(StreamTask.java:870)
    at org.apache.flink.streaming.runtime.tasks.StreamTask.initialize
State(StreamTask.java:856)
    at org.apache.flink.streaming.runtime.tasks.StreamTask.invoke(
StreamTask.java:292)
    at org.apache.flink.runtime.taskmanager.Task.run(Task.java:762)
    at java.lang.Thread.run(Thread.java:834)
Caused by: org.apache.flink.api.common.typeutils.SerializationExcepti
on: Cannot serialize/deserialize the object.
    at com.alibaba.blink.contrib.streaming.state.AbstractRocksDBRawSe
condaryState.deserializeStateEntry(AbstractRocksDBRawSecondaryState.
java:167)
```

```
    at com.alibaba.blink.contrib.streaming.state.RocksDBIncrementalRe
storeOperation.restoreRawStateData(RocksDBIncrementalRestoreOperation.
java:425)
    at com.alibaba.blink.contrib.streaming.state.RocksDBIncrementalRe
storeOperation.restore(RocksDBIncrementalRestoreOperation.java:119)
    at com.alibaba.blink.contrib.streaming.state.RocksDBKeyedStateBac
kend.restore(RocksDBKeyedStateBackend.java:216)
    at org.apache.flink.streaming.api.operators.AbstractStreamOperator
.createKeyedStateBackend(AbstractStreamOperator.java:986)
    at org.apache.flink.streaming.api.operators.AbstractStreamOperator
.initKeyedState(AbstractStreamOperator.java:675)
    ... 6 more
Caused by: java.io.EOFException
    at java.io.DataInputStream.readUnsignedByte(DataInputStream.java:
290)
    at org.apache.flink.types.StringValue.readString(StringValue.java:
770)
    at org.apache.flink.api.common.typeutils.base.StringSerializer.
deserialize(StringSerializer.java:69)
    at org.apache.flink.api.common.typeutils.base.StringSerializer.
deserialize(StringSerializer.java:28)
    at org.apache.flink.api.java.typeutils.runtime.RowSerializer.
deserialize(RowSerializer.java:169)
    at org.apache.flink.api.java.typeutils.runtime.RowSerializer.
deserialize(RowSerializer.java:38)
    at com.alibaba.blink.contrib.streaming.state.AbstractRocksDBRawSe
condaryState.deserializeStateEntry(AbstractRocksDBRawSecondaryState.
java:162)
    ... 11 more
```

7.4 AutoScale自动配置调优

为了解决您使用AutoConf自动配置调优功能时频繁启停作业的问题，实时计算3.0及以上版本提供了AutoScale自动配置调优功能。作业启动后，系统会根据资源配置规则，自动进行作业的调优，直到满足设定的调优目标，全程无需人工介入。

说明:

- AutoScale自动配置调优功能仅支持实时计算3.0及以上版本。
- 升级实时计算3.0版本前，需要删除所有低于3.0版本的PlanJson，并重新获取配置资源。
- 升级实时计算3.0版本前，需要删除所有低于3.0版本的PlanJson，并重新获取配置资源。

启动AutoScale自动配置调优

作业上线的过程中即可完成AutoScale自动配置调优的开启。

1. 登录作业编辑页面。
   a. 登录实时计算控制台。
   b. 单击页面顶部的开发。
   c. 在作业开发区域，双击目标文件夹或目标作业名，进入作业编辑页面。

2. 单击页面顶部菜单栏中的上线，进入上线新版本页面。

3. 在初始资源页面，选择初始资源类型，单击下一步。

- 上次自动调优：使用上次AutoScale的planjson配置启动作业。满足以下所有条件后即可选择上次自动调优功能。
  - 作业已开启AutoScale并完成迭代。
  - 作业为暂停状态。
  - 在资源配置页面获取AutoScale配置信息。

- 系统分配：使用上一次AutoScale自动生成的方法。新增作业或者作业逻辑没有修改且兼容实时计算版本时，可以选择系统分配。
- 手动资源配置：使用手动生成的资源配置方法。手动重新生成或者修改AutoScale时，需要选择手动资源配置。

4. 在数据检查，通过上线检查后，单击下一步。
5. 在资源配置，配置自动调优信息后，单击下一步。

参数	说明
自动调优	选择开启，开启自动调优功能。
planJson最大CU数	作业可用的资源上限，单位为CU，1CU=1 Core + 4G RAM。最大CU数需小于项目可用CU数。
调优策略	目前调优策略仅支持数据滞留时间。系统会根据选择的调优策略和期望值对作业自动调优。

参数	说明
期望值	数据滞留时间阈值。当数据源的数据滞留时间超过该值时，触发AutoScale，优化作业并发数。

> 说明:
> 例如，调优策略设置数据滞留时间的期望值为5（秒）。如果此时作业的数据滞留时间大于5秒，则系统会不断进行自动调优，直至数据滞留时间小于5秒。

6. 在上线作业，单击上线。
7. 启动作业。启动作业步骤参见启动。

关闭AutoScale

> 说明:
> 启动作业时开启AutoScale自动调优功能后，才可以进行关闭AutoScale自动调优操作。

您可以在作业运行期间，动态关闭AutoScale自动调优功能。

1. 登录作业运维页面。
   a. 登录实时计算控制台。
   b. 单击页面顶部的运维。
   c. 在作业列表区域，单击作业名称下的目标作业名。
2. 单击页面右上角的关闭自动调优。
3. 单击确认。

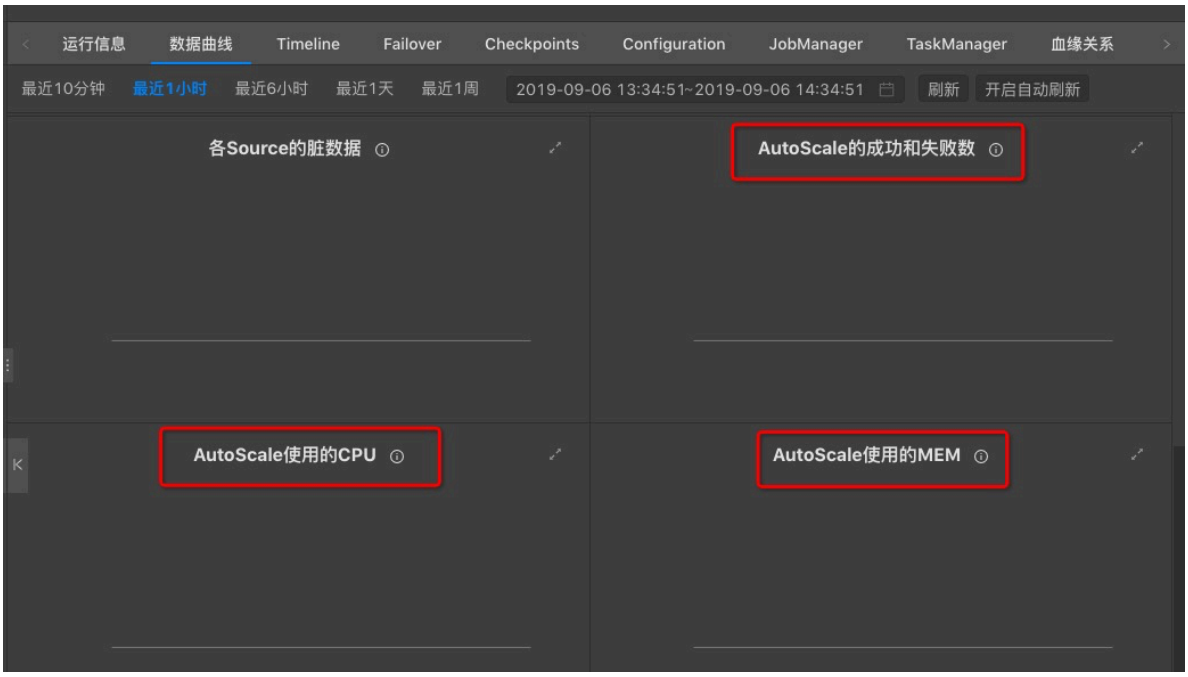
> 说明:
> 作业在上线时启动了AutoScale后，运维界面自动调优列下的操作按键才能生效。

AutoScale效果查看

> 说明:
> 请在运维界面查看。

- AutoScale Metric

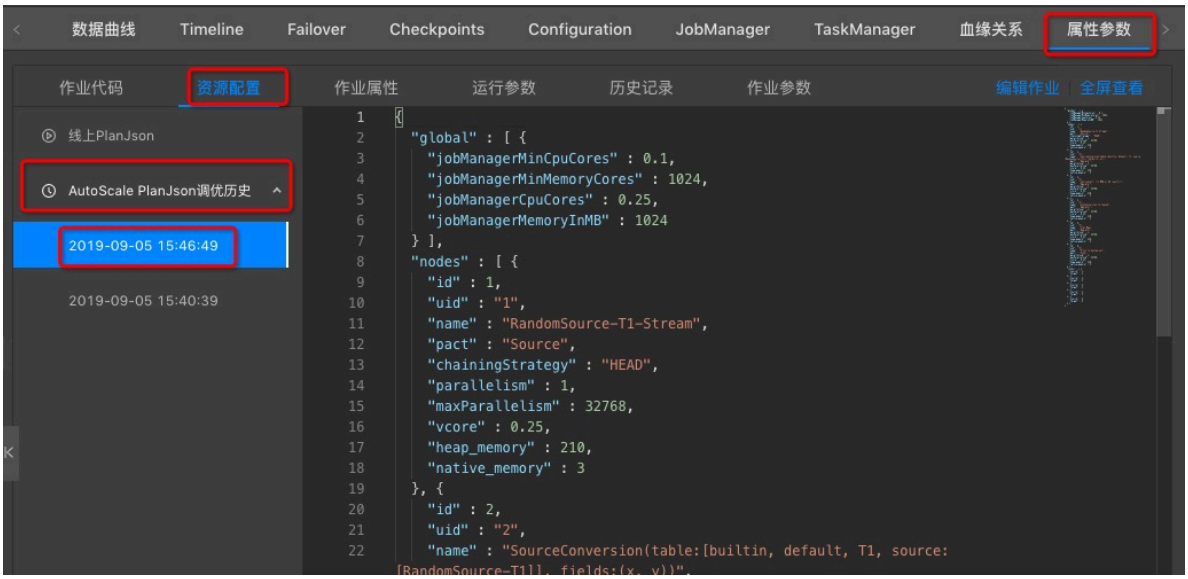
  在数据曲线 > OverView页面，可以查看AutoScale的信息。



曲线名称	说明
AutoScale的成功和失败数	AutoScale成功和未成功执行的次数。
AutoScale的成功和失败数	执行AutoScale时消耗的CPU量。
AutoScale的成功和失败数	执行AutoScale时消耗的内存量。

- AutoScale PlanJson信息

  在属性参数 > 资源配置 > AutoScale PlanJson调优历史，单击目标版本，查看AutoScale生成的PlanJson的信息。



AutoScale可能出现的问题

- 作业自动重启

  AutoScale根据实际流量来动态调整并发数以及资源，因此可能会在流量上涨以及流量下降时，通过自动重启触发资源调整。

- 短时间出现数据传输延迟

  流量低谷时AutoScale会降低并发数、减少资源分配。当流量上涨后，流量低谷时的资源配置，会导致吞吐能力不够和数据传输的延迟。

- 作业无法恢复

  在部分场景下AutoScale无法有效工作，导致作业延迟和作业频繁的进行配置调整。

- 并发数先减少后增多

  包含窗口函数或聚合函数的作业，在状态（State）数量的增加时，访问State的性能衰减，作业启动时的并发数减少。随着数据积累，并发数会逐渐增大，直至State数量达到稳定。

FAQ

- Q：作业启动时产生报错：`ERR_ID: CLI-00000001`。

  A：参见启动报错：CLI-00000001。

- Q：作业上线时参数校验报错：`resource validate failed! please modify your planJson or max CU and try again.`。

  A： 以下2种场景可能导致该报错：

  - PlanJson资源文件和实际作业不匹配
    - 报错原因：PlanJson资源文件和实际作业不匹配，导致能够关联（Chain）在一起的节点没有关联在一起，使作业所需资源变大。
    - 解决方案：重新生成配置信息，操作步骤参见启动报错：CLI-00000001。
  - 配置的可用资源过小
    - 报错原因：作业所需的CPU或者内存中的一个资源超过最大资源。
    - 解决方案：返回上一步，增加最大可用资源的CU数。

- Q：AutoScale没有启动。

  A： 排查方法如下：

  - 检查作业是否频繁运行失败（Failover）。AutoScale开启的前提是作业本身逻辑正确并且能够稳定运行。
  - 查看JobManager的相关日志，检查是否存在系统异常。

- Q：AutoScale没有效果。

  A：请按照以下顺序进行排查：

  1. 检查最大资源限制，确认作业已使用资源是否达到最大资源限制。
  2. 检查作业数据源（Source）节点的逻辑，查看Source节点是否关联（Chain）过多的Operator。若关联过多Operator，请手动编辑PlanJson，在一些关键的位置进行关联（Chain）拆分。解决方案参见#unique_19。
  3. 查看JobManager的相关日志，检查是否存在系统异常。

7.5 手动配置调优

您可以通过手动的调整实时计算的作业参数、资源参数和上下游参数，提升实时计算作业的性能。

> 说明:
> 建议您完成作业反压检测后，再判断是否需要进行配套调优。实时计算3.0以上版本作业反压检测方法，请参见作业反压检测步骤。

手动配置调优概述

手动配置调优的内容主要有3种类型：

- 上下游参数调优：主要对作业中的上下游存储参数进行调优。
- 作业参数调优：主要对作业中的`miniBatch`等参数进行调优。
- 资源参数调优：主要对作业中的Operator的并发数（Parallelism）、CPU（Core）、堆内存（Heap Memory）等参数进行调优。

下面通过3个章节对以上3类调优进行介绍。参数调优后将生成新的配置，Job需停止 > 启动或者暂停 > 恢复后才能使用新的配置，启动新配置的方法请参见重新启用新的配置。

上下游参数调优

由于实时计算的特性，每条数据均会触发上下游存储的读写，会对上下游存储形成性能压力。通过设置batchsize批量的读写上下游存储数据可以降低对上下游存储的压力。支持batchsize参数的上下游存储如下：

名称	参数	详情	设置参数值
DataHub源表	batchReadSize	单次读取的条数	可选，默认为10。
DataHub源表	batchSize	单次写入的条数	可选，默认为300。
日志服务 LOG源表	batchGetSize	单次读取logGroup的条数	可选，默认为10。

分析型数据库MySQL版 2.0结果表	batchSize	一次批量写入的条数	可选，默认为1000。
云数据库RDS版结果表	batchSize	一次批量写入的条数	可选，默认为50。
云数据库 HybridDB for MySQL结果表	batchSize	一次批量写入的条数	可选，默认值为1000，建议可设置的最大值为4096。
	bufferSize	去重的buffer大小，需要指定主键才生效。	可选，设置batchSize必须设置bufferSize，建议可设置的最大值为4096。

说明:

DDL的WITH参数列中增加batchSize相关参数，即可完成数据的批量读写设置。例如，`batchReadSize='<number>'`。

作业参数调优

`miniBatch`设置仅适用于优化GROUP BY。Flink SQL流模式下，每来一条数据都会执行State操作，IO消耗较大。设置`miniBatch`后，同一个Key的一批数据只访问一次State，且只输出最新的一条数据，即减少了State访问也减少了向下游的数据更新。`miniBatch`设置说明如下：

说明:

- 新增加的作业参数建议用户停止 > 启动。
- 更新作业参数值暂停 > 恢复即可。

```
#3.2及以上版本开启window miniBatch方法（3.2及以上版本默认不开启window miniBatch）。
sql.exec.mini-batch.window.enabled=true
# excatly-once语义。
blink.checkpoint.mode=EXACTLY_ONCE
# checkpoint间隔时间，单位毫秒。
blink.checkpoint.interval.ms=180000
blink.checkpoint.timeout.ms=600000
# 实时计算2.0及以上版本使用niagara作为statebackend，以及设定state数据生命周期，单位毫秒。
state.backend.type=niagara
state.backend.niagara.ttl.ms=129600000
# 实时计算2.0及以上版本开启5秒的microbatch（窗口函数不要设置这个参数。）
blink.microBatch.allowLatencyMs=5000
# 表示整个Job允许的延迟。
blink.miniBatch.allowLatencyMs=5000
#双流join节点优化参数
blink.miniBatch.join.enabled=true
# 单个Batch的size。
blink.miniBatch.size=20000
# local优化，实时计算2.0及以上版本默认已经开启，1.6.4需手动开启。
blink.localAgg.enabled=true
```

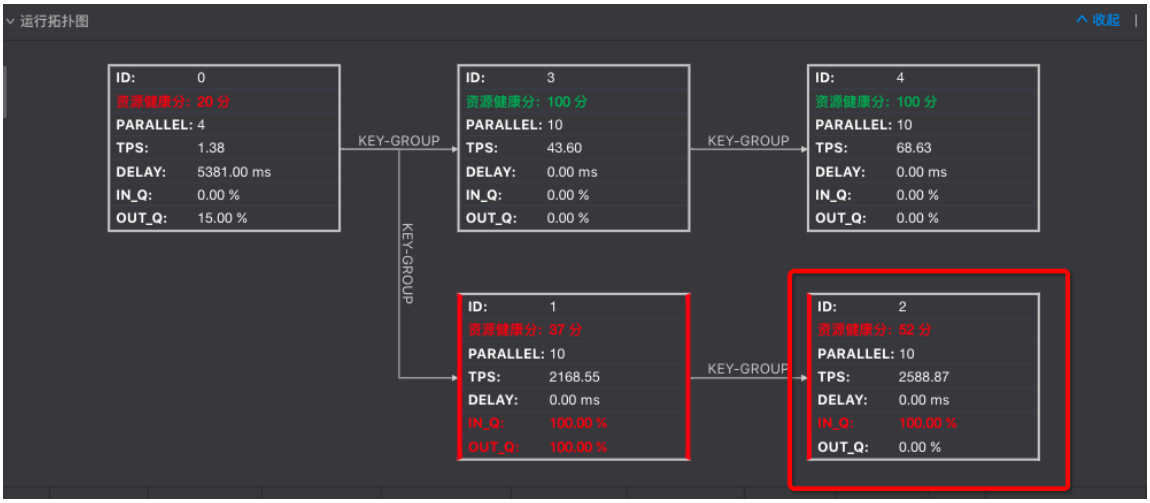
```
# 实时计算2.0及以上版本开启partial优化，解决count distinct效率低问题。
blink.partialAgg.enabled=true
# union all优化。
blink.forbid.unionall.as.breakpoint.in.subsection.optimization=true
# GC优化（源表为SLS时，不能设置改参数）。
blink.job.option=-yD heartbeat.timeout=180000 -yD env.java.opts='-
verbose:gc -XX:NewRatio=3 -XX:+PrintGCDetails -XX:+PrintGCDateStamps -
XX:ParallelGCThreads=4'
# 时区设置。
blink.job.timeZone=Asia/Shanghai
```

资源调优

下文通过示例为您介绍资源调优方法。

1. 分析问题

   a. 通过Job的拓扑图查看到2号的Task节点的输入队列已达到100%，造成上游1号节点的数据堆积，输出队列造成数据堆积。



   b. 单击目标Task节点（本示例为2号Task节点）。

   c. 在SubTask List，查看In Queue为`100%`的列。

   d. 单击目标列操作下的查看日志。

   e. 单击跳转到TaskExecutor。

   f. 在TaskExecutor > Metrics Graph，查看CPU和MEM的使用情况。

2. 性能调优

a. 单击作业编辑页面右侧的资源配置，进入调优窗口。

b. 对目标Operator或者Group进行参数修改。

- 单个Operator参数修改：

  A. GROUP框内，单击右上角的加号（+）。

  B. 将鼠标悬停至目标Operator框内。

  C. 单击目标Operator右侧的铅笔图标。

  D. 在修改Operator数据 修改参数。

- GROUP批量参数修改：

  A. 将鼠标悬停至GROUP框内。

  B. 单击GROUP右侧的铅笔图标。

  C. 在批量修改Operator数据 修改参数。

c. 完成配置参数修改后，单击资源配置右上角的配置信息操作 > 应用当前配置。

说明：

- 如果增加了Group的资源配置后，作业的运行效率提升不明显，需要按照以下顺序分析问题：

  a. 该节点是否存在数据倾斜现象。

  b. 复杂运算的Operator节点（如：GROUP BY、WINDOW、JOIN等）的子节点是否存在异常。

- Operator子节点拆解方法：

  a. 单击需要修改的Operator。

  b. 将chainingStrategy参数值修改为HEAD。若已为HEAD，需要将后一个Operator的chainingStrategy参数值修改为HEAD。chainingStrategy参数说明如下。

参数	说明
ALWAYS	算子会尽可能的链接在一起。为了优化性能，通常最佳实践是让算子尽可能链接在一起，同时增加并发度。
NEVER	算子将不会和上下游的算子链接在一起。
HEAD	算子将不会和上游的算子链接在一起，但是会和下游的算子链接在一起。

3. 资源参数的配置原则和建议

- 作业的资源配置建议为：`core:heap_memory=1:4`，即1个核对应4G内存。

  说明：

  - Operator的总`core=parallelism*core`。
  - Operator的总`heap_memory=parallelism*heap_memory`。
  - Group中的core取各个Operator中最大值，memory取各个Operator中memory之和。

  例如，`core`参数值为1CU，`heap memory`参数值为3G，则最终资源分配结果的是1CU+4G；`core`参数值为1CU，`heap memory`参数值为5G，则最终资源分配结果的是1.25CU+5G。

- parallelism
  - Source节点

    说明：

Source的并发数不能大于Source的Shard数。

■ Source数量和上游Partition数量相关。

■ 例如，Source的个数是16，Source的并发数可以配置为16、8或4等，不得超过16。

- 中间节点

■ 根据预估的QPS计算。

■ QPS低的任务，中间处理节点数和Source并发数一样。

■ QPS高的任务，中间处理节点数配置为比Source并发数大，例如，64、128或者256。

- Sink节点

■ 并发度和下游存储的Partition数相关，一般是下游Partition个数的2~3倍。

■ 如果配置过大会导致输入超时或失败。例如，下游Sink节点个数是16，建议Sink的并发数最大配置为48。

· core

默认值为0.1，根据实际CPU使用配置，建议参数值为0.25。

· heap_memory

堆内存，默认为值为256（单位为MB），请根据实际的内存状况使用进行配置。

· state_size

存在GROUP BY的Task节点中可配置参数state_size。state_size默认值为0，存在GROUP BY的Task节点需要把state_size参数值设置为`1`，表示该Operator会使用state功能，作业会为该Operator申请额外的内存。如果state_size参数值不设置为`1`，作业可能运行失败。state_size需要配成`1`的Operator包括GROUP BY、undefinedJOIN、OVER和WINDOW。

重新启用新的配置

完成配置后，需要暂停 > 恢复或者停止 > 启动后才能使新配置生效。完成作业重启或恢复后，可通过运维 > 运行信息 > Vertex拓扑查看新的配置是否生效。

 说明:

不建议采用停止 > 启动的方式触发新配置。停止作业后，作业状态会被清除，从而可能导致计算结果不一致。

· 暂停 > 恢复：仅更改资源配置、调整WITH参数大小或调整作业参数大小。

· 停止 > 启动：需要更改SQL逻辑、更改作业版本、增加WITH参数或增加作业参数。

- 暂停 > 启动步骤如下：

  1. 上线作业。上线步骤参见上线。配置方式选择使用上次资源配置（手动资源配置）。
  2. 在运维页面，单击目标作业操作列下的暂停。
  3. 在运维页面，单击目标作业操作列下的恢复。
  4. 在恢复作业运行，单击按最新配置恢复。

- 停止 > 启动步骤如下：

  1. 停止作业。停止作业步骤参见停止。
  2. 启动作业。启动作业步骤参见启动。

相关参数说明

- Global

  isChainingEnabled：是否启用Chain策略，默认为true。不需要修改。

- Nodes

参数	说明	是否需要修改
id	节点ID号，自动生成，ID号唯一	否
uid	节点UID号，用于计算Operator ID，如果不设置，会使用ID。	否
pact	节点类型，例如，Data Source、Operator或Data Sink等。	否
name	节点名称，可自定义。	可修改
slotSharingGroup	请保持默认配置。	否

参数	说明	是否需要修改
chainingStrategy	ChainingStrategy 是用来定义算子链接的策略。当一个算子和上游算子链接在一起，这意味着它们会运行在同一个线程。它们会合并为一个有多个运行步骤的算子。ChainingStrategy支持以下三种方式： - ALWAYS：算子会尽可能的链接在一起。为了优化性能，通常最佳实践是让算子尽可能链接在一起，同时增加并发度。 - NEVER：算子将不会和上下游的算子链接在一起。 - HEAD：算子将不会和上游的算子链接在一起，但是会和下游的算子链接在一起。	可修改
parallelism	并发数，默认值为 1，可以根据实际数据量增大并发数。	可修改
core	CPU，默认为0.1，可根据实际CPU使用情况进行配置。建议设置为0.25。	可修改
heap_memory	堆内存，默认为256（单位为MB），可根据实际内存使情况进行配置。	可修改
direct_memory	JVM堆外内存，默认0（单位为MB），建议不修改。	可修改
native_memory	JVM堆外内存，JNI使用，默认为0，建议配置为10（单位为MB）。	可修改

- Chain

  Flink SQL任务是一个DAG图，由多个节点（Operator）组成，部分上下游的节点在运行时可以合成为一个节点，称为Chain。Chain后的节点，总CPU为所有节点CPU的最大值，总内存为所有节点内存的总和。多节点合成一个节点可以有效的减少网络传输，降低成本。

   说明:

- 并发数相同的节点才能进行Chain操作。
- Group By节点不能进行Chain操作。

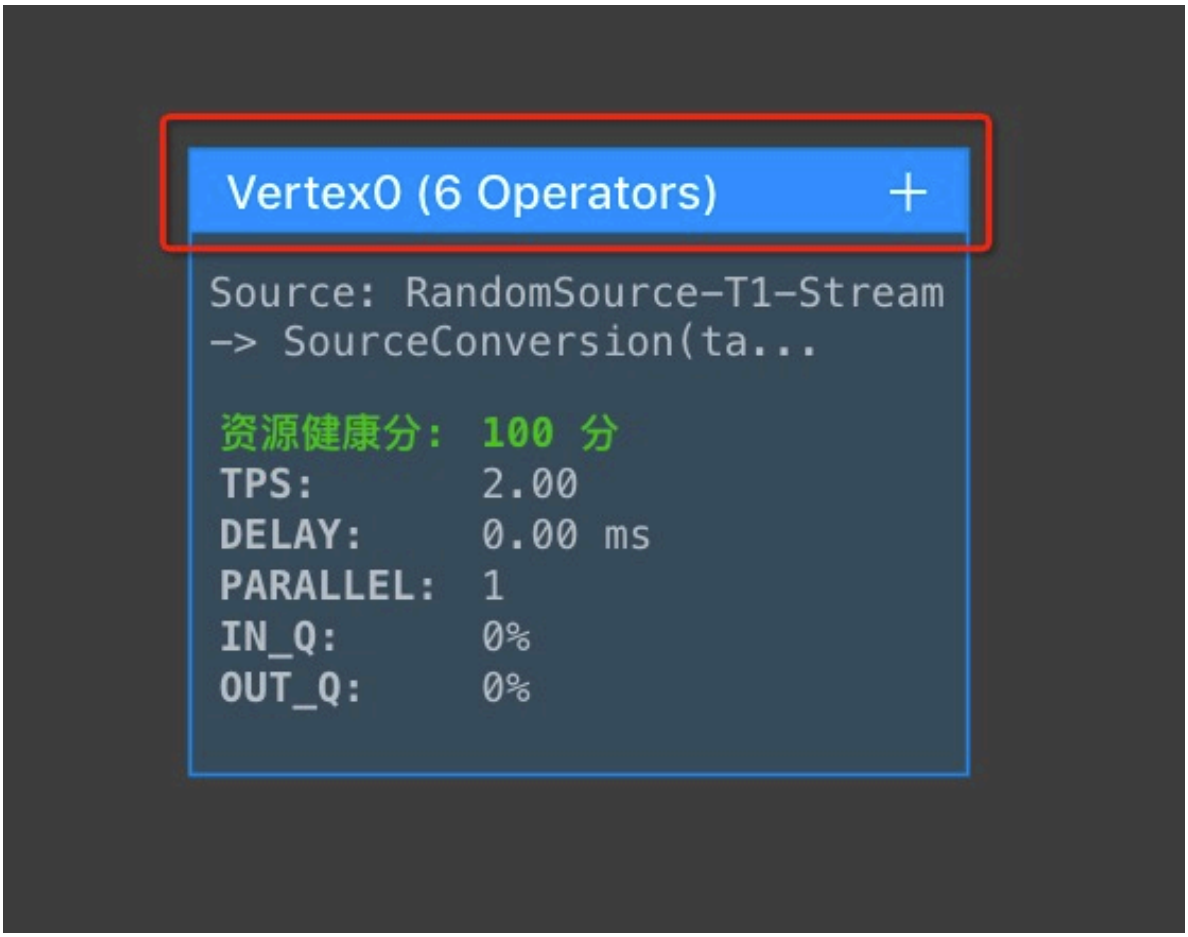
7.6 典型的反压场景及优化思路

反压是流式Shuffle中的一个重要概念，当下游处理能力不足时，会通知上游停止发送数据，从而避免数据丢失。本文为您介绍典型的反压场景及优化思路。

反压检测机制

实时计算3.0以上版本提供了作业反压检测机制，通过检测Vertex（可以理解为多个关联Chain在一起的Operator组）输出网络Buffer的拥塞情况，判断对应的Vertex是否存在反压。具体步骤如下：

1. 登录作业运维页面。
   a. 登录实时计算控制台。
   b. 单击页面顶部的运维。
   c. 在作业列表区域，单击作业名称下的目标作业名。
2. 在运行信息 > Vertex拓扑，单击对应的节点边框。



3. 在右侧信息区域的BackPressure > Status，查看反压状态。

   - 红色high：表示该节点存在反压。
   - 绿色ok：表示该节点不存在反压。

反压场景及优化思路

> 说明:
> 示意图中Vertex的颜色为绿色表示通过以上的检测显示不存在反压，红色表示存在反压。

- 仅存在1个Vertex，经检测无反压

由于Flink的特性，在最后一个Vertex的输出上没有设置网络Buffer（直接写出到下游存储），当作业仅存在一个（或最后一个）Vertex时，反压检测机制失效。因此以上Vertex拓扑示意图并不表示作业不存在反压，若需进一步判断反压点，需将Vertex0中的Operator拆分后再进行判断。拆分方法参见#unique_98/unique_98_Connect_42_section_dj3_clm_bgb。

- 存在多个Vertex，倒数第2个Vertex经检测存在反压

该场景表示：Vertex1被反压，性能瓶颈点在Vertex2。此时，可以通过查看Vertex2中的Operator组的名称进行判断：

- 若只涉及写出下游存储的操作，则可能是写出速度较慢所导致。建议增加Vertex2的并发数，或者配置对应结果表（Sink）的`batchsize`参数，具体操作步骤请参见#unique_98/unique_98_Connect_42_section_q6p_kx0_qrm。
- 若涉及到除写出下游存储外的其他操作，需要将其他操作对应的Operator拆分后再进一步判断。拆分方法参见#unique_98/unique_98_Connect_42_section_dj3_clm_bgb。

- 存在多个Vertex，非倒数第2个Vertex经检测存在反压

该场景表示：Vertex0被反压，性能瓶颈点在Vertex1。此时，可以通过查看Vertex1中的Operator组的名称来判断具体的操作。常见的操作和对应的优化方法如下：

- GROUP BY操作：可以考虑增加并发或设置`miniBatch`参数优化状态（State）的操作，具体方法参见#unique_98/unique_98_Connect_42_section_sxj_yjm_bgb。
- 维表JOIN操作：可以考虑增加并发数或者设置维表的Cache策略，具体请参见对应维表文档。
- UDX操作：可以考虑增加并发数或者优化对应的UDX代码。

- 存在多个Vertex，所有Vertex经检测不存在反压

该场景表示：可能的性能瓶颈点在Vertex0，可以通过查看Vertex0中的Operator组的名称判断具体的操作。

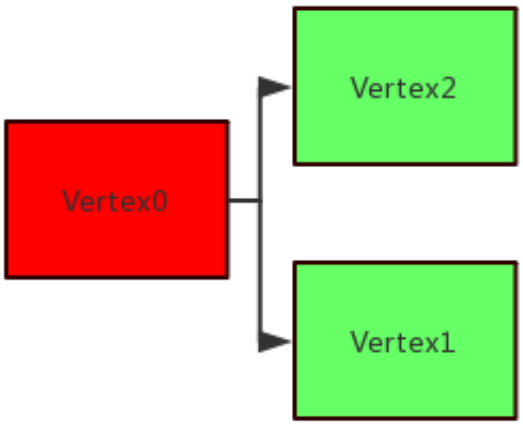
- 若其中只存在读取上游源表（Source）的操作，则可能实时计算本身没有性能瓶颈，只是读取Source的速度较慢而导致延时较大。此时可通过增加Source节点的并发数

或者设置读取Source的batchsize的方法进行优化，具体步骤参见#unique_98/unique_98_Connect_42_section_q6p_kx0_qrm。

> 说明:
> Source节点增加后的并发数不能大于上游存储的Shard数。

- 若其中除读取上游Source的操作外，还存在其他操作，建议将其他操作的Operator拆分后再进一步判断，拆分方法参见#unique_98/unique_98_Connect_42_section_dj3_clm_bgb。

· 存在多个Vertex，其中1个Vertex经检测存在反压，但其后续的多个并行Vertex经检测不存在反压



该场景表示：Vertex0被反压，但是无法直接判断性能瓶颈点是Vertex1还是Vertex2。您可以通过Vertex1和Vertex2的IN_Q指标做初步判断，如果对应的IN_Q长时间为100%，则此节点极可能存在性能瓶颈点。若需要进一步确认，则需将此节点拆分后进行进一步判断。拆分方法参见#unique_98/unique_98_Connect_42_section_dj3_clm_bgb。

8 Flink SQL

8.1 Flink SQL概述

Flink SQL是阿里云实时计算为简化计算模型，降低用户使用实时计算门槛而设计的一套符合标准SQL语义的开发语言。

本章节通过以下方面，为您介绍实时计算Flink SQL的使用方法。

- 基本概念
- 关键字
- 数据类型
- DDL语句
- DML语句
- QUERY语句
- 数据视图
- 窗口函数
- 逻辑函数
- 内置函数
- 自定义函数

8.2 关键字

本文为您介绍实时计算中已保留的关键字和使用关键字字符的方法。

关键字常用类型

常用类型	关键字
数据类型	VARCHAR、INT、BIGINT、DOUBLE、DATE、BOOLEAN、TINYINT、SMALLINT、FLOAT、DECIMAL、VARBINARY
DDL	CREATE TABLE、CREATE FUNCTION、CREATE VIEW
DML	INSERT INTO
SELECT 子句	SELECT FROM、WHERE、GROUP BY、JOIN

命名规则

源表名称、结果表名称、视图表名称、别名名称遵循标准数据库命名规则定义。必须以字母开头，并且只能包含字母、数字和下划线。

保留关键字

以下字符串组合已经被保留为关键字，以备将来使用。如果您想使用以下字符串作为字段名称，请在关键字两端添加`，例如`value`。

```
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,BEFORE,BEGIN
,BERNOULLI,BETWEEN,BIGINT,BINARY,BIT,BLOB,BOOLEAN,BOTH,BREADTH,BY,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_COD
E,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,DATETIME_INTERVAL_PR
ECISION,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,EACH,ELEMENT
,ELSE,END,END-EXEC,EPOCH,EQUALS,ESCAPE,EVERY,EXCEPT,EXCEPTION,EXCLUDE
,EXCLUDING,EXEC,EXECUTE,EXISTS,EXP,EXPLAIN,EXTEND,EXTERNAL,EXTRACT,
FALSE,FETCH,FILTER,FINAL,FIRST,FIRST_VALUE,FLOAT,FLOOR,FOLLOWING,FOR
```

```
,FOREIGN,FORTRAN,FOUND,FRAC_SECOND,FREE,FROM,FULL,FUNCTION,FUSION,G
,GENERAL,GENERATED,GET,GLOBAL,GO,GOTO,GRANT,GRANTED,GROUP,GROUPING,
HAVING,HIERARCHY,HOLD,HOUR,IDENTITY,IMMEDIATE,IMPLEMENTATION,IMPORT,IN
,INCLUDING,INCREMENT,INDICATOR,INITIALLY,INNER,INOUT,INPUT,INSENSITIV
E,INSERT,INSTANCE,INSTANTIABLE,INT,INTEGER,INTERSECT,INTERSECTION,
INTERVAL,INTO,INVOKER,IS,ISOLATION,JAVA,JOIN,K,KEY,KEY_MEMBER,KEY_TYPE
,LABEL,LANGUAGE,LARGE,LAST,LAST_VALUE,LATERAL,LEADING,LEFT,LENGTH,
LEVEL,LIBRARY,LIKE,LIMIT,LN,LOCAL,LOCALTIME,LOCALTIMESTAMP,LOCATOR,
LOWER,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,NAME,
NAMES,NATIONAL,NATURAL,NCHAR,NCLOB,NESTING,NEW,NEXT,NO,NONE,NORMALIZE,
NORMALIZED,NOT,NULL,NULLABLE,NULLIF,NULLS,NUMBER,NUMERIC,OBJECT,OCTETS,
OCTET_LENGTH,OF,OFFSET,OLD,ON,ONLY,OPEN,OPTION,OPTIONS,OR,ORDER,ORDERING
,ORDINALITY,OTHERS,OUT,OUTER,OUTPUT,OVER,OVERLAPS,OVERLAY,OVERRIDING
,PAD,PARAMETER,PARAMETER_MODE,PARAMETER_NAME,PARAMETER_ORDINAL_PO
SITION,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,QUARTER,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_L
ENGTH,RETURNED_OCTET_LENGTH,RETURNED_SQLSTATE,RETURNS,REVOKE,RIGHT,
ROLE,ROLLBACK,ROLLUP,ROUTINE,ROUTINE_CATALOG,ROUTINE_NAME,ROUTINE_SC
HEMA,ROW,ROWS,ROW_COUNT,ROW_NUMBER,SAVEPOINT,SCALE,SCHEMA,SCHEMA_NAM
E,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,SQLEXCEPTI
ON,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,TABLE
,TABLESAMPLE,TABLE_NAME,TEMPORARY,THEN,TIES,TIME,TIMESTAMP,TIMESTAMPA
DD,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
,UESCAPE,UNBOUNDED,UNCOMMITTED,UNDER,UNION,UNIQUE,UNKNOWN,UNNAMED,UNNEST
,UPDATE,UPPER,UPSERT,USAGE,USER,USER_DEFINED_TYPE_CATALOG,USER_DEFIN
ED_TYPE_CODE,USER_DEFINED_TYPE_NAME,USER_DEFINED_TYPE_SCHEMA,USING,VALUE
,VALUES,VARBINARY,VARCHAR,VARYING,VAR_POP,VAR_SAMP,VERSION,VIEW,WEEK,
WHEN,WHENEVER,WHERE,WIDTH_BUCKET,WINDOW,WITH,WITHIN,WITHOUT,WORK,WRAPPER
,WRITE,XML,YEAR,ZONE

8.3 基本概念

8.3.1 时区

实时计算作业支持配置时区参数调整时间类型数据的输出结果。本文为您介绍配置时区参数的方法。

配置时区参数

- 配置相同时区参数

  实时计算平台的作业参数中，您可以通过配置作业的时区（例如`blink.job.timeZone=America/New_York`）调整时间类型数据的输出结果。默认时区为东八区。

  - 时区列表参见支持的时区列表。
  - 时区参数配置方法参见#unique_114/unique_114_Connect_42_section_rwy_txm_jhb。

- 配置不同时区参数

  您可以对不同的源表或结果表配置不同的时区。例如，您需要读写的MySQL数据类型（例如TIME、DATE、TIMESTAMP类型）的时区为`America/New_York`，而作业计算需要的时区为`Asia/Shanghai`，则可以通过如下方式单独配置源表或结果表的时区。

```
CREATE TABLE mysql_source_my_table (
```

```
 -- ...
) WITH (
timeZone='America/New_York'
 -- ...
)
```

时区参数配置示例

实时计算时区相关函数语义上为自定义的时区。本示例以自定义时区为`Asia/Shanghai`进行说明。

- 字符串转时间类型（TO_TIMESTAMP、TIMESTAMP和UNIX_TIMESTAMP）

```
-- Scalar function
TO_TIMESTAMP("2018-03-14 19:01:02.123")
-- SQL Literal

TIMESTAMP '2018-03-14 19:01:02.123'
-- 输出：  `1521025262123`。
-- 类似的还有 UNIX_TIMESTAMP，区别是单位为秒。
```

- 时间类型转字符串（FROM_TIMESTAMP和DATA_FORMAT）

说明:

当参数为TIMESTAMP时，输出结果取决于自定义设定的时区。

```
SELECT DATE_FORMAT(TO_TIMESTAMP(1520960523000), 'yyyy-MM-dd HH:mm:ss
')
-- 输出：`2018-03-14 01:02:03`。

DATE_FORMAT('2018-03-14 01:02:03', 'yyyy-MM-dd HH:mm:ss', 'yyyy/MM/
dd HH:mm:ss')
FROM_UNIXTIME(1521025200000/1000)
-- 输出：`2018-03-14 19:00:00`。
```

- 时间相关计算函数

当参数为TIMESTMP时，EXTRACT、FLOOR、CEIL、DATE_DIFF等函数输出结果取决于自定义的时区。

```
-- 1521503999000 2018-03-19T23:59:59+0000, 2018-03-20T07:59:59+0800
 EXTRACT(DAY FROM TO_TIMESTAMP(1521503999000))
-- 输出：`20`。表示东八区的20日。
```

- 当前时间函数

当前时间函数包括`LOCALTIMESTAMP()`、`CURRENT_TIMESTAMP()`、`NOW()`和`UNIX_TIMESTAMP()`等。

```
-- 当前时间是 2018-04-03 16:56:10 (Asia/Shanghai)
SELECT DATE_FORMAT(CURRENT_TIMESTAMP, ‘yyyy-MM-dd HH:mm:ss’);
-- 输出：`2018-04-03 16:56:10`。

SELECT DATE_FORMAT(LOCALTIMESTAMP, ‘yyyy-MM-dd HH:mm:ss’);
-- 输出：`2018-04-03 16:56:10`。
```

```
SELECT FROM_UNIXTIME(NOW()); SELECT FROM_UNIXTIME(UNIX_TIMESTAMP
());
-- 输出：`2018-04-03 16:56:10`。
```

- DATA和TIME类型

  对于DATE和TIME类型，SQL内部用整数进行显示和计算。DATE指的是EPOCH DAYS；TIME指的是用户时区，从当天的零点开始计算的毫秒数。如果UDF中对DATE和TIME进行计算，从内部类型转换到`java.sql.Date`、`java.sql.Time`类型时，Java对象中已经完成了时区偏移操作。

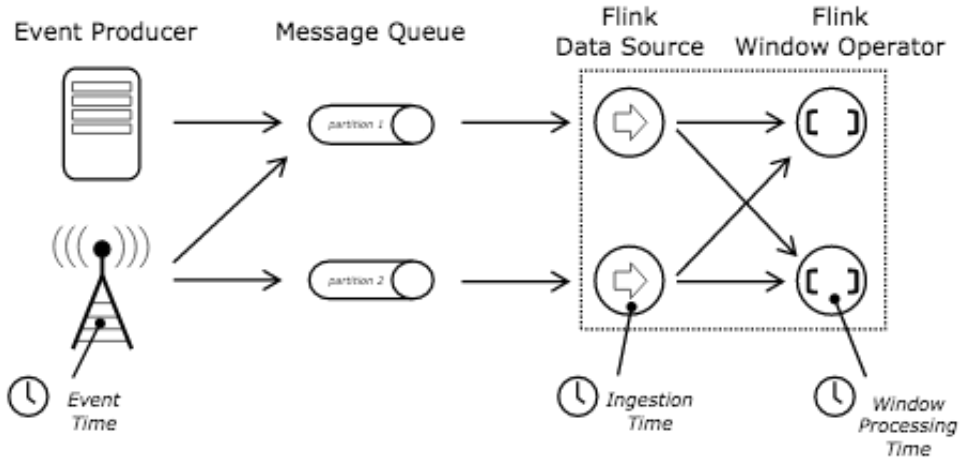
支持的时区列表

实时计算支持的时区列表参见附件TimeZone。

8.3.2 时间属性

本文为您介绍Flink SQL支持的Event Time和Processing Time时间类型。

Flink支持3种与流数据处理相关的时间概念：Processing Time、Event Time和Ingestion Time。



Flink SQL仅支持其中的2种时间类型Event Time和Processing Time：

- Event Time：您提供的事件时间（通常是数据的最原始的创建时间）。Event Time必须是您提供在数据储存里的数据。
- Processing Time：系统对事件进行处理的本地系统时间，单位为毫秒。

Event Time

Event Time也称为Row Time。EventTime时间属性必须在源表DDL中声明，可以将源表中的某一字段声明成Event Time。目前只支持将TIMESTAMP类型（将来会支持LONG类型）声明成Row Time字段。如果源表中需要声明为Event Time的列不是TIMESTAMP类型，需要借助#unique_116，基于现有列构造出一个TIMESTAMP类型的列。

由于数据本身的乱序、网络的抖动（网络堵塞导致的数据传输延迟的变化）或者其它原因，导致了数据到达的顺序和被处理的顺序，可能是不一致的（乱序）。因此定义一个Row Time字段，需要明文定义一个#unique_117计算方法。

窗口函数基于Event Time聚合的示例如下。

```
CREATE TABLE tt_stream(
  a VARCHAR,
  b VARCHAR,
  c TIMESTAMP,
  WATERMARK wk1 FOR c as withOffset(c, 1000)  --Watermark计算方法。
) WITH (
  type = 'sls',
  topic = '<yourTopicName>',
  accessId = '<yourAccessId>',
  accessKey = '<yourAccessSecret>'
);

CREATE TABLE rds_output(
  id VARCHAR,
  c TIMESTAMP,
  f TIMESTAMP,
  cnt BIGINT
) WITH (
  type = 'rds',
  url = 'jdbc:mysql://****3306/test',
  tableName = '<yourTableName>',
  userName = '<yourUserName>',
  password = '<yourPassword>'
);

INSERT INTO rds_output
SELECT a AS id,
     SESSION_START(c, INTERVAL '1' SECOND) AS c,
     CAST(SESSION_END(c, INTERVAL '1' SECOND) AS TIMESTAMP) AS f,
     COUNT(a) AS cnt
FROM tt_stream
GROUP BY SESSION(c, INTERVAL '1' SECOND), a
```

Processing Time

Processing Time是系统产生的，不在您的原始数据中，您需要在数据源表的声明中明文定义一个Processing Time列。

```
filedName as PROCTIME()
```

窗口函数基于Processing Time聚合的示例如下。

```
CREATE TABLE mq_stream (
  a VARCHAR,
  b VARCHAR,
  c BIGINT,
  d AS PROCTIME()   --在数据源表的声明中明文定义一个Processing Time列。
) WITH (
  type = 'mq',
  topic = '<yourTopic>',
  accessId = '<yourAccessId>',
  accessKey = '<yourAccessSecret>'
```

```
);

CREATE TABLE rds_output (
  id VARCHAR,
  c TIMESTAMP,
  f TIMESTAMP,
  cnt BIGINT
) with (
  type = 'rds',
  url = '<yourDatebaseURL>',
  tableName = '<yourDatabasTableName>',
  userName = '<yourUserName>',
  password = '<yourPassword>'
);


INSERT INTO rds_output
SELECT a AS id,
     SESSION_START(d, INTERVAL '1' SECOND) AS c,
     SESSION_END(d, INTERVAL '1' SECOND) AS f,
     COUNT(a) AS cnt
FROM mq_stream
GROUP BY SESSION(d, INTERVAL '1' SECOND), a
```

时间属性字段传递

时间属性字段经过如下4种操作后会失去时间属性特性：

- 对时间属性字段以外的字段进行GROUP BY（滚动窗口、#unique_119或会话窗口中的GROUP BY除外）操作。
- 双流JOIN操作。

  说明:
  双流JOIN详情请参见JOIN语句。
- #unique_122中的MATCH_RECOGNIZE操作。
- OVER窗口中的PARTITION BY操作。

如果经过以上4种操作后，继续使用该时间属性字段进行窗口函数运算，会出现类似`org.apache.flink.table.api.ValidationException: Window can only be defined over a time attribute column.`的报错。

8.3.3 Watermark

实时计算可以基于时间属性对数据进行窗口聚合。基于的Event Time时间属性的窗口函数作业中，数据源表的声明中需要使用Watermark方法。

Watermark是一种衡量Event Time进展的机制，它是数据本身的一个隐藏属性，Watermark的定义是数据源表DDL定义的一部分。Flink提供了如下语法定义Watermark。

说明:

实时计算时间属性详情，请参见#unique_125。

```
WATERMARK [watermarkName] FOR <rowtime_field> AS withOffset(<
rowtime_field>, offset)
```

参数	说明
watermarkName	标识Watermark的名字，可选。
<rowtime_field>	<rowtime_field>必须是表中已定义的一列（当前仅支持为TIMESTAMP类型），含义是基于该列生成Watermark，并且标识该列为Event Time列。您可以使用<rowtime_field>在作业代码中定义窗口。
withOffset	是目前提供的Watermark的生成策略，根据`<rowtime_field> - offset`生成Watermark的值。`withOffset`的第一个参数必须是<rowtime_field>。
offset	单位为毫秒，含义为Watermark值与Event Time值的偏移量。

通常一条记录中的某个字段就代表了该记录的发生时间。例如，表中`rowtime`字段的类型为TIMESTAMP，`1501750584000 (2017-08-03 08:56:24.000)`。定义一个基于该`rowtime`列，偏移4秒的Watermark的示例如下。

```
WATERMARK FOR rowtime AS withOffset(rowtime, 4000)
```

这条数据的Watermark时间为`1501750584000 - 4000 = 1501750580000 (2017-08-03 08:56:20.000)`。这条数据的Watermark时间含义：时间戳小于`1501750580000 (2017-08-03 08:56:20.000)`的数据已经全部到达。

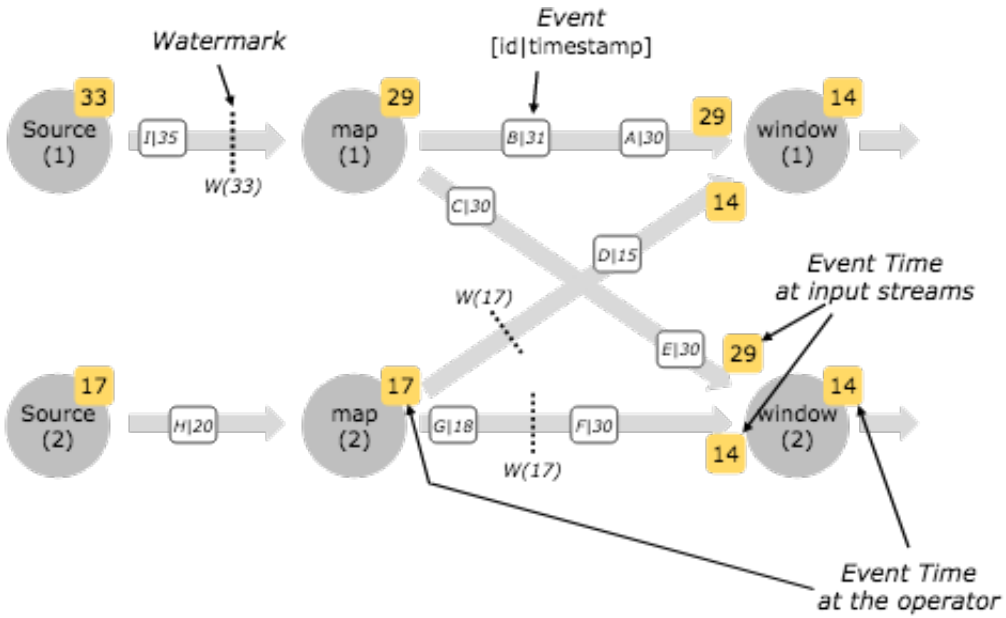
 说明:

- 在使用Event Time Watermark时的`rowtime`必须是TIMESTAMP类型。当前支持毫秒级别的、在Unix时间戳里是13位。如果是其他类型或是在Unix时间戳不是13位，建议使用计算列完成转换。
- Event Time和Processing Time的只能在源表上声明。

`Watermark`使用总结

- Watermark含义是所有时间戳`t'< t`的事件已经全部发生。若Watermark`t`已经生效，则后续Event Time小于`t`的记录将全部丢弃（后续支持用户配置，使Event Time小于`t`的数据也能继续更新）。

- 针对乱序的的流，Watermark至关重要。即使部分事件延迟到达，也不会过大影响窗口计算的正确性。
- 并行数据流中，当算子（Operator）有多个输入流时，算子的Event Time以最小流Event Time为准。



8.3.4 计算列

计算列可以使用其它列的数据，计算出其所属列的数值。如果您的数据源表中没有TIMESTAMP类型的列，可以使用计算列方法从其它类型的字段进行转换。

计算列概念

计算列是虚拟列，并非实际存储在表中。计算列可以通过表达式、内置函数、或是自定义函数等方式，使用其它列的数据，计算出其所属列的数值。计算列在Flink SQL中可以像普通字段一样被使用。

计算列的用途

目前Watermark的Event Time（也称为Rowtime）列只支持TIMESTAMP类型（未来将支持LONG类型）。Watermark只能定义在源表DDL中，如果您的源表中没有TIMESTAMP类型的列，可以使用计算列从其他类型的字段进行转换。

计算列语法

```
column_name AS computed_column_expression
```

计算列示例

Watermark的Rowtime必须是TIMESTAMP数据类型。当前实时计算支持毫秒级别的、在Unix时间戳里是13位TIMESTAMP数据类型。如果DataHub的TIME字段是微秒级别的（16位Unix时间戳），可以用计算列方法转换为13位的时间戳，如下所示。

```
CREATE TABLE test_stream(
  a INT,
  b BIGINT,
  `TIME` BIGINT,
  ts AS TO_TIMESTAMP(`TIME`/1000), --使用计算列方法，将16位时间戳转换为13位时间戳。
  WATERMARK FOR ts AS WITHOFFSET(ts, 1000)
) WITH (
  type = 'datahub',
  ...
);
```

源表数据中的字段`TIME`包含时间信息，为BIGINT类型。用计算列的功能将字段TIME转换成13位时间戳（TIMESTAMP）类型字段ts，并将ts字段作为Watermark的Rowtime字段。

8.4 数据类型

8.4.1 数据类型概述

本文为您介绍实时计算支持的数据类型以及数据类型之间的转换方法。

实时计算支持的数据类型

数据类型	说明	值域
VARCHAR	可变长度字符串	最大容量为4Mb
BOOLEAN	逻辑值	取值为TRUE、FALSE或UNKNOWN。
TINYINT	微整型，1字节整数。	-128到127
SMALLINT	短整型，2字节整数。	-32768到32767
INT	整型，4字节整数。	-2147483648到2147483647
BIGINT	长整型，8字节整数。	-9223372036854775808到9223372036854775807
FLOAT	4字节浮点型	6位数字精度

数据类型	说明	值域
DECIMAL	小数类型	示例：`123.45是DECIMAL(5,2)的值`。
DOUBLE	浮点型，8字节浮点型	15位十进制精度
DATE	日期类型	示例：`DATE'1969-07-20'`。
TIME	时间类型	示例：`TIME '20: 17: 40'`。
TIMESTAMP	时间戳，日期和时间	示例：`TIMESTAMP '1969-07-20 20:17:40'`。
VARBINARY	二进制数据	`byte[]` 数组

数据类型转换

是否可转	数据类型										
	VARCHAR	BOOLEAN	TINYINT	SMALLINT	INT	BIGINT	FLOAT	DOUBLE	DATE	TIMESTAMP	TIME
VARCHAR	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
BOOLEAN	Y	Y	N	N	N	N	N	N	N	N	N
TINYINT	Y	N	Y	Y	Y	Y	Y	Y	N	N	N
SMALLINT	Y	N	Y	Y	Y	Y	Y	Y	N	N	N
INT	Y	N	Y	Y	Y	Y	Y	Y	N	N	N
BIGINT	Y	N	Y	Y	Y	Y	Y	Y	N	N	N
FLOAT	Y	N	Y	Y	Y	Y	Y	Y	N	N	N
DOUBLE	Y	N	Y	Y	Y	Y	Y	Y	N	N	N
DATE	Y	N	N	N	N	N	N	N	Y	Y	N
TIMESTAMP	Y	N	N	N	N	N	N	N	Y	Y	Y
TIME	Y	N	N	N	N	N	N	N	N	Y	Y

类型转换示例

- 测试数据

var1（VARCHAR）	big1（BIGINT）
1000	323

- 测试语句

```
cast (var1 as bigint) as AA;
cast (big1 as varchar) as BB;
```

- 测试结果

AA（BIGINT）	BB（VARCHAR）
1000	323

8.4.2 数据类型之间运算关系

本文为您介绍实时计算数据类型之间的数学运算和逻辑运算。

数学运算和逻辑运算

运算语句	描述	numeric1和numeric2支持的数据类型	示例
numeric1 + numeric2	返回前后两数字数学运算的和	· INT · DOUBLE · DECIMAL · BIGINT	`2+4.2`
numeric1 - numeric2	返回前后两数字数学运算差值		`3-5.3`
numeric1 * numeric2	返回前后两数字数学运算积值		`2*4`
numeric1 / numeric2	返回前后两数字数学运算商值		2.4/5
numeric1 > numeric2	返回前后两数字是否大于的逻辑运算值		`2.4>5`
numeric1 < numeric2	返回前后两数字是否小于逻辑运算值		`2.4<5`
numeric1 >= numeric2	返回前后两数字是否大于等于的逻辑运算值		`2.4>=5`
numeric1 <= numeric2	返回前后两数字是否小于等于的逻辑运算值		`2.4<=5`
numeric1 = numeric2	返回前后两数字是否相同（等）逻辑运算值	· INT · DOUBLE · DECIMAL · BIGINT · VARCHAR	`'iphone' = 5`
numeric1 <> numeric2	返回前后两数字是否不相同（等）逻辑运算值		`'iphone' <> 5`

说明:

同一运算语句中numeric1和numeric2的数据类型需要保持一致。

8.5 创建数据视图

您可以通过创建实时计算数据视图简化开发过程。

语法

如果计算的逻辑比较复杂，您可以通过定义视图的方式创建数据视图，简化开发过程。

说明:

数据视图仅用于辅助计算逻辑的描述，不会产生数据的物理存储。

```
CREATE VIEW viewName[ (columnName[ , columnName]* ) ] AS queryStatement;
```

示例1

```
CREATE VIEW LargeOrders (r, t, c, u) AS
SELECT
    rowtime,
    productId,
    c,
    units
FROM
    orders;
INSERT INTO
    rds_output
SELECT
    r,
    t,
    c,
    u
FROM
    LargeOrders;
```

示例2

- **测试数据**

a（VARCHAR）	b（BIGINT）	c（TIMESTAMP）
test1	1	1506823820000
test2	1	1506823850000
test1	1	1506823810000
test2	1	1506823840000
test2	1	1506823870000
test1	1	1506823830000
test2	1	1506823860000

- **测试语句**

```
CREATE TABLE datahub_stream (
   a VARCHAR,
   b BIGINT,
   c TIMESTAMP,
   d AS PROCTIME()
) WITH (
  TYPE='datahub',
  ...
);
CREATE TABLE rds_output (
```

```
   a VARCHAR,
   b TIMESTAMP,
   cnt BIGINT,
   PRIMARY KEY(a)
)WITH(
  TYPE = 'rds',
  ...
);

CREATE VIEW rds_view AS
SELECT a,
   CAST(
      HOP_START(d, INTERVAL '5' SECOND, INTERVAL '30' SECOND) AS
TIMESTAMP
   ) AS cc,
   SUM(b) AS cnt
FROM
   datahub_stream
GROUP BY
    HOP(d, INTERVAL '5' SECOND, INTERVAL '30' SECOND),a;

INSERT INTO
   rds_output
SELECT
   a,
   cc,
   cnt
FROM
   rds_view
WHERE
   cnt=4
```

- **测试结果**

a(VARCHAR)	b (TIMESTAMP)	cnt (BIGINT)
test2	2017-11-06 16:54:10	4

8.6 DDL语句

8.6.1 DDL概述

本文为您介绍实时计算DDL语法以及在DDL使用过程中需要注意的字段映射和大小写敏感问题。

语法

```
CREATE TABLE tableName
      (columnName dataType [, columnName dataType ]*)
      [ WITH (propertyName=propertyValue [, propertyName=propertyValue
 ]*) ];
```

说明

阿里云实时计算本身不带有数据存储功能，所有涉及表创建DDL的操作，实际上均是对于外部数据表、存储的引用声明，如下所示。

```
CREATE TABLE mq_stream(
```

```
  a VARCHAR,
  b VARCHAR,
  c VARCHAR
) WITH (
  type='mq',
  topic='blink_mq_test',
  accessID='<yourAccessID>',
  accessKey='<yourAccessSecret>'
);
```

以上代码不是在Flink SQL中创建消息队列（MQ）源表的topic，而是声明了一个名称为mq_stream的表引用。下游所有对MQ的Topicblink_mq_test的相关DML操作，均可以用别名mq_stream代替。

- 实时计算声明表的作用域是当前作业（1个SQL文件提交后生成1个实时计算作业），即上述有关mq_stream的声明仅在当前SQL有效。相同实时计算项目下不同SQL文件（作业）中同样可以声明名称为mq_stream的表。
- 按照SQL标准定义，DDL语法中关键字、表名、列名等不区分大小写。
- 表名、列名必须以字母或者数字开头，并且名称中只能包含字母、数字或下划线。
- DDL声明不完全根据名称进行映射（取决于上游插件的性质）。建议您引用声明的字段名称、字段个数和外部表保持一致，避免因定义混乱而导致的数据错乱情况。

> 说明:
> 对于声明表和外部表：
> - 上下游插件支持根据KEY取值，则不要求两者字段数量一致，但字段名称需要一致。
> - 上下游插件不支持根据KEY取值，则需要字段数量和字段顺序一致。

字段映射

声明表的字段映射根据外部数据源是否有Shema，分为两大类别。

- 顺序映射

  适用于以MQ为代表的不带有Schema系统。这类系统通常是非结构化存储系统，不支持根据KEY取值。建议您在DDL SQL声明中对字段名称进行自定义，并且和外部表的字段类型、字段数量保持一致。

  如下以MQ的1条记录为例。

  ```
  asavfa,sddd32,sdfdsv
  ```

  按照命名规范设置MQ的字段名。

  ```
  CREATE TABLE mq_stream(
    a VARCHAR,
    b VARCHAR,
  ```

```
 c VARCHAR
) WITH (
 type='mq',
 topic='blink_mq_test',
 accessID='<yourAccessID>',
 accessKey='<yourAccessSecret>'
);
```

- 名称映射

适用于带有Schema的系统。这类系统在表存储级别定义了字段名称以及字段类型，支持根据KEY取值。建议您在Flink SQL的声明中保持和外部数据存储Schema定义一致，包括字段名称、字段数量以及字段的顺序。

说明:

如果外部数据存储的字段名称是大小写敏感类型（如表格存储），则需要在区分大小写的字段名称处使用反引号`` ` ``进行转换。在DDL语法中，声明表的字段名和外部表的字段名需要一致。

DataHub定义的Schema如下：

字段名	类型
name	STRING
age	BIGINT
value	STRING

说明:

DataHub中的STRING数据类型对应的是在实时计算中的VARCHAR类型。

关于上述DataHub声明的DDL如下：

```
create  table stream_result (
    `name` varchar,
    age bigint,
    `value` varchar
  ) with (
      type='datahub',
      endpoint='http://dh-cn-hangzhou.aliyuncs.com',
      accessID='<yourAccessID>',
      accessKey='<yourAccessSecret>'
      project='project',
      topic='topic'
  );
```

说明:

建议您将所有列进行声明引用。声明引用时可以减少字段，不能新增字段。

处理大小写敏感

SQL标准定义中，大小写是不敏感的。如下所示，2段语句的含义相同。

```
create table stream_result (
    name varchar,
    value varchar
);
```

```
create table STREAM_RESULT (
    NAME varchar,
    VALUE varchar
);
```

但实时计算引用的大量外部数据源中，存在要求大小写敏感的数据源。例如，表格存储（Table Store）对于大小写是敏感的。如果需要在Table Store定义大写`NAME`字段，我们应该如下定义。

```
create  table STREAM_RESULT (
    `NAME` varchar,
    `VALUE` varchar
);
```

在之后所有的DML操作中，对于这个字段的引用均需要添加反引号`` ` ``，如下所示。

```
INSERT INTO tableA
SELECT
  `NAME`,
  `VALUE`
FROM
  tableB;
```

相关章节

请参见以下文档，了解如何创建实时计算数据源表、数据结果表和数据维表。

- #unique_146
- #unique_147
- #unique_148

8.6.2 创建数据源表

8.6.2.1 数据源表概述

实时计算的数据源表是流式数据存储。流式数据存储驱动实时计算的运行，因此每个实时计算作业必须提供至少1个流式数据存储。

语法

```
  CREATE TABLE tableName
      (columnName dataType [, columnName dataType ]*)
```

```
      [ WITH (propertyName=propertyValue [, propertyName=propertyValue
 ]*) ];
```

示例

```
CREATE TABLE  datahub_stream(
  name VARCHAR,
  age BIGINT,
  birthday BIGINT
) WITH (
  type='datahub',
  endPoint='<yourEndpoint>',
  project='<yourProjectName>',
  topic='<yourDataHubTopic>',
  accessId='<yourAccessId>',
  accessKey='<yourAccessSecret>',
  startTime='2017-07-21 00:00:00'
);
```

获取数据源表属性字段

- **获取数据源表属性字段语法**

  实时计算在源表的DDL语句中提供 `HEADER` 关键字，用于获取源表中的属性字段。

  ```
  CREATE TABLE sourcetable
  (
    `timestamp`  VARCHAR HEADER,
    name         VARCHAR,
    MsgID        VARCHAR
  )WITH(
       type='<yourSourceTableType>'
  );
  ```

  上面示例中`timestamp`字段定义为HEADER，可从数据的属性字段读取数值，后续当成普通字段使用。

  说明:

  不同的源表（DataHub、Log Service或MQ等）存在不同的默认属性字段，部分源表支持设置自定义的属性字段，具体请参见对应的源表文档。

- **获取源表属性字段示例**

  以日志服务（Log service）为例，为您介绍如何获取源表属性字段。目前日志服务默认支持如下3个属性字段。

字段名	说明
`__source__`	消息源
`__topic__`	消息主题

字段名	说明
__timestamp__	日志时间

说明:

获取属性字段，除了按照正常逻辑声明外，还需要在类型声明后面加上HEADER。

示例如下：

- **示例数据**

```
__topic__:  ens_altar_flow
        result:  {"MsgID":"ems0a","Version":"0.0.1"}
```

- **示例语句**

```
CREATE TABLE sls_log (
  __topic__  VARCHAR HEADER,
  result     VARCHAR
)WITH(
  type ='sls'
);
CREATE TABLE sls_out (
  name     varchar,
  MsgID    varchar,
  Version  varchar
)WITH(
  type ='RDS'
);
INSERT INTO sls_out
SELECT
__topic__,
JSON_VALUE(result,'$.MsgID'),
JSON_VALUE(result,'$.Version')
FROM
sls_log
```

- **测试结果**

name(VARCHAT)	MsgID(VARCHAT)	Version(VARCHAT)
ens_altar_flow	ems0a	0.0.1

包含窗口函数的数据源表

实时计算可以基于Event Time和Processing Time这2种时间属性对数据进行窗口聚合。包含窗口函数的作业中，数据源表的声明中需要使用到#unique_117和#unique_116方法。实时计算基于时间属性的聚合详情，请参见时间属性。

支持创建的数据源表类型

实时计算支持创建多种类型的数据源表。详情请参见以下文档。

- #unique_151
- #unique_152
- #unique_144
- #unique_153
- 创建表格存储 TableStore源表
- 创建MaxCompute源表

8.6.2.2 创建数据总线 DataHub源表

本文为您介绍如何为实时计算创建数据总线 DataHub源表以及创建过程涉及到的属性字段、WITH参数和类型映射。

说明:

DataHub未正式商用。

什么是数据总线

DataHub作为流式数据总线，为阿里云数加平台提供了大数据的入口服务。实时计算可以使用DataHub作为流式数据存储的源头和输出的目的端。

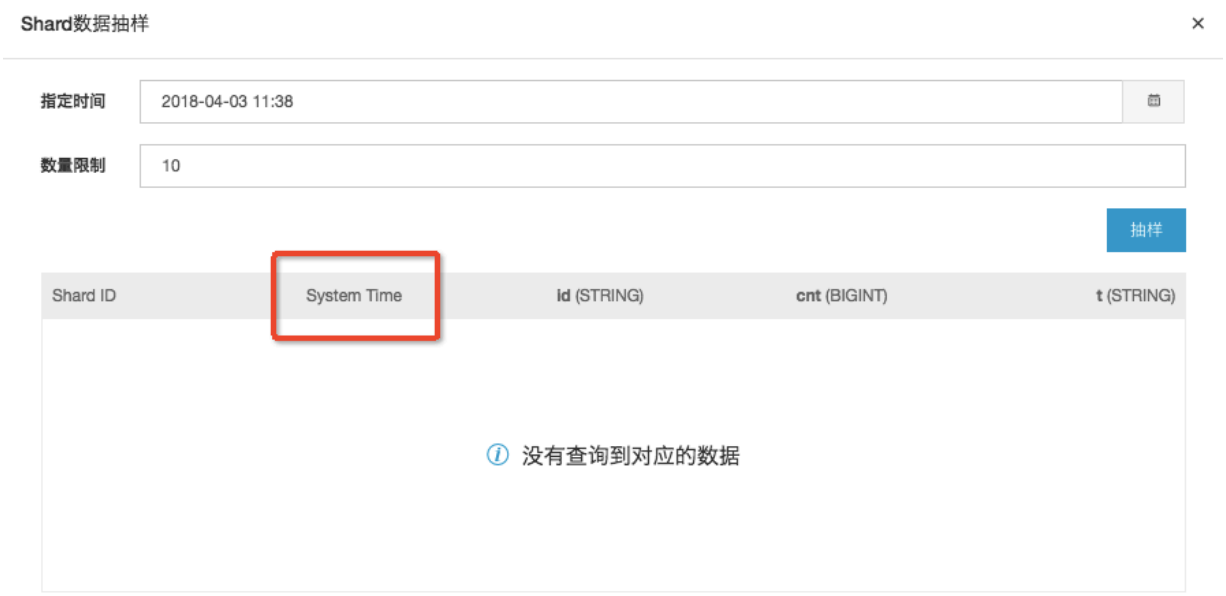
示例

DataHub可以作为实时计算的数据输入，示例如下。

```
CREATE TABLE datahub_stream(
  name VARCHAR,
  age BIGINT,
  birthday BIGINT
) WITH (
  type='datahub',
  endPoint='http://dh-cn-hangzhou.aliyun-inc.com',
  project='<yourProjectName>',
  topic='<yourTopic>',
  accessId='<yourAccessID>',
  accessKey='<yourAccessSecret>',
  startTime='2017-07-21 00:00:00'
);
```

属性字段

Flink SQL支持获取DataHub的属性字段。通过读取属性字段可以获得每条信息输入DataHub的系统时间（System Time）。



字段名	注释说明
timestamp	每条记录写入DataHub的系统时间（System Time）

说明:

获取属性字段的方法参见#unique_157/unique_157_Connect_42_section_v3l_32x_tgb。

WITH参数

目前只支持tuple模式的topic。

参数	注释说明	备注
endPoint	消费端点信息	参见DataHub域名列表 。
accessId	读取的accessId	无。
accessKey	读取的密钥	无。
project	读取的项目	无。
topic	project下的具体的topic	无。
startTime	启动位点的时间	格式为`yyyy-MM-dd hh:mm:ss`。
maxRetryTimes	读取最大尝试次数	可选，默认值为20。
retryIntervalMs	重试间隔	可选，默认值为1000，单位为毫秒。
batchReadSize	单次读取条数	可选，默认值为10，可设置的最大值为1000。

参数	注释说明	备注
lengthCheck	单行字段条数检查策略	可选，默认值为NONE，表示： · 解析出的字段数大于定义字段数时，按从左到右的顺序，取定义字段数量的数据。 · 解析出的字段数小于定义字段数时，跳过这行数据。 其它可选值为SKIP、EXCEPTION和PAD。 · SKIP：解析出的字段数和定义字段数不同时跳过这行数据。 · EXCEPTION：解析出的字段数和定义字段数不同时提示异常。 · PAD：按从左到右顺序填充。 - 解析出的字段数大于定义字段数时，按从左到右的顺序，取定义字段数量的数据。 - 解析出的字段数小于定义字段数时，在行尾用null填充缺少的字段。
columnErrorDebug	是否开启调试功能	可选，默认值为false，表示关闭调试功能。开启调试功能参数值为true，将打印解析异常的日志。

类型映射

DataHub和实时计算字段类型对应关系如下，建议使用该对应关系时进行DDL声明。

DataHub字段类型	实时计算字段类型
BIGINT	BIGINT
TIMESTAMP	
STRING	VARCHAR
DOUBLE	DOUBLE
BOOLEAN	BOOLEAN
DECIMAL	DECIMAL

 说明：

DataHub的TIMESTAMP精确到微秒，在Unix时间戳中为16位，但实时计算定义的TIMESTAMP精确到毫秒，在Unix时间戳中为13位，建议使用BIGINT进行映射。如果需要使用TIMESTAMP，建议使用#unique_116进行转换。

8.6.2.3 创建日志服务 LOG源表

本文为您介绍如何为实时计算创建日志服务 LOG源表以及创建过程涉及到的属性字段、WITH参数和类型映射。

什么是日志服务

日志服务 LOG是针对日志类数据的一站式服务，在阿里巴巴集团经历大量大数据场景锤炼而成。日志服务本身是流数据存储，实时计算能将其作为流式数据输入。对于日志服务而言，数据格式类似JSON，示例如下。

```
{
    "a": 1000,
    "b": 1234,
    "c": "li"
}
```

对于实时计算而言，我们需要定义的DDL如下（代码中的sls代表LogService）。

```
create table sls_stream(
  a int,
  b int,
  c VARCHAR
) with (
  type ='sls',
  endPoint ='<yourEndpoint>',
  accessId ='<yourAccessId>',
  accessKey ='<yourAccessKey>',
  startTime = '<yourStartTime>',
  project ='<yourProjectName>',
  logStore ='<yourLogStoreName>',
  consumerGroup ='<yourConsumerGroupName>'
);
```

属性字段

目前Flink SQL默认支持3个LogServic属性字段的获取，也支持其它自定义字段的写入。

字段名	注释说明
`__source__`	消息源
`__topic__`	消息主题
`__timestamp__`	日志时间

属性字段使用说明

为了获取这些属性字段，除了正常逻辑声明外，还需要在类型声明后加上HEADER关键字。示例如下。

- 测试数据

```
      __topic__:  ens_altar_flow
         result:  {"MsgID":"ems0a","Version":"0.0.1"}
```

- 测试语句

```
CREATE TABLE sls_log (
  __topic__  VARCHAR HEADER,
  `result`     VARCHAR
)
WITH
(
  type ='sls'
);
CREATE TABLE sls_out (
  `name`     VARCHAR,
  MsgID    VARCHAR,
  `Version`  VARCHAR
)
WITH
(
  type ='RDS'
);
INSERT INTO sls_out
SELECT
__topic__,
JSON_VALUE(`result`,'$.MsgID'),
JSON_VALUE(`result`,'$.`Version`')
FROM
sls_log
```

- 测试结果

name(VARCHAR)	MsgID(VARCHAR)	Version(VARCHAR)
ens_altar_flow	ems0a	0.0.1

WITH参数

参数	注释说明	备注
endPoint	消费端点信息	#unique_43。
accessId	LogService读取的accessKey	无。
accessKey	LogService读取的密钥	无。
project	读取的LogService项目	无。
logStore	Project下的具体的LogStore名称	无。
consumerGroup	消费组名	您可以自定义消费组名（没有固定格式）。

参数	注释说明	备注
startTime	消费日志开始的时间点	无。
heartBeatIntervalMills	可选，消费客户端心跳间隔时间	默认为10。
maxRetryTimes	读取最大尝试次数	可选，默认为5。
batchGetSize	单次读取logGroup条数	可选，默认为10。
lengthCheck	单行字段条数检查策略	可选，默认值为NONE，表示： · 解析出的字段数大于定义字段数时，按从左到右的顺序，取定义字段数量的数据。 · 解析出的字段数小于定义字段数时，跳过这行数据。 其它可选值为SKIP、EXCEPTION和PAD。 · SKIP：解析出的字段数和定义字段数不同时跳过这行数据。 · EXCEPTION：解析出的字段数和定义字段数不同时提示异常。 · PAD：按从左到右顺序填充。 - 解析出的字段数大于定义字段数时，按从左到右的顺序，取定义字段数量的数据。 - 解析出的字段数小于定义字段数时，在行尾用null填充缺少的字段。
columnErrorDebug	是否打开调试开关	可选，默认为false，不打开。如果选择打开，将打印解析异常的日志。
directMode	是否使用LogService的直连模式	可选，默认为false，不开启LogService直连模式。开启后作业直接连接LogService的nginx服务器，网络可通时，连接效率能够提升。

说明:

· LogService暂不支持MAP类型的数据。

· 字段顺序支持无序（建议字段顺序和表中定义一致）。

- 输入数据源为JSON形式时，注意定义分隔符，并且需要采用内置函数#unique_159分析，否则就会解析失败。报错如下:

```
2017-12-25 15:24:43,467 WARN [Topology-0 (1/1)] com.alibaba.blink
.streaming.connectors.common.source.parse.DefaultSourceCollector
 - Field missing error, table column number: 3, data column number
: 3, data filed number: 1, data: [{"lg_order_code":"LP00000005","
activity_code":"TEST_CODE1","occur_time":"2017-12-10 00:00:01"}]
```

- batchGetSize设置不能超过1000，否则会报错。
- batchGetSize指明的是logGroup获取的数量，如果单条logItem的大小和batchGetSize都很大，有可能会导致频繁的垃圾回收（Garbage Collection），这种情况下该参数应调小。

类型映射

日志服务和实时计算字段类型对应关系如下。建议您使用该对应关系进行DDL声明:

日志服务字段类型	实时计算字段类型
STRING	VARCHAR

8.6.2.4 创建消息队列 MQ源表

本文为您介绍实时计算创建消息队列 MQ源表以及创建过程涉及到的CSV类格式、WITH参数和类型映射。

什么是消息队列MQ

消息队列 MQ是阿里云专业消息中间件，是企业级互联网架构的核心产品。实时计算可以将消息队列作为流式数据输入。

示例

```
create table mq_stream(
 x varchar,
 y varchar,
 z varchar
) with (
 type='mq',
 topic='<yourTopicName>',
 endpoint='<yourEndpoint>',
 pullIntervalMs='1000',
 accessId='<yourAccessId>',
 accessKey='<yourAccessSecret>',
 startMessageOffset='1000',
 consumerGroup='<yourConsumerGroup>',
 fieldDelimiter='|'
);
```

 说明:

MQ实际上是一个非结构化存储格式，对于数据的Schema不提供强制定义，完全由业务层指定。目前实时计算支持类CSV格式文本和二进制格式。

CSV类格式

假设您的1条CSV格式消息记录如下。

```
1,name,male
2,name,female
```

说明:

1条MQ消息可以包括0条到多条数据记录，记录之间使用\n分隔。

在实时计算作业中，声明MQ数据源表的DDL如下。

```
create table mq_stream(
 id varchar,
 name varchar,
 gender varchar
) with (
 type='mq',
 topic='<yourTopicName>',
 endpoint='<ourEndpoint>',
 pullIntervalMs='1000',
 accessId='<yourAccessId>',
 accessKey='<yourAccessSecret>',
 startMessageOffset='1000',
 consumerGroup='<yourConsumerGroup>',
 fieldDelimiter='|'
);
```

二进制格式

二进制格式测试代码如下。

```
create table source_table (
  mess varbinary
) with (
  type = 'mq',
  endpoint = '<yourEndpoint>',
  pullIntervalMs='500',
  accessId='<yourAccessId>',
  accessKey='<yourAccessSecret>',
  topic = '<yourTopicName>',
  consumerGroup='<yourConsumerGroup>'
);
create table out_table (
  commodity varchar
)with(
  type='print'
);
INSERT INTO out_table
SELECT
  cast(mess as varchar)
```

```
FROM source_table
```

说明:

- `cast(mess as varbinary)` 需在实时计算2.0及以上版本使用，若版本低于2.0，请先升级。
- VARBINARY只能入参一次。

WITH参数

参数	注释说明	备注
topic	topic名称	无。
endPoint	endPoint地址	· 公共云内网（阿里云经典网络/VPC）接入地址： - 华东1、华东2、华北1华北2、华南1、中国（香港）：`onsaddr-internal.aliyun.com:8080` - 新加坡：`ap-southeastaddr-internal.aliyun.com:8080` - 迪拜：`ons-me-east-1-internal.aliyuncs.com:8080` - 孟买：`ons-ap-south-1-internal.aliyuncs.com:8080` - 吉隆坡：`ons-ap-southeast-3-internal.aliyun.com:8080` · 公共云公网接入地址：`onsaddr-internet.aliyun.com:80`
accessId	accessId	无。
accessKey	accessKey	无。
consumerGroup	订阅消费group名称	无。
pullIntervalMs	拉取时间间隔	单位为毫秒。
startTime	消息消费启动的时间点	可选。
startMessageOffset	消息开始的偏移量	可选，如果填写，将优先以startMessageoffset的位点开始加载。
tag	订阅的标签	可选。
lineDelimiter	解析block时行分隔符	可选，默认为 `\n`。

参数	注释说明	备注
fieldDelimiter	字段分隔符	可选，默认为\u0001。表示在只读模式下以\u0001（\u0001在只读模式不可见）作为分隔符；在编辑模式下以^A作为分隔符。
encoding	编码格式	可选，默认为 utf-8。
lengthCheck	单行字段条数检查策略	可选，默认值为NONE，表示： · 解析出的字段数大于定义字段数时，按从左到右的顺序，取定义字段数量的数据。 · 解析出的字段数小于定义字段数时，跳过这行数据。 其它可选值为SKIP、EXCEPTION和PAD。 · SKIP：解析出的字段数和定义字段数不同时跳过这行数据。 · EXCEPTION：解析出的字段数和定义字段数不同时提示异常。 · PAD：按从左到右顺序填充。 - 解析出的字段数大于定义字段数时，按从左到右的顺序，取定义字段数量的数据。 - 解析出的字段数小于定义字段数时，在行尾用null填充缺少的字段。
columnErrorDebug	是否打开调试开关	可选，默认为false。如果设置为true，则打印解析异常的log。
instanceID	Topic所属的分组	可选。若MQ实例为DEFAULT_INSTANCE，则不可使用instanceID参数。

类型映射

MQ字段类型	实时计算字段类型
STRING	VARCHAR

8.6.2.5 创建消息队列 Kafka源表

本文为您介绍如何创建实时计算消息队列 Kafka源表以及Kafka版本对应关系和Kafka消息解析示例。

 说明:

本文档仅适用于独享模式。

什么是Kafka源表

Kafka源表的实现迁移自社区的Kafka版本实现。Kafka源表数据解析流程：Kafka Source Table -> UDTF ->Realtime Compute -> Sink。从Kakfa读入的数据为VARBINARY（二进制）格式，需要使用#unique_162的方式，将二进制格式解析成格式化数据。

DDL定义

Kafka源表定义DDL部分必须与以下SQL完全一致，WITH参数中的设置可更改。

```
create table kafka_stream(   --必须和Kafka源表中的5个字段的顺序保持一致。
  messageKey VARBINARY,
  `message`    VARBINARY,
  topic      VARCHAR,
  `partition`  INT,
  `offset`     BIGINT
) with (
  type ='kafka010',
  topic = '<yourTopicName>',
  `group.id` = '<yourGroupId>',
  ...
);
```

WITH参数

- 通用配置

参数	注释说明	备注
type	kafka对应版本	必选，必须是Kafka08、Kafka09、Kafka010或Kafka011。版本对应关系见Kafka版本对应关系。
topic	读取的单个topic	无。
topicPattern	读取一批topic的表达式	无。

参数	注释说明	备注
startupMode	启动位点	默认参数为GROUP_OFFSETS。 - EARLIEST：从Kafka最早分区开始读取。 - GROUP_OFFSETS：根据Group读取。 - LATEST：从Kafka最新位点开始读取。 - TIMESTAMP：从指定的时间点读取。（Kafka010、Kafka011支持。） 说明: ■ 设置为TIMESTAMP模式时，需要在作业参数中明文指定时区。例如，`blink.job.timeZone=Asia/Shanghai`。 ■ Group_OFFSETS模式下，GroupID的第一次消费，没有设置偏移（Offset）值，默认从Kafka的最早分区开始读取数据。 ■ 阿里云Kafka产品基于开源Kafka 0.10.0版本，不支持`startupMode='TIMESTAMP'`模式。
partitionDiscoveryIntervalMS	定时检查是否有新分区产生	默认值为60000，单位为毫秒。
extraConfig	额外的kafkaConsumer配置项目	可选，未在可选配置项中，但是额外期望的配置。

- kafka08配置
  - kafka08必选配置

参数	注释说明	备注
group.id	组名	消费组id

参数	注释说明	备注
zookeeper.connect	zk链接地址	zk连接id

- 可选配置Key
  - consumer.id
  - socket.timeout.ms
  - fetch.message.max.bytes
  - num.consumer.fetchers
  - auto.commit.enable
  - auto.commit.interval.ms
  - queued.max.message.chunks
  - rebalance.max.retries
  - fetch.min.bytes
  - fetch.wait.max.ms
  - rebalance.backoff.ms
  - refresh.leader.backoff.ms
  - auto.offset.reset
  - consumer.timeout.ms
  - exclude.internal.topics
  - partition.assignment.strategy
  - client.id
  - zookeeper.session.timeout.ms
  - zookeeper.connection.timeout.ms
  - zookeeper.sync.time.ms
  - offsets.storage
  - offsets.channel.backoff.ms
  - offsets.channel.socket.timeout.ms
  - offsets.commit.max.retries
  - dual.commit.enabled
  - partition.assignment.strategy
  - socket.receive.buffer.bytes
  - fetch.min.bytes

- kafka09/kafka010/kafka011配置
  - kafka09/kafka010/kafka011必选配置

参数	注释说明	备注
group.id	组名	消费组ID
bootstrap.servers	Kafka集群地址	Kafka集群地址

  - kafka09/kafka010/kafka011可选配置

    请参见如下Kafka官方文档进行配置。

    ■ Kafka09

    ■ Kafka010

    ■ Kafka011

    当需要配置某选项时，在DDL中的with部分增加对应的参数即可。例如，配置SASL登录，需增加3个参数`security.protocol`，`sasl.mechanism`和`sasl.jaas.config`，示例如下。

```
create table kafka_stream(
  messageKey varbinary,
  `message` varbinary,
  topic varchar,
  `partition` int,
  `offset` bigint
) with (
  type ='kafka010',
  topic = '<yourTopicName>',
  `group.id` = '<yourGroupId>',
  ...,
  `security.protocol`=SASL_PLAINTEXT,
  `sasl.mechanism`=PLAIN,
  `sasl.jaas.config`='org.apache.kafka.common.security.plain
.PlainLoginModule required username="yourUserName" password="
yourPassword";'
);
```

Kafka版本对应关系

type	Kafka版本
Kafka08	0.8.22
Kafka09	0.9.0.1
Kafka010	0.10.2.1
Kafka011	0.11.0.2

Kafka消息解析示例

- 示例1

  将Kafka中的数据输入实时计算，并将计算结果输出到RDS。

  - Kafka中保存了JSON格式数据，需要用Realtime Compute进行计算，消息格式为：

    ```
    {
      "name":"Alice",
      "age":13,
      "grade":"A"
    }
    ```

    整个计算流程为：Kafka Source->UDTF->Realtime Compute->RDS Sink

  - 示例代码

    ■ SQL

    ```
    -- 定义解析Kakfa message的UDTF。
    CREATE FUNCTION kafkaparser AS 'com.alibaba.kafkaUDTF';

    -- 定义源表。注意：Kafka源表DDL字段必须与以下示例完全一致。WITH中参数可以修改。
    CREATE TABLE kafka_src (
        messageKey  VARBINARY,
        `message`   VARBINARY,
        topic       VARCHAR,
        `partition` INT,
        `offset`    BIGINT
    ) WITH (
        type = 'kafka010',    --请参见Kafka版本对应关系。
        topic = 'test_kafka_topic',
        `group.id` = 'test_kafka_consumer_group',
        bootstrap.servers = 'ip1:port1,ip2:port2,ip3:port3'
    );
    CREATE TABLE rds_sink (
      name       VARCHAR,
      age        INT,
      grade      VARCHAR,
      updateTime TIMESTAMP
    ) WITH(
     type='rds',
     url='jdbc:mysql://localhost:3306/test',
     tableName='test4',
     userName='test',
     password='<yourDatabasePassword>'
    );

    -- 使用UDTF，将二进制数据解析成格式化数据。
    CREATE VIEW input_view (
        name,
        age,
        grade,
        updateTime
    ) AS
    SELECT
        T.name,
        T.age,
        T.grade,
    ```

```
    T.updateTime
FROM
    kafka_src as S,
    LATERAL TABLE (kafkaparser (`message`)) as T (
        name,
        age,
        grade,
        updateTime
    );

-- 使用解析出的格式化数据进行计算，并将结果输出到RDS。
INSERT INTO rds_sink
  SELECT
      name,
      age,
      grade,
      updateTime
  FROM input_view;
```

■ UDTF

说明:

UDTF创建步骤请参见#unique_162。实时计算2.2.4版本Maven依赖，示例如下。

```
    <dependencies>
        <dependency>
            <groupId>org.apache.flink</groupId>
            <artifactId>flink-core</artifactId>
            <version>blink-2.2.4-SNAPSHOT</version>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>org.apache.flink</groupId>
            <artifactId>flink-streaming-java_2.11</artifactId>
            <version>blink-2.2.4-SNAPSHOT</version>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>org.apache.flink</groupId>
            <artifactId>flink-table_2.11</artifactId>
            <version>blink-2.2.4-SNAPSHOT</version>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>fastjson</artifactId>
            <version>1.2.9</version>
        </dependency>
    </dependencies>
```

```
package com.alibaba;

import com.alibaba.fastjson.JSONObject;
import org.apache.flink.table.functions.TableFunction;
import org.apache.flink.table.types.DataType;
import org.apache.flink.table.types.DataTypes;
import org.apache.flink.types.Row;
import java.io.UnsupportedEncodingException;
import java.sql.Timestamp;
```

```
public class kafkaUDTF extends TableFunction<Row> {
    public void eval(byte[] message) {
        try {
            /* input message :
                {
                  "name":"Alice",
                  "age":13,
                  "grade":"A",
                  "updateTime":1544173862
                }
            */
            String msg = new String(message, "UTF-8");
            try {
                JSONObject data = JSON.parseObject(msg);
                if (data != null) {
                    String name = data.getString("name") ==
null ? "null" : data.getString("name");
                    Integer age = data.getInteger("age") ==
null ? 0 : data.getInteger("age");
                    String grade = data.getString("grade") ==
null ? "null" : data.getString("grade");
                    Timestamp updateTime = data.getTimestamp("
updateTime");

                    Row row = new Row(4);
                    row.setField(0, name);
                    row.setField(1, age);
                    row.setField(2, grade);
                    row.setField(3,updateTime );

                    System.out.println("Kafka message str ==>"
 + row.toString());
                    collect(row);
                }
            } catch (ClassCastException e) {
                System.out.println("Input data format error.
Input data " + msg + "is not json string");
            }
        } catch (UnsupportedEncodingException e) {
            e.printStackTrace();
        }
    }
    @Override
    // 如果返回值是Row，就必须重载实现这个方法，显式地告诉系统返回的字段类
型。
    public DataType getResultType(Object[] arguments, Class[]
argTypes) {
        return DataTypes.createRowType(DataTypes.STRING,
DataTypes.INT, DataTypes.STRING, DataTypes.TIMESTAMP);
    }
}
```

- 示例2

  从Kafka读出数据，输入实时计算，进行窗口计算。

  - 按照实时计算目前的设计，滚窗/滑窗等窗口操作，需要（且必须）在源表DDL上定义#unique_117。Kafka源表比较特殊。如果要以Kafka中message字段中的的Event Time进行窗口操作，需要先从message字段，使用UDX解析出Event Time，才能定

义Watermark。 在Kafka源表场景中，需要使用#unique_116。 例如，Kafka中写入的数据如下：

`2018-11-11 00:00:00|1|Anna|female` 。计算流程为：Kafka Source->UDTF->Realtime Compute->RDS Sink。

- 示例代码

  ■ SQL

```
-- 定义解析Kakfa message的UDTF。
CREATE FUNCTION kafkapaser AS 'com.alibaba.kafkaUDTF';
CREATE FUNCTION kafkaUDF AS 'com.alibaba.kafkaUDF';

-- 定义源表，注意：Kfka源表DDL字段必须与以下例子一模一样。WITH中参数可改。
create table kafka_src (
    messageKey  VARBINARY,
    `message`   VARBINARY,
    topic       VARCHAR,
    `partition` INT,
    `offset`    BIGINT,
    ctime AS TO_TIMESTAMP(kafkaUDF(`message`)), -- 定义计算列，计算列可理解为占位符，源表中并没有这一列，其中的数据可经过下游计算得出。注意:计算列的类型必须为TIMESTAMP才能在做watermark。
    watermark for `ctime` as withoffset(`ctime`,0) -- 在计算列上定义watermark
) WITH (
    type = 'kafka010',    -- 请参见Kafka版本对应关系。
    topic = 'test_kafka_topic',
    `group.id` = 'test_kafka_consumer_group',
    bootstrap.servers = 'ip1:port1,ip2:port2,ip3:port3'
);

create table rds_sink (
  name       VARCHAR,
  age        INT,
  grade      VARCHAR,
  updateTime TIMESTAMP
) WITH(
 type='rds',
 url='jdbc:mysql://localhost:3306/test',
 tableName='test4',
 userName='test',
 password='******'
);

-- 使用UDTF，将二进制数据解析成格式化数据。
CREATE VIEW input_view (
    name,
    age,
    grade,
    updateTime
) AS
SELECT
    COUNT(*) as cnt,
    T.ctime,
    T.order,
    T.name,
    T.sex
from
```

```
    kafka_src as S,
    LATERAL TABLE (kafkapaser (`message`)) as T (
        ctime,
        order,
        name,
        sex
    )
Group BY T.sex,
        TUMBLE(ctime, INTERVAL '1' MINUTE);

-- 对input_view中输出的数据做计算。
CREATE VIEW view2 (
    cnt,
    sex
) AS
SELECT
    COUNT(*) as cnt,
    T.sex
from
    input_view
Group BY sex, TUMBLE(ctime, INTERVAL '1' MINUTE);

-- 使用解析出的格式化数据进行计算，并将结果输出到RDS。
insert into rds_sink
  SELECT
      cnt,sex
  from view2;
```

■ UDF&UDTF

说明:

UDF和UDTF创建步骤请参见自定义标量函数（UDF）和#unique_162。实时计算2.2.4版本Maven依赖，示例如下。

```
  <dependencies>
        <dependency>
            <groupId>org.apache.flink</groupId>
            <artifactId>flink-core</artifactId>
            <version>blink-2.2.4-SNAPSHOT</version>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>org.apache.flink</groupId>
            <artifactId>flink-streaming-java_2.11</artifactId>
            <version>blink-2.2.4-SNAPSHOT</version>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>org.apache.flink</groupId>
            <artifactId>flink-table_2.11</artifactId>
            <version>blink-2.2.4-SNAPSHOT</version>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>fastjson</artifactId>
            <version>1.2.9</version>
        </dependency>
```

```
    </dependencies>
```

■ UDTF

```
package com.alibaba;

import com.alibaba.fastjson.JSONObject;
import org.apache.flink.table.functions.TableFunction;
import org.apache.flink.table.types.DataType;
import org.apache.flink.table.types.DataTypes;
import org.apache.flink.types.Row;
import java.io.UnsupportedEncodingException;

/**
  以下例子解析输入Kafka中的JSON字符串，并将其格式化输出。
**/
public class kafkaUDTF extends TableFunction<Row> {

    public void eval(byte[] message) {
        try {
          // 读入一个二进制数据，并将其转换为String格式
            String msg = new String(message, "UTF-8");

                // 提取JSON Object中各字段。
                    String ctime = Timestamp.valueOf(data.
split('\\|')[0]);
                    String order = data.split('\\|')[1];
                    String name = data.split('\\|')[2];
                    String sex = data.split('\\|')[3];

                    // 将解析出的字段放到要输出的Row()对象。
                    Row row = new Row(4);
                    row.setField(0, ctime);
                    row.setField(1, age);
                    row.setField(2, grade);
                    row.setField(3, updateTime);

                    System.out.println("Kafka message str
 ==>" + row.toString());

                    // 输出一行
                    collect(row);

            } catch (ClassCastException e) {
                System.out.println("Input data format error.
Input data " + msg + "is not json string");
            }


        } catch (UnsupportedEncodingException e) {
            e.printStackTrace();
        }

    }

    @Override
    // 如果返回值是Row，就必须重载实现这个方法，显式地告诉系统返回的字
段类型。
    // 定义输出Row()对象的字段类型。
    public DataType getResultType(Object[] arguments, Class
[] argTypes) {
```

```
        return DataTypes.createRowType(DataTypes.TIMESTAMP
,DataTypes.STRING, DataTypes.Integer, DataTypes.STRING,
DataTypes.STRING);
    }

}
```

■ UDF

```
package com.alibaba;
package com.hjc.test.blink.sql.udx;
import org.apache.flink.table.functions.FunctionContext;
import org.apache.flink.table.functions.ScalarFunction;

public class KafkaUDF extends ScalarFunction {
    // 可选，open方法可以不写。
    // 需要import org.apache.flink.table.functions.FunctionCo
ntext;

    public String eval(byte[] message) {

         // 读入一个二进制数据，并将其转换为String格式。
        String msg = new String(message, "UTF-8");
        return msg.split('\\|')[0];
    }
    public long eval(String b, String c) {
        return eval(b) + eval(c);
    }
    //可选，close方法可以不写。
    @Override
    public void close() {
        }
}
```

自建Kafka

- 示例

```
create table kafka_stream(
  messageKey VARBINARY,
  `message` VARBINARY,
  topic varchar,
  `partition` int,
  `offset` bigint
) with (
  type ='kafka011',
  topic = 'kafka_01',
  `group.id` = 'CID_blink',
  bootstrap.servers = '192.168.0.251:****'
);
```

- WITH参数

  请参见文档开始部分WITH参数。

  说明:

  - bootstrap.servers参数需要填写自建的地址和端口号。

- 实时计算仅在2.2.6及以上版本中，支持阿里云Kafka或自建Kafka的TPS、RPS等指标信息的显示。

常见问题

Q：作业启动时产生如下报错：

```
ERR_ID:
     SQL-00010007
CAUSE:
     Could not create table 'kafka_source' as source table
ACTION:
     Please refer to details section for hint.
     If it doesn't help, please contact customer support
DETAIL:
     java.lang.IllegalArgumentException: Startup time[1566481803000]
must be before current time[1566453003356].
```

A：时区设置错误导致以上报错。请在作业参数中增加如下参数：

```
blink.job.timeZone=Asia/Shanghai
```

8.6.2.6 创建表格存储 TableStore源表

本文为您介绍如何创建实时计算表格存储 TableStore源表。

 说明：

本文仅适用于实时计算3.2.2及以上版本。

什么是表格存储 Table Store

表格存储 TableStore是构建在阿里云飞天分布式系统之上的分布式NoSQL数据存储服务。表格存储通过数据分片和负载均衡技术，实现数据规模与访问并发上的无缝扩展，提供海量结构化数据的存储和实时访问服务。

表格存储通道服务

表格存储通道服务是基于表格存储数据接口的全增量一体化服务，它通过Tunnel Service API和SDK为您提供了增量、全量和增量加全量，3种类型的分布式数据实时消费通道。通过Tunnel Service数据通道，您可以使用流式计算的方式，对表中存量或新增数据的进行消费。实时计算可以将Tunnel Service数据通道作为流式数据的输入，每条数据类似一个JSON格式。Tunnel Service数据通道的示例如下。

```
{
            "OtsRecordType": "PUT",  // 数据操作类型，包括PUT、UPDATE和
DELETE。
            "OtsRecordTimestamp": 1506416585740836, //数据写入时间（单位
为微秒）；全量数据时为0。
            "PrimaryKey": [
                {
                    "ColumnName": "pk_1", //第1主键列。
```

```
                    "Value": 1506416585881590900
                },
                {
                    "ColumnName": "pk_2", //第2主键列。
                    "Value": "string_pk_value"
                }
            ],
            "Columns": [
                {
                    "OtsColumnType": "Put", // 列操作类型，包括PUT、
DELETE_ONE_VERSION和DELETE_ALL_VERSION。
                    "ColumnName": "attr_0",
                    "Value": "hello_table_store",
                },
                {
                    "OtsColumnType": "DELETE_ONE_VERSION", // DELETE操
作没有Value字段。
                    "ColumnName": "attr_1"
                }
            ]
}
```

说明:

由于OTS源表中的Primary Key底层可进行Retract撤回机制，OTS源表暂不支持用Event Time作为窗口函数的时间类型。OTS源表仅支持Processing Time作为窗口函数的时间类型。Event Time和Processing Time详细说明，请参见#unique_164。

DDL定义

实时计算支持使用TableStore作为结果输出。针对本文Tunnel Service数据通道示例的TableStore源表DDL，示例如下。

```
create table tablestore_stream(
  pk_1 BIGINT,
  pk_2 VARCHAR,
  attr_0 VARCHAR,
  attr_1 DOUBLE,
  OtsRecordType  VARCHAR  HEADER, --属性字段后边需要增加HEADER。
  primary key(pk_1, pk_2)
) with (
  type ='ots',
  endPoint ='http://blink-demo.cn-hangzhou.vpc.tablestore.aliyuncs.com
',
  instanceName = "blink-demo",
  tableName ='demo_table',
  tunnelName = 'blink-demo-stream',
  accessId ='<yourAccessID>',
  accessKey ='<yourAccessSecret>',
  ignoreDelete = 'false' //是否忽略DELETE操作的数据。
);
```

属性字段

表格存储源表属性字段的获取和使用方法，请参见#unique_157/unique_157_Connect_42_section_v3l_32x_tgb。

字段名	注释说明
OtsRecordType	数据操作类型
OtsRecordTimestamp	数据操作时间（全量数据时为0）
<列名>_OtsColumnType	某列的操作类型

WITH参数

参数	注释说明	备注
endPoint	表格存储的实例访问地址	VPC网络环境需要选择实例的VPC Endpoint。
instanceName	表格存储的实例名称	无。
tableName	表格存储的数据表名	无。
tunnelName	表格存储数据表的数据通道名	无。
accessId	表格存储读取的accessKey	无。
accessKey	表格存储读取的秘钥	无。
ignoreDelete	是否忽略DELETE操作类型的实时数据	可选，默认为false。

CACHE参数

参数	注释说明	备注
cache	缓存策略	默认 None，可选 LRU，ALL。 · None：无缓存。 · LRU：最近使用策略缓存。需要配置cacheSize参数和cacheTTLMs参数。 · ALL：全量缓存策略。
cacheSize	缓存大小	选择LRU缓存策略后，可以设置缓存大小，默认为10000行。
cacheTTLMs	缓存超时时间，单位为毫秒。	默认缓存不超时，单位为毫秒。不同缓存策略下的功能如下： · LRU：设置缓存失效的超时时长。 · ALL：设置缓存Reload的间隔时长，默认不重新加载。

8.6.2.7 创建MaxCompute源表

本文为您介绍如何创建MaxCompute（ODPS）源表。

DDL定义

实时计算支持使用MaxCompute作为源表输入，示例代码如下。

```
create table odps_source(
    id INT,
    user_name VARCHAR,
    content VARCHAR
) with (
    type = 'odps',
    endPoint = 'http://service.cn.maxcompute.aliyun-inc.com/api',
    project = '<projectName>',
    tableName = '<tableName>',
    accessId = '<yourAccessKeyId>',
    accessKey = '<yourAccessKeySecret>',
    `partition` = 'ds=2018****'
);
```

WITH参数

参数	说明	备注
endPoint	ODPS服务地址	必选，参见#unique_165/unique_165_Connect_42_section_f2d_51y_5c
tunnelEndpoint	MaxCompute Tunnel服务的连接地址	可选，参见#unique_165/unique_165_Connect_42_section_f2d_51y_5c 说明： VPC环境下为必填。
project	MaxCompute项目名称	必选。
tableName	MaxCompute表名	必选。
accessId	accessId	必选。
accessKey	accessKey	必选。
partition	分区名	可选，如果存在分区表则必填。具体分区信息可在数据地图查看。例如：一个表的分区信息为ds=20180905，则可以写`partition` = 'ds=20180905'。多级分区之间用逗号分隔，`partition` = 'ds=20180912,dt=xxxyyy'。

常见问题

- Q：DDL中endpoint和tunnelEndpont参数配置错误的影响?

  A：point和tunnelEndpont参数介绍请参见#unique_166/unique_166_Connect_42_section_f2d_51y_5db。参数配置错误会导致以下问题：

  - Endpoint配置错误：任务上线进度停滞在91%处。
  - tunnelEndpoint配置错误：任务运行失败。
- Q：MaxCompute作为源表是否支持分区裁剪?

  A：支持分区裁剪。如果MaxCompute源表是分区表，而且对分区列具有筛选要求，即可筛选出MaxCompute对应分区的数据。例如，`SELECT a FROM odps_test where ds = '20180926'`，其中ds是分区列，即可筛选出`ds='20180926'`这个分区的数据。
- Q：提交任务阶段出现Akka超时报错，但是MaxCompute源表获取Metadata速率正常，应该怎样处理?

  A：请合并小文件，具体步骤请参见文档MaxCompute小文件有关场景及解决方案。
- Q: MaxCompute中的数据类型和实时计算中数据类型的对应关系是什么?

  A：如下表。

MaxCompute	Blink
TINYINT	TINYINT
SMALLINT	SMALLINT
INT	INT
BIGINT	BIGINT
FLOAT	FLOAT
DOUBLE	DOUBLE
BOOLEAN	BOOLEAN
DATETIME	TIMESTAMP
TIMESTAMP	TIMESTAMP
STRING	VARCHAR
DECIMAL	DECIMAL
BINARY	VARBINARY

 说明:

- 其他MaxCompute类型，ODPS Connector暂时还不支持转换。
- VARCHAR是ODPS新增加的类型，Blink Connectors暂不支持，建议您将ODPS的Schema中的VARCHAR类型设置为STRING类型。

· Q：是否支持指定读取MaxCompute表所有分区的字典序号最大的分区？

A：实时计算3.2.1及以上版本支持该功能。在DDL的WITH参数中输入参数`` `partition` = 'ds=max_pt()' ``即可，示例如下。

```
with (
  type='odps',
  `partition` = 'ds=max_pt()',
  ...
)
```

说明：

- 请不要在`'ds=max_pt()'`等号的两侧添加空格，否则可能出现如下报错信息。

```
org.apache.flink.table.api.TableException: Partition specific
format is invalid!
  at com.alibaba.blink.connectors.odps.util.PartitionConditionPa
rser.validateAndParseMaxPartStr
```

- 实时计算3.2.1以下版本如需此功能，请先升级至实时计算3.2.1及以上版本。

8.6.3 创建数据结果表

8.6.3.1 数据结果表概述

实时计算使用`CREATE TABLE`作为输出结果数据的格式定义，同时定义数据如何写入到目的数据结果表。

实时计算结果表有Append类型和Update类型。

· Append类型：如果输出存储是日志系统、消息系统、或未定义主键的RDS数据库，数据流的输出结果会以追加的方式写入到存储中，不会修改存储中原有的数据。

· Update类型：如果输出存储是声明了主键（primary key）的数据库（例如，RDS和HBase），数据流的输出结果有以下2种情况。

- 如果根据主键查询的数据在数据库中不存在，则会将该数据插入数据库。
- 如果根据主键查询的数据在数据库中存在，则会根据主键更新数据。

结果表语法

```
CREATE TABLE tableName
    (columnName dataType [, columnName dataType ]*)
```

```
    [ WITH (propertyName=propertyValue [, propertyName=propertyValue
 ]*) ];
```

结果表示例

```
CREATE TABLE rds_output(
id INT,
len INT,
content VARCHAR,
PRIMARY KEY(id)
) WITH (
type='rds',
url='yourDatabaseURL',
tableName='yourTableName',
userName='yourDatabaseUserName',
password='yourDatabasePassword'
);
```

8.6.3.2 创建分析型数据库MySQL版 2.0结果表

本文为您介绍如何创建实时计算分析型数据库MySQL版 2.0版本结果表以及分析型数据库和实时计算字段类型之间的映射关系。

什么是分析型数据库MySQL版

分析型数据库MySQL版是阿里巴巴自主研发的海量数据实时高并发在线分析（Realtime OLAP）云计算服务，支持在毫秒级时单位时间内，对千亿级数据进行即时的多维分析透视和业务探索。

DDL定义

说明:

分析型数据库MySQL版 3.0版本（ADS 3.0版本）实时计算结果表DDL请参见#unique_170。

实时计算支持使用分析型数据库MySQL版 2.0作为结果输出。示例代码如下。

```
CREATE TABLE stream_test_hotline_agent (
id INTEGER,
len BIGINT,
content VARCHAR,
PRIMARY KEY(id)
) WITH (
type='ads',
url='yourDatabaseURL',
tableName='<yourDatabaseTableName>',
userName='<yourDatabaseUserName>',
password='<yourDatabasePassword>',
batchSize='20'
);
```

说明:

- 建议使用存储注册功能，参见#unique_27。

- 分析型数据库MySQL版结果表声明中的主键`primary key`要和分析型数据库MySQL版里的主键一致，大小写敏感。不一致会导致数组索引越界的异常现象。

WITH参数

参数	注释说明	备注
url	jdbc连接地址	分析型数据库MySQL版数据库地址。示例：`url ='jdbc:mysql://databaseName****-cn-shenzhen-a.ads.aliyuncs.com:10014/databaseName'`。 说明： - 分析型数据库MySQL版数据库连接信息参见#unique_171中URL地址查询。 - AnalyticDB数据库名称（databaseName）即AnalyticDB实例名称。
tableName	表名	无。
username	账号	无。
password	密码	无。
maxRetryTimes	写入重试次数	可选，默认为10。
bufferSize	流入多少条数据后开始去重	可选，默认为5000，表示输入的数据达到5000条就开始输出。
batchSize	一次批量写入的条数	可选，默认值为1000。
batchWriteTimeoutMs	写入超时时间	可选，单位为毫秒，默认值为5000。表示如果缓存中的数据在等待5秒后，依然没有达到输出条件，系统会自动输出缓存中的所有数据。
connectionMaxActive	单连接池最大连接数	可选，默认值30
ignoreDelete	是否忽略delete操作	默认为false。

说明：

如果错代码是20015，则表示batchSize设置的过大。分析型数据库MySQL版单次batch不能超过1M。例如，batchSize设置为`1000`，平均每条记录大小不能超过1KB。

类型映射

建议使用分析型数据库MySQL版和实时计算字段类型对应关系进行DDL声明。

分析型数据库MySQL版字段类型	实时计算字段类型
BOOLEAN	BOOLEAN
TINYINT	INT
SAMLLINT	
INT	
BIGINT	BIGINT
DOUBEL	DOUBLE
VARCHAR	VARCHAR
DATE	DATE

8.6.3.3 创建数据总线 DataHub结果表

本文为您介绍如何创建实时计算数据总线 DataHub结果表。

说明:

DataHub未正式商用。

什么是数据总线 DataHub

DataHub作为流式数据总线，为阿里云数加平台提供了大数据的入口服务。实时计算通常使用DataHub作为流式数据存储输入源和输出目的端。

DDL定义

实时计算支持使用DataHub作为数据输出结果表，DataHub结果表声明示例如下。

```
create table datahub_output(
  name VARCHAR,
  age BIGINT,
  birthday BIGINT
)with(
  type='datahub',
  endPoint='<yourEndpoint>',
  project='<yourProjectName>',
  topic='<yourTopicName>',
  accessId='<yourAccessId>',
  accessKey='<yourAccessKey>',
  batchSize='<yourBatchSize>',
  batchWriteTimeoutMs='1000'
);
```

```
```

说明:

建议使用存储注册功能，参见#unique_173。

WITH参数

参数	注释说明	备注
endPoint	Endpoint地址	参见DataHub的Endpoint地址。
project	DataHub项目名称	无
topic	DataHub中topic名称	无
accessId	accessId	无
accessKey	accessKey	无
maxRetryTimes	最大重试次数	可选，默认值为3。
batchSize	一次批量写入的条数	可选，默认值为300。
batchWriteTimeoutMs	缓存数据的超时时间	可选，单位为毫秒，默认值为5000。表示如果缓存中的数据在等待5秒后，依然没有达到输出条件，系统会自动输出缓存中的所有数据。
maxBlockMessages	每次写入的最大Block数	可选，默认值为100。

8.6.3.4 创建日志服务 LOG结果表

本文为您介绍如何创建实时计算日志服务 LOG结果表。

DDL定义

实时计算支持使用日志服务作为结果输出。日志服务结果表声明示例如下。

```
create table sls_stream(
 `name` VARCHAR,
 age BIGINT,
 birthday BIGINT
)with(
 type='sls',
 endPoint='<yourEndpoint>',
 accessId='<yourAccessId>',
 accessKey='<yourAccessKey>',
 project='<yourProjectName>',
 logstore='<yourLogstoreName>'
);
```

说明:

建议使用日志服务存储注册功能，参见#unique_175。

WITH参数

参数	注释说明	备注
endPoint	endPoint地址	#unique_43。
project	项目名	无。
logstore	logstore表名	无。
accessId	accessId	无。
accessKey	accessKey	无。
mode	写入方式	可选，默认为`random`模式，选择`partition`则会按分区写入。
partitionColumn	分区列	如果`mode`为`partition`，则必填。
source	日志的来源地，例如产生该日志机器的IP地址。	可选，默认为空。

8.6.3.5 创建消息队列 MQ结果表

本文为您介绍如何创建实时计算消息队列 MQ结果表。

什么是消息列队 MQ

消息队列（Message Queue）简称MQ，是阿里云商用的专业消息中间件，是企业级互联网架构的核心产品。消息列队是基于高可用分布式集群技术，搭建了包括发布订阅、消息轨迹、资源统计、定时（延时）、监控报警等一套完整的消息云服务。 实时计算可以将消息队列作为流式数据输出，如下所示。

```
CREATE TABLE stream_test_hotline_agent (
id INTEGER,
len BIGINT,
content VARCHAR
) WITH (
type='mq',
endpoint='<yourEndpoint>',
accessID='<yourAccessId>',
accessKey='<yourAccessSecret>',
topic='<yourTopicName>',
producerGroup='<yourGroupName>',
tag='<yourTagName>',
encoding='utf-8',
fieldDelimiter=',',
retryTimes='5',
sleepTimeMs='500'
```

```
);
```

CSV类格式

```
CREATE TABLE stream_test_hotline_agent (
id INTEGER,
len BIGINT,
content VARCHAR
) WITH (
type='mq',
endpoint='<yourEndpoint>',
accessID='<yourAccessId>',
accessKey='<yourAccessSecret>',
topic='<yourTopicName>',
producerGroup='<yourGroupName>',
tag='<yourTagName>',
encoding='utf-8',
fieldDelimiter=',',
retryTimes='5',
sleepTimeMs='500'
);
```

二进制格式

二进制格式测试代码如下。

```
CREATE TABLE source_table (
  commodity VARCHAR
)WIHT(
  type='random'
);

CREATE TABLE result_table (
  mess VARBINARY
) WITH (
  type = 'mq',
  endpoint='<yourEndpoint>',
  accessID='<yourAccessId>',
  accessKey='<yourAccessSecret>',
  topic='<yourTopicName>',
  producerGroup='<yourGroupName>'
);

INSERT INTO result_table
SELECT
CAST(SUBSTRING(commodity,0,5) AS VARBINARY) AS mess
FROM source_table
```

说明:

`cast(varchar as varbinary)` **需在实时计算2.0及以上版本使用，若版本低于2.0，请先完成版本升级。**

WITH参数

参数	说明	备注
topic	Message Queue队列名称	无。
endpoint	地址	· 公共云内网（阿里云经典网络/VPC）接入地址： - 华东1、华东2、华北1、华北2、华南1、中国（香港）：onsaddr-internal.aliyun.com:8080 - 新加坡：ap-southeastaddr-internal.aliyun.com:8080 - 迪拜：ons-me-east-1-internal.aliyuncs.com:8080 - 孟买：ons-ap-south-1-internal.aliyuncs.com:8080 - 吉隆坡：ons-ap-southeast-3-internal.aliyun.com:8080 · 公共云公网接入地址：onsaddr-internet.aliyun.com:80
accessID	填写阿里云accessID	无。
accessKey	填写阿里云accessKey	无。
producerGroup	写入的群组	无。
tag	写入的标签	可选，默认为空。
fieldDelimiter	字段分割符	可选，默认为\u0001。表示在只读模式下以\u0001（\u0001在只读模式不可见）作为分隔符；在编辑模式下以^A作为分隔符。
encoding	编码类型	可选，默认为utf-8。

参数	说明	备注
retryTimes	写入的重试次数	可选，默认为10。
sleepTimeMs	重试间隔时间	可选，默认为1000（毫秒）。
instanceID	Topic所属的分组	可选。若MQ实例为DEFAULT_INSTANCE，则不可使用instanceID参数。

8.6.3.6 创建表格存储 TableStore结果表

本文为您介绍如何创建实时计算表格存储（Table Store）结果表以及表格存储和实时计算字段类型之间的映射关系。

什么是表格存储 TableStore

表格存储 TableStore，简称OTS，是基于阿里云飞天分布式系统的分布式NoSQL数据存储服务。表格存储通过数据分片和负载均衡技术，实现数据规模与访问并发上的无缝扩展，提供海量结构化数据的存储和实时访问服务。

DDL定义

实时计算支持使用TableStore作为结果输出，示例代码如下。

```
CREATE TABLE stream_test_hotline_agent (
 name VARCHAR,
 age BIGINT,
 birthday BIGINT,
 PRIMARY KEY (name,age)
) WITH (
 type='ots',
 instanceName='<yourInstanceName>',
 tableName='<yourTableName>',
 accessId='<yourAccessId>',
 accessKey='<yourAccessSecret>',
 endPoint='<yourEndpoint>',
 valueColumns='birthday'
);
```

说明:

- 推荐使用数据存储注册功能，参见表格存储#unique_28。
- valueColumns的值不能是声明的主键，可以是主键之外的任意字段。
- OTS结果表声明中，除主键列外，至少包含一个属性列。

WITH参数

参数	说明	备注
instanceName	实例名	无

参数	说明	备注
tableName	表名	无
endPoint	实例访问地址	参见 #unique_38。
accessId	访问的id	无
accessKey	访问的键	无
valueColumns	指定插入的字段列名	插入多个字段以,分割。如'ID,NAME'。
bufferSize	流入多少条数据后开始去重	可选，默认值5000，表示输入的数据达到5000条就开始输出。
batchWriteTimeoutMs	写入超时的时间	可选，单位为毫秒，默认值为5000。表示如果缓存中的数据在等待5秒后，依然没有达到输出条件，系统会自动输出缓存中的所有数据。
batchSize	一次批量写入的条数	可选，默认值100。
retryIntervalMs	重试间隔时间	可选，单位毫秒，默认值1000。
maxRetryTimes	最大重试次数	可选，默认值100。
ignoreDelete	是否忽略delete操作	默认为false。

类型映射

OTS字段类型	实时计算字段类型
INTEGER	BIGINT
STRING	VARCHAR
BOOLEAN	BOOLEAN
DOUBLE	DOUBLE

说明:

TableStore结果表须定义有primary key，输出数据以update方式写入到现有TableStore表。update方式说明请参见数据结果表概述中Update类型。

8.6.3.7 创建云数据库RDS版结果表

本文为您介绍如何创建实时计算云数据库RDS（DRDS）版结果表以及云数据库RDS版和实时计算字段类型之间的映射关系。

说明:

实时计算暂不支持引用RDS for MySQL 8.0版本作为数据存储。

什么是云数据库RDS版

阿里云关系型数据库（Relational Database Service）简称RDS，是一种稳定可靠、可弹性伸缩的在线数据库服务。RDS基于阿里云分布式文件系统和高性能存储，支持MySQL、SQL Server、PostgreSQL和PPAS（Postgre Plus Advanced Server）引擎，并且提供了容灾、备份、恢复、监控、迁移等方面的全套解决方案。

DDL定义

实时计算支持使用RDS/DRDS作为结果输出（目前仅支持MySQL数据存储类型）。示例代码如下。

```
CREATE TABLE rds_output(
 id INT,
 len INT,
 content VARCHAR,
 PRIMARY KEY (id,len)
) WITH (
 type='rds',
 url='<yourDatabaseURL>',
 tableName='<yourDatabaseTable>',
 userName='<yourDatabaseUserName>',
 password='<yourDatabasePassword>'
);
```

说明:

- 实时计算写入RDS/DRDS数据库结果表原理：针对实时计算每行结果数据，拼接成一行SQL语句，输入至目标端数据库，进行执行。如果使用批量写，需要在URL后面加上参数`?rewriteBatchedStatements=true`，以提高系统性能。
- RDS MySQL数据库支持自增主键。如果需要让实时计算写入数据支持自增主键，在DDL中不声明该自增字段即可。例如，ID是自增字段，实时计算DDL不写出该自增字段，则数据库在一行数据写入过程中会自动填补相关的自增字段。
- 如果DRDS有分区表，拆分键必须在实时计算DDL里`primary key ()`中声明，否则拆分的表无法写入。
- 建议使用数据存储注册方式，参见#unique_29。
- DDL声明的字段必选至少存在一个非主键的字段，否则产生报错。

WITH参数

参数	说明	备注
url	jdbc连接地址	url的格式为：`jdbc:mysql://<内网地址>/<databaseName>`，其中databaseName为对应的数据库名称。内网地址参见如下链接： · RDS的内网地址 · DRDS的内网地址  说明： 若访问通过VPC访问授权过的RDS，对应的url配置请参见#unique_5。
tableName	表名	无。
userName	用户名	无。
password	密码	无。
maxRetryTimes	最大重试次数	可选，默认值为3。
batchSize	一次批量写入的条数	可选，默认值为50。
bufferSize	流入多少条数据后开始去重	可选，需要指定主键才生效。默认值为1000，表示输入的数据达到1000条即开始输出。
flushIntervalMs	清空缓存的时间间隔	可选，单位为毫秒，默认值为5000。表示如果缓存中的数据在等待5秒后，依然没有达到输出条件，系统会自动输出缓存中的所有数据。
excludeUpdateColumns	忽略指定字段的更新	可选，默认值为空（默认忽略primary key字段）。表示更新主键值相同的数据时，忽略指定字段的更新。
ignoreDelete	是否忽略delete操作	可选，默认为false，表示支持delete功能。
partitionBy	分区	可选，默认为空。表示写入Sink节点前，会根据该值做hash。数据会流向相应的hash节点。

类型映射

RDS字段类型	实时计算字段类型
INTEGER	INT
FLOAT	FLOAT
DECIMAL	DECIAML
LONG	BIGINT
DOUBLE	DOUBLE
TEXT	VARCHAR
BYTE	
DATE	
DATETIME	
TIMESTAMP	
TIME	
YEAR	
CHAR	

JDBC连接参数

参数名称	参数说明	默认值	最低版本要求
useUnicode	是否使用Unicode字符集，如果参数characterEncoding设置为gb2312或gbk，本参数值必须设置为true。	false	1.1g
characterEncoding	当useUnicode设置为true时，指定字符编码。比如可设置为gb2312或gbk。	false	1.1g
autoReconnect	当数据库连接异常中断时，是否自动重新连接。	false	1.1
autoReconnectForPools	是否使用针对数据库连接池的重连策略。	false	3.1.3
failOverReadOnly	自动重连成功后，连接是否设置为只读。	true	3.0.12

参数名称	参数说明	默认值	最低版本要求
maxReconnects	autoReconnect设置为true时，重试连接的次数。	3	1.1
initialTimeout	autoReconnect设置为true时，两次重连之间的时间间隔，单位为秒。	2	1.1
connectTimeout	和数据库服务器建立socket连接时的连接超时时长，单位为毫秒。 0表示永不超时，适用于JDK 1.4及更高版本。	0	3.0.1
socketTimeout	socket操作（读写）超时，单位：毫秒。 0表示永不超时。	0	3.0.1

FAQ

- Q：实时计算的结果数据写入RDS表，是按主键更新的，还是生成1条新的记录?

  A：如果在DDL中定义了主键，会采用`insert into on duplicate key update`的方式更新记录，也就意味着对于不存在的主键字段会直接插入，存在的主键字段则更新相应的值。 如果DDL中没有声明primary key，则会用`insert into`方式插入记录，追加数据。

- Q：使用RDS表中的唯一索引做group by需要注意什么?

  A：

  - 需要在作业中的primary key中声明这个唯一索引。
  - RDS中只有一个自增主键，实时计算作业中不能声明为primary key。

8.6.3.8 创建MaxCompute结果表

本文为您介绍如何创建MaxCompute结果表以及创建过程中的常见问题。

DDL定义

实时计算支持使用MaxCompute作为结果输出，示例代码如下。

```
create table odps_output(
    id INT,
    user_name VARCHAR,
    content VARCHAR
) with (
    type = 'odps',
```

```
    endPoint = 'http://service.cn.maxcompute.aliyun-inc.com/api',
    project = '<projectName>',
    tableName = '<tableName>',
    accessId = '<yourAccessKeyId>',
    accessKey = '<yourAccessKeySecret>',
    `partition` = 'ds=2018****'
);
```

WITH参数

参数	说明	备注
endPoint	MaxCompute服务地址	必选，参见#unique_165/unique_165_Connect_42_section_f2d_51y_5c
tunnelEndpoint	MaxCompute Tunnel服务的连接地址	可选，参见#unique_165/unique_165_Connect_42_section_f2d_51y_5c 说明: VPC环境下为必填。
project	MaxCompute项目名称	必选
tableName	表名	必选
accessId	accessId	必选
accessKey	accessKey	必选

参数	说明	备注
partition	分区名	可选，如果存在分区表则必填。具体分区信息可在数据地图查看。例如：一个表的分区信息为ds=20180905，则可以写`partition` = 'ds=20180905'。多级分区之间用逗号分隔，*`partition` = 'ds=20180912,dt=xxxyyy'。 说明： 如果分区的值不做填写，将会根据数据源的值写入不同的分区，即动态分区。（仅实时计算3.2.1及以上版本支持。）例如，`partition`='ds'，即根据ds字段的数据源的值写入不同分区。 对于动态分区字段为空的情况，如果数据源中ds=null或者ds=''，则输出结果为： · 3.2.1及以前的版本会抛出空指针异常（NPE）。 · 3.2.2及以后的版本会创建ds=NULL的分区。
isOverwrite	写入Sink前，是否将结果表或者结果表的数据清空。	· 实时计算3.2以下版本默认参数值为true。 · 实时计算3.2及以上版本默认参数值为false。声明MaxCompute的流式作业结果表时，isOverwrite参数必须为false，否则在编译时会抛出异常。 说明： 实时计算3.2及以上版本支持isOverwrite参数值变更。 实时计算3.2以下版本如需变更isOverwrite参数值，请升级版本。

常见问题

1. Q: Stream模式的MaxCompute结果表是否支持`isOverwrite`为`true`?

   A：实时计算3.2以下版本支持；实时计算3.2及以上版本不支持。

   `isOverwrite`为`true`，即写入结果表之前会把结果表或者结果数据清空。作业每次启动后和暂停恢复后、写入之前会把原来结果表或者结果分区里的内容删除。

   - 实时计算3.2以下版本isOverwrite默认值是true，且不支持修改。流式作业完成暂停/恢复操作后，会造成数据丢失。
   - 实时计算3.2及以上版本isOverwrite默认值是false，且声明MaxCompute流式作业结果表时isOverwrite参数值需要设置为false，否则在编译时会抛出异常。Stream模式MaxCompute结果表具备At Least Once数据保障机制，在作业运行失败后，可能会出现数据重复。

2. Q: MaxCompute中的数据类型和实时计算中数据类型的对应关系是什么?

   A：如下表。

MaxCompute	Blink
TINYINT	TINYINT
SMALLINT	SMALLINT
INT	INT
BIGINT	BIGINT
FLOAT	FLOAT
DOUBLE	DOUBLE
BOOLEAN	BOOLEAN
DATETIME	TIMESTAMP
TIMESTAMP	TIMESTAMP
STRING	VARCHAR
DECIMAL	DECIMAL
BINARY	VARBINARY

    说明:

   - 其他MaxCompute类型，ODPS Connector暂时还不支持转换。
   - VARCHAR是ODPS新增加的类型，Blink Connectors暂不支持，建议您将ODPS的Schema中的VARCHAR类型设置为STRING类型。

8.6.3.9 创建云数据库 HBase版结果表

本文为您介绍如何创建实时计算云数据库HBase版结果表。

说明:

- 本文档仅适用于实时计算独享模式。
- blink-3.3.0以下版本仅支持HBase企业标准版
- blink-3.3.0及以上版本同时支持HBase企业标准版和HBase性能增强版。

DDL定义

实时计算支持使用HBase作为结果输出。

- HBase企业标准版示例代码如下。

```
create table liuxd_user_behavior_test_front (
    row_key varchar,
    from_topic varchar,
    origin_data varchar,
    record_create_time varchar,
    primary key (row_key)
) with (
    type = 'cloudhbase',
    zkQuorum = '2',
    columnFamily = '<yourColumnFamily>',
    tableName = '<yourTableName>',
    batchSize = '500'
)
```

- HBase性能增强版示例代码如下。

```
create table liuxd_user_behavior_test_front (
    row_key varchar,
    from_topic varchar,
    origin_data varchar,
    record_create_time varchar,
    primary key (row_key)
) with (
    type = 'cloudhbase',
    endPoint = '2',
    columnFamily = '<yourColumnFamily>',
    tableName = '<yourTableName>',
    batchSize = '500'
)
```

说明:

- PRIMARY KEY支持定义多字段。多字段以`rowkeyDelimiter`（默认为:）作为分隔符进行连接。
- HBase执行撤回删除操作时，如果COLUMN定义了多版本，将清空所有版本的COLUMN值。

- HBase企业标准版和HBase性能增强版DDL的区别为连接参数不同：
  - HBase企业标准版使用连接参数`zkQuorum`。
  - HBase性能增强版使用连接参数`endPoint`。

WITH参数

参数	注释说明	备注
zkQuorum	HBase集群配置的zk地址	可以在*hbase-site.xml*文件中找到hbase.zookeeper.quorum相关配置。 说明： 仅在HBase企业标准版中生效。
zkNodeParent	集群配置在zk上的路径	可以在*hbase-site.xm*文件中找到hbase.zookeeper.quorum相关配置。 说明： 仅在HBase企业标准版中生效。
endPoint	HBase地域名称	可在购买的HBase实例控制台中获取。 说明： 仅在HBase性能增强版中生效。
userName	userName	可选。 说明： 仅在HBase性能增强版中生效。
password	密码	可选。 说明： 仅在HBase性能增强版中生效。
tableName	HBase表名	无。
partitionBy	是否使用joinKey进行分区	可选，默认为false。设置为true时，使用joinKey进行分区，将数据分发到各JOIN节点，提高缓存命中率。

参数	注释说明	备注
shuffleEmptyKey	是否将上游EMPTY KEY随机发送到下游节点	可选，默认为false。参数意义如下： · false：如果上游有多个EMPTY KEY，将会将所有EMPTY KEY发送至一个JOIN节点。 · true：如果上游有多个EMPTY KEY，将会将所有EMPTY KEY随机发送到各个JOIN节点。 说明: shuffleEmptyKey在partitionedJoin生效后才是使用。
columnFamily	列族名	目前只支持插入同一列族。
maxRetryTimes	最大尝试次数	可选，默认为10。 说明: 仅实时计算3.2.3及以上版本支持maxRetryTimes参数。
bufferSize	流入多少条数据后进行去重	默认为5000。
batchSize	一次批量写入的条数	可选，默认为100。 说明: 建议batchSize参数值为200~300。过大的batchSize值可能导致任务OOM（内存不足）报错。
flushIntervalMs	最长插入时间	可选，默认为2000。
writePkValue	是否写入主键值	可选，默认为false。
stringWriteMod	是否都按照string插入	可选，默认为false。
rowkeyDelimiter	rowKey的分隔符	可选，默认为:。
isDynamicTable	是否为动态表	可选，默认为false。

动态表

实时计算部分结果数据需要按某列的值，作为动态列输入HBase。HBase中，以每小时的成交额作为动态列的数据，示例如下。

rowkey	cf:0	cf:1	...
20170707	100	200	...

当isDynamicTable参数值为true时，表明该表为支持动态列的HBase表。

动态表仅支持3列输出，例如，ROW_KEY, COLUMN和VALUE。此时第2列（本示例中的COLUMN）为动态列，动态表中的其它参数与HBase的WITH参数一致。

说明:

使用动态表时，所有数据类型需要先转换为STRING类型，再进行输入。

```
CREATE TABLE stream_test_hotline_agent (
  name varchar,
  age varchar,
  birthday varchar,
  primary key (name)
) WITH (
  type = 'cloudhbase',
  ...
columnFamily = 'cf',
 isDynamicTable ='true'
)
```

说明:

- 以上声明将把birthday插入到以name为ROW_KEY的cf:age列中。例如，(wang,18,2016-12-12)会插入ROW_KEY为wang的行，cf:18列。
- DDL中必选按照从上到下的顺序，声明ROW_KEY（本示例中的name）、COLUMN（本示例中的age）和VALUE（本示例中的birthday），且声明ROW_KEY为PRIMARY KEY。

常见问题

Q：HBase结果表作业运行时报错cloudHbase update error, No columns to insert for #10 item?

A：HBase结果表要求写入的单条记录的列数据（不包括rowkey）不能全为NULL。请先过滤全为NULL的数据，再输入HBase。

8.6.3.10 创建ElasticSearch结果表

本文为您介绍如何创建实时计算ElasticSearch（ES）结果表。

说明:

本文仅适用于实时计算3.2.2及以上版本。

DDL定义

ElasticSearch结果表的实现使用REST API，理论上兼容ElasticSearch的各个版本。实时计算支持使用ES作为结果输出。示例代码如下。

```
 CREATE TABLE es_stream_sink(
  field1 LONG,
  field2 VARBINARY,
  field3 VARCHAR,
  PRIMARY KEY(field1)
)WIHT(
  type ='elasticsearch',
  endPoint = 'http://es-cn-mp****.public.elasticsearch.aliyuncs.com
:****',
  accessId = '<yourAccessId>',
  accessKey = '<yourAccessSecret>',
  index = '<yourIndex>',
  typeName = '<yourTypeName>'
);
```

说明:

- ES支持根据PRIMARY KEY进行UPDATE，且PRIMARY KEY只能为1个字段。
- 指定PRIMARY KEY后，Document的ID为PRIMARY KEY字段的值。
- 未指定PRIMARY KEY的Document对应的ID为随机，详情请参见Index API。
- `full`更新模式下，新增的`doc`会完全覆盖已存在的`doc`。
- `inc`更新模式下，会依据输入的字段值更新对应的字段。
- 所有的更新默认为UPSERT语义，即INSERT或UPDATE。

WITH参数

通用配置:

参数	注释说明	默认值	Required
endPoint	server地址，例入：*http://127.0.0.1:9211*。	无	是

参数	注释说明	默认值	Required
accessId	访问实例ID。  说明: 如果您通过Kibana插件操作ES，请填写Kibana登录ID。	无	是
accessKey	访问实例密钥。  说明: 如果您通过Kibana插件操作ES，请填写Kibana登录密码。	无	是
index	索引名称，类似于数据库Database的名称。	无	是
typeName	Type名称，类似于数据库的Table名称。	无	是
bufferSize	流入多少条数据后开始去重。	1000	否
maxRetryTimes	异常重试次数。	30	否
timeout	读超时，单位为毫秒。	600000	否
discovery	是否开启节点发现。如果开启，客户端每5分钟刷新一次Server List。	false	否
compression	是否使用GZIP压缩Request Bodies。	true	否
multiThread	是否开启JestClient多线程。	true	否
ignoreWriteError	是否忽略写入异常。	false	否
settings	创建Index的Settings配置。	无	否

参数	注释说明	默认值	Required
updateMode	指定主键（PRIMARY KEY）后的更新模式。	full 说明: · full：全量覆盖 · inc：增量更新	否

动态索引相关WITH参数

参数	注释说明	默认值	Required
dynamicIndex	是否开启动态索引	false(true/false)	否。
indexField	抽取索引的字段名	无	dynamicIndex为true时必填，只支持timestamp（以秒为单位的）、date和long3种数据类型。
indexInterval	切换索引的周期	d	dynamicIndex为true时必填，可选参数值如下： · d：天 · m：月 · w：周

说明:

· 仅实时计算2.2.7及以上版本支持动态索引功能。

· 当开启动态索引后，基本配置中的`index`名称会作为后续创建索引的统一Alias，Alias和索引为一对多关系。

· 不同的`indexInterval`对应的真实索引名称：

- d -> Alias + "yyyyMMdd"
- m -> Alias + "yyyyMM"
- w -> Alias + "yyyyMMW"

· 对于单个的真实索引可使用Index API进行修改，但对于Alias只支持`get`功能。若需要更新Alias，请参见Index Aliases。

8.6.3.11 创建时序时空数据库结果表

本文为您介绍如何创建实时计算时序时空数据库（TSDB）结果表。

什么是时序时空数据库（TSDB）

时序时空数据库（Time Series Database）简称TSDB。是一种高性能，低成本，稳定可靠的在线时序数据库服务。TSDB提供高效读写，高压缩比存储、时序数据插值及聚合计算，广泛应用于物联网（IoT）设备监控系统，企业能源管理系统（EMS），生产安全监控系统，电力检测系统等行业场景。

DDL定义

实时计算支持使用TSDB作为结果输出，示例代码如下。

说明:

实时计算引用时序时空数据库（TSDB）结果表需要#unique_183。

```
CREATE TABLE stream_test_hitsdb (
    metric VARCHAR,
    timestamp INTEGER,
    value DOUBLE,
    tagk1 VARCHAR,
    tagk2 VARCHAR,
    tagk3 VARCHAR
) WITH (
    type='hitsdb',
    host='<yourHostName>',
    virtualDomainSwitch = 'false',
    httpConnectionPool = '20',
    batchPutSize = '1000'
);
```

建表默认格式：

- 第0列：metric（VARCHAR）
- 第1列：timestamp（INTEGER），单位为秒 。
- 第2列：value（DOUBLE）
- 第3~N列：TagKey，即时序时空数据库中的fieldName。

说明:

- metric、timestamp、value必须声明，且字段名称、字段顺序和字段数据类型必须和TSDB保持完全一致。
- tag可以为多列。

参数设置详情请参见TSDB建表规范。

WITH参数

参数	注释说明	备注
host	IP或VIP域名	填写注册实例的Host，详情请参看：连接实例。
port	端口	默认为8242。
virtualDomainSwitch	是否使用VIPServer	默认为false，若需要使用VIPServer，则设置为true。
httpConnectionPool	HTTP连接池	默认为10。
httpCompress	使用GZIP压缩	默认为false，即不压缩。
httpConnectTimeout	HTTP连接超时时间	默认为0。
ioThreadCount	IO线程数	默认为1。
batchPutBufferSize	缓冲区大小	默认为10000。
batchPutRetryCount	写入重试次数	默认为3。
batchPutSize	每次提交数据量	默认每次提交500个数据点。
batchPutTimeLimit	缓冲区等待时间	默认为200，单位为毫秒。
batchPutConsumerThreadCount	序列化线程数	默认为1。

常见问题

Q：Failover中报错，LONG类型不能转换成INT类型。

A：实时计算2.2.5以下版本只能支持到INT类型。实时计算2.2.5及以上版本支持BIGINT类型。

8.6.3.12 创建消息队列 Kafka结果表

本文档为您介绍如何创建实时计算消息队列 Kafka结果表。

说明:

本文档仅适用于独享模式。

DDL定义

消息队列 Kafka结果表需要定义的DDL如下。

```
create table sink_kafka (
    messageKey VARBINARY,
    `message` VARBINARY,
    PRIMARY KEY (messageKey)
) with (
    type = 'kafka010',
    topic = '<yourTopicName>',
```

```
    bootstrap.servers = '<yourServerAddress>'
);
```

说明:

- 创建Kafka结果表时，必须显示的指定Primary Key (messageKey)。
- 实时计算仅在2.2.6及以上版本中，支持阿里云Kafka或自建Kafka的TPS、RPS等指标信息的显示。

WITH参数

通用配置

参数	注释说明	备注
type	Kafka对应版本	必选，必须是 Kafka08、Kafka09、Kafka010、Kafka011中的一种，版本对应关系参见Kafka版本对应关系。
topic	写入的topic	topic名称

- 必选配置
  - Kafka08必选配置:

参数	注释说明	备注
zookeeper.connect	zk链接地址	zk连接ID

  - Kafka09/Kafka010/Kafka011必选配置:

参数	注释说明	备注
bootstrap.servers	Kafka集群地址	Kafka集群地址

说明:

Kafka集群地址

- 如果您使用的Kafka是阿里云商业版，请参见Kafka商业版准备配置文档。
- 如果您使用的Kafka是阿里云公测版，请参见Kafka公测版准备配置文档。

- **可选配置参数**
  - `consumer.id`
  - `socket.timeout.ms`
  - `fetch.message.max.bytes`
  - `num.consumer.fetchers`
  - `auto.commit.enable`
  - `auto.commit.interval.ms`
  - `queued.max.message.chunks`
  - `rebalance.max.retries`
  - `fetch.min.bytes`
  - `fetch.wait.max.ms`
  - `rebalance.backoff.ms`
  - `refresh.leader.backoff.ms`
  - `auto.offset.reset`
  - `consumer.timeout.ms`
  - `exclude.internal.topics`
  - `partition.assignment.strategy`
  - `client.id`
  - `zookeeper.session.timeout.ms`
  - `zookeeper.connection.timeout.ms`
  - `zookeeper.sync.time.ms`
  - `offsets.storage`
  - `offsets.channel.backoff.ms`
  - `offsets.channel.socket.timeout.ms`
  - `offsets.commit.max.retries`
  - `dual.commit.enabled`
  - `partition.assignment.strategy`
  - `socket.receive.buffer.bytes`
  - `fetch.min.bytes`

 **说明:**

其它可选配置项参考Kafka官方文档进行配置。

- Kafka09
- Kafka010
- Kafka011

Kafka版本对应关系

type	Kafka 版本
Kafka08	0.8.22
Kafka09	0.9.0.1
Kafka010	0.10.2.1
Kafka011	0.11.0.2

示例

```
create table datahub_input (
id VARCHAR,
nm VARCHAR
) with (
type = 'datahub'
);

create table sink_kafka (
 messageKey VARBINARY,
`message` VARBINARY,
 PRIMARY KEY (messageKey)
) with (
    type = 'kafka010',
    topic = 'yourTopicName',
    bootstrap.servers = 'yourServerAddress'
);


INSERT INTO
    sink_kafka
SELECT
   cast(id as VARBINARY) as messageKey,
   cast(nm as VARBINARY) as `message`
FROM
    datahub_input;
```

8.6.3.13 创建云数据库 HybridDB for MySQL结果表

本文为您介绍如何创建实时计算云数据库（HybridDB for MySQL）结果表。

什么是云数据库 HybridDB for MySQL

云数据库 HybridDB for MySQL（原名PetaData）是同时支持在线事务（OLTP）和在线分析（OLAP）的关系型 HTAP 类数据库。HTAP是Hybrid Transaction/Analytical Processing的简写，意为将数据的事务处理（TP）与分析（AP）混合处理，从而实现对数据的实时处理分析。

HybridDB for MySQL兼容MySQL的语法及函数，并且增加了对Oracle常用分析函数的支持。完全兼容TPC-H和TPC-DS测试标准。

DDL定义

实时计算支持使用HybridDB for MySQL作为结果输出。示例代码如下。

```
create table petadata_output(
 id INT,
 len INT,
 content VARCHAR,
 primary key(id,len)
) with (
 type='petaData',
 url='yourDatabaseURL',
 tableName='yourTableName',
 userName='yourDatabaseUserName',
 password='yourDatabasePassword'
);
```

 说明:

- 实时计算写入PetaData数据库结果表原理：针对实时计算每行结果数据，拼接成一行SQL向目标端数据库进行执行。
- bufferSize默认值是1000，如果到达bufferSize阈值的话（buffer hashmap size），也会触发写出。因此您配置batchSize的同时还需要配上bufferSize。bufferSize和batchSize大小相同即可。
- 建议设置`batchSize='4096'`，batchSize数值不建议设置过大。

WITH参数

参数	注释说明	备注
url	地址	#unique_187
tableName	表名	无
userName	用户名	无
password	密码	无
maxRetryTimes	最大重试次数	可选，默认为3。
batchSize	一次批量写入的条数	可选，默认值1000，表示每次写多少条。
bufferSize	流入多少条数据后开始去重	可选

参数	注释说明	备注
flushIntervalMs	写超时时间	可选，单位为毫秒，默认值为3000。表示如果缓存中的数据在等待5秒后，依然没有达到输出条件，系统会自动输出缓存中的所有数据。
ignoreDelete	是否忽略delete操作	默认为false

8.6.3.14 创建云数据库RDS SQL Server版结果表

本文为您介绍如何创建云数据库RDS SQL Server版结果表。

DDL定义

实时计算支持使用云数据库RDS SQL Server版作为结果表输出。示例代码如下。

```
create table ss_output(
 id INT,
 len INT,
 content VARCHAR,
 primary key(id,len)
) with (
 type='jdbc',
 url='jdbc:sqlserver://ip:port;database=****',
 tableName='<yourDatabaseTableName>',
 userName='<yourDatabaseUserName>',
 password='<yourDatabasePassword>'
);
```

说明:

实时计算引用的云数据库RDS SQL Server版真实表需要定义主键，否则初始化阶段会产生NPE（NullPointerException）报错。

WITH参数

参数	注释说明	备注
url	jdbc连接地址	地址请参见： · RDS的URL地址 · DRDS的URL地址
tableName	表名	无
username	账号	无
password	密码	无
maxRetryTimes	写入重试次数	可选，默认为3

参数	注释说明	备注
bufferSize	流入多少条数据后开始去重	可选，默认为500，表示输入的数据达到500条就开始输出。
flushIntervalMs	清空缓存的时间间隔	可选，单位为毫秒，默认值为5000。表示如果缓存中的数据在等待5秒后，依然没有达到输出条件，系统会自动输出缓存中的所有数据。
excludeUpdateColumns	忽略指定字段的更新	可选，默认值为空（默认忽略primary key字段）。表示更新主键值相同的数据时，忽略指定字段的更新。
ignoreDelete	是否忽略delete操作	默认为false

8.6.3.15 创建云数据库 Redis版结果表

本文为您介绍如何创建实时计算云数据库 Redis版结果表。

说明:

本文档仅支持实时计算3.0及以上版本。

DDL定义

目前Redis结果表支持5种Redis数据结构，其DDL定义如下：

- STRING类型

  DDL为2列：第1列为key；第2列为value。Redis插入数据的命令为`set key value`。

```
create table resik_output (
  a varchar,
  b varchar,
  primary key(a)
) with (
  type = 'redis',
  mode = 'string',
  host = '${redisHost}', -- 例如，'127.0.0.1'。
  port = '${redisPort}', -- 例如，'6379'。
  dbNum = '${dbNum}', -- 默认为0。
  ignoreDelete = 'true' -- 收到Retraction时，是否删除之前插入的数据，默认为false。
)
```

- LIST类型

  DDL为2列：第1列为key；第2列为value。Redis插入数据的命令为`lpush key value`。

```
create table resik_output (
  a varchar,
  b varchar,
  primary key(a)
) with (
  type = 'redis',
  mode = 'list',
```

```
  host = '${redisHost}', -- 例如，'127.0.0.1'。
  port = '${redisPort}', -- 例如，'6379'。
  dbNum = '${dbNum}', -- 默认为0。
  ignoreDelete = 'true' -- 收到Retraction时，是否删除之前插入的数据，默认
为false。
)
```

- **SET类型**

  DDL为2列：第1列为key；第2列为value。Redis插入数据的命令为`sadd key value`**。**

```
create table resik_output (
  a varchar,
  b varchar,
  primary key(a)
) with (
  type = 'redis',
  mode = 'set',
  host = '${redisHost}', -- 例如，'127.0.0.1'。
  port = '${redisPort}', -- 例如，'6379'。
  dbNum = '${dbNum}', -- 默认为0。
  ignoreDelete = 'true' -- 收到Retraction时，是否删除之前插入的数据，默认
为false。
)
```

- **HASHMAP类型**

  DDL为3列：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 resik_output (
  a varchar,
  b varchar,
  c varchar,
  primary key(a)
) with (
  type = 'redis',
  mode = 'hashmap',
  host = '${redisHost}', -- 例如，'127.0.0.1'。
  port = '${redisPort}', -- 例如，'6379'。
  dbNum = '${dbNum}', -- 默认为0。
  ignoreDelete = 'true' -- 收到Retraction时，是否删除之前插入的数据，默认
为false。
)
```

- **SORTEDSET类型**

  DDL为3列：第1列为key；第2列为score；第3列为value。Redis插入数据的命令为`add key score value`**。**

```
create table resik_output (
  a varchar,
  b double,  --必须为DOUBLE类型。
  c varchar,
  primary key(a)
) with (
  type = 'redis',
  mode = 'sortedset',
  host = '${redisHost}', -- 例如，'127.0.0.1'。
```

```
  port = '${redisPort}', -- 例如，'6379'。
  dbNum = '${dbNum}', -- 默认为0。
  ignoreDelete = 'true' -- 收到Retraction时，是否删除之前插入的数据，默认为false。
)
```

WITH参数

参数	参数说明	是否必选	取值
type	结果表类型	是	redis
mode	对应Redis的数据结构		可取以下值： · stirng · list · set · hashmap · sortedset
host	Redis server对应地址		取值示例：`127.0.0.1`
port	Redis server对应端口	否	默认值为6379
dbNum	Redis server对应数据库序号		默认值为0
ignoreDelete	是否忽略Retraction消息		默认值为false，可取值为true或false。如果设置为true，收到Retraction时，同时删除数据对应的key及之前插入的数据。
password	Redis Server 对应的密码		用户配置参数

8.6.3.16 创建云数据库 MongoDB版结果表

本文为您介绍实时计算云数据库MongoDB版结果表的DDL定义和WITH参数。

说明：

本文仅适用于实时计算3.2.2及以上版本。

DDL定义

实时计算支持使用MongoDB作为数据输出的结果表，示例代码如下。

```
CREATE TABLE mongodb_sink (
  `a`              VARCHAR
) WITH (
   type = 'mongodb'
    ,database = '<yourDatabaseName>'
```

```
    ,collection= '<yourCollectionName>'
    ,uri='mongodb://{<atabaseAccount>}:{<atabasePassword>}@{host
}:****?replicaSet=mgset-1224****'
    ,keepAlive='true'
    ,maxConnectionIdleTime='20000'
    ,batchSize='2000'
);
```

WITH参数

参数	说明	是否必填	备注
type	Connector类型	是	固定参数值，`type = 'mongodb'`。
database	数据库	是	无。
collection	数据集合	是	无。
uri	mongodb连接串	是	无。
keepAlive	是否保持长连接	否	默认值为true。
maxConnectionIdleTime	连接超时时长	否	整型值，单位为毫秒。0表示无连接超时限制，不能为负值，默认值为60000。
batchSize	1次批量写入的条数	否	整型值，默认值为1024。

8.6.3.17 创建分析型数据库MySQL版 3.0结果表

本文为您介绍如何创建实时计算分析型数据库MySQL版3.0版本的结果表。

说明:

- 暂不支持注册存储功能。
- 本文档仅适用于`blink 3.3.0`及以上版本。
- AnalyticDB for MySQL 2.0版本结果表创建说明参见#unique_192。

DDL定义

实时计算支持使用分析型数据库MySQL版3.0版本（AnalyticDB for MySQL 3.0）作为结果输出。示例代码如下。

```
CREATE TABLE rds_output (
id INT,
len INT,
content VARCHAR,
PRIMARY KEY(id,len)
) WITH (
type='ADB30',
url='jdbc:mysql://<yourNetworkAddress>:<PortId>/<yourDatabaseName>',
```

```
tableName='<yourDatabaseTableName>',
userName='<yourDatabaseUserName>',
password='<yourDatabasePassword>'
);
```

说明:

实时计算写入AnalyticDB for MySQL 3.0结果表的原理:

1. 将流式计算每行的结果数据拼接为一行SQL。

2. 将拼接后的SQL在目标数据库执行。

WITH参数

参数	注释说明	是否必选	备注
type	类型	是	固定值，为ADB30。
url	jdbc连接地址	是	分析型数据库MySQL版数据库地址 。示例：url='jdbc:mysql://databaseName****-cn-shenzhen-a.ads.aliyuncs.com:10014/databaseName‘。 **说明:** · 分析型数据库MySQL版数据库连接信息参见#unique_171中URL地址查询。 · AnalyticDB数据库名称（databaseName）即AnalyticDB实例名称。
tableName	表名	是	无。
username	账号	是	无。
password	密码	是	无。
maxRetryTimes	写入重试次数	否	默认为3。
bufferSize	流入多少条数据后开始去重	否	默认为1000，表示输入的数据达到1000条则开始输出。 **说明:** 需要指定主键后才能生效。
batchSize	一次批量写入的条数	否	默认值为1000。 **说明:** 需要指定主键后才能生效。

参数	注释说明	是否必选	备注
flushInterValMs	清空缓存的时间间隔	否	单位为毫秒，默认值为3000。表示如果缓存中的数据在等待3秒后，依然没有达到输出条件，系统会自动输出缓存中的所有数据。
ignoreDelete	是否忽略delete操作	否	默认为false，表示支持delete功能。
reserveMilliSecond	TimeStamp类型是否保留毫秒	否	默认false，不保留毫秒数值。

8.6.3.18 创建自定义结果表

为满足各种差异化的输出需求，实时计算平台提供用户自定义结果表开发功能。本文为您介绍如何创建实时计算自定义结果表。

 说明:
本文档仅适用于独享模式。

环境搭建

您可以通过以下2种方式搭建自定义结果表的开发环境。

- 直接使用DEMO中的环境

  实时计算团队为您提供自定义结果表DEMO便于您快速进行业务开发。

   说明:
  DEMO示例中已为您配置对应版本的开发环境，您无需进行环境搭建。

  - 实时计算3.0版本

    blink_customersink_3x
  - 实时计算2.0版本

    blink_customersink_2x
  - 实时计算1.0版本

    blink_customersink_1x
- 下载JAR包自行搭建环境

   说明:

Maven工程中引用以下依赖包时，Scope设置为`<scope>provided</scope>`。

- 实时计算3.0版本
  - JAR包下载
    - blink-connector-custom-blink-3.2.1
    - blink-connector-common-blink-3.2.1

说明:

请在POM文件中添加如下信息，完成`flink-table_2.11`JAR包的自动下载。

```
<profiles>
  <profile>
     <id>allow-snapshots</id>
        <activation><activeByDefault>true</activeByDefault></
activation>
     <repositories>
       <repository>
         <id>snapshots-repo</id>
         <url>https://oss.sonatype.org/content/repositories/
snapshots</url>
         <releases><enabled>false</enabled></releases>
         <snapshots><enabled>true</enabled></snapshots>
       </repository>
     </repositories>
   </profile>
</profiles>
```

  - 依赖包

```
<dependencies>
 <dependency>
   <groupId>com.alibaba.blink</groupId>
   <artifactId>blink-connector-common</artifactId>
   <version>blink-3.2.1-SNAPSHOT</version>
   <scope>provided</scope>
 </dependency>
 <dependency>
   <groupId>com.alibaba.blink</groupId>
   <artifactId>blink-connector-custom</artifactId>
   <version>blink-3.2.1-SNAPSHOT</version>
   <scope>provided</scope>
 </dependency>
 <dependency>
   <groupId>com.alibaba.blink</groupId>
   <artifactId>flink-table_2.11</artifactId>
   <version>blink-3.2.1-SNAPSHOT</version>
   <scope>provided</scope>
 </dependency>
```

```
</dependencies>
```

- 实时计算2.0版本

  - JAR包下载

    - blink-connector-common-blink-2.2.4
    - blink-connector-custom-blink-2.2.4
    - blink-table-blink-2.2.4
    - flink-table_2.11-blink-2.2.4
    - flink-core-blink-2.2.4

  - 依赖包

```
<dependencies>
  <dependency>
    <groupId>com.alibaba.blink</groupId>
    <artifactId>blink-table</artifactId>
    <version>blink-2.2.4-SNAPSHOT</version>
    <scope>provided</scope>
  </dependency>
  <dependency>
    <groupId>org.apache.flink</groupId>
    <artifactId>flink-table_2.11</artifactId>
    <version>blink-2.2.4-SNAPSHOT</version>
    <scope>provided</scope>
  </dependency>
  <dependency>
    <groupId>org.apache.flink</groupId>
    <artifactId>flink-core</artifactId>
    <version>blink-2.2.4-SNAPSHOT</version>
    <scope>provided</scope>
  </dependency>
  <dependency>
    <groupId>com.alibaba.blink</groupId>
    <artifactId>blink-connector-common</artifactId>
    <version>blink-2.2.4-SNAPSHOT</version>
    <scope>provided</scope>
  </dependency>
  <dependency>
    <groupId>com.alibaba.blink</groupId>
    <artifactId>blink-connector-custom</artifactId>
    <version>blink-2.2.4-SNAPSHOT</version>
    <scope>provided</scope>
  </dependency>
```

```
    </dependencies>
```

- 实时计算1.0版本

  - JAR包下载

    - blink-connector-common-blink-1.4
    - blink-connector-custom-blink-1.4
    - blink-table-blink-1.4
    - flink-core-blink-1.4
    - flink-streaming-scala_2.11-blink-1.4

  - 依赖包

```
      <dependencies>
        <dependency>
            <groupId>com.alibaba.blink</groupId>
            <artifactId>blink-connector-common</artifactId>
            <version>blink-1.4-SNAPSHOT</version>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>com.alibaba.blink</groupId>
            <artifactId>blink-connector-custom</artifactId>
            <version>blink-1.4-SNAPSHOT</version>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>org.apache.flink</groupId>
            <artifactId>flink-streaming-scala_${scala.binary.
version}</artifactId>
            <version>blink-1.4-SNAPSHOT</version>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>org.apache.flink</groupId>
            <artifactId>flink-core</artifactId>
            <version>blink-1.4-SNAPSHOT</version>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>com.alibaba.blink</groupId>
            <artifactId>blink-table</artifactId>
            <version>blink-1.4-SNAPSHOT</version>
            <scope>provided</scope>
        </dependency>
    </dependencies>
```

接口说明

自定义结果表Class需要继承自定义Sink插件的基类CustomSinkBase，并实现如下方法。

```
protected Map<String,String> userParamsMap;// userParamsMap是自定义SQL的
WITH语句中定义的键值对，所有的键均为小写。
protected Set<String> primaryKeys;// primaryKeys是自定义的主键字段名。
protected List<String> headerFields;// headerFields是标记为header的字段列
表。
```

```
protected RowTypeInfo rowTypeInfo;//字段类型和名称。
/**
 * 初始化方法。每次初始建立和Failover的时候会调用一次。
 *
 * @param taskNumber 当前节点的编号。
 * @param numTasks   Sink节点的总数。
 * @throws IOException
 */
public abstract void open(int taskNumber,int numTasks) throws
IOException;

/**
 * lose方法，释放资源。
 *
 * @throws IOException
 */
public abstract void close() throws IOException;

/**
 * 处理插入单行数据。
 *
 * @param row
 * @throws IOException
 */
public abstract void writeAddRecord(Row row) throws IOException;

/**
 * 处理删除单行数据。
 *
 * @param row
 * @throws IOException
 */
public abstract void writeDeleteRecord(Row row) throws IOException;

/**
 * 如果进行批量插入，该方法需要把线程中缓存的数据全部刷入下游存储；若无，可不实现此
方法。
 *
 * @throws IOException
 */
public abstract void sync() throws IOException;

/**
* 返回类名。
*/
public String getName();
```

在实时计算平台上传JAR包，引用资源之后，对于自定义的Sink插件，需要指明`type = 'custom'`，并且指明实现接口的class，如下以自定义的Redis结果表（DEMO）为例。

```
create table in_table(
    kv varchar
)with(
    type = 'random'
);

create table out_table(
    `key` varchar,
    `value` varchar
)with(
    type = 'custom',
    class = 'com.alibaba.blink.connector.custom.demo.RedisSink',
```

```
    -- **1. 可以定义更多的自定义参数，在open函数中可以通过userParamsMap获
取。**
    -- **2. WITH参数里key大小写不敏感，在实时计算中，参数key值会直接处理成全小
写。建议您在引用数据存储的DDL中使用小写声明key。**
    host = 'r-uf****.redis.rds.aliyuncs.com',
    port = '6379',
    db = '0',
    batsize = '10',
    password = '<yourHostPassword>'
);

insert into out_table
select
substring(kv,0,4) as `key`,
substring(kv,0,6) as `value`
from in_table;
```

Redis Sink插件的参数说明如下。

参数	说明	是否必填	备注
host	Redis实例内网连接地址（host）	是	无。
port	Redis实例端口号	是	无。
password	Redis连接密码	是	无。
db	Redis Database编号	否	默认值为0，代表db0。
batchsize	批量写入大小	否	默认值为1，代表不攒批写入。

8.6.4 创建数据维表

8.6.4.1 数据维表概述

在结果表DDL语法中增加1行`PERIOD FOR SYSTEM_TIME`的声明，即可使用标准的`CREATE TABLE`语法定义实时计算维表。`PERIOD FOR SYSTEM_TIME`定义了维表的变化周期，表明该表是变化的表。

示例

```
CREATE TABLE white_list (
  id varchar,
  name varchar,
  age int,
  PRIMARY KEY (id),  -- 用作维表时，必须有声明的主键。
  PERIOD FOR SYSTEM_TIME  -- 定义维表的变化周期。
) with (
  type = 'RDS',
  ...
```

```
)
```

说明:

- 声明维表时，必须指名主键。维表JOIN时，ON的条件必须包含所有主键的等值条件。
- 目前仅支持源表INNER JOIN或LEFT JOIN维表。
- 维表的唯一键（UK）必须为数据库中的唯一键。维表声明的唯一键如果不是数据库表的唯一键会产生以下影响：
  - 维表的读取变慢。
  - 维表JOIN时，会从第一条数据做JOIN，在加入Job的过程中，相同KEY的多条记录在数据库中按顺序发生变化，可能导致JOIN结果错误。

INDEX语法

说明:

建议在实时计算2.2.7及以上版本使用INDEX语法。

实时计算2.2以下版本，维表定义要求声明PRIMARY KEY，这种情况下只能实现一对一连接。为支持一对多连接的需求，引入了INDEX语法。（非cache All的维表JOIN目前是通过INDEX LOOKUP的方式实现的，Batch作业后续可能根据COST选择SCAN的方式）。

```
CREATE TABLE Persons (
    ID bigint,
    LastName varchar,
    FirstName varchar,
    Nick varchar,
    Age int,
    [UNIQUE] INDEX(LastName,FirstName,Nick), --定义index，不需要具体的类型(例如，fulltext或clustered等)。
    PERIOD FOR SYSTEM_TIME
) with (
  type='RDS',
  ...
);
```

UNIQUE INDEX表示一对一连接，而INDEX表示一对多连接。

说明:

1. 实时计算2.2.7及以后版本支持UNIQUE CONSTRAINT（UNIQUE KEY），实时计算2.2.7以下版本版本可以使用PRIMARY KEY的定义。
2. 在生成执行计划时，引擎优先采用UNIQUE INDEX，即若DDL中使用INDEX，但JOIN等值连接条件中同时包含UNIQUE/NON-UNIQUE INDEX时，会优先使用UNIQUE INDEX进行右表数据查找。

3. 支持一对多连接的维表类型包括RDS，MaxComput（仅支持`cache All`模式，不支持随机访问）。
4. 可增加了`maxJoinRows`参数，表示一对多连接时，左表一条记录连接右表的最大记录数（默认值为1024），注意一对多连接的记录数过多时，需要考虑对Cache的内存需求变大（`cacheSize`限制的是左表key个数），单条左表记录对应的右表记录较多时，可能会极大的影响流任务的性能。

维表、源表和结果表的区别

	源表	结果表	维表
是否能驱动计算	是	否	否
是否能读取数据	是，直接读取。	否	是，仅通过源表和维表join读取。
是否能写入数据	否	是	否
是否支持Cache	否	否	是

8.6.4.2 创建表格存储 TableStore维表

本文为您介绍如何创建实时计算表格存储 TableStore维表。

什么是表格存储 TableStore

表格存储 TableStore简称OTS，是根据99.99%的高可用以及11个9的数据可靠性的设计标准，构建在阿里云飞天分布式系统之上的分布式NoSQL数据存储服务。表格存储通过数据分片和负载均衡技术，实现数据规模与访问并发上的无缝扩展，提供海量结构化数据的存储和实时访问服务。

示例

实时计算支持表格存储 TableStore作为维表，示例如下。

```
CREATE TABLE ots_dim_table (
 id int,
 len int,
 content VARCHAR,
 PRIMARY KEY (id),
 PERIOD FOR SYSTEM_TIME--定义了维表的变化周期，即表明该表是一张会变化的表。
) WITH (
 type='ots',
 endPoint='<yourEndpoint>'
 instanceName='<yourInstanceName>',
 tableName='<yourTableName>',
 accessId='<yourAccessId>',
 accessKey='<yourAccessKey>',
);
```

 说明:

声明一个维表的时候，必须要指名主键。维表JOIN的时候，ON的条件必须包含所有主键的等值条件。OTS的主键即表的rowkey。

WITH参数

参数	注释说明
instanceName	实例名
tableName	表名
endPoint	实例访问地址
accessId	访问的ID
accessKey	访问的键

CACHE参数

参数	注释说明	备注
cache	缓存策略	默认为 None，可选 LRU。
cacheSize	缓存大小	当选择 LRU缓存策略后，可以设置缓存大小，默认为10000 行。
cacheTTLMs	缓存超时时间，单位为毫秒。	当选择 LRU缓存策略后，可以设置缓存失效的超时时间。

测试案例

```
CREATE TABLE datahub_input1 (
id            BIGINT,
name        VARCHAR,
age           BIGINT
) WITH (
type='datahub'
);

create table phoneNumber(
name VARCHAR,
phoneNumber bigint,
primary key(name),
PERIOD FOR SYSTEM_TIME--定义维表的变化周期
)with(
type='ots'
);

CREATE table result_infor(
id bigint,
phoneNumber bigint,
name VARCHAR
)with(
type='rds'
);
```

```
INSERT INTO result_infor
SELECT
t.id
,w.phoneNumber
,t.name
FROM datahub_input1 as t
JOIN phoneNumber FOR SYSTEM_TIME AS OF PROCTIME() as w --维表JOIN必须指
定
ON t.name = w.name;
```

关于维表详细语法请参见#unique_197。

8.6.4.3 创建云数据库RDS版维表

本文为您介绍如何创建实时计算云数据库RDS（DRDS）版维表。

说明:

实时计算暂不支持引用RDS for MySQL 8.0版本作为数据存储。

云数据库RDS版

阿里云关系型数据库（Relational Database Service）简称RDS，是一种稳定可靠、可弹性伸缩的在线数据库服务。基于阿里云分布式文件系统和高性能存储，RDS支持MySQL、SQL Server、PostgreSQL和PPAS（Postgre Plus Advanced Server，一种高度兼容Oracle的数据库）引擎，并且提供了容灾、备份、恢复、监控、迁移等方面的全套解决方案，彻底解决数据库运维的烦恼。

说明:

云数据库RDS（DRDS）版插件中的WITH参数一致，可以通用。 在使用云数据库（RDS和DRDS）作为维表时，RDS或DRDS中必须要有真实的表存在。

示例

实时计算支持使用RDS或DRDS作为维表（目前仅支持MySQL数据存储类型），示例代码如下。

```
CREATE TABLE rds_dim_table(
 id INT,
 len INT,
 content VARCHAR,
 PRIMARY KEY (id),
 PERIOD FOR SYSTEM_TIME--定义维表的变化周期，表明该表是一张会变化的表。
) with (
 type='rds',
 url='<yourDatabaseURL>',
 tableName='<yourDatabaseTableName>',
 userName='<yourDatabaseUserName>',
 password='<yourDatabasePassword>'
```

```
);
```

说明:
声明一个维表时，必须要指名主键。维表JOIN的时候，ON的条件必须包含所有主键的等值条件。RDS或DRDS的主键可以定义为表的主键或是唯一索引列。

WITH参数

参数	注释说明	备注
url	jdbc连接地址	url的格式为：`jdbc:mysql://<内网地址>/<databaseName>`，其中databaseName为对应的数据库名称。内网地址参见如下链接： · RDS的内网地址 · DRDS的内网地址 **说明:** 若访问通过VPC访问授权过的RDS，对应的url配置请参见#unique_5。
tableName	表名	无
userName	用户名	无
password	密码	无
maxRetryTimes	最大尝试插入次数	可选，默认为3。

Cache 参数

参数	注释说明	备注
cache	缓存策略	默认为 `None`，可选 `LRU`或`ALL`。
cacheSize	缓存大小	当选择 `LRU` 缓存策略后，可以设置缓存大小，默认 10000 行。

参数	注释说明	备注
cacheTTLMs	缓存超时时间，单位为毫秒。	当选择 LRU 缓存策略后，可以设置缓存失效的超时时间，默认不过期。当选择 ALL 策略，则为缓存reload 的间隔时间，默认不重新加载。
cacheReloadTimeBlackList	缓存策略选择ALL 时启用。更新时间黑名单，防止在此时间内做cache 更新。（如双11场景。）	可选，默认为空。自定义输入格式为2017-10-24 14:00 -> 2017-10-24 15:00 , 2017-11-10 23:30 -> 2017-11-11 08:00。用逗号,来分隔多个黑名单，用箭头->来分割黑名单的起始结束时间。

目前RDS/DRDS提供如下三种缓存策略。

- None：无缓存。
- LRU：最近使用策略缓存。需要配置相关参数：缓存大小（cacheSize）和 缓存超时时间（cacheTTLMs）。
- ALL：全量缓存策略。在Job运行前会将远程表中所有数据load到内存中，之后所有的维表查询都会通过cache进行。cache命中不到则不存在，并在缓存过期后重新加载一遍全量缓存。全量缓存策略适合远程表数据量小、miss key多的场景。全量缓存相关配置：缓存更新间隔（cacheTTLMs），更新时间黑名单（cacheReloadTimeBlackList）。

 说明:

- 因为会异步Reload，使用cache All的时候，需要将维表JOIN的节点增加一些内存，增加的内存大小为远程表两倍的数据量。
- 使用cache All，请特别注意节点的内存，防止内存溢出。

测试案例

```
CREATE TABLE datahub_input1 (
id            BIGINT,
name         VARCHAR,
age           BIGINT
) WITH (
type='datahub'
);
create table phoneNumber(
name VARCHAR,
phoneNumber BIGINT,
primary key(name),
PERIOD FOR SYSTEM_TIME--定义维表的变化周期。
```

```
)with(
type='rds'
);

CREATE table result_infor(
id BIGINT,
phoneNumber BIGINT,
name VARCHAR
)with(
type='rds'
);

INSERT INTO result_infor
SELECT
t.id
,w.phoneNumber
,t.name
FROM datahub_input1 as t
JOIN phoneNumber FOR SYSTEM_TIME AS OF PROCTIME() as w --维表JOIN时必须指定此声明。
ON t.name = w.name;
```

关于维表详细语法请参见#unique_197。

8.6.4.4 创建云数据库 HBase版维表

本文为您介绍如何创建实时计算云数据库 HBase版维表。

说明:

- blink-3.3.0以下版本仅支持HBase企业标准版
- blink-3.3.0及以上版本同时支持HBase企业标准版和HBase性能增强版。
- 云数据库 HBase版维表的JOIN语法详见： #unique_197。

示例

- HBase企业标准版示例代码如下。

```
CREATE TABLE hbase (
    `key` varchar,
    `name` varchar,
     PRIMARY KEY (`key`), -- HBase中的rowkey字段。
     PERIOD FOR SYSTEM_TIME --维表标识。
    ) with (
     TYPE = 'cloudhbase',
     zkQuorum = '<yourzkQuorum>',
     columnFamily = '<yourColumnFamilyName>',
     tableName = '<yourTableName>'
);
```

- HBase性能增强版示例代码如下。

```
CREATE TABLE hbase (
    `key` varchar,
    `name` varchar,
     PRIMARY KEY (`key`), -- HBase中的rowkey字段。
     PERIOD FOR SYSTEM_TIME --维表标识。
```

```
    ) with (
     TYPE = 'cloudhbase',
     endPoint = '<yourEndpoint>',
     columnFamily = '<yourColumnFamilyName>',
     tableName = '<yourTableName>'
);
```

说明:

- 声明维表时，必须要指名主键。维表JOIN的时候，ON的条件必须包含所有主键的等值条件。HBase中的主键即rowkey。
- HBase企业标准版和HBase性能增强版DDL的区别为连接参数不同：
  - HBase企业标准版使用连接参数`zkQuorum`。
  - HBase性能增强版使用连接参数`endPoint`。

参数

参数	注释说明	备注
zkQuorum	HBase集群配置的zk地址，是以,分隔的主机列表。	可以在*hbase-site.xml*文件中找到hbase.zookeeper.quorum相关配置。 **说明:** 仅在HBase企业标准版中生效。
zkNodeParent	集群配置在zk上的路径	可以在*hbase-site.xm*文件中找到hbase.zookeeper.quorum相关配置。 **说明:** 仅在HBase企业标准版中生效。
endPoint	HBase地域名称	可在购买的HBase实例控制台中获取。 **说明:** 仅在HBase性能增强版中生效。

参数	注释说明	备注
userName	用户名	可选。 说明: 仅在HBase性能增强版中生效。
password	密码	可选。 说明: 仅在HBase性能增强版中生效。
tableName	HBase表名	无。
columnFamily	列族名	目前只支持插入同一列族。
userName	用户名	无。
password	密码	无 。
maxRetryTimes	最大尝试次数	默认10次。
partitionedJoin	设置为true之后会在用joinKey做partition，将数据分发到join节点，提高缓存命中率。	可选，默认关闭。
shuffleEmptyKey	设置为true之后遇到空key会随机往下游做shuffle，否则往0号下游发。	建议打开。

CACHE参数

参数	注释说明	备注
cache	缓存策略	默认 `None`，可选 `LRU`，`ALL`。
cacheSize	缓存大小	当选择 `LRU` 缓存策略后，可以设置缓存大小，默认 10000 行。
cacheTTLMs	缓存超时时间，单位为毫秒。	当选择 `LRU` 缓存策略后，可以设置缓存失效的超时时间，默认不过期。当选择 `ALL` 策略，则为缓存reload 的间隔时间，默认不重新加载。

参数	注释说明	备注
cacheReloadTimeBlackList	缓存策略选择ALL时启用。更新时间黑名单，防止在此时间内做cache更新。（如双11场景。）	可选，默认为空。自定义输入格式为 `为2017-10-24 14:00 -> 2017-10-24 15:00, 2017-11-10 23:30 -> 2017-11-11 08:00` 。用逗号,来分隔多个黑名单，用箭头->来分割黑名单的起始结束时间。
cacheScanLimit	缓存策略选择ALL时启用。load全量HBase数据，服务端一次RPC返回给客户端的行数。	可选，默认100条。

目前RDS和DRDS提供如下三种缓存策略。

- None：无缓存。
- LRU：最近使用策略缓存。需要配置相关参数：缓存大小（cacheSize）和 缓存超时时间（cacheTTLMs）。
- ALL：全量缓存策略。在Job运行前会将远程表中所有数据load到内存中，之后所有的维表查询都会通过cache进行。cache命中不到，则会在缓存过期后重新加载一遍全量缓存。全量缓存策略适合远程表数据量小、miss key多的场景。全量缓存相关配置：缓存更新间隔（cacheTTLMs），更新时间黑名单（cacheReloadTimeBlackList）。

> 说明:
> 因为会异步reload，使用cache all的时候，需要将维表JOIN的节点增加一些内存，增加的内存大小为远程表2倍的数据量。

8.6.4.5 创建MaxCompute维表

本文为您介绍如何创建实时计算MaxCompute维表。

> 说明:
> - 推荐实时计算2.1.1及以上版本使用。
> - 维表的Query语法参见： #unique_201。
> - 使用MaxCompute表作为维表，要先赋权读权限给MaxCompute账号。

创建MaxCompute维表示例

```
CREATE TABLE white_list (
  id varchar,
  name varchar,
  age int,
  PRIMARY KEY (id),
  PERIOD FOR SYSTEM_TIME --定义了维表的标识。
) WITH (
  type = 'odps',
  endPoint = 'http://service.cn.maxcompute.aliyun-inc.com/api',
  project = '<projectName>',
  tableName = '<tableName>',
  accessId = '<yourAccessKeyId>',
  accessKey = '<yourAccessKeySecret>',
  `partition` = 'ds=2018****',
  cache = 'ALL'
)
```

说明:

- 声明维表时，必须要指名主键，维表JOIN的时候，ON的条件必须包含所有主键的等值条件。
- MaxCompute维表主键必须有唯一性，否则会被去重。
- parition是关键字，使用时需要使用反引号注释`partition`。
- 如果是分区表，目前不支持将分区列写入到schema定义中。
- 实时计算2.2.0及以上版本支持加载最新分区表，配置方法为partition='max_pt()'，max_pt()为所有分区的字典序最大值。

WITH参数

参数	注释说明	备注
endPoint	ODPS服务地址	必选，参见#unique_165/unique_165_Connect_42_section_f2d_5
tunnelEndpoint	MaxCompute Tunnel服务的连接地址	可选，参见#unique_165/unique_165_Connect_42_section_f2d_5 **说明:** VPC环境下为必填。
project	ODPS项目名称	必选
tableName	表名	必选
accessId	accessId	必选
accessKey	accessKey	必选

参数	注释说明	备注
partition	分区名	可选，分区表必填，具体分区信息到数据地图查看。例如: 一个表的分区信息为 `ds=20180905`，则可以写 `` `partition` = 'ds=20180905' ``。多级分区之间用逗号分隔，示例: `` `partition` = 'ds=20180912,dt=xxxyyy' ``
maxRowCount	可加载的最大表格数量	可选，默认为100000。 说明: 如果您的数据超过100000，需要设置maxRowCount参数。建议设定值比实际值大。

CACHE参数

参数	注释说明	备注
cache	缓存策略	仅支持`ALL`策略。
cacheSize	缓存大小	可以设置缓存大小，ODPS默认缓存值为100000行。
cacheTTLMs	缓存超时时间，单位为毫秒。	当选择 `ALL` 策略，则为缓存Reload的间隔时间，默认为不重新加载。
cacheReloadTimeBlackList	ALL Cache时启用，更新时间黑名单，防止在此时间内做cache更新（如双11场景）。	可选，默认为空，格式为`2017-10-24 14:00 -> 2017-10-24 15:00, 2017-11-10 23:30 -> 2017-11-11 08:00`。用逗号,来分隔多个黑名单，用箭头`->`来分割黑名单的起始结束时间。
cacheScanLimit	ALL Cache 时启用，读取全量HBase数据，服务端一次RPC返回给客户端的行数。	可选，默认为100条。

目前ODPS维表只支持`ALL`全量缓存策略，必须显式声明。

ALL: 全量缓存策略。即在Job运行前会将远程表中所有数据load到内存中，之后所有的维表查询都会走cache，cache命中不到则不存在，并在缓存过期后重新加载一遍全量缓存。全量缓存策略适合远程表不大、miss key特别多的场景。全量缓存策略需要配置：缓存更新间隔（`cacheTTLMs`）和更新时间黑名单（`cacheReloadTimeBlackList`）。

说明:

注意：使用ALL Cache的时候，由于会进行异步Reload，需要将维表JOIN的节点增加内存，增加的内存大小为远程表两倍的数据量。

METRIC

维表JOIN可以查看关联率，缓存命中率等METRIC信息。目前实时计算控制台还不支持查看，青铜狗通过Kmonitor查看。

查询语句	说明
fetch qps	查询维表总qps，包括命中和不命中。对应的Metric Name：`blink.projectName.jobName.dimJoin.fetchQPS` 。
fetchHitQPS	维表命中qps，包括换成命中和查询物理维表命中，对应Metric Name：`blink.projectName.jobName.dimJoin.fetchHitQPS`。
cacheHitQPS	维表缓存命中qps，对应Metric Name：`blink.projectName.jobName.dimJoin.cacheHitQPS`
dimJoin.fetchHit	维表关联率，对应Metric Name： `blink.projectName.jobName.dimJoin.fetchHit` 。
dimJoin.cacheHit	维表缓存关联率，对应Metric Name：`blink.projectName.jobName.dimJoin.cacheHit` 。

如何查看MaxCompute分区名

1. 进入数据地图。
2. 搜索表名。

3. 在数据表详情界面的明细信息 > 分区信息中进行查看。例如：adm_dim_csn_trans_shift的分区是`ds=20180905`

。

常见问题

Q：任务出现了`RejectedExecutionException: Task java.util.concurrent.ScheduledThreadPoolExecutor$ScheduledFutureTas`的Failover该怎么处理?

A：实时计算1.x版本中维表JOIN存在一定的问题，推荐升级到2.1.1及以上实时计算版本。如果需要继续使用原有版本，需要对作业进行暂停恢复操作，根据failover history中第一次出现failover的具体报错信息进行排查。

Q：MaxCompute中的数据类型和实时计算中数据类型的对应关系是什么?

A：如下表。

MaxCompute	Blink
TINYINT	TINYINT
SMALLINT	SMALLINT
INT	INT
BIGINT	BIGINT
FLOAT	FLOAT
DOUBLE	DOUBLE
BOOLEAN	BOOLEAN
DATETIME	TIMESTAMP
TIMESTAMP	TIMESTAMP

MaxCompute	Blink
VARCHAR	VARCHAR
STRING	STRING
DECIMAL	DECIMAL
BINARY	VARBINARY

说明:

其他MaxCompute类型，ODPS connector暂时还没有支持转换。

8.6.4.6 创建云数据库 Redis维表

本文为您介绍如何创建实时计算Redis维表。

说明:

- 本文仅适用于实时计算3.2.2及以上版本。
- 实时计算Redis维表仅支持引用Redis数据存储中String类型的数据。

DDL

实时计算支持使用Redis作为数据存储结果表。Redis结果表DDL定义如下。

```
CREATE TABLE white_list (
  id VARCHAR,
  name VARCHAR,
  PRIMARY KEY (id),  --Redis中的Row Key字段。
  PERIOD FOR SYSTEM_TIME --维表的标识。
) WITH (
  type = 'redis',
  host = '<yourHostName>',
  port = '<yourPort>',
  password = '<yourPassword>',
  dbNum = '<yourDatabaseNumber>'
)
```

说明:

- Redis维表必须声明且只能声明一个主键。
- 维表JOIN时，ON的条件必须包含所有主键的等值条件。
- Redis维表仅支持声明两个字段，且字段类型必须为VARCHAR。

WITH参数

参数	注释书名	备注
host	Redis连接地址	无。

参数	注释书名	备注
port	Redis连接端口	默认为6379。
dbNum	选择操作的数据库	默认为0。
password	Redis密码	默认为空，不进行权限验证。

CACHE参数

参数	注释说明	备注
cache	缓存策略	默认 `None`，可选 `LRU`。 · None：无缓存。 · LRU：最近使用策略缓存。需要配置cacheSize参数和cacheTTLMs参数。
cacheSize	缓存大小	选择`LRU`缓存策略后，可以设置缓存大小，默认为10000行。
cacheTTLMs	缓存超时时长	默认缓存不超时，单位为毫秒。可选LRU缓存策略，即设置缓存失效的超时时长。

METRIC

维表JOIN可以查看关联率，缓存命中率等Metric信息。

> 说明:
> 目前仅支持通过Kmonitor查看Metric信息。

查询语句	说明	metric name
fetch qps	查询维表总QPS，包括命中或不命中。	blink.projectName.jobName.dimJoin.fetchQPS
fetchHitQPS	维表命中QPS，包括缓存命中和查询物理维表命中。	blink.projectName.jobName.dimJoin.fetchHitQPS
cacheHitQPS	维表缓存命中QPS。	blink.projectName.jobName.dimJoin.cacheHitQPS
dimJoin.fetchHit	维表关联率。	blink.projectName.jobName.dimJoin.fetchHit

查询语句	说明	metric name
dimJoin.cacheHit	维表缓存关联率	blink.projectName. jobName.dimJoin.cacheHit

8.6.4.7 创建ElasticSearch维表

本文为您介绍如何创建实时计算ElasticSearch（ES）维表。

说明:

本文仅适用于实时计算3.2.2及以上版本。

DDL定义

实时计算支持使用ElasticSearch（ES）作为维表，示例代码如下。

```
 CREATE TABLE es_stream_sink(
  field1 LONG,
  field2 VARBINARY,
  field3 VARCHAR,
  PRIMARY KEY(field1),
  PERIOD FOR SYSTEM_TIME
) WIHT (
  type ='elasticsearch',
  endPoint = '<yourEndPoint>',
  accessId = '<yourAccessId>',
  accessKey = '<yourAccessSecret>',
  index = '<yourIndex>',
  typeName = '<yourTypeName>'
);
```

说明:

ES支持根据PRIMARY KEY进行UPDATE，且PRIMARY KEY只能为1个字段。

WITH参数

通用配置:

参数	注释说明	默认值	Required
endPoint	Server地址，例如：*http://127.0.0.1:9211*。	无	是
accessId	访问实例ID。	无	是
accessKey	访问实例密钥。	无	是
index	索引名称，类似于数据库Database的名称。	无	是

参数	注释说明	默认值	Required
typeName	Type名称，类似于数据库的Table名称。	无	是
maxRetryTimes	异常重试次数。	30	否
timeout	读取超时时长，单位为毫秒。	600000	否
discovery	是否开启节点发现。如果开启，客户端每5分钟刷新一次Server List。	false	否
compression	是否使用GZIP压缩Request Bodies。	true	否
multiThread	是否开启JestClient多线程。	true	否

CACHE参数

参数	注释说明	备注
cache	缓存策略	默认 `None`，可选 `LRU`，`ALL`。 · None：无缓存。 · LRU：最近使用策略缓存。需要配置cacheSize参数和cacheTTLMs参数。 · ALL：全量缓存策略。
cacheSize	缓存大小	选择`LRU`缓存策略后，可以设置缓存大小，默认为10000行。
cacheTTLMs	缓存超时时长	默认缓存不超时，单位为毫秒。不同缓存策略下的功能如下： · LRU：设置缓存失效的超时时长。 · ALL：设置缓存Reload的间隔时长，默认不重新加载。

8.7 DML语句

8.7.1 EMIT语句

EMIT语法可以使QUERY根据不同场景，定义不同的输出策略，从而达到控制延迟或提高数据准确性的效果。

说明:

EMIT语法仅支持实时计算2.0以上版本。

什么是EMIT策略

EMIT策略是指在Flink SQL中，QUERY根据不同场景选择不同的输出策略（例如最大延迟时长）。传统的ANSI SQL语法不支持这类输出策略。例如，1小时的时间窗口，窗口触发之前希望每分钟都能看到最新的结果，窗口触发之后希望不丢失迟到一天内的数据。针对这类场景，实时计算抽象出EMIT语法，并扩展到SQL语法。以下为不同场景下EMIT策略的示例：

- 窗口结束之前，按1分钟延迟输出，窗口结束之后无延迟输出。

```
EMIT
  WITH DELAY '1'MINUTE BEFORE WATERMARK,
  WITHOUT DELAY AFTER WATERMARK
```

- 窗口结束之前不输出，窗口结束之后无延迟输出。

```
EMIT WITHOUT DELAY AFTER WATERMARK
```

- 全局都按1分钟的延迟输出 (minibatch)。

```
EMIT WITH DELAY '1'MINUTE
```

- 窗口结束之前按1分钟延迟输出。

```
EMIT WITH DELAY '1'MINUTE BEFORE WATERMARK
```

EMIT语法的用途

EMIT语法能够实现以下2种功能：

- 控制延迟：针对窗口，设置窗口触发之前的EMIT输出频率，减少结果输出延迟。
- 提高数据精确性：不丢弃窗口触发之后的迟到的数据，修正输出结果。

说明:

选择EMIT策略时，请权衡业务复杂程度和资源消耗情况。越低的输出延迟、越高的数据精确性，需要提供越高的计算开销。

EMIT语法

EMIT语法是定义输出的策略，即定义在输出（INSERT INTO）中。当未配置EMIT语法时，保持原有默认行为，即WINDOW只在WATERMARK触发时EMIT一个结果。

说明:

EMIT语句只能置于INSERT INTO语句中的所有QUERY语句之后，不能置于VIEW语句中。

```
INSERT INTO tableName
query
EMIT strategy [, strategy]*

strategy ::= {WITH DELAY timeInterval | WITHOUT DELAY}
                [BEFORE WATERMARK |AFTER WATERMARK]

timeInterval ::='string' timeUnit
```

参数	说明
`WITH DELAY`	可延迟输出，即按指定时长进行间隔输出。
`WITHOUT DELAY`	不可以延迟输出，即每来一条数据就进行输出。
`BEFORE WATERMARK`	窗口结束之前的策略配置，即WATERMARK触发之前。
`AFTER WATERMARK`	窗口结束之后的策略配置，即WATERMARK触发之后。

说明:

- 其中`strategy`可以定义多个，同时定义BEFORE和BEFORE的策略。 但不能同时定义两个BEFORE或两个AFTER的策略。
- 若配置了AFTER WATERMARK策略，需要使用明文方式声明 `blink.state.ttl.ms` ，标识最大延迟时长。

生命周期

AFTER策略允许接收迟到的数据，窗口的状态（State）允许保留一定时长，等待迟到的数据。这段保留的时长称为生命周期TTL。运用AFTRE策略后，通过明文声明 `blink.state.ttl.ms`参数，您可以设置状态允许的生命周期。例如， `blink.state.ttl.ms=3600000` 表示状态允许保留超时时长为1小时内的数据，超时时长大于1小时的数据不被录入状态。

EMIT语法示例

窗口区间为1小时的滚动窗口 `tumble_window`的语法示例如下。

```
CREATE VIEW tumble_window AS
SELECT
```

```
  id,
  TUMBLE_START(rowtime, INTERVAL '1' HOUR) as start_time,
  COUNT(*) as cnt
FROM source
GROUP BY id, TUMBLE(rowtime, INTERVAL '1' HOUR)
```

默认 tumble_window 的输出需要等到1小时结束才能显示。如果您需要尽早看到窗口的结果（即使是不完整的结果），例如每分钟看到最新的窗口结果，可以添加如下语句：

```
INSERT INTO result
SELECT * FROM tumble_window
EMIT WITH DELAY '1' MINUTE BEFORE WATERMARK -- 窗口结束之前，每隔1分钟输出一次更新结果。
```

默认 tumble_window 会忽略并丢弃窗口结束后到达的数据，如果您需要将窗口结束后1天到达的数据统计进入结果，并且需要每接收1条数据后立刻更新结果，可以添加如下语句：

```
INSERT INTO result
SELECT * FROM tumble_window
EMIT WITH DELAY '1' MINUTE BEFORE WATERMARK,
     WITHOUT DELAY AFTER WATERMARK  -- 窗口结束后，每收到一条数据输出一次更新结果。

blink.state.ttl.ms = 86400000    --增加1天状态生命周期的配置。
```

DELAY概念

EMIT策略中的DELAY指的是用户可接受的数据延迟时长，该延迟是指从用户的数据从进入实时计算，到看到结果数据（从实时计算系统输出）的时间（Event Time或Processing Time）。延迟的计算基于系统时间。动态表（流式数据在实时计算内部的存储）中的数据发生变化的时间和结果表（实时计算外部的存储）中显示新记录的时间的间隔，称为延迟。

假设，实时计算系统的处理耗时是0，则在流式数据积攒的过程和Window等待窗口数据的过程可能会导致延迟。如果，用户指定了最多延迟30秒，则30秒可用于流式数据的积攒。如果Query是1小时的窗口，则最多延迟30秒的含义是每隔30秒更新结果数据。

- 配置EMIT WITH DELAY '1' MINUTE

  如果是Group By聚合，则会用1分钟积攒流式数据。如果有Window且Window的Size大于1分钟，则Window会每隔1分钟更新一次结果数据，否则忽略这个配置，因为窗口依靠Watermark的输出就能保证这个Latency SLA。
- 配置EMIT WITHOUT DELAY

  如果是Group By函数，则不会启用minibatch，每来一条数据都触发计算和输出。如果是Window函数，则也是每来一条数据都触发计算和输出。

目前状态和未来计划

目前状态和未来计划：

1. 目前EMIT策略只支持TUMBLE，HOP窗口，暂不支持SESSION窗口。
2. 目前一个Job若有多个输出，多个输出的EMIT需要定义成一样的策略。未来会支持不一样的策略。
3. 目前EMIT语法还不能用来配置minibatch的`allowLateness`，未来打算使用EMIT策略来声明`allowLateness`。

8.7.2 INSERT INTO语句

本文为您介绍如何在实时使计算中使用INSERT INTO语句以及INSERT INTO的操作约束。

语法

```
INSERT INTO tableName
      [ (columnName[ , columnName]*) ]
          queryStatement;
```

示例

```
INSERT INTO LargeOrders
SELECT * FROM Orders WHERE units > 1000;
INSERT INTO Orders(z, v)
SELECT c,d FROM OO;
```

说明:

- 单个实时计算作业支持在一个SQL作业里面包含多个DML操作，同样也允许包含多个数据源、多个数据目标端、多个维表。例如，在一个作业文件里面包含两段完全业务上独立的SQL，分别写出到不同的数据目标端。
- 实时计算不支持单独的SELECT查询，必须有CREATE VIEW或是在INSERT INTO内才能操作。
- INSERT INTO支持UPDATE更新。例如，向RDS的表插入一个KEY值。如果这个KEY值存在就会更新，如果不存在就会插入一条新的KEY值。

操作约束

表类型	操作约束
源表	只能引用（FROM），不可执行INSERT。
维表	只能引用（JOIN），不可执行INSERT。
结果表	仅支持INSERT操作。
视图	只能引用（FROM）。

8.8 QUERY语句

8.8.1 SELECT语句

SELECT语句用于从表中选取数据。

语法

```
SELECT [ DISTINCT ]
{ * | projectItem [, projectItem ]* }
FROM tableExpression;
```

测试数据

a（VARCHAR）	b（INT）	c（DATE）
a1	211	1990-02-20
b1	120	2018-05-12
c1	89	2010-06-14
a1	46	2016-04-05

示例一

- 测试语句

```
SELECT * FROM 表名;
```

- 测试结果

a（VARCHAR）	b（INT）	c（DATE）
a1	211	1990-02-20
b1	120	2018-05-12
c1	89	2010-06-14
a1	46	2016-04-05

示例二

- 测试语句

```
SELECT a, c AS d FROM 表名;
```

- 测试结果

a（VARCHAR）	d（DATE）
a1	1990-02-20

a（VARCHAR）	d（DATE）
b1	2018-05-12
c1	2010-06-14
a1	2016-04-05

示例三

- 测试语句

```
SELECT DISTINCT a FROM 表名;
```

- 测试结果

a（VARCHAR）
a1
b1
c1

子查询

普通的SELECT是从几张表中读数据，如`SELECT column_1, column_2 … FROM table_name`，但查询的对象也可以是另外一个SELECT操作。

说明:

当查询的对象是另一个SELECT操作时，必须为子查询加别名。示例如下。

- 示例语句

```
INSERT INTO result_table
SELECT * FROM
               (SELECT   t.a,
                         sum(t.b) AS sum_b
                FROM     t1 t
                GROUP BY t.a
               ) t1
WHERE  t1.sum_b > 100;
```

- 示例结果

a（VARCHAR）	b（INT）
a1	211
b1	120
a1	257

8.8.2 WHERE语句

WHERE语句可用于对SELECT语句中的数据进行筛选。

语法

```
SELECT [ ALL | DISTINCT ]
{ * | projectItem [, projectItem ]* }
FROM tableExpression
[ WHERE booleanExpression ];
```

下面的运算符可在WHERE语句中使用。

操作符	描述
=	等于
<>	不等于
>	大于
>=	大于等于
<	小于
<=	小于等于

示例

- 测试数据

Address	City
Oxford Street	Beijing
Fifth Avenue	Beijing
Changan Street	shanghai

- 测试语句

```
SELECT * FROM XXXX WHERE City='Beijing'
```

- 测试结果

Address	City
Oxford Street	Beijing
Fifth Avenue	Beijing

8.8.3 HAVING语句

在使用聚合函数时，您需要添加HAVING语句，以实现与WHERE语句一样的过滤效果。

语法

```
SELECT [ ALL | DISTINCT ]{ * | projectItem [, projectItem ]* }
FROM tableExpression
[ WHERE booleanExpression ]
[ GROUP BY { groupItem [, groupItem ]* } ]
[ HAVING booleanExpression ];
```

示例

- 测试数据

Customer	OrderPrice
Bush	1000
Carter	1600
Bush	700
Bush	300
Adams	2000
Carter	100

- 测试语句

```
SELECT Customer,SUM(OrderPrice) FROM XXX
GROUP BY Customer
HAVING SUM(OrderPrice)<2000;
```

- 测试结果

Customer	SUM（OrderPrice）
Carter	1700

8.8.4 GROUP BY语句

GROUP BY语句用于根据一个或多个列对结果集进行分组。

语法

```
SELECT [ DISTINCT ]
{ * | projectItem [, projectItem ]* }
FROM tableExpression
```

```
[ GROUP BY { groupItem [, groupItem ]* } ];
```

示例

- **测试数据**

Customer	OrderPrice
Bush	1000
Carter	1600
Bush	700
Bush	300
Adams	2000
Carter	100

- **测试语句**

```
SELECT Customer,SUM(OrderPrice) FROM xxx
GROUP BY Customer;
```

- **测试结果**

Customer	SUM(OrderPrice)
Bush	2000
Carter	1700
Adams	2000

8.8.5 JOIN语句

实时计算的JOIN和传统批处理JOIN的语义一致，都用于将两张表联系起来。区别的是实时计算联系的是两张动态表，联系的结果也会动态更新以保证最终结果和批处理结果保持一致。

语法

```
tableReference [, tableReference ]* | tableexpression
[ LEFT ] JOIN tableexpression [ joinCondition ];
```

说明:

- **只支持等值连接，不支持不等连接。**
- **只支持INNER JOIN和LEFT OUTER JOIN两种JOIN方式。**

示例1

- **测试数据**

  Orders：

rowtime	productId	orderId	units
10:17:00	30	5	4
10:17:05	10	6	1
10:18:05	20	7	2
10:18:07	30	8	20
11:02:00	10	9	6
11:04:00	10	10	1
11:09:30	40	11	12
11:24:11	10	12	4

  Products：

productId	name	unitPrice
30	Cheese	17
10	Beer	0.25
20	Wine	6
30	Cheese	17
10	Beer	0.25
10	Beer	0.25
40	Bread	100
10	Beer	0.25

- **测试语句**

```
  SELECT o.rowtime, o.productId, o.orderId, o.units,p.name, p.
unitPrice
  FROM Orders AS o
  JOIN Products AS p
  ON o.productId = p.productId;
```

```
```

- 测试结果

rowtime	productId	orderId	units	name	unitPrice
10:17:00	30	5	4	Cheese	17
10:17:05	10	6	1	Beer	0.25
10:18:05	20	7	2	Wine	6
10:18:07	30	8	20	Cheese	17
11:02:00	10	9	6	Beer	0.25
11:04:00	10	10	1	Beer	0.25
11:09:30	40	11	12	Bread	100
11:24:11	10	12	4	Beer	0.25

示例2

- 测试数据

datahub_stream1：

a（BIGINT）	b（BIGINT）	c（VARCHAR）
0	10	test11
1	10	test21

datahub_stream2：

a（BIGINT）	b（BIGINT）	c（VARCHAR）
0	10	test11
1	10	test21
0	10	test31
1	10	test41

- 测试语句

```
SELECT s1.c,s2.c
FROM datahub_stream1 AS s1
JOIN datahub_stream2 AS s2
ON s1.a =s2.a
WHERE s1.a = 0;
```

- 测试结果

s1.c（VARCHAR）	s2.c（VARCHAR）
test11	test11
test11	test31

8.8.6 维表JOIN语句

维表是一张不断变化的表，因此在JOIN维表的时候，需指明这条记录关联维表快照的时刻。目前维表JOIN仅支持对当前时刻维表快照的关联。（未来会支持关联左表rowtime所对应的维表快照。）

维表JOIN语法

```
SELECT column-names
FROM table1  [AS <alias1>]
[LEFT] JOIN table2 FOR SYSTEM_TIME AS OF PROCTIME() [AS <alias2>]
ON table1.column-name1 = table2.key-name1
```

例如，事件流JOIN白名单维表的SQL如下。

```
SELECT e.*, w.*
FROM event AS e
JOIN white_list FOR SYSTEM_TIME AS OF PROCTIME() AS w
ON e.id = w.id
```

说明:

- 维表支持INNER JOIN和LEFT JOIN，不支持RIGHT JOIN或FULL JOIN。
- 必须加上FOR SYSTEM_TIME AS OF PROCTIME()，表示JOIN维表当前时刻所看到的每条数据。
- JOIN行为只发生在处理时间（processing time），即使维表中的数据新增、更新或删除，已关联的数据也不会被改变或撤回。
- ON条件必须包含维表PRIMARY KEY的等值条件（且要求与真实表定义一致），但除此之外，ON条件中也可以有其他等值条件。
- 维表和维表不能做JOIN。

示例

- **测试数据**

  nameinfo：

id（bigint）	name（varchar）	age（bigint）
1	lilei	22
2	hanmeimei	20
3	libai	28

  phoneNumber：

name（varchar）	phoneNumber（bigint）
dufu	18867889855
baijuyi	18867889856
libai	18867889857
lilei	18867889858

- **测试语句**

```
CREATE TABLE datahub_input1 (
id            BIGINT,
name        VARCHAR,
age           BIGINT
) WITH (
type='datahub'
);

create table phoneNumber(
name VARCHAR,
phoneNumber bigint,
primary key(name),
PERIOD FOR SYSTEM_TIME
)with(
type='rds'
);

CREATE table result_infor(
id bigint,
phoneNumber bigint,
name VARCHAR
)with(
type='rds'
);

INSERT INTO result_infor
SELECT
t.id,
w.phoneNumber,
t.name
FROM datahub_input1 as t
```

```
JOIN phoneNumber FOR SYSTEM_TIME AS OF PROCTIME() as w
ON t.name = w.name;
```

- 测试结果

id（bigint）	phoneNumber（bigint）	name（varchar）
1	18867889858	lilei
3	18867889857	libai

8.8.7 UNION ALL语句

UNION ALL语句将两个流式数据合并。要求两个流式数据的字段完全一致，包括字段类型和字段顺序。

语法

```
select_statement
UNION ALL
select_statement;
```

说明:

实时计算同样支持UNION函数。`UNION ALL`允许重复值，`UNION`不允许重复值。在实时计算底层，`UNION`是`UNION ALL`+`Distinct`，运行效率较低，一般不推荐使用`UNION`。

示例

- 测试数据

test_source_union1：

a（varchar）	b（bigint）	c（bigint）
test1	1	10

test_source_union2：

a（varchar）	b（bigint）	c（bigint）
test1	1	10
test2	2	20

test_source_union3：

a（varchar）	b（bigint）	c（bigint）
test1	1	10
test2	2	20

a（varchar）	b（bigint）	c（bigint）
test1	1	10

- 测试语句

```
SELECT
    a,
    sum(b),
    sum(c)
FROM
    (SELECT * from test_source_union1
    UNION ALL
    SELECT * from test_source_union2
    UNION ALL
    SELECT * from test_source_union3
    )t
 GROUP BY a;
```

- 测试结果

d（varchar）	e（bigint）	f（bigint）
test1	1	10
test2	2	20
test1	2	20
test1	3	30
test2	4	40
test1	4	40

8.8.8 TopN语句

TopN语句用于对实时数据中某个指标的前N个最值的筛选。Flink SQL可以基于OVER窗口操作灵活地完成TopN的功能。

语法

```
SELECT *
FROM (
  SELECT *,
    ROW_NUMBER() OVER ([PARTITION BY col1[, col2..]]
    ORDER BY col1 [asc|desc][, col2 [asc|desc]...]) AS rownum
  FROM table_name)
WHERE rownum <= N [AND conditions]
```

说明:

- ROW_NUMBER()：行号计算函数OVER的窗口，行号计算从1开始。
- PARTITION BY col1[, col2..]：指定分区的列，可以不指定。

- `ORDER BY col1 [asc|desc][, col2 [asc|desc]...]`：指定排序的列和每列的排序方向。

如上语法所示，TopN需要两层Query：

- 查询中使用 `ROW_NUMBER()` 窗口函数来对数据根据排序列进行排序并标上排名。
- 外层查询中对排名进行过滤，只取前N条。例如N=10即为取前10条的数据。

在执行过程中，Flink SQL会对输入的数据流根据排序键进行排序。如果某个分区的前N条记录发生了改变，则会将改变的那几条数据以更新流的形式发给下游。

说明:

如果需要将TopN的数据输出到外部存储，后接的结果表必须是一个带主键的表。

WHERE条件的限制

为了使Flink SQL能识别出这是一个TopN的query，外层循环中必须要指定 `rownum <= N`的格式来指定前N条记录，不能使用`rownum - 5 <= N` 这种将`rownum`至于某个表达式中。WHERE条件中，可以额外带上其他条件，但是必须是以`AND`连接。

示例1

如下示例，先统计查询流中每小时、每个城市和关键字被查询的次数。然后输出每小时、每个城市被查询最多的前100个关键字。在输出表中，小时、城市、排名三者可以唯一确定一条记录，所以需要将这三列声明成联合主键（需要在外部存储中也有同样的主键设置）。

```
CREATE TABLE rds_output (
  rownum BIGINT,
  start_time BIGINT,
  city VARCHAR,
  keyword VARCHAR,
  pv BIGINT,
  PRIMARY KEY (rownum, start_time, city)
) WITH (
  type = 'rds',
  ...
)

INSERT INTO rds_output
SELECT rownum, start_time, city, keyword, pv
FROM (
  SELECT *,
     ROW_NUMBER() OVER (PARTITION BY start_time, city ORDER BY pv desc
) AS rownum
  FROM (
        SELECT SUBSTRING(time_str,1,12) AS start_time,
            keyword,
            count(1) AS pv,
            city
        FROM tmp_search
        GROUP BY SUBSTRING(time_str,1,12), keyword, city
    ) a
```

```
) t
WHERE rownum <= 100
```

示例2

- **测试数据**

ip（VARCHAR）	time（VARCHAR）
192.168.1.1	100000000
192.168.1.2	100000000
192.168.1.2	100000000
192.168.1.3	100030000
192.168.1.3	100000000
192.168.1.3	100000000

- **测试语句**

```
CREATE TABLE source_table (
  IP VARCHAR,
  `TIME` VARCHAR
)WITH(
  type='datahub',
  endPoint='<yourEndpoint>',
  project='<yourProjectName>',
  topic='<yourTopicName>',
  accessId='<yourAccessId>',
  accessKey='<yourAccessSecret>'
);

CREATE TABLE result_table (
  rownum BIGINT,
  start_time VARCHAR,
  IP VARCHAR,
  cc BIGINT,
  PRIMARY KEY (start_time, IP)
) WITH (
  type = 'rds',
  url='<yourDatabaseAddress>',
  tableName='blink_rds_test',
  userName='<yourDatabaseUserName>',
  password='<yourDatabasePassword>'
);
INSERT INTO result_table
SELECT rownum,start_time,IP,cc
FROM (
  SELECT *,
     ROW_NUMBER() OVER (PARTITION BY start_time ORDER BY cc DESC) AS
 rownum
  FROM (
        SELECT SUBSTRING(`TIME`,1,2) AS start_time,--可以根据真实时间取
相应的数值，这里取得是测试数据。
        COUNT(IP) AS cc,
        IP
        FROM  source_table
        GROUP BY SUBSTRING(`TIME`,1,2), IP
```

```
    )a
) t
WHERE rownum <= 3 --可以根据真实top值取相应的数值，这里取得是测试数据。
```

- 测试结果

rownum（BIGINT）	start_time（VARCHAR）	ip（VARCHAR）	cc（BIGINT）
1	10	192.168.1.3	6
2	10	192.168.1.2	4
3	10	192.168.1.1	2

无排名优化

- 无排名优化解决数据膨胀问题
  - 数据膨胀问题

    根据TopN的语法，rownum字段会作为结果表的主键字段之一写入结果表。但是这可能导致数据膨胀的问题。例如，收到一条原排名9的更新数据，更新后排名上升到1，则从1到9的数据排名都发生变化了。需要将这些数据作为更新都写入结果表。这样就产生了数据膨胀，可能导致结果表因为收到了太多的数据而降低更新速度。
  - 无排名优化方法

    结果表中不保存 rownum，最终的 rownum 由前端计算。因为TopN的数据量通常不会很大，前端排序100个数据很快。当收到一条原排名9，更新后排名上升到1的数据，也只需要发送这一条数据，而不用把排名1到9的数据全发送下去。这种优化能够提升结果表的更新速度。
- 无排名优化语法

```
SELECT col1, col2, col3
FROM (
 SELECT col1, col2, col3
   ROW_NUMBER() OVER ([PARTITION BY col1[, col2..]]
   ORDER BY col1 [asc|desc][, col2 [asc|desc]...]) AS rownum
 FROM table_name)
WHERE rownum <= N [AND conditions]
```

语法与上文类似，只是在外层查询中将rownum字段裁剪掉即可。

说明:

在无rownum的场景，结果表主键的定义一定要特别注意。如果定义有误，会直接导致TopN结果的不正确。 无rownum场景的主键应为TopN上游GROUP BY节点的KEY列表。

- 无排名优化示例

  本示例来自某视频行业用户的真实业务（精简后）。用户每个视频在分发时会产生大量流量，依据视频产生的流量可以分析出最热门的视频。如下示例用于统计出每分钟流量最大的Top5的视频。

  - 测试语句

```
--从SLS读取数据原始存储表。
CREATE TABLE sls_cdnlog_stream (
vid VARCHAR, -- video id
rowtime TIMESTAMP, -- 观看视频的时间。
response_size BIGINT, -- 观看产生的流量。
WATERMARK FOR rowtime as withOffset(rowtime, 0)
) WITH (
type='sls',
...
);

--1分钟窗口统计vid带宽数。
CREATE VIEW cdnvid_group_view AS
SELECT vid,
TUMBLE_START(rowtime, INTERVAL '1' MINUTE) AS start_time,
SUM(response_size) AS rss
FROM sls_cdnlog_stream
GROUP BY vid, TUMBLE(rowtime, INTERVAL '1' MINUTE);

--存储表。
CREATE TABLE hbase_out_cdnvidtoplog (
vid VARCHAR,
rss BIGINT,
start_time VARCHAR,
   -- 注意结果表中不存储rownum字段。
   -- 特别注意该主键的定义，为TopN上游GROUP BY的KEY。
PRIMARY KEY(start_time, vid)
) WITH (
type='RDS',
...
);

-- 统计每分钟Top5消耗流量的vid，并输出。
INSERT INTO hbase_out_cdnvidtoplog

-- 注意次外层查询，不选出rownum字段。
SELECT vid, rss, start_time FROM
(
SELECT
vid, start_time, rss,
ROW_NUMBER() OVER (PARTITION BY start_time ORDER BY rss DESC) as
rownum
FROM
cdnvid_group_view
)
```

```
WHERE rownum <= 5;
```

- 测试数据

vid（VARCHAR）	rowtime（Timestamp）	response_size（BIGINT）
10000	2017-12-18 15:00:10	2000
10000	2017-12-18 15:00:15	4000
10000	2017-12-18 15:00:20	3000
10001	2017-12-18 15:00:20	3000
10002	2017-12-18 15:00:20	4000
10003	2017-12-18 15:00:20	1000
10004	2017-12-18 15:00:30	1000
10005	2017-12-18 15:00:30	5000
10006	2017-12-18 15:00:40	6000
10007	2017-12-18 15:00:50	8000

- 测试结果

start_time（VARCHAR）	vid（VARCHAR）	rss（BIGINT）
2017-12-18 15:00:00	10000	9000
2017-12-18 15:00:00	10007	8000
2017-12-18 15:00:00	10006	6000
2017-12-18 15:00:00	10005	5000
2017-12-18 15:00:00	10002	4000

8.8.9 复杂事件处理（CEP）语句

复杂事件处理（CEP）语句MATCH_RECOGNIZE用于从输入流中识别符合指定规则的事件，并按照指定的方式输出。

语法

```
SELECT [ ALL | DISTINCT ]
{ * | projectItem [, projectItem ]* }
FROM tableExpression
[MATCH_RECOGNIZE (
[PARTITION BY {partitionItem [, partitionItem]*}]
[ORDER BY {orderItem [, orderItem]*}]
[MEASURES {measureItem AS col [, measureItem AS col]*}]
[ONE ROW PER MATCH|ALL ROWS PER MATCH|ONE ROW PER MATCH WITH TIMEOUT
ROWS|ALL ROWS PER MATCH WITH TIMEOUT ROWS]
[AFTER MATCH SKIP]
```

```
PATTERN (patternVariable[quantifier] [ patternVariable[quantifier]]*)
WITHIN intervalExpression
DEFINE {patternVariable AS patternDefinationExpression [, patternVar
iable AS patternDefinationExpression]*}
)];
```

参数	说明
PARTITION BY	分区的列，可选项。
ORDER BY	可指定多列，但是必须以EVENT TIME列或者PROCESS TIME列作为排序的首列，可选项。
MEASURES	定义如何根据匹配成功的输入事件构造输出事件。
ONE ROW PER MATCH	对于每一次成功的匹配，只会产生一个输出事件。
ONE ROW PER MATCH WITH TIMEOUT ROWS	除了匹配成功的时候产生输出外，超时的时候也会产生输出。超时时间由PATTERN语句中的WITHIN语句定义。
ALL ROWS PER MATCH	对于每一次成功的匹配，对应于每一个输入事件，都会产生一个输出事件。
ALL ROWS PER MATCH WITH TIMEOUT ROWS	除了匹配成功的时候产生输出外，超时的时候也会产生输出。超时时间由PATTERN语句中的WITHIN语句定义。
[ONE ROW PER MATCH|ALL ROWS PER MATCH|ONE ROW PER MATCH WITH TIMEOUT ROWS|ALL ROWS PER MATCH WITH TIMEOUT ROWS]	为可选项，默认为ONE ROW PER MATCH。
AFTER MATCH SKIP TO NEXT ROW	匹配成功之后，从匹配成功的事件序列中的第一个事件的下一个事件开始进行下一次匹配。
AFTER MATCH SKIP PAST LAST ROW	匹配成功之后，从匹配成功的事件序列中的最后一个事件的下一个事件开始进行下一次匹配。
AFTER MATCH SKIP TO FIRST patternItem	匹配成功之后，从匹配成功的事件序列中第一个对应于patternItem的事件开始下一次匹配。
AFTER MATCH SKIP TO LAST patternItem	匹配成功之后，从匹配成功的事件序列中最后一个对应于patternItem的事件开始下一次匹配。

参数	说明
PATTERN	定义待识别的事件序列需要满足的规则，需要定义在()中，由一系列自定义的patternVariable构成。  说明: · patternVariable之间若以空格间隔，表示符合这两种patternVariable的事件中间不存在其他事件。 · patternVariable之间若以->间隔，表示符合这两种patternVariable的事件之间可以存在其它事件。

· quantifier

quantifier用于指定符合patternVariable定义的事件的出现次数。

参数	参数意义
*	0次或多次
+	1次或多次
?	0次或1次
{n}	n次
{n,}	大于等于n次
{n, m}	大于等于n次，小于等于m次

参数	参数意义
{,m}	小于等于m次

默认为贪婪匹配。比如对于`pattern: A -> B+ -> C`，输入为`a bc1 bc2 c`（其中bc1和bc2表示既匹配B也匹配C），则输出为：`a bc1 bc2 c`。可以在quantifier符号后面加`?`来表示非贪婪匹配。

- `*?`
- `+?`
- `{n}?`
- `{n,}?`
- `{n, m}?`
- `{,m}?`

此时对于上面例子中的pattern及输入，产生的输出为：`a bc1 bc2,a bc1 bc2 c`。

说明:

- WITHIN 定义符合规则的事件序列的最大时间跨度。
- 静态窗口格式：`INTERVAL ‘string’ timeUnit [ TO timeUnit ]` 示例：`INTERVAL ‘10’ SECOND, INTERVAL ‘45’ DAY, INTERVAL ‘10:20’ MINUTE TO SECOND, INTERVAL ‘10:20.10’ MINUTE TO SECOND, INTERVAL ‘10:20’ HOUR TO MINUTE, INTERVAL ‘1-5’ YEAR TO MONTH`。
- 动态窗口格式：`INTERVAL intervalExpression` 示例：`INTERVAL A.windowTime + 10`，其中A为pattern定义中第一个patternVariable。在intervalExpression的定义中，可以使用pattern定义中出现过的patternVariable。当前只能使用第一个patternVariable。intervalExpression中可以使用UDF，intervalExpression的结果必须为long，单位为millisecond，表示窗口的大小。
- DEFINE 定义在PATTERN中出现的patternVariable的具体含义，若某个patternVariable在DEFINE中没有定义，则认为对于每一个事件，该patternVariable都成立。

- MEASURES和DEFINE语句函数

函数	函数意义
Row Pattern Column References	形式为：`patternVariable.col`。表示访问`patternVariable`所对应的事件的指定的列。

函数	函数意义
PREV	只能用在DEFINE语句中，一般与Row Pattern Column References合用。用于访问指定的pattern所对应的事件之前偏移指定的offset所对应的事件的指定的列。示例：对于DOWN AS DOWN.price < PREV(DOWN.price)，PREV(A.price)表示当前事件的前一个事件的price列的值。注意，DOWN.price等价于PREV(DOWN.price, 0)。 PREV(DOWN.price)等价于PREV(DOWN.price, 1)。
FIRST、LAST	一般与Row Pattern Column References合用，用于访问指定的PATTERN所对应的事件序列中的指定偏移位置的事件。示例：FIRST(A.price, 3)表示PATTERN A所对应的事件序列中的第4个事件。LAST(A.price, 3)表示PATTERN A所对应的事件序列中的倒数第4个事件。

- 输出列

函数	输出列
ONE ROW PER MATCH	包括PARTITION BY中指定的列及MEASURES中定义的列。 对于PARTITION BY中已经指定的列，在MEASURES中无需重复定义。
ONE ROW PER MATCH WITH TIMEOUT ROWS	除匹配成功的时候产生输出外，超时的时候也会产生输出，超时时间由PATTERN语句中的WITHIN语句定义。

说明:

- 定义PATTERN的时候，最好也定义WITHIN，否则可能会造成STATE越来越大。
- ORDER BY中定义的首列必须为EVENT TIME列或者PROCESS TIME列。

示例

- 示例语法

```
SELECT *
FROM Ticker MATCH_RECOGNIZE (
PARTITION BY symbol
ORDER BY tstamp
MEASURES STRT.tstamp AS start_tstamp,
```

```
LAST(DOWN.tstamp) AS bottom_tstamp,
LAST(UP.tstamp) AS end_tstamp
ONE ROW PER MATCH
AFTER MATCH SKIP TO NEXT ROW
PATTERN (STRT DOWN+ UP+) WITHIN INTERVAL '10' SECOND
DEFINE
DOWN AS DOWN.price < PREV(DOWN.price),
UP AS UP.price > PREV(UP.price)
)
```

- **测试数据**

timestamp（TIMESTAMP）	card_id(VARCHAR)	location（VARCHAR）	action（VARCHAR）
2018-04-13 12:00:00	1	WW	Tom
2018-04-13 12:05:00	1	WW1	Tom
2018-04-13 12:10:00	1	WW2	Tom
2018-04-13 12:20:00	1	WW	Tom

- **测试案例语法**

```
CREATE TABLE datahub_stream (
    `timestamp`               TIMESTAMP,
    card_id                   VARCHAR,
    location                  VARCHAR,
    `action`                  VARCHAR,
    WATERMARK wf FOR `timestamp` AS withOffset(`timestamp`, 1000)
) WITH (
    type = 'datahub'
    ...
);
CREATE TABLE rds_out (
    start_timestamp               TIMESTAMP,
    end_timestamp                 TIMESTAMP,
    card_id                       VARCHAR,
    event                         VARCHAR
) WITH (
    type= 'rds'
    ...
);

--案例描述
-- 当相同的card_id在十分钟内，在两个不同的location发生刷卡现象，就会触发报警机制，以便于监测信用卡盗刷等现象。

-- 定义计算逻辑
insert into rds_out
select
`start_timestamp`,
`end_timestamp`,
card_id, `event`
```

```
from datahub_stream
MATCH_RECOGNIZE (
    PARTITION BY card_id   -- 按card_id分区，将相同卡号的数据分到同一个计
算节点上。
    ORDER BY `timestamp`   -- 在窗口内，对事件时间进行排序。
    MEASURES               --定义如何根据匹配成功的输入事件构造输出事件。
        e2.`action` as `event`,
        e1.`timestamp` as `start_timestamp`,   --第一次的事件时间为
start_timestamp。
        LAST(e2.`timestamp`) as `end_timestamp`--最新的事件时间为
end_timestamp。
    ONE ROW PER MATCH           --匹配成功输出一条。
    AFTER MATCH SKIP TO NEXT ROW--匹配后跳转到下一行。
    PATTERN (e1 e2+) WITHIN INTERVAL '10' MINUTE  -- 定义两个事件，e1和
e2。
    DEFINE                      --定义在PATTERN中出现的patternVariable的
具体含义。
        e1 as e1.action = 'Tom',    --事件一的action标记为Tom。
        e2 as e2.action = 'Tom' and e2.location <> e1.location --事件
二的action标记为Tom，且事件一和事件二的location不一致。
);
```

· 测试结果

start_timestamp（TIMESTAMP）	end_timestamp（TIMESTAMP）	card_id（VARCHAR）	event（VARCHAR）
2018-04-13 12:00:00.0	2018-04-13 12:05:00.0	1	Tom
2018-04-13 12:05:00.0	2018-04-13 12:10:00.0	1	Tom

8.9 窗口函数

8.9.1 窗口函数概述

本文为您介绍Flink SQL支持的窗口函数以及窗口函数支持的时间属性和窗口类型。

窗口函数

窗口函数Flink SQL支持基于无限大窗口的聚合（无需显式定义在SQL Query中添加任何的窗口）以及对一个特定的窗口的聚合。例如，需要统计在过去的1分钟内有多少用户点击了某个的网页，可以通过定义一个窗口来收集最近1分钟内的数据，并对这个窗口内的数据进行计算。

Flink SQL支持的窗口聚合主要是两种：Window聚合和Over聚合。本文档主要介绍Window聚合。Window聚合支持两种时间属性定义窗口：Event Time和Processing Time。每种时间属性类型支持三种窗口类型：滚动窗口（TUMBLE）、滑动窗口（HOP）和会话窗口（SESSION）。

时间属性

Flink SQL支持以下2种时间属性。实时计算可以基于这2种时间属性对数据进行窗口聚合。

- Event Time：您提供的事件时间（通常是数据的最原始的创建时间）Event Time一定是您提供在Schema里的数据。
- Processing Time：对事件进行处理的本地系统时间。

说明:

实时计算时间属性详情，请参见时间属性。

级联窗口

Rowtime列在经过窗口操作后，其Event Time属性将丢失。您可以使用辅助函数TUMBLE_ROWTIME、HOP_ROWTIME或SESSION_ROWTIME获取窗口中的Rowtime列的最大值max(rowtime)作为时间窗口的Rowtime，其类型是具有Rowtime属性的TIMESTAMP，取值为 window_end - 1 。 例如[00:00, 00:15) 的窗口，返回值为00:14:59.999 。

级联窗口示例如下。示例逻辑为：基于1分钟的滚动窗口聚合结果，进行1小时的滚动窗口聚合。

```
CREATE TABLE user_clicks(
  username varchar,
  click_url varchar,
  ts timeStamp,
  WATERMARK wk FOR ts as withOffset(ts, 2000)  --为Rowtime定义Watermark。
) with (
  type='datahub',
  ...
);

CREATE TABLE tumble_output(
  window_start TIMESTAMP,
  window_end TIMESTAMP,
  username VARCHAR,
  clicks BIGINT
) with (
  type='print'
);

CREATE VIEW one_minute_window_output as
SELECT
  // 使用TUMBLE_ROWTIME作为二级Window的聚合时间
  TUMBLE_ROWTIME(ts, INTERVAL '1' MINUTE) as rowtime,
  username,
  COUNT(click_url) as cnt
FROM user_clicks
GROUP BY TUMBLE(ts, INTERVAL '1' MINUTE), username;

INSERT INTO tumble_output
SELECT
  TUMBLE_START(rowtime, INTERVAL '1' HOUR),
  TUMBLE_END(rowtime, INTERVAL '1' HOUR),
  username,
  SUM(cnt)
FROM one_minute_window_output
```

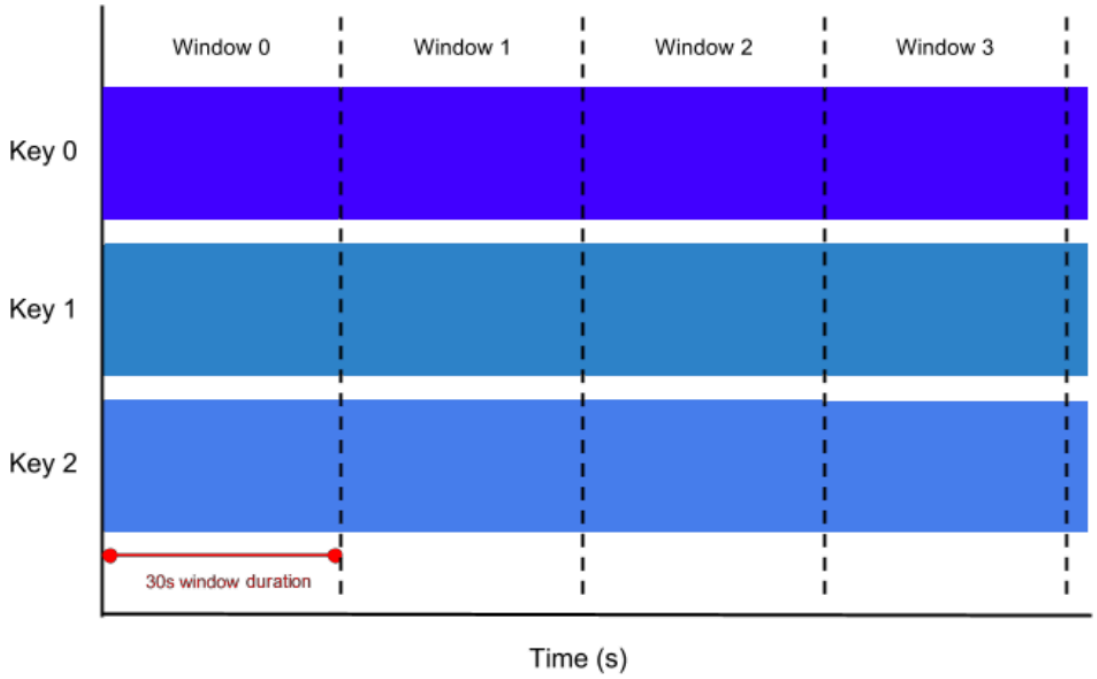
```
GROUP BY TUMBLE(rowtime, INTERVAL '1' HOUR), username
```

8.9.2 滚动窗口

本文为您介绍如何使用实时计算滚动窗口函数。

什么滚动窗口

滚动窗口（TUMBLE）将每个元素分配到一个指定大小的窗口中。通常滚动窗口有一个固定的大小，并且不会出现重叠。例如：如果指定了一个5分钟大小的滚动窗口，无限流的数据会根据时间划分成`[0:00 - 0:05)`、`[0:05, 0:10)`、`[0:10, 0:15)`等窗口。如下图，展示了一个大小划分为30秒的滚动窗口。



语法

TUMBLE函数用在GROUP BY子句中，用来定义滚动窗口。

```
TUMBLE(<time-attr>, <size-interval>)
<size-interval>: INTERVAL 'string' timeUnit
```

说明:

`<time-attr>` 参数必须是流中的一个合法的时间属性字段，指定为Processing Time或Event Time。 请参见#unique_218，了解如何定义时间属性和Watermark。

标识函数

使用标识函数选出窗口的起始时间或者结束时间，窗口的时间属性用于下级Window的聚合。

窗口标识函数	返回类型	描述
TUMBLE_START(time-attr, size-interval)	TIMESTAMP	返回窗口的起始时间（包含边界）。如[00:10, 00:15)的窗口，返回00:10。
TUMBLE_END(time-attr, size-interval)	TIMESTAMP	返回窗口的结束时间（包含边界）。如[00:00, 00:15]的窗口，返回00:15。
TUMBLE_ROWTIME(time-attr, size-interval)	TUMBLE_ROWTIME(time-attr, size-interval)	返回窗口的结束时间（不包含边界）。如[00:00, 00:15]的窗口，返回00:14:59.999。返回值是一个 rowtime attribute，即可以基于该字段做时间属性的操作，如级联窗口。只能用在基于event time的window 上。
TUMBLE_PROCTIME(time-attr, size-interval)	TUMBLE_ROWTIME(time-attr, size-interval)	返回窗口的结束时间（不包含边界）。如[00:00, 00:15]的窗口，返回00:14:59.999。返回值是一个 proctime attribute，即可以基于该字段做时间属性的操作，如级联窗口。 只能用在基于processing time 的window上。

示例1

以下示例为使用event time统计每个用户每分钟在指定网站的点击数。

- 测试数据

username （VARCHAR）	click_url （VARCHAR）	ts （TIMESTAMP）
Jark	*http://taobao.com/xxx*	2017-10-10 10:00:00.0
Jark	*http://taobao.com/xxx*	2017-10-10 10:00:10.0
Jark	*http://taobao.com/xxx*	2017-10-10 10:00:49.0
Jark	*http://taobao.com/xxx*	2017-10-10 10:01:05.0
Jark	*http://taobao.com/xxx*	2017-10-10 10:01:58.0
Timo	*http://taobao.com/xxx*	2017-10-10 10:02:10.0

- 测试语句

```
CREATE TABLE user_clicks(
username varchar,
click_url varchar,
ts timeStamp,
WATERMARK wk FOR ts as withOffset(ts, 2000) -- 为rowtime定义watermark
) with (
type='datahub',
...
);

CREATE TABLE tumble_output(
window_start TIMESTAMP,
window_end TIMESTAMP,
username VARCHAR,
clicks BIGINT
) with (
type='RDS'
);

INSERT INTO tumble_output
SELECT
TUMBLE_START(ts, INTERVAL '1' MINUTE),
TUMBLE_END(ts, INTERVAL '1' MINUTE),
username,
COUNT(click_url)
FROM user_clicks
GROUP BY TUMBLE(ts, INTERVAL '1' MINUTE), username
```

- 测试结果

window_start（TIMESTAMP）	window_end（TIMESTAMP）	username（VARCHAR）	clicks（BIGINT）
2017-10-10 10:00:00.0	2017-10-10 10:01:00.0	Jark	3
2017-10-10 10:01:00.0	2017-10-10 10:02:00.0	Jark	2
2017-10-10 10:02:00.0	2017-10-10 10:03:00.0	Timo	1

示例2

以下示例为使用processing time统计每个用户每分钟在指定网站的点击数。

- 测试数据

username（VARCHAR）	click_url（VARCHAR）
Jark	*http://taobao.com/xxx*
Jark	*http://taobao.com/xxx*

username （VARCHAR）	click_url （VARCHAR）
Jark	http://taobao.com/xxx
Jark	http://taobao.com/xxx
Jark	http://taobao.com/xxx
Timo	http://taobao.com/xxx

- 测试语句

```
CREATE TABLE window_test (
  username               VARCHAR,
  click_url              VARCHAR,
    ts as PROCTIME()
) WITH (
  type = 'datahu ...c = xxx'
);


CREATE TABLE tumble_output(
window_start TIMESTAMP,
window_end TIMESTAMP,
username VARCHAR,
clicks BIGINT
) with (
type='print'
);

INSERT INTO tumble_output
SELECT
TUMBLE_START(ts, INTERVAL '1' MINUTE),
TUMBLE_END(ts, INTERVAL '1' MINUTE),
username,
COUNT(click_url)
FROM window_test
GROUP BY TUMBLE(ts, INTERVAL '1' MINUTE), username
```

- 测试结果

window_start （TIMESTAMP）	window_end （TIMESTAMP）	username （VARCHAR）	clicks （BIGINT）
2019-04-11 14:43:00.000	2019-04-11 14:44:00.000	Jark	5
2019-04-11 14:43:00.000	2019-04-11 14:44:00.000	Timo	1

8.9.3 滑动窗口

本文为您介绍如何使用实时计算滑动窗口函数。

说明:

实时计算滑动窗口（HOP）暂不支持与LAST_VALUE、FIRST_VALUE或TopN函数共同使用。

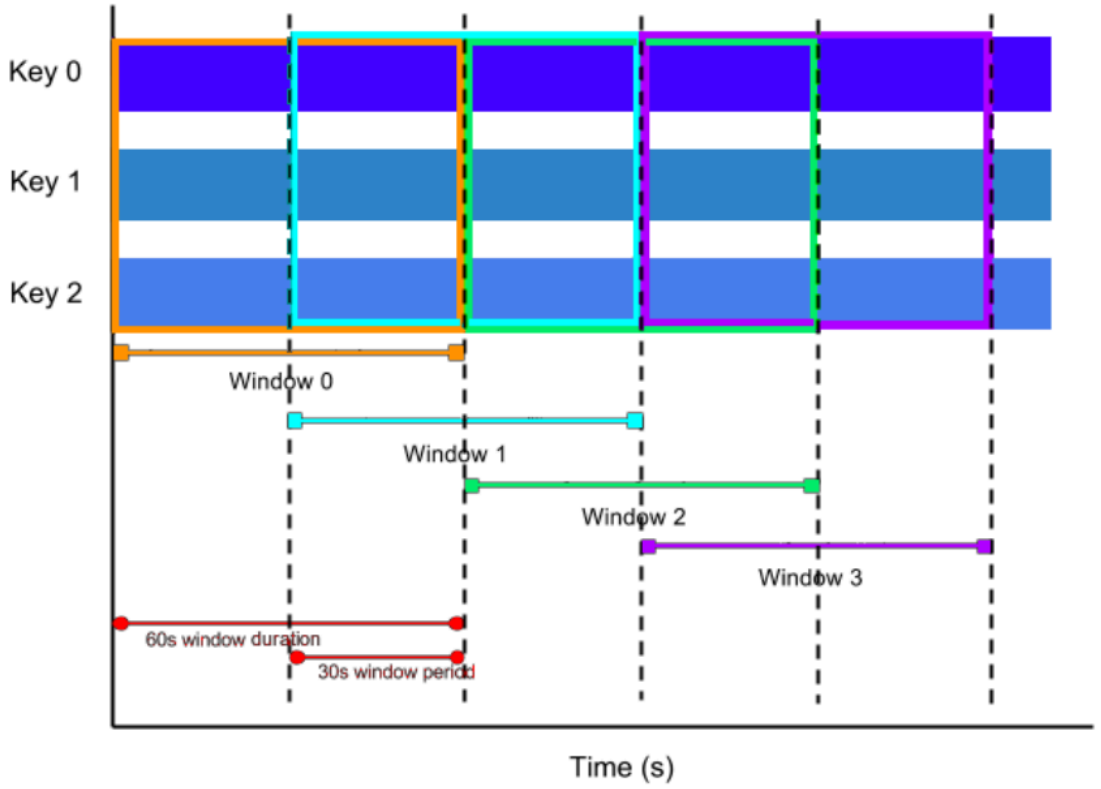
什么是滑动窗口

滑动窗口（HOP），也被称作Sliding Window。不同于滚动窗口，滑动窗口的窗口可以重叠。

滑动窗口有两个参数：slide和size。slide为每次滑动的步长，size为窗口的大小。

- `slide < size`，则窗口会重叠，每个元素会被分配到多个窗口。
- `slide = size`，则等同于滚动窗口（TUMBLE）。
- `slide > size`，则为跳跃窗口，窗口之间不重叠且有间隙。

通常情况下大部分元素符合多个窗口情景，窗口是重叠的。因此，滑动窗口在计算移动平均数（moving averages）时很实用。例如，计算过去5分钟数据的平均值，每10秒钟更新一次，可以设置slide为10秒，size为5分钟。下图为您展示间隔为30秒，窗口大小为1分钟的滑动窗口。



滑动窗口函数语法

HOP函数用在group by子句中，用来定义滑动窗口。

```
HOP(<time-attr>, <slide-interval>,<size-interval>)
<slide-interval>: INTERVAL 'string' timeUnit
<size-interval>: INTERVAL 'string' timeUnit
```

说明:

`<time-attr>` 参数必须是流中的一个合法的时间属性字段，指定为Processing Time或Event Time。 请参见#unique_218，了解如何定义#unique_87和#unique_117。

滑动窗口标识函数

使用滑动窗口标识函数选出窗口的起始时间或者结束时间，窗口的时间属性用于下级Window的聚合。

窗口标识函数	返回类型	描述
`HOP_START (<time-attr>, <slide-interval>, <size-interval>)`	TIMESTAMP	返回窗口的起始时间（包含边界）。如`[00:10, 00:15)`的窗口，返回`00:10` 。
`HOP_END (<time-attr>, <slide-interval>, <size-interval>)`	TIMESTAMP	返回窗口的结束时间（包含边界）。如`[00:00, 00:15)`的窗口，返回`00:15`。
`HOP_ROWTIME (<time-attr>, <slide-interval>, <size-interval>)`	TIMESTAMP（rowtime-attr）	返回窗口的结束时间（不包含边界）。如`[00:00, 00:15)`的窗口，返回`00:14:59.999`。返回值是一个rowtime attribute，也就是可以基于该字段做时间类型的操作，如#unique_220/unique_220_Connect_42_section_cwf_只能用在基于event time的window上。
`HOP_PROCTIME (<time-attr>, <slide-interval>, <size-interval>)`	TIMESTAMP（rowtime-attr）	返回窗口的结束时间（不包含边界）。如`[00:00, 00:15)`的窗口，返回`00:14:59.999` 。返回值是一个proctime attribute，也就是可以基于该字段做时间类型的操作，如#unique_220/unique_220_Connect_42_section_cwf_只能用在基于 processing time的window 上。

示例

统计每个用户过去1分钟的点击次数，每30秒更新1次。即1分钟的窗口，30秒滑动1次。

- 测试数据

username（VARCHAR）	click_url（VARCHAR）	ts（TIMESTAMP）
Jark	http://taobao.com/xxx	2017-10-10 10:00:00.0
Jark	http://taobao.com/xxx	2017-10-10 10:00:10.0
Jark	http://taobao.com/xxx	2017-10-10 10:00:49.0
Jark	http://taobao.com/xxx	2017-10-10 10:01:05.0
Jark	http://taobao.com/xxx	2017-10-10 10:01:58.0
Timo	http://taobao.com/xxx	2017-10-10 10:02:10.0

- 测试语句

```
CREATE TABLE user_clicks (
    username VARCHAR,
    click_url VARCHAR,
    ts TIMESTAMP,
    WATERMARK wk FOR ts AS WITHOFFSET (ts, 2000)--为rowtime定义
Watermark。
) WITH ( TYPE = 'datahub',
         ...);
CREATE TABLE hop_output (
    window_start TIMESTAMP,
    window_end TIMESTAMP,
    username VARCHAR,
    clicks BIGINT
) WITH (TYPE = 'rds',
         ...);
INSERT INTO
    hop_output
SELECT
    HOP_START (ts, INTERVAL '30' SECOND, INTERVAL '1' MINUTE),
    HOP_END (ts, INTERVAL '30' SECOND, INTERVAL '1' MINUTE),
    username,
    COUNT (click_url)
FROM
    user_clicks
GROUP BY
    HOP (ts, INTERVAL '30' SECOND, INTERVAL '1' MINUTE),
    username
```

- 测试结果

window_start （TIMESTAMP）	window_end （TIMESTAMP）	username （VARCHAR）	clicks （BIGINT）
2017-10-10 09:59:30.0	2017-10-10 10:00:30.0	Jark	2
2017-10-10 10:00:00.0	2017-10-10 10:00:00.0	Jark	3

window_start （TIMESTAMP）	window_end （TIMESTAMP）	username （VARCHAR）	clicks （BIGINT）
2017-10-10 10:00:30.0	2017-10-10 10:01:30.0	Jark	2
2017-10-10 10:01:00.0	2017-10-10 10:02:00.0	Jark	2
2017-10-10 10:01:30.0	2017-10-10 10:02:30.0	Jark	1
2017-10-10 10:01:30.0	2017-10-10 10:02:30.0	Timo	1
2017-10-10 10:02:00.0	2017-10-10 10:03:00.0	Timo	1

 说明:

HOP窗口无法读取数据进入的时间，第一个窗口的开启时间会前移。前移时长=窗口时长-滑动步长，示例如下表。

窗口时长（秒）	滑动步长（秒）	Event Time	第一个窗口StartTime	第一个窗口EndTime
120	30	2019-07-31 10:00:00.0	2019-07-31 09:58:30.0	2019-07-31 10:00:30.0
60	10	2019-07-31 10:00:00.0	2019-07-31 09:59:10.0	2019-07-31 10:00:10.0

8.9.4 会话窗口

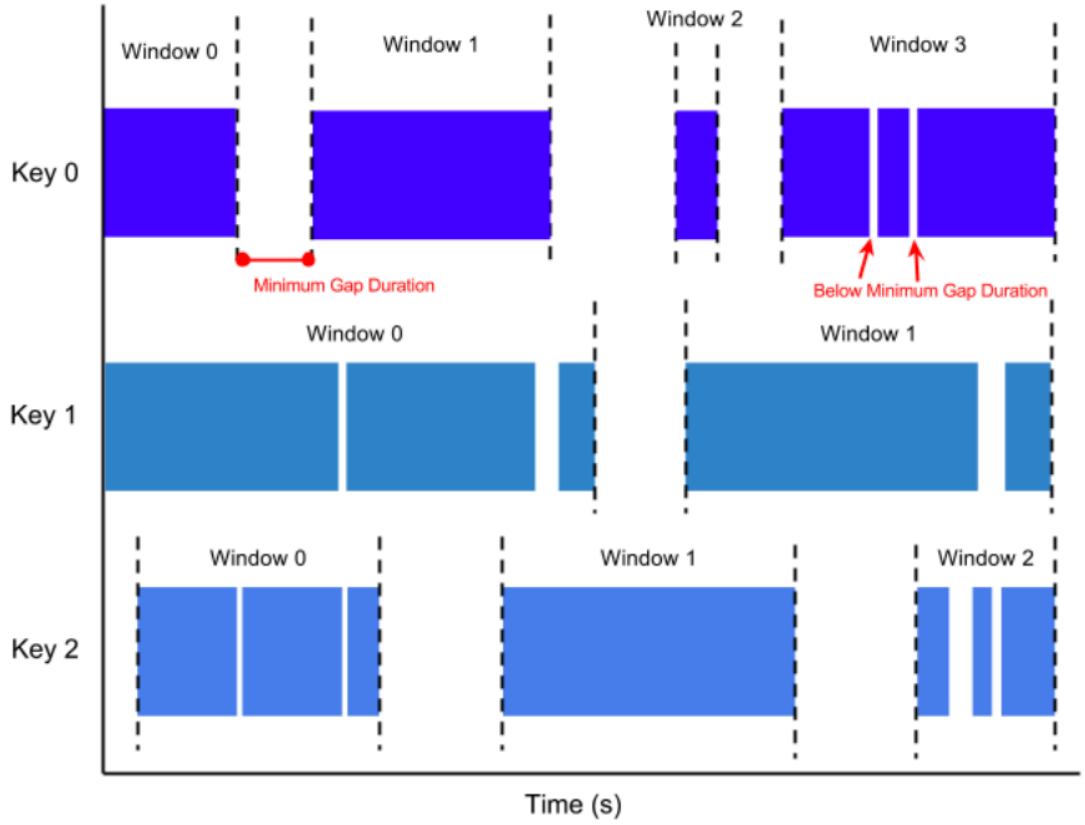
本文为您介绍如何使用实时计算会话窗口函数。

什么是会话窗口

会话窗口（SESSION）通过Session活动来对元素进行分组。会话窗口与滚动窗口和滑动窗口相比，没有窗口重叠，没有固定窗口大小。相反，当它在一个固定的时间周期内不再收到元素，即会话断开时，这个窗口就会关闭。

会话窗口通过一个间隔时间（Gap）来配置，这个间隔定义了非活跃周期的长度。例如，一个表示鼠标点击活动的数据流可能具有长时间的空闲时间，并在其间散布着高浓度的点击。 如果数据在最短指定的间隔持续时间之后到达，则会开始一个新的窗口。

下图展示了一个会话窗口，注意每个Key由于不同的数据分布有不同的Window。



会话窗口函数语法

SESSION函数用在GROUP BY子句中定义会话窗口。

```
SESSION(<time-attr>, <gap-interval>)
<gap-interval>: INTERVAL 'string' timeUnit
```

说明:

`<time-attr>` 参数必须是流中的一个合法的时间属性字段，指定为Processing Time或Event Time。 请参见#unique_218，了解如何定义时间属性和Watermark。

会话窗口标识函数

使用标识函数选出窗口的起始时间或者结束时间，窗口的时间属性用于下级Window的聚合。

窗口标识函数	返回类型	描述
`SESSION_START (<time-attr>, <gap-interval>)`	Timestamp	返回窗口的起始时间（包含边界）。如`[00:10, 00:15)`的窗口，返回`00:10`。

窗口标识函数	返回类型	描述
SESSION_END (<time-attr>, <gap-interval>)	Timestamp	返回窗口的结束时间（包含边界）。如[00:00, 00:15)的窗口，返回00:15。
SESSION_ROWTIME (<time-attr>, <gap-interval>)	Timestamp（rowtime-attr）	返回窗口的结束时间（不包含边界）。如[00:00, 00:15)的窗口，返回00:14:59.999 。返回值是一个 rowtime attribute，也就是可以基于该字段做时间类型的操作，如级联窗口。只能用在基于event time 的window 上。
SESSION_PROCTIME (<time-attr>, <gap-interval>)	Timestamp（rowtime-attr）	返回窗口的结束时间（不包含边界）。如[00:00, 00:15）的窗口，返回00:14:59.999。返回值是一个proctime attribute，也就是可以基于该字段做时间类型的操作，如级联窗口。只能用在基于 processing time的window 上。

示例

统计每个用户在每个活跃会话期间的点击次数，会话超时时间30秒。

- 测试数据

username （VARCHAR）	click_url （VARCHAR）	ts （TIMESTAMP）
Jark	http://taobao.com/xxx	2017-10-10 10:00:00.0
Jark	http://taobao.com/xxx	2017-10-10 10:00:10.0
Jark	http://taobao.com/xxx	2017-10-10 10:00:49.0
Jark	http://taobao.com/xxx	2017-10-10 10:01:05.0
Jark	http://taobao.com/xxx	2017-10-10 10:01:58.0
Timo	http://taobao.com/xxx	2017-10-10 10:02:10.0

- 测试语句

```
CREATE TABLE user_clicks(
username varchar,
click_url varchar,
ts timeStamp,
```

```
WATERMARK wk FOR ts as withOffset(ts, 2000) -- 为rowtime定义watermark
) with (
type='datahub',
...
);

CREATE TABLE session_output(
window_start TIMESTAMP,
window_end TIMESTAMP,
username VARCHAR,
clicks BIGINT
) with (
type='rds'
);

INSERT INTO session_output
SELECT
SESSION_START(ts, INTERVAL '30' SECOND),
SESSION_END(ts, INTERVAL '30' SECOND),
username,
COUNT(click_url)
FROM user_clicks
GROUP BY SESSION(ts, INTERVAL '30' SECOND), username
```

- 测试结果

window_start （TIMESTAMP）	window_end （TIMESTAMP）	username （VARCHAR）	clicks （BIGINT）
2017-10-10 10:00:00.0	2017-10-10 10:00:40.0	Jark	2
2017-10-10 10:00:49.0	2017-10-10 10:01:35.0	Jark	2
2017-10-10 10:01:58.0	2017-10-10 10:02:28.0	Jark	1
2017-10-10 10:02:10.0	2017-10-10 10:02:40.0	Timo	1

8.9.5 OVER窗口

OVER窗口（OVER Window）是传统数据库的标准开窗，OVER Window不同于Group By Window，OVER Window中每1个元素都对应1个窗口。窗口元素是与当前元素相邻的元素集合，流上元素会在多个窗口中。在Flink SQL Window的实现中，每个触发计算的元素所确定的行，都是该元素所在窗口的最后一行。

在应用OVER Window的流式数据中，每1个元素都对应1个OVER Window。每1个元素都触发一次数据计算。在实时计算的底层实现中，OVER Window的数据进行全局统一管理（数据只存储一份），逻辑上为每1个元素维护1个OVER Window，为每1个元素进行窗口计算，完成计算后会清除过期的数据。

语法

```
SELECT
    agg1(col1) OVER (definition1) AS colName,
    ...
    aggN(colN) OVER (definition1) AS colNameN
FROM Tab1
```

说明:

- agg1到aggN所对应的OVER definition1必须相同。
- AS的别名可供外层SQL进行查询。

类型

Flink SQL中对OVER Window的定义遵循标准SQL的定义语法，传统OVER Window没有对其进行更细粒度的窗口类型命名划分。本节为了方便您理解OVER Window的语义，将OVER Window按照计算行的定义方式划分为如下两类。

- ROWS OVER Window：每一行元素都视为新的计算行，即每一行都是一个新的窗口。
- RANGE OVER Window：具有相同时间值的所有元素行视为同一计算行，即具有相同时间值的所有行都是同一个窗口。

属性

正交属性	proctime	eventtime
rows	√	√
range	√	√

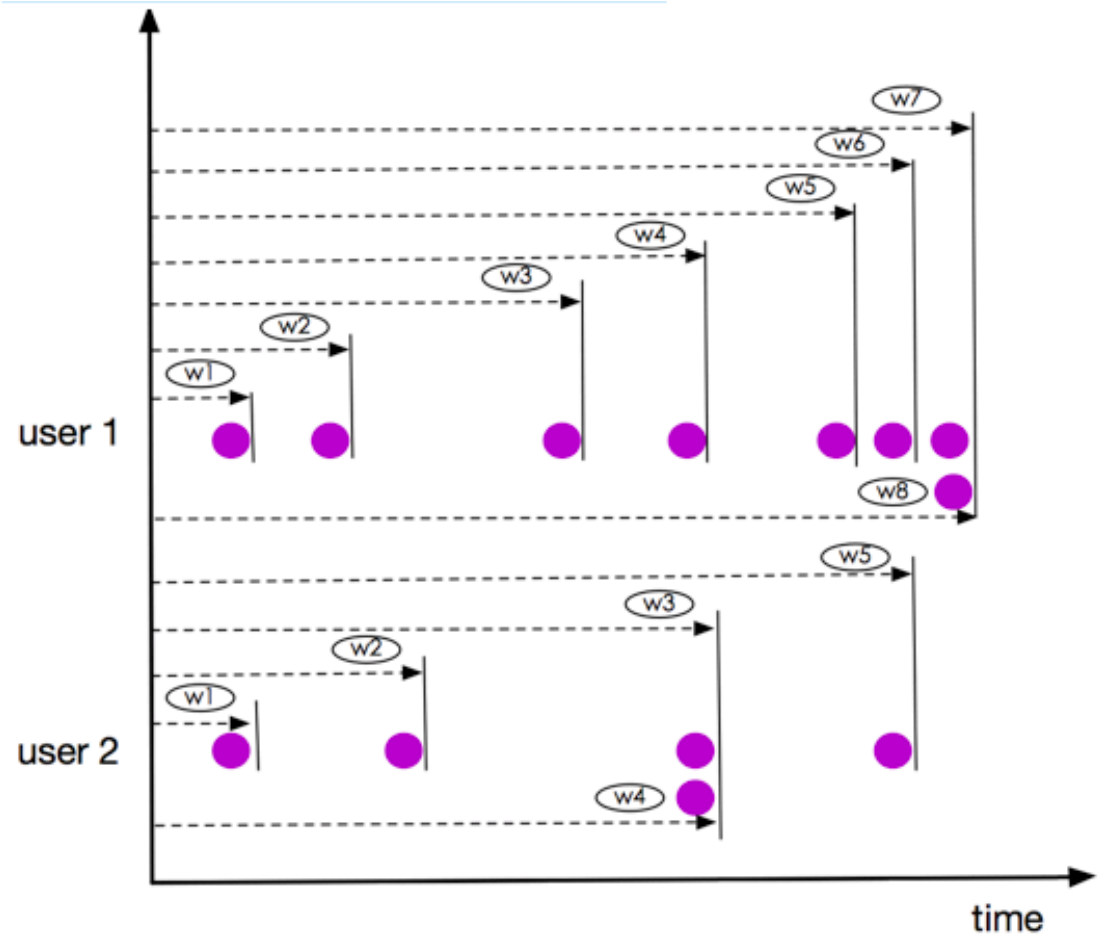
- rows：按照实际元素的行进行确定窗口。
- range：按照实际的元素值（时间戳值）进行确定窗口。

Rows OVER Window语义

- 窗口数据

  ROWS OVER Window的每个元素都确定一个窗口。ROWS OVER Window也有Unbounded和Bounded的两种情况。

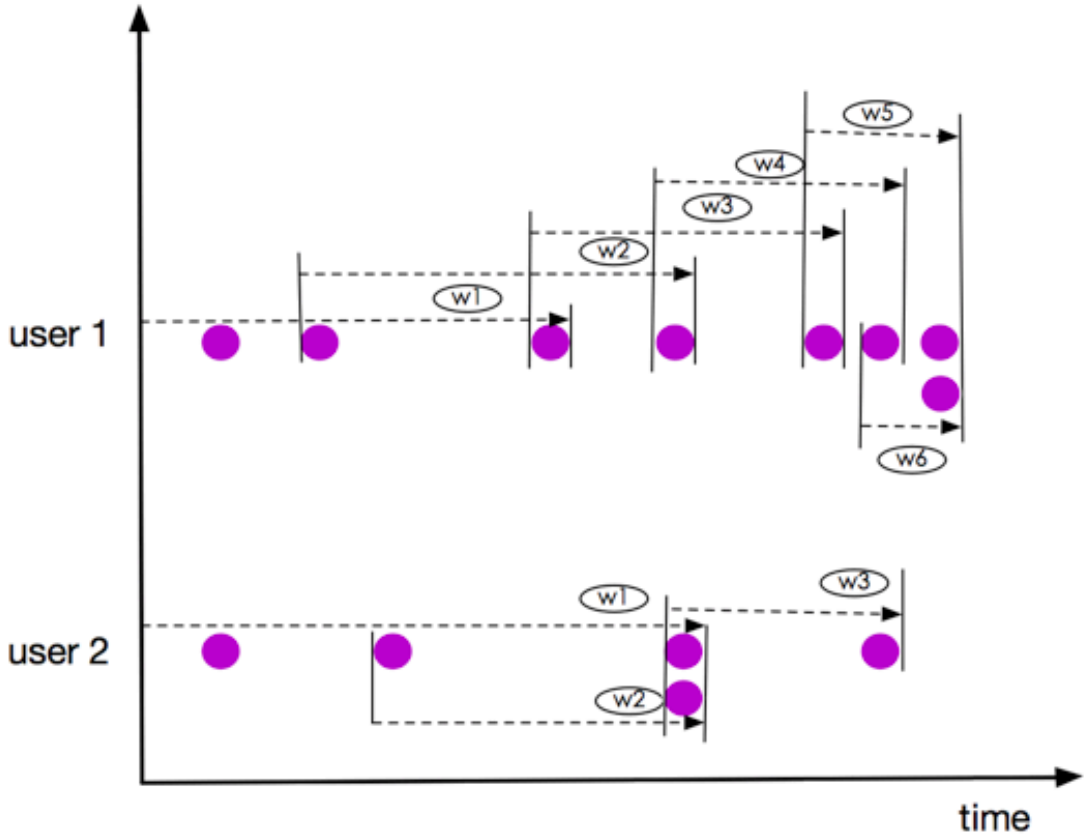
  Unbounded ROWS OVER Window数据如下图所示。



说明:

上图所示窗口user1的w7和w8，user2的窗口w3和w4，虽然元素都是同一时刻到达，但是它们仍然是在不同的窗口，这一点有别于RANGE OVER Window。

Bounded ROWS OVER Window数据以3个元素（2 PRECEDING）的窗口为例，如下图所示。



说明:

上图所示窗口user1的w5和w6，user2的窗口w2和w3，虽然有元素都是同一时刻到达，但是它们仍然是在不同的窗口，这一点有别于RANGE OVER Window。

- 窗口语法

```
SELECT
    agg1(col1) OVER(
     [PARTITION BY (value_expression1,..., value_expressionN)]
     ORDER BY timeCol
     ROWS
     BETWEEN (UNBOUNDED | rowCount) PRECEDING AND CURRENT ROW) AS
colName, ...
```

```
FROM Tab1
```

- value_expression：分区值表达式。
- timeCol：用于元素排序的时间字段。
- rowCount：是定义根据当前行开始向前追溯几行元素。

· 案例

以Bounded ROWS OVER Window场景示例，假设，有一张商品上架表，包含有商品ID、商品类型、商品上架时间、商品价格数据。假设，求在当前商品上架之前同类的3个商品中的最高价格。

测试数据

商品ID	商品类型	上架时间	销售价格
ITEM001	Electronic	2017-11-11 10:01:00	20
ITEM002	Electronic	2017-11-11 10:02:00	50
ITEM003	Electronic	2017-11-11 10:03:00	30
ITEM004	Electronic	2017-11-11 10:03:00	60
ITEM005	Electronic	2017-11-11 10:05:00	40
ITEM006	Electronic	2017-11-11 10:06:00	20
ITEM007	Electronic	2017-11-11 10:07:00	70
ITEM008	Clothes	2017-11-11 10:08:00	20

测试代码

```
CREATE TABLE tmall_item(
   itemID VARCHAR,
   itemType VARCHAR,
   onSellTime TIMESTAMP,
   price DOUBLE,
   WATERMARK onSellTime FOR onSellTime as withOffset(onSellTime, 0)
)
WITH (
  type = 'sls',
```

```
   ...
) ;

SELECT
    itemID,
    itemType,
    onSellTime,
    price,
    MAX(price) OVER (
        PARTITION BY itemType
        ORDER BY onSellTime
        ROWS BETWEEN 2 preceding AND CURRENT ROW) AS maxPrice
  FROM tmall_item
```

测试结果

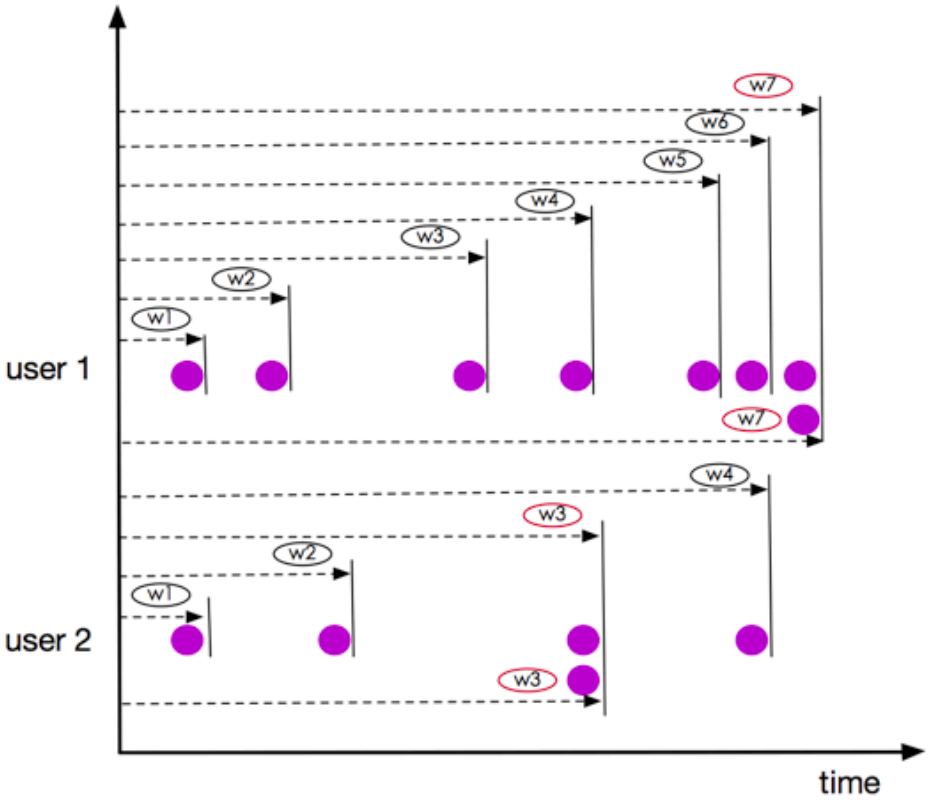
itemID	itemType	onSellTime	price	maxPrice
ITEM001	Electronic	2017-11-11 10:01:00	20	20
ITEM002	Electronic	2017-11-11 10:02:00	50	50
ITEM003	Electronic	2017-11-11 10:03:00	30	50
ITEM004	Electronic	2017-11-11 10:03:00	60	60
ITEM005	Electronic	2017-11-11 10:05:00	40	60
ITEM006	Electronic	2017-11-11 10:06:00	20	60
ITEM007	Electronic	2017-11-11 10:07:00	70	70
ITEM008	Clothes	2017-11-11 10:08:00	20	20

RANGE OVER Window语义

- 窗口数据

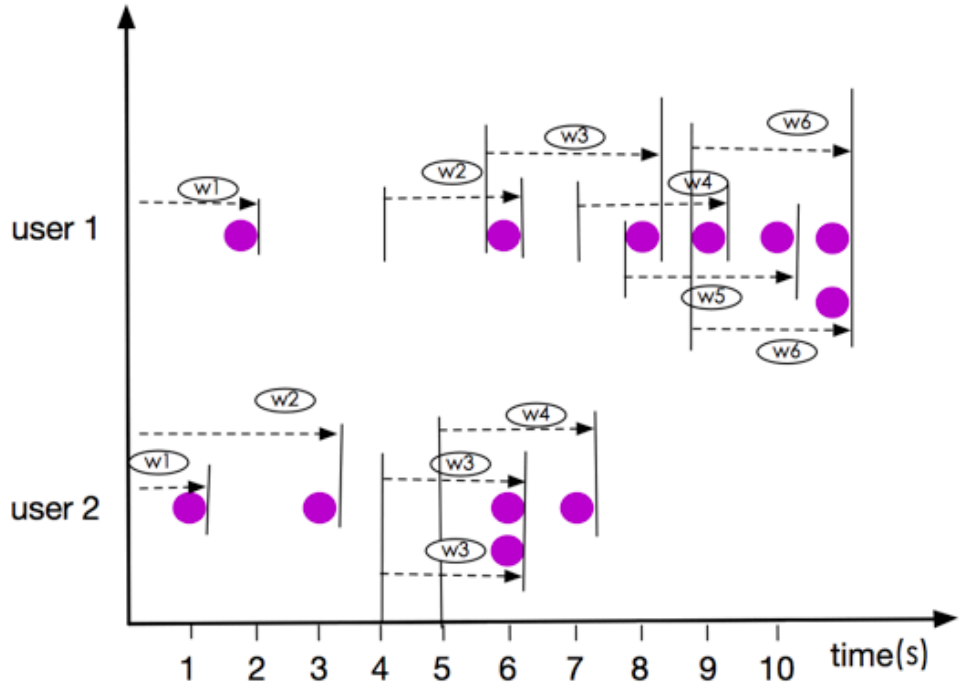
  RANGE OVER Window所有具有共同元素值（元素时间戳）的元素行确定一个窗口，RANGE OVER Window也有Unbounded和Bounded的两种情况。

  Unbounded RANGE OVER Window 数据如下图所示。



> 注意: 上图所示窗口user1的w7， user2的窗口w3，两个元素同一时刻到达，他们属于相同的window，这一点有别于ROWS OVER Window。

Bounded RANGE OVER Window 数据，以3秒中数据(`INTERVAL '2' SECOND`)的窗口为例，如下图所示。



说明:

上图所示窗口user1的w6， user2的窗口w3，元素都是同一时刻到达，他们属于相同的window，这一点有别于ROWS OVER Window。

- **窗口语法**

```
SELECT
    agg1(col1) OVER(
     [PARTITION BY (value_expression1,..., value_expressionN)]
     ORDER BY timeCol
     RANGE
     BETWEEN (UNBOUNDED | timeInterval) PRECEDING AND CURRENT ROW)
AS colName,
...
FROM Tab1
```

  - **value_expression：进行分区的字表达式。**
  - **timeCol：用于元素排序的时间字段。**
  - **timeInterval：是定义根据当前行开始向前追溯指定时间的元素行。**

- 案例

  以Bounded RANGE OVER Window场景示例，假设有一张商品上架表，包含有商品ID、商品类型、商品上架时间、商品价格数据。假设求比当前商品上架时间早2分钟的同类商品中的最高价格。

  测试数据

商品ID	商品类型	上架时间	销售价格
ITEM001	Electronic	2017-11-11 10:01:00	20
ITEM002	Electronic	2017-11-11 10:02:00	50
ITEM003	Electronic	2017-11-11 10:03:00	30
ITEM004	Electronic	2017-11-11 10:03:00	60
ITEM005	Electronic	2017-11-11 10:05:00	40
ITEM006	Electronic	2017-11-11 10:06:00	20
ITEM007	Electronic	2017-11-11 10:07:00	70
ITEM008	Clothes	2017-11-11 10:08:00	20

  测试代码

```
CREATE TABLE tmall_item(
   itemID VARCHAR,
   itemType VARCHAR,
   onSellTime TIMESTAMP,
   price DOUBLE,
   WATERMARK onSellTime FOR onSellTime as withOffset(onSellTime, 0)
)
WITH (
  type = 'sls',
   ...
) ;

SELECT
    itemID,
    itemType,
    onSellTime,
    price,
```

```
    MAX(price) OVER (
        PARTITION BY itemType
        ORDER BY onSellTime
        RANGE BETWEEN INTERVAL '2' MINUTE preceding AND CURRENT ROW
) AS maxPrice
  FROM tmall_item
```

测试结果

itemID	itemType	onSellTime	price	maxPrice
ITEM001	Electronic	2017-11-11 10:01:00	20	20
ITEM002	Electronic	2017-11-11 10:02:00	50	50
ITEM003	Electronic	2017-11-11 10:03:00	30	50
ITEM004	Electronic	2017-11-11 10:03:00	60	60
ITEM005	Electronic	2017-11-11 10:05:00	40	60
ITEM006	Electronic	2017-11-11 10:06:00	20	40
ITEM007	Electronic	2017-11-11 10:07:00	70	70
ITEM008	Clothes	2017-11-11 10:08:00	20	20

8.10 逻辑函数

8.10.1 =

本文为您介绍如何使用实时计算逻辑运算函数=。

语法

```
A = B
```

入参

参数	数据类型
A	INT

参数	数据类型
B	INT

功能描述

如果A等于B，返回TRUE，否则返回FALSE。

示例

- 测试数据

int1(INT)	int2(INT)	int3(INT)
97	65	65

- 测试语句

```
SELECT int1 as aa
FROM T1
WHERE int3 = int2;
```

- 测试结果

aa(int)
97

8.10.2 >

本文为您介绍如何使用实时计算逻辑运算函数>。

语法

```
A > B
```

入参

参数	数据类型
A	INT
B	INT

功能描述

如果A大于B，返回TRUE，否则返回FALSE。

示例

- **测试数据**

int1(INT)	int2(INT)	int3(INT)
97	65	100

- **测试语句**

```
SELECT int1 as aa
FROM T1
WHERE int3 > int2;
```

- **测试结果**

aa(int)
97

8.10.3 >=

本文为您介绍如何使用实时计算逻辑运算函数>=。

语法

```
A >= B
```

入参

参数	数据类型
A	INT
B	INT

功能描述

如果A大于等于B，返回TRUE，否则返回FALSE。

示例

- **测试数据**

int1(INT)	int2(INT)	int3(INT)
97	65	65
9	6	61

- **测试语句**

```
SELECT int1 as aa
FROM T1
```

```
WHERE int3 >= int2;
```

- 测试结果

aa(int)
97
9

8.10.4 <=

本文为您介绍如何使用实时计算逻辑运算函数<=。

语法

```
A <= B
```

入参

参数	数据类型
A	INT
B	INT

功能描述

如果A小于等于B，返回TRUE，否则返回FALSE。

示例

- 测试数据

int1(INT)	int2(INT)	int3(INT)
97	66	65
9	6	5

- 测试语句

```
SELECT int1 as aa
FROM T1
WHERE int3 <= int2;
```

- 测试结果

aa(int)
97
9

8.10.5 <

本文为您介绍如何使用实时计算逻辑运算函数<。

语法

```
A < B
```

入参

参数	数据类型
A	INT
B	INT

功能描述

如果A小于B，返回TRUE，否则返回FALSE。

示例

- 测试数据

int1(INT)	int2(INT)	int3(INT)
97	66	65
9	6	5

- 测试语句

```
SELECT int1 as aa
FROM T1
WHERE int3 < int2;
```

- 测试结果

aa(int)
97
9

8.10.6 <>

本文为您介绍如何使用实时计算逻辑运算函数<>。

语法

```
A <> B
```

入参

参数	数据类型
A	INT
B	INT

功能描述

如果A不等于B，返回TRUE，否则返回FALSE。

示例

- 测试数据

int1(INT)	int2(INT)	int3(INT)
97	66	6

- 测试语句

```
SELECT int1 as aa
FROM T1
WHERE int3 <> int2;
```

- 测试结果

aa(int)
97

8.10.7 AND

本文为您介绍如何使用实时计算逻辑运算函数AND。

语法

```
A AND B
```

入参

参数	数据类型
A	BOOLEAN

参数	数据类型
B	BOOLEAN

功能描述

如果A和B均为TRUE，则为TRUE，否则为FALSE。

示例

- 测试数据

int1(INT)	int2(INT)	int3(INT)
255	97	65

- 测试语句

```
SELECT int2 as aa
FROM T1
WHERE int1=255 AND int3=65;
```

- 测试结果

aa(int)
97

8.10.8 BETWEEN AND

本文为您介绍如何使用实时计算逻辑运算函数BETWEEN AND。

语法

```
A BETWEEN AND B
```

入参

参数	数据类型
A	DOUBLE，BIGINT，INT，VARCHAR，DATE，TIMESTAMP，TIME
B	DOUBLE，BIGINT，INT，VARCHAR，DATE，TIMESTAMP，TIME
C	DOUBLE，BIGINT，INT，VARCHAR，DATE，TIMESTAMP，TIME

功能描述

BETWEEN操作符用于选取介于两个值之间的数据范围内的值。

示例1

- **测试数据**

int1(INT)	int2(INT)	int3(INT)
90	80	100
11	10	7

- **测试语句**

```
SELECT int1 as aa
FROM T1
WHERE int1 BETWEEN int2 AND int3;
```

- **测试结果**

aa(int)
90

示例2

- **测试数据**

var1(varchar)	var2(varchar)	var3(varchar)
b	a	c

- **测试语句**

```
SELECT var1 as aa
FROM T1
WHERE var1 BETWEEN var2 AND var3;
```

- **测试结果**

aa(varchar)
b

示例3

- **测试数据**

TIMESTAMP1(TIMESTAMP)	TIMESTAMP2(TIMESTAMP)	TIMESTAMP3(TIMESTAMP)
1969-07-20 20:17:30	1969-07-20 20:17:20	1969-07-20 20:17:45

- 测试语句

```
SELECT TIMESTAMP1 as aa
FROM T1
WHERE TIMESTAMP1 BETWEEN TIMESTAMP2 AND TIMESTAMP3;
```

- 测试结果

aa(TIMESTAMP)
1969-07-20 20:17:30

8.10.9 IS NOT FALSE

本文为您介绍如何使用实时计算逻辑运算函数IS NOT FALSE。

语法

```
A IS NOT FALSE
```

入参

参数	数据类型
A	BOOLEAN

功能描述

如果A是TRUE，返回TRUE。如果A是FALSE，返回FALSE。

示例

- 测试数据

int1(INT)	int2(INT)
255	97

- 测试语句

```
SELECT int2 as aa
FROM T1
WHERE int1=255 IS NOT FALSE;
```

- 测试结果

aa(int)
97

8.10.10 IS NOT NULL

本文为您介绍如何使用实时计算逻辑运算函数IS NOT NULL。

语法

```
value IS NOT NULL
```

入参

参数	数据类型
value	任意数据类型

功能描述

如果 value为NULL，返回FALSE，否则返回TRUE。

示例

- 测试数据

int1(INT)	int2(VARCHAR)
97	无
9	ww123

- 测试语句

```
SELECT int1 as aa
FROM T1
WHERE int2 IS NOT NULL;
```

- 测试结果

aa(int)
9

8.10.11 IS NOT TRUE

本文为您介绍如何使用实时计算逻辑运算函数IS NOT TRUE。

语法

```
A IS NOT TRUE
```

入参

参数	数据类型
A	BOOLEAN

功能描述

如果A是TRUE，返回FALSE。如果A是FALSE，返回TRUE。

示例

- 测试数据

int1(INT)	int2(INT)
255	97

- 测试语句

```
SELECT int1 as aa
FROM T1
WHERE int1=25 IS NOT TRUE;
```

- 测试结果

aa(int)
97

8.10.12 IS NOT UNKNOWN

本文为您介绍如何使用实时计算逻辑运算函数IS NOT UNKNOWN。

语法

```
A IS NOT UNKNOWN
```

入参

参数	数据类型
A	BOOLEAN

功能描述

A为逻辑比较表达式，例如： 6<8。

正常情况下数值型与数值型作逻辑比较时，A值为TRUE或者FALSE。当其中一个不为数值型数据类型时，就会出现无法比较的情况。`IS NOT UNKNOWN` 就是判断这种情况是否存在。 当两边无法进行正常的逻辑判断时，即A值既不是`TRUE`也不是`FALSE`，返回 `FALSE`。可正常逻辑判断时，即A值为`TRUE`或者`FALSE`，返回 `TRUE` 。

示例一

- **测试数据**

int1(INT)	int2(INT)
255	97

- **测试语句**

```
SELECT int2 as aa
FROM T1
WHERE int1=25 IS NOT UNKNOWN;
```

- **测试结果**

aa(int)
97

示例二

- **测试数据**

int1(INT)	int2(INT)
255	97

- **测试语句**

```
SELECT int2 as aa
FROM T1
WHERE int1 < null IS NOT UNKNOWN;
```

- **测试结果**

aa(int)
空

8.10.13 IS NULL

本文为您介绍如何使用实时计算逻辑运算函数IS NULL。

语法

```
value IS NULL
```

入参

参数	数据类型
value	任意数据类型

功能描述

如果 value为NULL，返回TURE，否则返回FALSE。

示例

- 测试数据

int1(INT)	int2(VARCHAR)
97	无
9	www

- 测试语句

```
SELECT int1 as aa
FROM T1
WHERE int2 IS NULL;
```

- 测试结果

aa(int)
97

8.10.14 IS TRUE

本文为您介绍如何使用实时计算逻辑运算函数IS TURE。

语法

```
A IS TRUE
```

入参

参数	数据类型
A	BOOLEAN

功能描述

如果A是TRUE，返回TURE。如果A是FALSE，返回FALSE。

示例

- 测试数据

int1(INT)	int2(INT)
255	97

- 测试语句

```
SELECT int1 as aa
FROM T1
WHERE int1=255 IS TRUE;
```

- 测试结果

aa(int)
97

8.10.15 IS UNKNOWN

本文为您介绍如何使用实时计算逻辑运算函数IS UNKNOWN。

语法

```
A IS UNKNOWN
```

入参

参数	数据类型
A	BOOLEAN

功能描述

当两边无法进行正常的逻辑判断时，即A(逻辑比较表达式)值既不是TRUE也不是FALSE，返回TRUE。可正常逻辑判断时，即A值为TRUE或者FALSE，返回 FALSE。正常情况下数值型 与数值型作逻辑比较时(例如6<>8)，A值为TRUE或者FALSE。但是，当其中一个不为数值型数据类型时，就会出现无法比较的情况。IS UNKNOWN就是判断这种情况是否存在。

示例一

- 测试数据

int1(INT)	int2(INT)
255	97

- 测试语句

```
SELECT int2 as aa
FROM T1
```

```
WHERE int1=25 IS UNKNOWN;
```

- 测试结果

aa(int)
空

示例二

- 测试数据

int1(INT)	int2(INT)
255	97

- 测试语句

```
SELECT int2 as aa
FROM T1
WHERE int1 > null IS UNKNOWN;
```

- 测试结果

aa(int)
97

8.10.16 LIKE

本文为您介绍如何使用实时计算逻辑运算函数LIKE。

语法

```
A LIKE B
```

入参

参数	数据类型
A	VARCHAR
B	VARCHAR

功能描述

如果匹配，返回TRUE，否则返回FALSE。

说明:

%可用于定义通配符。

示例1

- **测试数据**

int1(INT)	VARCHAR2(VARCHAR)	VARCHAR3(VARCHAR)
90	ss97	97ss
99	ss10	7ho7

- **测试语句**

```
SELECT int1 as aa
FROM T1
WHERE VARCHAR2 LIKE 'ss%';
```

- **测试结果**

aa(int)
90
99

示例2

- **测试数据**

int1(INT)	VARCHAR2(VARCHAR)	VARCHAR3(VARCHAR)
90	ss97	97ss
99	ss10	7ho7

- **测试语句**

```
SELECT int1 as aa
FROM T1
WHERE VARCHAR3 LIKE '%ho%';
```

- **测试结果**

aa(int)
99

8.10.17 NOT

本文为您介绍如何使用实时计算逻辑运算函数NOT。

语法

```
NOT A
```

入参

参数	数据类型
A	BOOLEAN

功能描述

如果A是TRUE，返回FALSE。如果A是FALSE，返回TRUE。

示例

- 测试数据

int2(INT)	int3(INT)
97	65

- 测试语句

```
SELECT int2 as aa
FROM T1
WHERE NOT int3=62;
```

- 测试结果

aa(int)
97

8.10.18 NOT BETWEEN AND

本文为您介绍如何使用实时计算逻辑运算函数NOT BETWEEN AND。

语法

```
A NOT BETWEEN B AND C
```

入参

参数	数据类型
A	DOUBLE、BIGINT、INT、VARCHAR、DATE、TIMESTAMP、TIME

参数	数据类型
B	DOUBLE、BIGINT、INT、VARCHAR、DATE、TIMESTAMP、TIME
C	DOUBLE、BIGINT、INT、VARCHAR、DATE、TIMESTAMP、TIME

功能描述

`NOT BETWEEN AND`操作符用于选取不存在与两个值之间的数据范围内的值。

示例1

- 测试数据

int1(INT)	int2(INT)	int3(INT)
90	97	80
11	10	7

- 测试语句

```
SELECT int1 as aa
FROM T1
WHERE int1 NOT BETWEEN int2 AND int3;
```

- 测试结果

aa(int)
11

示例2

- 测试数据

var1(varchar)	var2(varchar)	var3(varchar)
d	a	c

- 测试语句

```
SELECT int1 as aa
FROM T1
WHERE var1 NOT BETWEEN var2 AND var3;
```

```
```

- 测试结果

aa(varchar)
d

示例3

- 测试数据

TIMESTAMP1(TIMESTAMP)	TIMESTAMP2(TIMESTAMP)	TIMESTAMP3(TIMESTAMP)
1969-07-20 20:17:30	1969-07-20 20:17:40	1969-07-20 20:17:45

- 测试语句

```
SELECT TIMESTAMP1 as aa
FROM T1
WHERE TIMESTAMP1 NOT BETWEEN TIMESTAMP2 AND TIMESTAMP3;
```

- 测试结果

aa(TIMESTAMP)
1969-07-20 20:17:30

8.10.19 OR

本文为您介绍如何使用实时计算逻辑运算函数OR。

语法

```
A OR B
```

入参

参数	数据类型
A	BOOLEAN
B	BOOLEAN

功能描述

如果A和B中至少有一个为TRUE，则为TRUE，否则为FALSE。

示例

- **测试数据**

int1(INT)	int2(INT)	int3(INT)
255	97	65

- **测试语句**

```
SELECT int2 as aa
FROM T1
WHERE int1=255 OR int3=65;
```

- **测试结果**

aa(int)
97

8.10.20 IN

本文为您介绍如何使用实时计算逻辑运算函数IN。

语法

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (value1,value2,...)
```

入参

参数	数据类型
value1	常数
value2	常数

功能描述

用来查找在参数中的记录。

示例

- **测试数据**

id(Int)	LastName(Varchar)
1	Adams
2	Bush
3	Carter

- 测试语句

```
SELECT *
FROM T1
WHERE LastName IN ('Adams','Carter')
```

- 测试结果

id(Int)	LastName(Varchar)
1	Adams
3	Carter

8.10.21 IS DISTINCT FROM

本文为您介绍如何使用实时计算逻辑运算函数IS DISTINCT FROM。

语法

```
A IS DISTINCT FROM B
```

入参

参数	数据类型
A	任意数据类型
B	任意数据类型

功能描述

- A和B的数据类型、值不完全相同则返回TURE。
- A和B的数据类型、值都相同返回FALSE。
- 将空值视为相同。

示例

- 测试数据

A(int)	B(varchar)
97	97
null	sss
null	null

- 测试语句

```
SELECT
A IS DISTINCT FROM B as `result`
```

```
FROM T1
```

- 测试结果

result(BOOLEAN)
true
true
false

8.10.22 IS NOT DISTINCT FROM

本文为您介绍如何使用实时计算逻辑运算函数IS NOT DISTINCT FROM。

语法

```
A IS NOT DISTINCT FROM B
```

入参

参数	数据类型
A	任意数据类型
B	任意数据类型

功能描述

- A和B的数据类型、值不完全相同则返回FALSE。
- A和B的数据类型、值都相同返回TURE。
- 将空值视为相同。

示例

- 测试数据

A(int)	B(varchar)
97	97
null	sss
null	null

- 测试语句

```
SELECT
A IS NOT DISTINCT FROM B as `result`
```

```
FROM T1
```

- 测试结果

result(BOOLEAN)
false
false
true

8.10.23 NOT IN

本文为您介绍如何使用实时计算逻辑运算函数NOT IN。

语法

```
SELECT column_name(s)
FROM table_name
WHERE column_name NOT IN (value1,value2,...)
```

入参

参数	数据类型
value1	常数
value2	常数

功能描述

用来查找在不在参数中的记录。

示例

- 测试数据

id(Int)	LastName(Varchar)
1	Adams
2	Bush
3	Carter

- 测试语句

```
SELECT *
FROM T1
```

```
WHERE LastName NOT IN ('Adams','Carter')
```

- 测试结果

id(Int)	LastName(Varchar)
2	Bush

8.11 内置函数

8.11.1 字符串函数

8.11.1.1 REGEXP_EXTRACT

本文为您介绍如何使用实时计算字符串函数REGEXP_EXTRACT。

语法

```
VARCHAR REGEXP_EXTRACT(VARCHAR str, VARCHAR pattern, INT index)
```

入参

参数	数据类型	说明
str	VARCHAR	指定的字符串。
pattern	VARCHAR	匹配的字符串。
index	INT	第几个被匹配的字符串。

 注意:

正则常量请按照Java代码来写。codegen会将SQL常量字符串自动转化成Java代码。如果要描述一个数字(\d)，需要写成 '\d'，也就是像在Java中写正则一样。

功能描述

使用正则模式pattern匹配抽取字符串str中的第index个子串，index从1开始，正则匹配提取。参数为null或者正则不合法返回null。

示例

- 测试数据

str1 (VARCHAR)	pattern1(VARCHAR)	index1 (INT)
foothebar	foo(.*?)(bar)	2
100-200	(\\d+)-(\\d+)	1
null	foo(.*?)(bar)	2

str1 (VARCHAR)	pattern1(VARCHAR)	index1 (INT)
foothebar	null	2
foothebar	空	2
foothebar	(	2

- 测试语句

```
SELECT  REGEXP_EXTRACT(str1, pattern1, index1) as result
FROM T1
```

- 测试结果

result(VARCHAR)
bar
100
null
null
null
null

8.11.1.2 REGEXP_REPLACE

本文为您介绍如何使用实时计算字符串函数REGEXP_REPLACE。

语法

```
VARCHAR REGEXP_REPLACE(VARCHAR str, VARCHAR pattern, VARCHAR
replacement)
```

入参

参数	数据类型	说明
str	VARCHAR	指定的字符串。
pattern	VARCHAR	被替换的字符串。
replacement	VARCHAR	用于替换的字符串。

注意:

请您按照Java代码编写正则常量。Codegen会自动将SQL常量字符串转化为Java代码。描述一个数值（\d）的正则表达式和Java中一样，为'\d'。

功能描述

用字符串`replacement`替换字符串`str`中正则模式为`pattern`的部分，并返回新的字符串。参数为NULL或者正则不合法返回NULL。

示例

- 测试数据

str1(VARCHAR)	pattern1(VARCHAR)	replace1(VARCHAR)
2014-03-13	-	空
NULL	-	空
2014-03-13	-	NULL
2014-03-13	空	s
2014-03-13	(	s
100-200	(\d+)	num

- 测试语句

```
SELECT  REGEXP_REPLACE(str1, pattern1, replace1) as result
FROM T1
```

- 测试结果

result(VARCHAR)
20140313
null
null
2014-03-13
null
num-num

8.11.1.3 REPEAT

本文为您介绍如何使用实时计算字符串函数REPEAT。

语法

```
VARCHAR REPEAT(VARCHAR str, INT n)
```

入参

参数	数据类型	说明
str	VARCHAR	重复字符串值。
n	INT	重复次数。

功能描述

返回以字符串值为str，重复次数为N的新的字符串。参数为null，则返回null。重复次数为0或负数，则返回空串。

示例

- 测试数据

str(VARCHAR)	n(INT)
J	9
Hello	2
Hello	-9
null	9

- 测试语句

```
SELECT  REPEAT(str,n) as var1
FROM T1
```

- 测试结果

var1(VARCHAR)
JJJJJJJJ
HelloHello
空
null

8.11.1.4 REPLACE

本文为您介绍如何使用实时计算字符串函数REPLACE。

语法

```
VARCHAR REPLACE(str1, str2, str3)
```

入参

参数	数据类型	说明
str1	VARCHAR	原字符
str2	VARCHAR	目标字符
str3	VARCHAR	替换字符

功能描述

字符串替换函数。

示例

- 测试数据

str1(INT)	str2(INT)	str3(INT)
alibaba blink	blink	flink

- 测试语句

```
SELECT REPLACE(str1, str2, str3) as `result`
FROM T1
```

- 测试结果

result(VARCHAR)
alibaba flink

8.11.1.5 REVERSE

本文为您介绍如何使用实时计算字符串函数REVERSE。

语法

```
VARCHAR REVERSE(VARCHAR str)
```

入参

参数	数据类型	说明
str	VARCHAR	普通字符串值。

功能描述

反转字符串，返回字符串值的相反顺序。任一参数为null，返回null。

示例

- 测试数据

str1(VARCHAR)	str2(VARCHAR)	str3(VARCHAR)	str4(VARCHAR)
iPhoneX	Alibaba	World	null

- 测试语句

```
SELECT  REVERSE(str1) as var1,REVERSE(str2) as var2,
        REVERSE(str3) as var3,REVERSE(str4) as var4
FROM T1
```

- 测试结果

var1(VARCHAR)	var2(VARCHAR)	var3(VARCHAR)	var4(VARCHAR)
XenohPi	ababilA	dlroW	null

8.11.1.6 RPAD

本文为您介绍如何使用实时计算字符串函数RPAD。

语法

```
VARCHAR RPAD(VARCHAR str, INT len, VARCHAR pad)
```

入参

参数	数据类型	说明
str	VARCHAR	启始的字符串。
len	INT	新的字符串的长度。
pad	VARCHAR	需要重复补充的字符串。

功能描述

字符串str右端填充若干个字符串pad，直到新的字符串达到指定长度`len`为止。任意参数为null时返回null。

`len`长度为负数时为null。

`pad`为空串时，若`len`不大于`str`长度，返回`str`裁剪后的结果。若`len`大于`str`长度，返回null。

示例

- **测试数据**

str(VARCHAR)	len(INT)	pad(VARCHAR)
空	-2	空
HelloWorld	15	John
John	2	C
C	4	HelloWorld
null	2	C
c	2	null
asd	2	空
空	2	s
asd	4	空
空	0	空

- **测试语句**

```
SELECT  RPAD(str, len, pad) as result
FROM T1
```

- **测试结果**

result(VARCHAR)
null
HelloWorldJohnJ
Jo
CHel
null
null
as
ss
null
空

8.11.1.7 SPLIT_INDEX

本文为您介绍如何使用实时计算字符串函数SPLIT_INDEX。

语法

```
VARCHAR SPLIT_INDEX(VARCHAR str, VARCHAR sep, INT index)
```

入参

参数	数据类型	说明
str	VARCHAR	被分隔的字符串。
sep	VARCHAR	分隔符的字符串。
index	INT	截取的字段位置。

功能描述

以sep作为分隔符，将字符串str分隔成若干段，取其中的第index段。index从0开始，取不到字段，则返回null。

任一参数为NULL，则返回null。

示例

- 测试数据

str(VARCHAR)	sep(VARCHAR)	index(INT)
Jack,John,Mary	,	2
Jack,John,Mary	,	3
Jack,John,Mary	null	0
null	,	0

- 测试语句

```
SELECT  SPLIT_INDEX(str, sep, index) as var1
FROM T1
```

- 测试结果

var1(VARCHAR)
Mary
null
null
null

8.11.1.8 STR_TO_MAP

本文为您介绍如何使用实时计算字符串函数STR_TO_MAP。

语法

```
MAP STR_TO_MAP(VARCHAR text)
MAP STR_TO_MAP(VARCHAR text, VARCHAR listDelimiter, VARCHAR keyValueDe
limiter)
```

功能描述

使用listDelimiter将text分隔成K-V对，然后使用keyValueDelimiter分隔每个K-V对，组装成MAP返回。默认listDelimiter为（,）, keyValueDelimiter为（=）。

入参

参数	数据类型	说明
text	VARCHAR	输入文本。
listDelimiter	VARCHAR	用来将text分隔成K-V对。默认为（,）。
keyValueDelimiter	VARCHAR	用来分隔每个key和value。默认为（=）。

注意:

这里的Delimiter使用的是java的正则表达式，遇到特殊字符需要转义。

测试语句

```
SELECT
  STR_TO_MAP('k1=v1,k2=v2')['k1'] as a
FROM T1
```

测试结果

a(VARCHAR)
v1

8.11.1.9 SUBSTRING

本文为您介绍如何使用实时计算字符串函数SUBSTRING。

语法

```
VARCHAR SUBSTRING(VARCHAR a, INT start)
```

```
VARCHAR SUBSTRING(VARCHAR a, INT start, INT len)
```

入参

参数	数据类型	说明
a	VARCHAR	指定的字符串。
start	INT	在字符串a中开始截取的位置。
len	INT	类截取的长度。

功能描述

获取字符串子串。截取从位置start开始，长度为len的子串。若未指定len则截取到字符串结尾。start从1开始，start为0当1看待，为负数时表示从字符串末尾倒序计算位置。

示例

- 测试数据

str(VARCHAR)	nullstr(VARCHAR)
k1=v1;k2=v2	null

- 测试语句

```
SELECT SUBSTRING('', 222222222) as var1,
       SUBSTRING(str, 2) as var2,
       SUBSTRING(str, -2) as var3,
       SUBSTRING(str, -2, 1) as var4,
       SUBSTRING(str, 2, 1) as var5,
       SUBSTRING(str, 22) as var6,
       SUBSTRING(str, -22) as var7,
       SUBSTRING(str, 1) as var8,
       SUBSTRING(str, 0) as var9,
       SUBSTRING(nullstr, 0) as var10
FROM T1
```

- 测试结果

var1(VARCHAR)	var2(VARCHAR)	var3(VARCHAR)	var4(VARCHAR)	var5(VARCHAR)	var6(VARCHAR)	var7(VARCHAR)	var8(VARCHAR)	var9(VARCHAR)	var10(VARCHAR)
空	1=v1;k2=v2	v2	v	1	空	空	k1=v1;k2=v2	k1=v1;k2=v2	null

8.11.1.10 TO_BASE64

本文为您介绍如何使用实时计算字符串函数TO_BASE64。

语法

```
VARCHAR TO_BASE64(bin)
```

入参

参数	数据类型
bin	BINARY

功能描述

将BINARY类型数据转换成对应base64编码的字符串输出。

示例

- 测试数据

c(VARCHAR)
SGVsbG8gd29ybGQ=
SGk=
SGVsbG8=

- 测试语句

```
SELECT  TO_BASE64(FROM_BASE64(c)) as var1
FROM T1
```

- 测试结果

var1(VARCHAR)
SGVsbG8gd29ybGQ=
SGk=
SGVsbG8=

8.11.1.11 TRIM

本文为您介绍如何使用实时计算字符串函数TRIM。

语法

```
VARCHAR TRIM( VARCHAR x )
```

入参

参数	数据类型
x	VARCHAR

功能描述

除掉一个字串中的字头或字尾。最常见的用途是移除字首或字尾的空格。

示例

- 测试语句

```
SELECT TRIM('   Sample   ') as result
FROM   T1
```

- 测试结果

result(VARCHAR)
Sample

> 说明:
> 结果值为 'Sample'。

8.11.1.12 UPPER

本文为您介绍如何使用实时计算字符串函数UPPER。

语法

```
VARCHAR UPPER(A)
```

入参

参数	数据类型
A	VARCHAR

功能描述

返回转换为大写字符的字符串。

示例

- **测试数据**

var1(VARCHAR)
ss
ttee

- **测试语句**

```
SELECT UPPER(var1) as aa
FROM T1;
```

- **测试结果**

aa(VARCHAR)
SS
TTEE

8.11.1.13 CHAR_LENGTH

本文为您介绍如何使用实时计算字符串函数CHAR_LENGTH。

语法

```
INT CHAR_LENGTH(A)
```

入参

参数	数据类型
A	INT

功能描述

返回字符串中的字符的数量。

示例

- **测试数据**

var1(INT)
ss
231ee

- **测试语句**

```
SELECT  CHAR_LENGTH(var1) as aa
```

```
FROM T1;
```

- 测试结果

aa(INT)
2
5

8.11.1.14 CHR

本文为您介绍如何使用实时计算字符串函数CHR。

语法

```
VARCHAR CHR(INT ascii)
```

入参

参数	数据类型	说明
ascii	INT	是0到255之间的整数。如果不在此范围内，则返回NULL。

功能描述

将ASCII码转换为字符。

示例

- 测试数据

int1(INT)	int2(INT)	int3(INT)
255	97	65

- 测试语句

```
SELECT CHR(int1) as var1, CHR(int2) as var2, CHR(int3) as var3
FROM T1
```

- 测试结果

var1(VARCHAR)	var2(VARCHAR)	var3(VARCHAR)
ÿ	a	A

8.11.1.15 CONCAT

本文为您介绍如何使用实时计算字符串函数CONCAT。

语法

```
VARCHAR CONCAT(VARCHAR var1, VARCHAR var2, ...)
```

入参

参数	数据类型	说明
var1	VARCHAR	普通字符串值
var2	VARCHAR	普通字符串值

功能描述

连接两个或多个字符串值从而组成一个新的字符串。任一参数为NULL，跳过该参数。

示例

- 测试数据

var1(VARCHAR)	var2(VARCHAR)	var3(VARCHAR)
Hello	My	World
Hello	null	World
null	null	World
null	null	null

- 测试语句

```
SELECT CONCAT(var1, var2, var3) as var
FROM T1
```

- 测试结果

var(VARCHAR)
HelloMyWorld
HelloWorld
World
N/A

8.11.1.16 CONCAT_WS

本文为您介绍如何使用实时计算字符串函数CONCAT_WS。

语法

```
VARCHAR CONCAT_WS(VARCHAR separator, VARCHAR var1, VARCHAR var2, ...)
```

入参

参数	数据类型	说明
separator	VARCHAR	指的是分隔符号
var1	VARCHAR	/
var2	VARCHAR	/

功能描述

将每个参数值和第一个参数separator指定的分隔符依次连接到一起组成新的字符串,长度和类型取决于输入值。

 说明:

当separator取值为null，则将separator视作空串进行拼接。当其它参数为NULL，在执行拼接过程中跳过该为NULL的参数。

示例

- 测试数据

sep(VARCHAR)	str1(VARCHAR)	str2(VARCHAR)	str3(VARCHAR)
|	Jack	Harry	John
null	Jack	Harry	John
|	null	Harry	John
|	Jack	null	null

- 测试语句

```
SELECT  CONCAT_WS(sep, str1, str2, str3) as var
FROM T1
```

- 测试结果

var(VARCHAR)
Jack|Harry|John
JackHarryJohn

var(VARCHAR)
Harry|John
Jack

8.11.1.17 FROM_BASE64

本文为您介绍如何使用实时计算字符串函数FROM_BASE64。

语法

```
BINARY FROM_BASE64(str)
```

入参

参数	数据类型	说明
str	VARCHAR	base64编码的字符串。

功能描述

将base64编码的字符串str解析成对应的binary类型数据输出。

示例

- 测试数据

a(INT)	b(BIGINT)	c(VARCHAR)
1	1L	null

- 测试语句

```
SELECT
from_base64(c) as var1,from_base64('SGVsbG8gd29ybGQ=') as var2
FROM T1
```

- 测试结果

var1(BINARY)	var2(BINARY)
null	Byte Array: [72('H'), 101('e'), 108('l'), 108('l'), 111('o'), 32(' '), 119('w'), 111('o'), 114('r'), 108('l'), 100('d')]

8.11.1.18 HASH_CODE

本文为您介绍如何使用实时计算字符串函数HASH_CODE。

语法

```
INT HASH_CODE(VARCHAR str)
```

入参

str	VARCHAR

功能描述

返回字符串的HASH_CODE()的绝对值。

示例

- 测试数据

str1(VARCHAR)	str2(VARCHAR)	nullstr(VARCHAR)
k1=v1;k2=v2	k1:v1,k2:v2	null

- 测试语句

```
SELECT  HASH_CODE(str1) as var1, HASH_CODE(str2) as var2, HASH_CODE(
nullstr) as var3
FROM T1
```

- 测试结果

var1(INT)	var2(INT)	var3(INT)
1099348823	401392878	null

8.11.1.19 INITCAP

本文为您介绍如何使用实时计算字符串函数INITCAP。

语法

```
VARCHAR INITCAP(A)
```

入参

A	VARCHAR

功能描述

返回字符串，每个字转换器的第一个字母大写，其余为小写。

示例

- **测试数据**

var1(VARCHAR)
aADvbn

- **测试语句**

```
SELECT INITCAP(var1)as aa
FROM T1;
```

- **测试结果**

aa(VARCHAR)
Aasvbn

8.11.1.20 INSTR

本文为您介绍如何使用实时计算字符串函数INSTR。

说明:

INSTR函数仅支持实时计算2.2.0及以上版本。

语法

```
INT instr( string1, string2 )
INT instr( string1, string2 [, start_position [, nth_appearance ] ] )
```

入参

参数	数据类型	说明
string1	VARCHAR	源字符串
string2	VARCHAR	目标字符串
start_position	INT	起始位置
nth_appearance	INT	匹配序号

功能描述

返回子字符串在源字符串中的位置，如果在源串中没有找到子串，则返回0。

示例

- **测试数据T1**

string1(VARCHAR)
helloworld

- **测试语句**

```
SELECT
instr('helloworld','lo') as res1,
instr('helloworld','l',-1,1) as res2,
instr('helloworld','l',3,2) as res3
FROM T1
```

- **测试结果**

res1(INT)	res2(INT)	res3(INT)
4	9	4

8.11.1.21 JSON_VALUE

本文为您介绍如何使用实时计算字符串函数JSON_VALUE。

语法

```
VARCHAR JSON_VALUE(VARCHAR content, VARCHAR path)
```

入参

- **content**

  VARCHAR类型, 需要解析的JSON对象，使用字符串表示。

- **path**

  VARCHAR类型，解析JSON的路径表达式。 目前pth支持如下表达式。

符号	功能
$	根对象
[]	数组下标
*	数组通配符
.	取子元素

功能描述

从JSON字符串中提取指定path的值，不合法的JSON和null都统一返回null。

说明:

自定义path需要使用单引号（'），示例如下。

```
JSON_VALUE(json,'$.passenger_name') AS ABC
```

示例

- **测试数据**

id(INT)	json(VARCHAR)	path1(VARCHAR)
1	[10, 20, [30, 40]]	$[2][*]
2	{"aaa":"bbb","ccc":{"ddd":"eee","fff":"ggg","hhh":["h0","h1","h2"]},"iii":"jjj"}	$.ccc.hhh[*]
3	{"aaa":"bbb","ccc":{"ddd":"eee","fff":"ggg",hhh":["h0","h1","h2"]},"iii":"jjj"}	$.ccc.hhh[1]
4	[10, 20, [30, 40]]	NULL
5	NULL	$[2][*]
6	"{xx]"	"$[2][*]"

- **测试语句**

```
SELECT
    id,
    JSON_VALUE(json, path1) AS `value`
FROM
    T1
```

- **测试结果**

id (INT)	value (VARCHAR)
1	[30,40]
2	["h0","h1","h2"]
3	H1
4	NULL
5	NULL
6	NULL

8.11.1.22 KEYVALUE

本文为您介绍如何使用实时计算字符串函数KEYVALUE。

语法

```
VARCHAR KEYVALUE(VARCHAR str, VARCHAR split1, VARCHAR split2, VARCHAR
key_name)
```

入参

参数	数据类型	说明
str	VARCHAR	字符串中的key-value（kv）对。
split1	VARCHAR	kv对的分隔符。
split2	VARCHAR	kv的分隔符。
key_name	VARCHAR	键的名称

功能描述

解析str字符串中，匹配有split1(kv对的分隔符)和split2(kv的分隔符)的key-value对，根据key_name返回对应的数值。key_name值不存在或者异常时返回NULL。

示例

- 测试数据

str(VARCHAR)	split1(VARCHAR)	split2(VARCHAR)	key1(VARCHAR)
k1=v1;k2=v2	;	=	k2
null	;	|	:
k1:v1|k2:v2	null	=	:
k1:v1|k2:v2	|	=	null
k1:v1|k2:v2	|	=	:

- 测试语句

```
SELECT  KEYVALUE(str, split1, split2, key1) as `result`
FROM T1
```

- 测试结果

result(VARCHAR)
v2
null

result(VARCHAR)
null
null
null

8.11.1.23 LOCALTIMESTAMP

本文为您介绍如何使用实时计算字符串函数LOCALTIMESTAMP。

语法

```
timestamp LOCALTIMESTAMP
```

入参

- 无

功能描述

返回当前系统的时间戳。

示例

- 测试语句

```
SELECT
LOCALTIMESTAMP as `result`
FROM T1
```

- 测试结果

result (TIMESTAMP)
2018-07-27 14:04:38.998

8.11.1.24 LOWER

本文为您介绍如何使用实时计算字符串函数LOWER。

语法

```
VARCHAR LOWER(A)
```

入参

- A

  VARCHAR类型。

功能描述

返回转换为小写字符的字符串。

示例

- 测试数据

var1(VARCHAR)
Ss
yyT

- 测试语句

```
SELECT LOWER(var1) as aa
FROM T1;
```

- 测试结果

aa(VARCHAR)
ss
yyt

8.11.1.25 LPAD

本文为您介绍如何使用实时计算字符串函数LPAD。

语法

```
VARCHAR LPAD(VARCHAR str, INT len, VARCHAR pad)
```

入参

参数	数据类型	说明
str	VARCHAR	启始的字符串。
len	INT	新的字符串的长度。
pad	VARCHAR	需要重复补充的字符串。

功能描述

字符串str左端填充若干个字符串pad, 直到新的字符串达到指定长度len为止。

任意参数为null时返回null。

`len`为负数时返回为null。

`pad`为空串时，若`len`不大于`str`长度，返回`str`裁剪后的结果。若`len`大于`str`长度，返回null。

示例

- **测试数据**

str(VARCHAR)	len(INT)	pad(VARCHAR)
空	-2	空
HelloWorld	15	John
John	2	C
C	4	HelloWorld
null	2	C
c	2	null
asd	2	空
空	2	s
asd	4	空
空	0	空

- **测试语句**

```
SELECT LPAD(str, len, pad) AS result
FROM T1
```

- **测试结果**

result(VARCHAR)
null
JohnJHelloWorld
Jo
HelC
null
null
as
ss
null
空

8.11.1.26 MD5

本文为您介绍如何使用实时计算字符串函数MD5。

语法

```
VARCHAR MD5(VARCHAR str)
```

入参

- str

  VARCHAR类型

功能描述

返回字符串的MD5值。如果参数为空串（即参数为''）返回空串。

示例

- 测试数据

str1(VARCHAR)	str2(VARCHAR)
k1=v1;k2=v2	空

- 测试语句

```
SELECT
   MD5(str1) as var1,
   MD5(str2) as var2
FROM T1
```

- 测试结果

var1(VARCHAR)	var2(VARCHAR)
19c17f42b4d6a90f7f9ffc2ea9bdd775	空

8.11.1.27 OVERLAY

本文为您介绍如何使用实时计算字符串函数OVERLAY。

语法

```
VARCHAR OVERLAY ( (VARCHAR x PLACING VARCHAR y FROM INT start_position
 [ FOR INT length ]) )
```

入参

参数	数据类型
x	VARCHAR
y	VARCHAR

参数	数据类型
start_position	INT
length（可选）	INT

功能描述

用y替换x的子串。下标从start_position开始，替换length+1个字符。

示例

- 测试语句

```
OVERLAY('abcdefg' PLACING 'hij' FROM 2 FOR 2) as result
FROM   T1
```

- 测试结果

result(VARCHAR)
ahijdefg

8.11.1.28 PARSE_URL

本文为您介绍如何使用实时计算字符串函数PARSE_URL。

语法

```
VARCHAR PARSE_URL(VARCHAR urlStr, VARCHAR partToExtract [, VARCHAR key
])
```

入参

参数	数据类型	说明
urlStr	VARCHAR	链接
partToExtract	VARCHAR	指定链接中解析的部分。可取值如下： · HOST · PATH · QUERY · REF · PROTOCOL · FILE · AUTHORITY · USERINFO

参数	数据类型	说明
key	VARCHAR	可选，指定截取部分中，具体的键。

功能描述

返回urlStr中指定的部分解析后的值。

说明:

urlStr参数值为null，则返回值为null。

示例

- 测试数据

url1(VARCHAR)	nullstr(VARCHAR)
http://facebook.com/path/p1.php?query=1	null

- 测试语句

```
SELECT PARSE_URL(url1, 'QUERY', 'query') as var1,
       PARSE_URL(url1, 'QUERY') as var2,
       PARSE_URL(url1, 'HOST') as var3,
       PARSE_URL(url1, 'PATH') as var4,
       PARSE_URL(url1, 'REF') as var5,
       PARSE_URL(url1, 'PROTOCOL') as var6,
       PARSE_URL(url1, 'FILE') as var7,
       PARSE_URL(url1, 'AUTHORITY') as var8,
       PARSE_URL(nullstr, 'QUERY') as var9,
       PARSE_URL(url1, 'USERINFO') as var10,
       PARSE_URL(nullstr, 'QUERY', 'query') as var11
FROM T1
```

- 测试结果

var1(VARCHAR)	var2(VARCHAR)	var3(VARCHAR)	var4(VARCHAR)	var5(VARCHAR)	var6(VARCHAR)	var7(VARCHAR)	var8(VARCHAR)	var9(VARCHAR)	var10(VARCHAR)	var11(VARCHAR)
1	query=1	facebook.com	/path/p1.php	null	http	/path/p1.php?query=1	facebook.com	null	null	null

8.11.1.29 POSITION

本文为您介绍如何使用实时计算字符串函数POSITION。

语法

```
INTEGER POSITION( x IN  y)
```

入参

参数	数据类型
x	VARCHAR
y	VARCHAR

功能描述

返回要查询的字符串x在被查询字符串y里第一次出现的位置。类似于LOCATE(substr,str)。

示例

- 测试语句

```
POSITION('in' IN 'china') as result
FROM T1
```

- 测试结果

result(INT)
3

8.11.1.30 REGEXP

本文为您介绍如何使用实时计算字符串函数REGEXP。

语法

```
BOOLEAN REGEXP(VARCHAR str, VARCHAR pattern)
```

入参

参数	数据类型	说明
str	VARCHAR	指定的字符串。
pattern	VARCHAR	指定的匹配模式。

功能描述

指定str的字符串是否匹配指定的pattern进行正则匹配,str或者pattern为空或NULL返回false。

示例

- **测试数据**

str1(VARCHAR)	pattern1(VARCHAR)
k1=v1;k2=v2	k2*
k1:v1|k2:v2	k3
null	k3
k1:v1|k2:v2	null
k1:v1|k2:v2	(

- **测试语句**

```
SELECT  REGEXP(str1, pattern1) AS result
FROM T1
```

- **测试结果**

result(BOOLEAN)
true
false
false
false
false

8.11.2 数学函数

8.11.2.1 加

本文为您介绍如何使用实时计算数学函数加法。

语法

```
A + B
```

入参

参数	数据类型
A	INT
B	INT

功能描述

返回A加B的结果。

示例

- 测试数据

int1(INT)	int2(INT)	int3(INT)
10	20	30

- 测试语句

```
SELECT int1+int2+int3 as aa
FROM T1
```

- 测试结果

aa(int)
60

8.11.2.2 减

本文为您介绍如何使用实时计算数学函数减法。

语法

```
A - B
```

入参

参数	数据类型
A	INT
B	INT

功能描述

返回A减B的结果。

示例

- 测试数据

int1(INT)	int2(INT)	int3(INT)
10	10	30

- 测试语句

```
SELECT int3 - int2 - int1 as aa
```

```
FROM T1
```

- 测试结果

aa(int)
10

8.11.2.3 乘

本文为您介绍如何使用实时计算数学函数乘法。

语法

```
A * B
```

入参

参数	数据类型
A	INT
B	INT

功能描述

返回A乘以B的结果。

示例

- 测试数据

int1(INT)	int2(INT)	int3(INT)
10	20	3

- 测试语句

```
SELECT int1*int2*int3 as aa
FROM T1
```

- 测试结果

aa(int)
600

8.11.2.4 除

本文为您介绍如何使用实时计算数学函数除法。

语法

```
A / B
```

入参

参数	数据类型
A	INT
B	INT

功能描述

返回A除以B的结果。

示例

- 测试数据

int1(INT)	int2(INT)
8	4

- 测试语句

```
SELECT int1 / int2 as aa
FROM T1
```

- 测试结果

aa(DOUBLE)
2.0

8.11.2.5 ABS

本文为您介绍如何使用实时计算数学函数ABS。

语法

```
DOUBLE ABS(A)
```

入参

参数	数据类型
A	DOUBLE

功能描述

返回A的绝对值。

示例

- 测试数据

in1(DOUBLE)
4.3

- 测试语句

```
SELECT ABS(in1) as aa
FROM T1
```

- 测试结果

aa(DOUBLE)
4.3

8.11.2.6 ACOS

本文为您介绍如何使用实时计算数学函数ACOS。

语法

```
ACOS(A)
```

入参

参数	数据类型
A	DOUBLE

功能描述

返回A的反余弦值。

示例

- 测试数据

in1(DOUBLE)
0.7173560908995228
0.4

- 测试语句

```
SELECT ACOS(in1) as aa
```

```
FROM T1
```

- 测试结果

aa(DOUBLE)
0.7707963267948966
1.1592794807274085

8.11.2.7 BIN

本文为您介绍如何使用实时计算数学函数BIN。

语法

```
VARCHAR BIN(BIGINT number)
```

入参

参数	数据类型
number	BIGINT  说明: 参数的隐式转换支持INT，不支持其他类型。

功能描述

将长整型参数转换为二进制形式，返回类型为字符串。

示例

- 测试数据

id(INT)	x(BIGINT)
1	12L
2	10L
3	0L
4	10000000000L

 说明:
测试数据x列中的字母L代表的是数据类型Long，并不是做二进制转换的数值的一部分。

- 测试语句

```
SELECT id, bin(x) as var1
```

```
FROM T1
```

- 测试结果

id(INT)	var1(VARCHAR)
1	1100
2	1010
3	0
4	10010101000000101111100100000000000

8.11.2.8 ASIN

本文为您介绍如何使用实时计算数学函数ASIN。

语法

```
DOUBLE ASIN(A)
```

入参

参数	数据类型
A	DOUBLE

功能描述

返回A的反正弦值。

示例

- 测试数据

in1(DOUBLE)
0.7173560908995228
0.4

- 测试语句

```
SELECT ASIN(in1) as aa
FROM T1
```

- 测试结果

aa(DOUBLE)
0.8
0.41151684606748806

8.11.2.9 ATAN

本文为您介绍如何使用实时计算数学函数ATAN。

语法

```
DOUBLE ATAN(A)
```

入参

参数	数据类型
A	DOUBLE

功能描述

返回A的反正切值。

示例

- 测试数据

in1(DOUBLE)
0.7173560908995228
0.4

- 测试语句

```
SELECT ATAN(in1) as aa
FROM T1
```

- 测试结果

aa(DOUBLE)
0.6222796222326533
0.3805063771123649

8.11.2.10 BITAND

本文为您介绍如何使用实时计算数学函数BITAND。

语法

```
INT BITAND(INT number1,INT number2)
```

入参

参数	数据类型
number1	INT

参数	数据类型
number2	INT

功能描述

运算符按位“与”操作。输入和输出类型均为INT整型，且类型一致。

示例

- 测试数据

a(INT)	b(INT)
2	3

- 测试语句

```
SELECT BITAND(a, b) as intt
FROM T1
```

- 测试结果

intt(INT)
2

8.11.2.11 BITNOT

本文为您介绍如何使用实时计算数学函数BITNOT。

语法

```
INT BITNOT(INT number)
```

入参

参数	数据类型
number	INT

功能描述

按位取反。输入和输出类型均为整型，且类型一致。

示例

- 测试数据

a(INT)
7

- 测试语句

```
SELECT BITNOT(a) as var1
FROM T1
```

- 测试结果

var1(INT)
0xfff8

8.11.2.12 BITOR

本文为您介绍如何使用实时计算数学函数BITOR。

语法

```
INT BITOR(INT number1, INT number2)
```

入参

参数	数据类型
number1	INT
number2	INT

功能描述

按位取“或”。输入和输出类型均为整型，且类型一致。

示例

- 测试数据

a(INT)	b(INT)
2	3

- 测试语句

```
SELECT BITOR(a, b) as var1
FROM T1
```

- 测试结果

var1(INT)
3

8.11.2.13 BITXOR

本文为您介绍如何使用实时计算数学函数BITXOR。

语法

```
INT BITXOR(INT number1, INT number2)
```

入参

参数	数据类型
number1	INT
number2	INT

功能描述

按位取“异或”。输入和输出类型均为整型，且类型一致。

示例

- 测试数据

a(INT)	b(INT)
2	3

- 测试语句

```
SELECT BITXOR(a, b) as var1
FROM T1
```

- 测试结果

var1(INT)
1

8.11.2.14 CARDINALITY

本文为您介绍如何使用实时计算数学函数CARDINALITY。

语法

```
CARDINALITY(str)
```

入参

参数	数据类型
number1	数组

功能描述

返回一个集合中的元素数量。

示例

- 测试语句

```
SELECT cardinality(array[1,2,3]) AS `result`
FROM T1
```

- 测试结果

result(INT)
3

8.11.2.15 COS

本文为您介绍如何使用实时计算数学函数COS。

语法

```
DOUBLE COS(A)
```

入参

参数	数据类型
A	DOUBLE

功能描述

返回A的余弦值。

示例

- 测试数据

in1(DOUBLE)
0.8
0.4

- 测试语句

```
SELECT COS(in1) as aa
```

```
FROM T1
```

- 测试结果

aa(DOUBLE)
0.6967067093471654
0.9210609940028851

8.11.2.16 COT

本文为您介绍如何使用实时计算数学函数COT。

语法

```
DOUBLE COT(A)
```

入参

参数	数据类型
A	DOUBLE

功能描述

返回A的余切值。

示例

- 测试数据

in1(DOUBLE)
0.8
0.4

- 测试语句

```
SELECT COT(in1) as aa
FROM T1
```

- 测试结果

aa(DOUBLE)
1.0296385570503641
0.4227932187381618

8.11.2.17 EXP

本文为您介绍如何使用实时计算数学函数EXP。

语法

```
DOUBLE EXP(A)
```

入参

参数	数据类型
A	DOUBLE

功能描述

返回自然常数e的A次幂的DOUBLE类型数值。

示例

- 测试数据

in1(DOUBLE)
8.0
10.0

- 测试语句

```
SELECT EXP(in1) as aa
FROM T1
```

- 测试结果

aa(DOUBLE)
2980.9579870417283
22026.465794806718

8.11.2.18 E

本文为您介绍如何使用实时计算数学函数E。

语法

```
DOUBLE E()
```

入参

参数	数据类型
A	DOUBLE

功能描述

返回自然常数e的DOUBLE类型值。

示例

- 测试数据

in1(DOUBLE)
8.0
10.0

- 测试语句

```
SELECT  e() as dou1, E() as dou2
FROM T1
```

- 测试结果

dou1(DOUBLE)	dou2(DOUBLE)
2.718281828459045	2.718281828459045

8.11.2.19 FLOOR

本文为您介绍如何使用实时计算数学函数FLOOR。

语法

```
B FLOOR(A)
```

入参

参数	数据类型
A	INT、BIGINT、FLOAT、DOUBLE

功能描述

返回小于或等于A的最大整数数值，即舍去小数位的数值。B的数据类型与A的数据类型一致。

示例

- 测试数据

in1(DOUBLE)	in2(BIGINT)
8.123	3

- 测试语句

```
SELECT
```

```
FLOOR(in1) as out1,
FLOOR(in2) as out2
FROM T1
```

- 测试结果

out1(DOUBLE)	out2(BIGINT)
8.0	3

8.11.2.20 LN

本文为您介绍如何使用实时计算数学函数LN。

语法

```
DOUBLE ln(DOUBLE number)
```

入参

参数	数据类型
number	DOUBLE 说明: 若输入为VARCHAR类型或BIGINT类型，会隐式转换到DOUBLE类型后参与运算。

功能描述

返回number的自然对数。返回值是DOUBLE类型的对数值。

示例

- 测试数据

ID(INT)	X(DOUBLE)
1	100.0
2	8.0

- 测试语句

```
SELECT id, ln(x) as dou1, ln(e()) as dou2
FROM T1
```

- 测试结果

ID(INT)	dou1(DOUBLE)	dou2(DOUBLE)
1	4.605170185988092	1.0

ID(INT)	dou1(DOUBLE)	dou2(DOUBLE)
2	2.0794415416798357	1.0

8.11.2.21 LOG

本文为您介绍如何使用实时计算数学函数LOG。

语法

```
DOUBLE LOG(DOUBLE base, DOUBLE x)
DOUBLE LOG(DOUBLE x)
```

入参

参数	数据类型
base	DOUBLE
x	DOUBLE

功能描述

返回以base为底的x的自然对数。返回值是DOUBLE类型的对数值。如果是只有一个参数的，返回以x为底的自然对数。

示例

- 测试数据

ID(INT)	BASE(DOUBLE)	X(DOUBLE)
1	10.0	100.0
2	2.0	8.0

- 测试语句

```
SELECT id, LOG(base, x) as dou1, LOG(2) as dou2
FROM T1
```

- 测试结果

ID(INT)	dou1(DOUBLE)	dou2(DOUBLE)
1	2.0	0.6931471805599453
2	3.0	0.6931471805599453

8.11.2.22 LOG10

本文为您介绍如何使用实时计算数学函数LOG10。

语法

```
DOUBLE LOG10(DOUBLE x)
```

入参

参数	数据类型
x	DOUBLE

功能描述

返回以10为底的自然对数。若x为NULL，则返回NULL。若x为负数会引发异常。

示例

- 测试数据

id(INT)	X(INT)
1	100
2	10

- 测试语句

```
SELECT id, log10(x) as dou1
FROM T1
```

- 测试结果

id(INT)	dou1(DOUBLE)
1	2.0
2	1.0

8.11.2.23 LOG2

本文为您介绍如何使用实时计算数学函数LOG2。

语法

```
DOUBLE LOG2(DOUBLE x)
```

入参

参数	数据类型
x	DOUBLE

功能描述

返回以2为底的自然对数。若x为NULL，则返回NULL。若x为负数，会引发异常。

示例

- 测试数据

id(INT)	X(INT)
1	8
2	2

- 测试语句

```
SELECT id, log2(x) as dou1
FROM T1
```

- 测试结果

id(INT)	dou1(DOUBLE)
1	3.0
2	1.0

8.11.2.24 PI

本文为您介绍如何使用实时计算数学函数PI。

语法

```
DOUBLE PI()
```

功能描述

获取圆周率常量PI值。

示例

- 测试数据

ID(INT)	X(INT)
1	8

- 测试语句

```
SELECT id, PI() as dou1
```

```
FROM T1
```

- 测试结果

ID(INT)	dou1(DOUBLE)
1	3.141592653589793

8.11.2.25 POWER

本文为您介绍如何使用实时计算数学函数POWER。

语法

```
DOUBLE POWER(A, B)
```

入参

参数	数据类型
A	DOUBLE
B	DOUBLE

功能描述

返回A的B次幂。

示例

- 测试数据

in1(DOUBLE)	in2(DOUBLE)
2.0	4.0

- 测试语句

```
SELECT POWER(in1, in2) as aa
FROM T1
```

- 测试结果

aa(DOUBLE)
16.0

8.11.2.26 RAND

本文为您介绍如何使用实时计算数学函数RAND。

语法

```
DOUBLE RAND([BIGINT seed])
```

入参

参数	数据类型	说明
seed	BIGINT	Seed取值为随机数，决定随机数序列的起始值。

功能描述

返回大于等于0小于1的DOUBLE类型随机数。

示例

- 测试数据

id(INT)	X(INT)
1	8

- 测试语句

```
SELECT id, rand(1) as dou1, rand(3) as dou2
FROM T1
```

- 测试结果

id(INT)	dou1(DOUBLE)	dou2(DOUBLE)
1	0.7308781907032909	0.731057369148862

8.11.2.27 SIN

本文为您介绍如何使用实时计算数学函数SIN。

语法

```
DOUBLE SIN(A)
```

入参

参数	数据类型
A	DOUBLE

功能描述

返回A的正弦值。

示例

- 测试数据

in1(DOUBLE)
8.0
0.4

- 测试语句

```
SELECT SIN(in1) as aa
FROM T1
```

- 测试结果

aa(DOUBLE)
0.9893582466233818
0.3894183423086505

8.11.2.28 SQRT

本文为您介绍如何使用实时计算数学函数SQRT。

语法

```
DOUBLE SQRT(A)
```

入参

参数	数据类型
A	DOUBLE

功能描述

返回A的平方根。

示例

- 测试数据

in1(DOUBLE)
8.0

- 测试语句

```
SELECT SQRT(in1) as aa
FROM T1
```

- 测试结果

aa(DOUBLE)
2.8284271247461903

8.11.2.29 TAN

本文为您介绍如何使用实时计算数学函数TAN。

语法

```
DOUBLE TAN(A)
```

入参

参数	数据类型
A	DOUBLE

功能描述

计算A的正切值。

示例

- 测试数据

in1(DOUBLE)
0.8
0.4

- 测试语句

```
SELECT TAN(in1) as aa
FROM T1
```

- 测试结果

aa(DOUBLE)
1.0296385570503641
0.4227932187381618

8.11.2.30 CEIL

本文为您介绍如何使用实时计算数学函数CEIL。

语法

```
B CEIL(A)
```

入参

参数	数据类型
A	INT、BIGINT、FLOAT或DOUBLE
B	INT、BIGINT、FLOAT或DOUBLE

功能描述

输出B为大于或等于输入值A的最小整数数值。输出B的数据类型与输入参数A的数据类型一致。

示例

- 测试数据

in1(INT)	in2（DOUBLE）
1	2.3

- 测试语句

```
SELECT
CEIL  (in1)  as out1
CEIL  (in2)  as out2
FROM T1
```

- 测试结果

out1(INT)	out2(DOUBLE)
1	3.0

8.11.2.31 CHARACTER_LENGTH

本文为您介绍如何使用实时计算数学函数CHARACTER_LENGTH。

语法

```
INTEGER CHARACTER_LENGTH( VARCHAR x )
```

入参

参数	数据类型
x	VARCHAR

功能描述

返回字符串x中的字符数。

示例

- 测试数据

ID(INT)	X(VARCHAR)
1	StreamCompute

- 测试语句

```
SELECT CHARACTER_LENGTH( x ) as result
FROM T1
```

- 测试结果

ID(INT)	result(INT)
1	13

8.11.2.32 DEGREES

本文为您介绍如何使用实时计算数学函数DEGREES。

语法

```
DOUBLE DEGREES( double x )
```

入参

参数	数据类型
x	DOUBLE

功能描述

将弧度x转换为数值。

示例

- 测试语句

```
SELECT DEGREES (PI()) as result
FROM T1
```

- 测试结果

result(DOUBLE)
180.0

8.11.2.33 MOD

本文为您介绍如何使用实时计算数学函数MOD。

语法

```
INTEGER MOD(INTEGER x,INTEGER y)
```

入参

参数	数据类型
x	INTEGER
y	INTEGER

功能描述

整数运算中，求整数x除以整数y的余数。当x为负值时，或者x、y均为负值时，结果为负值。

示例

- 测试数据

X（INT）	Y（INT）
29	3
-29	3
-29	-3

- 测试语句

```
SELECT MOD(x,y) as result
FROM T1
```

- 测试结果

ID(INT)	result(INT)
1	2
2	-2
3	-2

8.11.2.34 ROUND

本文为您介绍如何使用实时计算数学函数ROUND。

语法

```
DECIMAL ROUND( DECIMAL x, INT n)
```

入参

参数	数据类型
x	DECIMAL
n	INT

功能描述

把数值x字段舍入为指定的小数n位数。

示例

- 测试数据

in1(DECIMAL)
0.7173560908995228
0.4

- 测试语句

```
SELECT ROUND(in1,2) as `result`
FROM T1
```

- 测试结果

result(DECIMAL)
0.72
0.40

8.11.3 日期函数

8.11.3.1 CURRENT_DATE

本文为您介绍如何使用实时计算日期函数CURRENT_DATE。

语法

```
CURRENT_DATE
```

功能描述

返回当前系统日期。

示例

- 测试语句

```
SELECT CURRENT_DATE as res
FROM T1
```

- 测试结果

res(DATE)
2018-09-20

8.11.3.2 CURRENT_TIMESTAMP

本文为您介绍如何使用实时计算日期函数CURRENT_TIMESTAMP。

语法

```
TIMESTAMP CURRENT_TIMESTAMP
```

功能描述

返回当前UTC（GMT+0）时间戳，时间戳单位为毫秒。

示例

- 测试语句

```
SELECT CURRENT_TIMESTAMP as var1
FROM T1
```

- 测试结果

var1(TIMESTAMP)
2007-04-30 13:10:02.047

8.11.3.3 DATEDIFF

本文为您介绍如何使用实时计算日期函数DATEDIFF。

语法

```
INT DATEDIFF(VARCHAR enddate, VARCHAR startdate)
INT DATEDIFF(TIMESTAMP enddate, VARCHAR startdate)
INT DATEDIFF(VARCHAR enddate, TIMESTAMP startdate)
INT DATEDIFF(TIMESTAMP enddate, TIMESTAMP startdate)
```

入参

参数	数据类型
startdate	TIMESTAMP或VARCHAR
enddate	TIMESTAMP或VARCHAR

说明:

VARCHAR日期格式：yyyy-MM-dd或yyyy-MM-dd HH:mm:ss。

功能描述

计算从enddate到startdate两个时间的天数差值，返回整数。若有参数为null或解析错误，返回null。

示例

- 测试数据

datetime1(VARCHAR)	datetime2(VARCHAR)	nullstr(VARCHAR)
2017-10-15 00:00:00	2017-09-15 00:00:00	null

- 测试语句

```
SELECT ROUND(in1,2) as `result`
 DATEDIFF(TIMESTAMP '2017-10-15 23:00:00',datetime2) as int2,
 DATEDIFF(datetime2,TIMESTAMP '2017-10-15 23:00:00') as int3,
 DATEDIFF(datetime2,nullstr) as int4,
 DATEDIFF(nullstr,TIMESTAMP '2017-10-15 23:00:00') as int5,
 DATEDIFF(nullstr,datetime2) as int6,
 DATEDIFF(TIMESTAMP '2017-10-15 23:00:00',TIMESTAMP '2017-9-15 00:00
:00')as int7
FROM T1
```

- 测试结果

int1(INT)	int2(INT)	int3(INT)	int4(INT)	int5(INT)	int6(INT)	int7(INT)
30	31	-31	null	null	null	31

8.11.3.4 DATE_ADD

本文为您介绍如何使用实时计算日期函数DATE_ADD。

语法

```
VARCHAR DATE_ADD(VARCHAR startdate, INT days)
VARCHAR DATE_ADD(TIMESTAMP time, INT days)
```

入参

参数	数据类型
startdate	TIMESTAMP或VARCHAR 说明: VARCHAR类型日期格式：yyyy-MM-dd 或 yyyy-MM-dd HH:mm:ss。
enddate	TIMESTAMP
days	INT

功能描述

返回指定startdate日期days天数后的VARCHAR类型日期，返回string格式的日期为`yyyy-MM-dd`。若有参数为null或解析错误，返回null。

示例

- 测试数据

datetime1(VATCHAR)	nullstr(VATCHAR)
2017-09-15 00:00:00	null

- 测试语句

```
SELECT DATE_ADD(datetime1, 30) as var1,
 DATE_ADD(TIMESTAMP '2017-09-15 23:00:00',30) as var2,
 DATE_ADD(nullstr,30) as var3
FROM T1
```

- 测试结果

var1(VARCHAR)	var2(VARCHAR)	var3(VARCHAR)
2017-10-15	2017-10-15	null

8.11.3.5 DATE_FORMAT

本文为您介绍如何使用实时计算日期函数DATE_FORMAT。

语法

```
VARCHAR DATE_FORMAT(TIMESTAMP time, VARCHAR to_format)
VARCHAR DATE_FORMAT(VARCHAR date, VARCHAR to_format)
VARCHAR DATE_FORMAT(VARCHAR date, VARCHAR from_format, VARCHAR
to_format)
```

入参

参数	数据类型
date	VARCHAR 说明: 默认日期格式：yyyy-MM-dd HH:mm:ss。
time	TIMESTAMP
from_format	VARCHAR
to_format	VARCHAR

功能描述

将字符串类型的日期从源格式转换至目标格式。第一个参数（time 或 date）为源字符串。第二个参数from_format可选，为源字符串的格式，默认为yyyy-MM-dd hh:mm:ss。第三个参数为返回日期的的格式，返回值为转换格式后的字符串类型日期。若有参数为null或解析错误，返回null。

示例

- 测试数据

date1(VARCHAR)	datetime1(VARCHAR)	nullstr(VARCHAR)
0915-2017	2017-09-15 00:00:00	null

- 测试语句

```
SELECT DATE_FORMAT(datetime1, 'yyMMdd') as var1,
 DATE_FORMAT(nullstr, 'yyMMdd') as var2,
 DATE_FORMAT(datetime1, nullstr) as var3,
 DATE_FORMAT(date1, 'MMdd-yyyy', nullstr) as var4,
 DATE_FORMAT(date1, 'MMdd-yyyy', 'yyyyMMdd') as var5,
 DATE_FORMAT(TIMESTAMP '2017-09-15 23:00:00', 'yyMMdd') as var6
```

```
FROM T1
```

- 测试结果

var1(VARCHAR)	var2(VARCHAR)	var3(VARCHAR)	var4(VARCHAR)	var5(VARCHAR)	var6(VARCHAR)
170915	null	null	null	20170915	170915

8.11.3.6 DATE_SUB

本文为您介绍如何使用实时计算日期函数DATE_SUB。

语法

```
VARCHAR DATE_SUB(VARCHAR startdate, INT days)
VARCHAR DATE_SUB(TIMESTAMP time, INT days)
```

入参

参数	数据类型
startdate	VARCHAR 说明: VARCHAR类型日期格式：yyyy-MM-dd 或 yyyy-MM-dd HH:mm:ss。
time	TIMESTAMP
days	INT

功能描述

返回startdate减去days天数的日期。返回VARCHAR类型的yyyy-MM-dd日期格式。若有参数为null或解析错误，返回null。

示例

- 测试数据

date1(VARCHAR)	nullstr(VARCHAR)
2017-10-15	null

- 测试语句

```
SELECT DATE_SUB(date1, 30) as var1,
 DATE_SUB(TIMESTAMP '2017-10-15 23:00:00',30) as var2,
 DATE_SUB(nullstr,30) as var3
```

```
FROM T1
```

- 测试结果

var1(VARCHAR)	var2(VARCHAR)	var3(VARCHAR)
2017-09-15	2017-09-15	null

8.11.3.7 DAYOFMONTH

本文为您介绍如何使用实时计算日期函数DAYOFMONTH。

语法

```
BIGINT DAYOFMONTH(TIMESTAMP time)
BIGINT DAYOFMONTH(DATE date)
```

入参

参数	数据类型
date	DATE
time	TIMESTAMP

功能描述

返回输入时间参数date或time中所指代的“日”。返回值范围为1～31。

示例

- 测试数据

tsStr(VARCHAR)	dateStr(VARCHAR)	tdate(DATE)	ts(TIMESTAMP)
2017-10-15 00:00:00	2017-09-15	2017-11-10	2017-10-15 00:00:00

- 测试语句

```
SELECT DAYOFMONTH(TIMESTAMP '2016-09-15 00:00:00') as int1,
 DAYOFMONTH(DATE '2017-09-22') as int2,
 DAYOFMONTH(tdate) as int3,
 DAYOFMONTH(ts) as int4,
 DAYOFMONTH(CAST(dateStr AS DATE)) as int5,
 DAYOFMONTH(CAST(tsStr AS TIMESTAMP)) as int6
FROM T1
```

- 测试结果

int1(BIGINT)	int2(BIGINT)	int3(BIGINT)	int4(BIGINT)	int5(BIGINT)	int6(BIGINT)
15	22	10	15	15	15

8.11.3.8 EXTRACT

本文为您介绍如何使用实时计算日期函数EXTRACT。

语法

```
BIGINT EXTRACT(unit FROM time)
```

入参

参数	数据类型
time	任意日期表达式
unit	可取值如下。 · MICROSECOND · SECOND · MINUTE · HOUR · DAY · WEEK · MONTH · QUARTER · YEAR · SECOND_MICROSECOND · MINUTE_MICROSECOND · MINUTE_SECOND · HOUR_MICROSECOND · HOUR_SECOND · HOUR_MINUTE · DAY_MICROSECOND · DAY_SECOND · DAY_MINUTE · DAY_HOUR · YEAR_MONTH

功能描述

返回日期/时间的单独部分，比如年、月、日、小时、分钟、周数等。

示例

· 测试语句

```
EXTRACT(YEAR FROM CURRENT_TIMESTAMP) AS OrderYear,
EXTRACT(MONTH FROM CURRENT_TIMESTAMP) AS OrderMonth,
EXTRACT(DAY FROM CURRENT_TIMESTAMP) AS OrderDay,
```

```
EXTRACT(WEEK FROM CURRENT_TIMESTAMP) AS OrderWeek
```

- 测试结果

OrderYear(BIGINT)	OrderMonth(BIGINT)	OrderDay(BIGINT)	OrderWeek(BIGINT)
2018	10	11	41

8.11.3.9 FROM_UNIXTIME

本文为您介绍如何使用实时计算日期函数FROM_UNIXTIME。

语法

```
VARCHAR FROM_UNIXTIME(BIGINT unixtime[, VARCHAR format])
```

入参

参数	数据类型
unixtime	BIGINT
format	VARCHAR

说明:

- 参数unixtime为长整型，是以秒为单位的时间戳。
- 参数format可选，为日期格式，默认格式为yyyy-MM-dd HH:mm:ss，表示返回VARCHAR类型的符合指定格式的日期，若有参数为null或解析错误，返回null。

功能描述

返回值为VARCHAR类型的日期值，默认日期格式：yyyy-MM-dd HH:mm:ss，若指定日期格式按指定格式输出任一输入参数是NULL，返回NULL。

示例

- 测试数据

unixtime1(BIGINT)	nullstr(VARCHAR)
1505404800	null

- 测试语句

```
SELECT FROM_UNIXTIME(unixtime1) as var1,
 FROM_UNIXTIME(unixtime1,'MMdd-yyyy') as var2,
 FROM_UNIXTIME(unixtime1,nullstr) as var3
```

```
FROM T1
```

- 测试结果

var1(VARCHAR)	var2(VARCHAR)	var3(VARCHAR)
2017-09-15 00:00:00	0915-2017	null

8.11.3.10 HOUR

本文为您介绍如何使用实时计算日期函数HOUR。

语法

```
BIGINT HOUR(TIME time)
BIGINT HOUR(TIMESTAMP timestamp)
```

入参

参数	数据类型
time	TIME
timestamp	TIMESTAMP

功能描述

返回输入时间参数time或timestamp中的24小时制的小时数，范围0～23。

示例

- 测试数据

datetime1(VARCHAR)	time1(VARCHAR)	time2(TIME)	timestamp1(TIMESTAMP)
2017-10-15 11:12:13	22:23:24	22:23:24	2017-10-15 11:12:13

- 测试语句

```
SELECT HOUR(TIMESTAMP '2016-09-20 23:33:33') AS int1,
 HOUR(TIME '23:30:33') AS int2,
 HOUR(time2) AS int3,
 HOUR(timestamp1) AS int4,
 HOUR(CAST(time1 AS TIME)) AS int5,
 HOUR(TO_TIMESTAMP(datetime1)) AS int6
FROM T1
```

- 测试结果

int1(BIGINT)	int2(BIGINT)	int3(BIGINT)	int4(BIGINT)	int5(BIGINT)	int6(BIGINT)
23	23	22	11	22	11

8.11.3.11 LOCALTIME

本文为您介绍如何使用实时计算日期函数LOCALTIME。

语法

```
TIME LOCALTIME
```

功能描述

以数据类型TIME的值返回会话时区中的当前时间。LOCALTIME可当变量直接使用。

示例

- 测试语句

```
SELECT LOCALTIME as `result`
FROM T1
```

- 测试结果

result(TIME)
19:00:47

8.11.3.12 MINUTE

本文为您介绍如何使用实时计算日期函数MINUTE。

语法

```
BIGINT MINUTE(TIME time)
BIGINT MINUTE(TIMESTAMP timestamp)
```

入参

参数	数据类型
time	TIME
timestamp	TIMESTAMP

功能描述

返回输入时间参数中time或timestamp中的“分钟”部分。取值范围0～59。

示例

- **测试数据**

datetime1(VARCHAR)	time1(VARCHAR)	time2(TIME)	timestamp1(TIMESTAMP)
2017-10-15 11:12:13	22:23:24	22:23:24	2017-10-15 11:12:13

- **测试语句**

```
SELECT MINUTE(TIMESTAMP '2016-09-20 23:33:33') as int1,
 MINUTE(TIME '23:30:33') as int2,
 MINUTE(time2) as int3,
 MINUTE(timestamp1) as int4,
 MINUTE(CAST(time1 AS TIME)) as int5,
 MINUTE(CAST(datetime1 AS TIMESTAMP)) as int6
FROM T1
```

- **测试结果**

int1(BIGINT)	int2(BIGINT)	int3(BIGINT)	int4(BIGINT)	int5(BIGINT)	int6(BIGINT)
33	30	23	12	23	12

8.11.3.13 MONTH

本文为您介绍如何使用实时计算日期函数MONTH。

语法

```
BIGINT MONTH(TIMESTAMP timestamp)
BIGINT MONTH(DATE date)
```

入参

参数	数据类型
time	TIME
timestamp	TIMESTAMP

功能描述

返回输入时间参数中的“月”，范围1～12。

示例

- **测试数据**

datetime1(VARCHAR)	time1(VARCHAR)	time2(TIME)	timestamp1(TIMESTAMP)
2017-10-15 11:12:13	22:23:24	22:23:24	2017-10-15 11:12:13

- **测试语句**

```
SELECT MONTH(TIMESTAMP '2016-09-15 00:00:00') as int1,
 MONTH(DATE '2017-09-22') as int2,
 MONTH(tdate) as int3,
 MONTH(ts) as int4,
 MONTH(CAST(dateStr AS DATE)) as int5,
 MONTH(CAST(tsStr AS TIMESTAMP)) as int6
FROM T1
```

- **测试结果**

int1(BIGINT)	int2(BIGINT)	int3(BIGINT)	int4(BIGINT)	int5(BIGINT)	int6(BIGINT)
9	9	11	10	9	10

8.11.3.14 NOW

本文为您介绍如何使用实时计算日期函数NOW。

语法

```
BIGINT NOW()
BIGINT NOW(a)
```

入参

参数	数据类型
a	INT

功能描述

- **未指定参数时返回当前时区时间的时间戳，单位为秒。**
- **可在括号内输入INT类型参数作为偏移值（单位：秒），返回偏移后的时间戳。例如，`now(100)`返回当前时间戳加100秒的时间戳。**

> **说明:**
> **偏移值a为NULL时，NOW(a)返回值为NULL。**

示例

- **测试数据**

  表 8-1: T1

a(INT)
null

- **测试语句**

```
SELECT
 NOW() as now,
 NOW(100) as now_100,
 NOW(a) as now_null
FROM T1
```

- **测试结果**

now(BIGINT)	now_100(BIGINT)	now_null(BIGINT)
1403006911	1403007011	null

8.11.3.15 SECOND

本文为您介绍如何使用实时计算日期函数SECOND。

语法

```
BIGINT SECOND(TIMESTAMP timestamp)
 BIGINT SECOND(TIME time)
```

入参

参数	数据类型
time	TIME
timestamp	TIMESTAMP

功能描述

返回输入时间参数中的“秒”部分，范围0～59。

示例

- **测试数据**

datetime1(VARCHAR)	time1(VARCHAR)	time2(TIME)	timestamp1(TIMESTAMP)
2017-10-15 11:12:13	22:23:24	22:23:24	2017-10-15 11:12:13

- 测试语句

```
SELECT SECOND(TIMESTAMP '2016-09-20 23:33:33') as int1,
 SECOND(TIME '23:30:33') as int2,
 SECOND(time2) as int3,
 SECOND(timestamp1) as int4,
 SECOND(CAST(time1 AS TIME)) as int5,
 SECOND(CAST(datetime1 AS TIMESTAMP)) as int6
FROM T1
```

- 测试结果

int1(BIGINT)	int2(BIGINT)	int3(BIGINT)	int4(BIGINT)	int5(BIGINT)	int6(BIGINT)
33	33	24	13	24	13

8.11.3.16 TIMESTAMPADD

本文为您介绍如何使用实时计算日期函数TIMESTAMPADD。

语法

```
TIMESTAMP TIMESTAMPADD(interval,INT int_expr,TIMESTAMP datetime_expr)
DATE TIMESTAMPADD(interval,INT int_expr,DATE datetime_expr)
```

入参

参数	数据类型
interval	VARCHAR
int_expr	INT
datetime_expr	TIMESTAMP或DATE

说明:

interval可取值如下。

interval参数	时间间隔单位
FRAC_SECOND	毫秒
SECOND	秒
MINUTE	分钟
HOUR	小时
DAY	天
WEEK	星期

interval参数	时间间隔单位
MONTH	月
QUARTER	季度
YEAR	年

功能描述

返回类型与datetime_expr类型相同。

将整型表达式int_expr添加到日期或日期时间表达式datetime_expr中。以数据类型TIME的值返回会话时区中的当前时间。

示例

- 测试数据

a(TIMESTAMP)	b(DATE)
2018-07-09 10:23:56	1990-02-20

- 测试语句

```
SELECT
TIMESTAMPADD(HOUR,3,a) AS `result1`
TIMESTAMPADD(DAY,3,b) AS `result2`
FROM T1
```

- 测试结果

result1(TIMESTAMP)	result2(DATE)
2018-07-09 13:23:56.0	1990-02-23

8.11.3.17 TO_DATE

本文为您介绍如何使用实时计算日期函数TO_DATE。

语法

```
Date TO_DATE(INT time)
Date TO_DATE(VARCHAR date)
```

```
Date TO_DATE(VARCHAR date, VARCHAR format)
```

入参

参数	数据类型
time	INT  说明: 表示从1970-1-1到所表示时间之间天数。
date	VARCHAR  说明: 默认格式为yyyy-MM-dd。
format	VARCHAR

功能描述

将INT类型的日期或者VARCHAR类型的日期转换成DATE类型。

示例

- 测试数据

date1(INT)	date2(VARCHAR)	date3(VARCHAR)
100	2017-09-15	20170915

- 测试语句

```
SELECT TO_DATE(date1) as var1,
 TO_DATE(date2) as var2,
 TO_DATE(date3,'yyyyMMdd') as var3
FROM T1
```

- 测试结果

var1(DATE)	var2(DATE)	var3(DATE)
1970-04-11	2017-09-15	2017-09-15

8.11.3.18 TO_TIMESTAMP

本文为您介绍如何使用实时计算日期函数TO_TIMESTAMP。

语法

```
TIMESTAMP TO_TIMESTAMP(BIGINT time)
TIMESTAMP TO_TIMESTAMP(VARCHAR date)
```

```
TIMESTAMP TO_TIMESTAMP(VARCHAR date, VARCHAR format)
```

入参

参数	数据类型
time	BIGINT 说明: 单位为毫秒。
date	VARCHAR 说明: 默认格式为yyyy-MM-dd HH:mm:ss。
format	VARCHAR

功能描述

将BIGINT类型的日期或者VARCHAR类型的日期转换成TIMESTAMP类型。

示例

- 测试数据

timestamp1(BIGINT)	timestamp2(VARCHAR)	timestamp3(VARCHAR)
1513135677000	2017-09-15 00:00:00	20170915000000

- 测试语句

```
SELECT TO_TIMESTAMP(timestamp1) as var1,
 TO_TIMESTAMP(timestamp2) as var2,
 TO_TIMESTAMP(timestamp3, 'yyyyMMddHHmmss') as var3
FROM T1
```

- 测试结果

var1(TIMESTAMP)	var2(TIMESTAMP)	var3(TIMESTAMP)
2017-12-13 03:27:57.0	2017-09-15 00:00:00.0	2017-09-15 00:00:00.0

8.11.3.19 UNIX_TIMESTAMP

本文为您介绍如何使用实时计算日期函数UNIX_TIMESTAMP。

语法

```
BIGINT UNIX_TIMESTAMP()
BIGINT UNIX_TIMESTAMP(VARCHAR date)
BIGINT UNIX_TIMESTAMP(TIMESTAMP timestamp)
```

```
BIGINT UNIX_TIMESTAMP(VARCHAR date, VARCHAR format)
```

入参

参数	数据类型
timestamp	TIMESTAMP
date	VARCHAR 说明: 默认日期格式为yyyy-MM-dd HH:mm:ss。
format	VARCHAR 说明: 默认格式为yyyy-MM-dd hh:mm:ss。

功能描述

返回第一个参数date转换成的长整型的时间戳，单位为秒。无参数时返回当前时间的时间戳，单位为秒，与now语义相同。若有参数为null或解析错误，返回null。

示例

- 测试数据

nullstr（VARCHAR）
null

- 测试语句

```
SELECT UNIX_TIMESTAMP() as big1,
       UNIX_TIMESTAMP(nullstr) as big2
FROM T1
```

- 测试结果

big1（BIGINT）	big2（BIGINT）
1403006911	null

8.11.3.20 WEEK

本文为您介绍如何使用实时计算日期函数WEEK。

语法

```
BIGINT WEEK(DATE date)
```

```
BIGINT WEEK(TIMESTAMP timestamp)
```

入参

参数	数据类型
date	DATE
timestamp	TIMESTAMP

功能描述

计算指定日期在一年中的第几周，周数取值区间 1~53。

示例

- 测试数据

dateStr(VARCHAR)	date1(DATE)	ts1(TIMESTAMP)
2017-09-15	2017-11-10	2017-10-15 00:00:00

- 测试语句

```
SELECT WEEK(TIMESTAMP '2017-09-15 00:00:00') as int1,
 WEEK(date1) as int2,
 WEEK(ts1) as int3,
 WEEK(CAST(dateStr AS DATE)) as int4
FROM T1
```

- 测试结果

int1(BIGINT)	int2(BIGINT)	int3(BIGINT)	int4(BIGINT)
37	45	41	37

8.11.3.21 YEAR

本文为您介绍如何使用实时计算日期函数YEAR。

语法

```
BIGINT YEAR(TIMESTAMP timestamp)
BIGINT YEAR(DATE date)
```

入参

参数	数据类型
date	DATE
timestamp	TIMESTAMP

功能描述

返回输入时间的年份。

示例

- 测试数据

tsStr(VARCHAR)	dateStr(VARCHAR)	tdate(DATE)	ts(TIMESTAMP)
2017-10-15 00:00:00	2017-09-15	2017-11-10	2017-10-15 00:00:00

- 测试语句

```
SELECT YEAR(TIMESTAMP '2016-09-15 00:00:00') as int1,
 YEAR(DATE '2017-09-22') as int2,
 YEAR(tdate) as int3,
 YEAR(ts) as int4,
 YEAR(CAST(dateStr AS DATE)) as int5,
 YEAR(CAST(tsStr AS TIMESTAMP)) as int6
FROM T1
```

- 测试结果

int1(BIGINT)	int2(BIGINT)	int3(BIGINT)	int4(BIGINT)	int5(BIGINT)	int6(BIGINT)
2016	2017	2017	2017	2015	2017

8.11.4 条件函数

8.11.4.1 CASE WHEN

本文为您介绍如何使用实时计算条件函数CASE WHEN。

语法

```
CASE WHEN a THEN b [WHEN c THEN d]* [ELSE e] END
```

入参

参数	数据类型
a	BOOLEAN
b	BOOLEAN
c	BOOLEAN
d	BOOLEAN
e	BOOLEAN

功能描述

如果a为TRUE,则返回b。如果a为FALSE，c为TRUE，则返回d。如果a，c都为FALSE,则返回e。

示例

说明:

CASE WHEN返回常量字符串的时候，会在字符串后面补全空格。如下面的例子，当满足else条件的时候返回值是 ‘ios ‘, 后面会多几个空格。

```
case when device_type = 'android'
then 'android'
else 'ios'
end as os
```

两种解决方法:

- 利用TRIM函数，去除空格。对这个例子来说，在用到os字段的地方，用TRIM(os)
- 利用CAST将常量字符串转为VARCHAR类型。

- 测试数据

device_type(VARCHAR)
android
ios
win

- 测试语句一（解决方法1：利用TRIM函数）

```
SELECT
   trim(os), -- 这里加了trim
   CHAR_LENGTH(trim(os)) -- 这里加了trim
from(
   SELECT
     case when device_type = 'android'
     then 'android'
     else 'ios'
end as os
FROM T1
);
```

测试语句二（解决方法2：利用CAST函数）

```
SELECT
 os,
CHAR_LENGTH(os)
from
(SELECT
 case when device_type = 'android'
```

```
 then cast('android' as varchar) -- 这里加了cast
 else cast('ios' as varchar) -- 这里加了cast
end as os
FROM T1
);
```

- 测试结果

os(VARCHAR)	length(INT)
android	7
ios	3
ios	3

8.11.4.2 COALESCE

本文为您介绍如何使用实时计算条件函数COALESCE。

语法

```
COALESCE(A,B,...)
```

入参

参数	数据类型
A	任意数据类型
B	任意数据类型

说明:

所有这些值类型必须相同或为null，否则会引发异常。

功能描述

返回列表中第一个非null的值，返回值类型和参数类型相同。如果列表中所有的值都是null，则返回null。

说明:

至少要有一个参数，否则引发异常。

示例

- 测试数据

var1(VARCHAR)	var2(VARCHAR)
null	30

- 测试语句

```
SELECT COALESCE(var1,var2) as aa
FROM T1
```

- 测试结果

aa(VARCHAR)
30

8.11.4.3 IF

本文为您介绍如何使用实时计算条件函数IF。

语法

```
T IF(BOOLEAN testCondition, T valueTrue, T valueFalseOrNull)
```

说明:

T代表任意类型的返回值。

入参

参数	数据类型
testCondition	BOOLEAN
valueTrue	可以为任意类型，但valueTrue和valueFalseOrNull类型需保持一致。
valueFalseOrNull	可以为任意类型，但valueTrue和valueFalseOrNull类型需保持一致。

功能描述

以第一个参数的布尔值为判断标准。若为true，则返回第二个参数valueTrue。若为false，则返回第三个参数valueFalseOrNull。第一个参数为null时，判定结果为false。其他参数为null时，按照正常语义运行。返回值类型以T为准。

示例

- 测试数据

int1(INT)	int2(INT)	str1(VARCHAR)	str2(VARCHAR)
1	2	Jack	Harry
1	2	Jack	null
1	2	null	Harry

- 测试语句

```
SELECT IF(int1 < int2,str1, str2) as int1
FROM T1
```

- 测试结果

int1(VARCHAR)
Jack
Jack
null

8.11.4.4 IS_ALPHA

本文为您介绍如何使用实时计算条件函数IS_ALPHA。

语法

```
BOOLEAN IS_ALPHA(VARCHAR str)
```

入参

参数	数据类型
str	VARCHAR

功能描述

str中只包含字母则返回TRUE，否则返回FALSE。

示例

- 测试数据

e(VARCHAR)	f(VARCHAR)	g(VARCHAR)
3	asd	null

- 测试语句

```
SELECT IS_ALPHA(e) as boo1,IS_ALPHA(f) as boo2,IS_ALPHA(g) as boo3
FROM T1
```

- 测试结果

boo1(BOOLEAN)	boo2(BOOLEAN)	boo3(BOOLEAN)
false	true	false

8.11.4.5 IS_DECIMAL

本文为您介绍如何使用实时计算条件函数IS_DECIMAL。

语法

```
BOOLEAN IS_DECIMAL(VARCHAR str)
```

入参

参数	数据类型
str	VARCHAR

功能描述

str字符串如果可以转换为十进制数值返回TRUE；否则返回FALSE。

示例

- 测试数据

a(VARCHAR)	b(VARCHAR)	c(VARCHAR)	d(VARCHAR)	e(VARCHAR)	f(VARCHAR)	g(VARCHAR)
1	123	2	11.4445	3	asd	null

- 测试语句

```
SELECT
IS_DECIMAL(a) as boo1,
IS_DECIMAL(b) as boo2,
IS_DECIMAL(c) as boo3,
IS_DECIMAL(d) as boo4,
IS_DECIMAL(e) as boo5,
IS_DECIMAL(f) as boo6,
IS_DECIMAL(g) as boo7
FROM T1
```

- 测试结果

boo1(BOOLEAN)	boo2(BOOLEAN)	boo3(BOOLEAN)	boo4(BOOLEAN)	boo5(BOOLEAN)	boo6(BOOLEAN)	boo7(BOOLEAN)
true	true	true	true	true	false	false

8.11.4.6 IS_DIGIT

本文为您介绍如何使用实时计算条件函数IS_DIGIT。

语法

```
BOOLEAN IS_DIGIT(VARCHAR str)
```

入参

参数	数据类型
str	VARCHAR

功能描述

str中只包含数字则返回true，否则返回false。返回值为BOOLEAN类型。

示例

- 测试数据

e(VARCHAR)	f(VARCHAR)	g(VARCHAR)
3	asd	null

- 测试语句

```
SELECT
IS_DIGIT(e) as boo1,
IS_DIGIT(f) as boo2,
IS_DIGIT(g) as boo3
FROM T1
```

- 测试结果

boo1(BOOLEAN)	boo2(BOOLEAN)	boo3(BOOLEAN)
true	false	false

8.11.4.7 NULLIF

本文为您介绍如何使用实时计算条件函数NULLIF。

语法

```
NULLIF(A,B)
```

入参

参数	数据类型
A	INT

参数	数据类型
B	INT

功能描述

两个参数的值相同，则返回null，值不同则返回第一个参数的值。

示例

- 测试数据

var1(INT)	var2(INT)
30	30

- 测试语句

```
SELECT NULLIF(var1,var2) as aa
FROM T1
```

- 测试结果

aa(INT)
null

8.11.5 表值函数

8.11.5.1 GENERATE_SERIES

本文为您介绍如何使用实时计算表值函数GENERATE_SERIES。

语法

```
GENERATE_SERIES(INT from, INT to)
```

入参

参数	数据类型
from	INT类型,指定下界，包含下界值。
to	INT类型,指定上界，不包含上界值。

功能描述

生成一串连续的数值，分别为 from, from+1, from+2, … , to-1。

示例

- **测试数据**

s(INT)	e(INT)
1	3
-2	1

- **测试语句**

```
SELECT s, e, v
FROM T1, lateral table(GENERATE_SERIES(s, e))
as T(v)
```

- **测试结果**

s(INT)	e(INT)	v(INT)
1	3	1
1	3	2
-2	1	-2
-2	1	-1
-2	1	-0

8.11.5.2 JSON_TUPLE

本文为您介绍如何使用实时计算表值函数JSON_TUPLE。

语法

```
JSON_TUPLE(str, path1, path2 ..., pathN)
```

入参

参数	数据类型	说明
str	VARCHAR	JSON字符串
path1～pathN	VARCHAR	表示路径的字符串，前面不需要`$.`。

功能描述

从JSON字符串中取出各路径字符串所表示的值。

示例

- **测试数据**

d(VARCHAR)	s(VARCHAR)
{ “qwe” :” asd” ,” qwe2” :” asd2” ,” qwe3” :” asd3” }	qwe3
{ “qwe” :” asd4” ,” qwe2” :” asd5” ,” qwe3” :” asd3” }	qwe2

- **测试语句**

```
SELECT d, v
FROM T1, lateral table(JSON_TUPLE(d, 'qwe', s))
 as T(v)
```

- **测试结果**

d(VARCHAR)	v(VARCHAR)
{ “qwe” :” asd” ,” qwe2” :” asd2” ,” qwe3” :” asd3” }	asd
{ “qwe” :” asd” ,” qwe2” :” asd2” ,” qwe3” :” asd3” }	asd3
{ “qwe” :” asd4” ,” qwe2” :” asd5” ,” qwe3” :” asd3” }	asd4
{ “qwe” :” asd4” ,” qwe2” :” asd5” ,” qwe3” :” asd3” }	asd5

8.11.5.3 STRING_SPLIT

本文为您介绍如何使用实时计算表值函数STRING_SPLIT。

语法

```
string_split(varchar string , varchar separator)
```

入参

参数	数据类型
string	varchar
separator	varchar

功能描述

将字符串string按照separator分隔符分割成多个部分。

示例

- **测试数据**

d(varchar)	s(varchar)
abc-bcd	-
hhh	-

- **测试语句**

```
select d, v
from T1,
lateral table(string_split(d, s)) as T(v)
```

- **测试结果**

d(varchar)	v(varchar)
abc-bcd	abc
abc-bcd	bcd
hhh	hhh

说明:
分隔符separator暂不支持多字符串形式，只支持单个字符串形式。

8.11.5.4 MULTI_KEYVALUE

本文为您介绍如何使用实时计算表值函数MULTI_KEYVALUE。

说明:
仅支持实时计算2.2.2及以上版本。

语法

```
MULTI_KEYVALUE(VARCHAR str, VARCHAR split1, VARCHAR split2, VARCHAR
key_name1, VARCHAR key_name2, ...)
```

入参

参数	数据类型	说明
str	VARCHAR	字符串中的key-value（kv）对。

参数	数据类型	说明
split1	VARCHAR	kv对的分隔符。当split1为null，表示按照whitespace作为kv对的分割符。当split1的长度>1时，split1仅表示分隔符的集合，每个字符都表示一个有效的分隔符。
split2	VARCHAR	kv的分隔符。当split2为null，表示按照whitespace作为kv的分割符。当split2的长度>1时，split2仅表示分隔符的集合，每个字符都表示一个有效的分隔符。
key_name1, key_name2, ...	VARCHAR	需要获取value的key值列表。

功能描述

解析str字符串中的key-value对，匹配有split1和split2的key-value对，并返回参数列表里key_name1，key_name2等对应的value值列表。key_name值不存在时，对应的value值是null。

示例

- 测试数据

str(VARCHAR)	split1(VARCHAR)	split2(VARCHAR)	key1(VARCHAR)	key2(VARCHAR)
k1=v1;k2=v2	;	=	k1	k2
null	;	=	k1	k2
k1:v1;k2:v2	;	:	k1	k3
k1:v1;k2:v2	;	=	k1	k2
k1:v1;k2:v2	,	:	k1	k2
k1:v1;k2=v2	;	:	k1	k2
k1:v1abck2:v2	cab	:	k1	k2
k1:v1;k2=v2	;	:=	k1	k2
k1:v1 k2:v2	null	:	k1	k2
k1 v1;k2 v2	;	null	k1	k2

- 测试语句

```
SELECT c1, c2
FROM T1, lateral table(MULTI_KEYVALUE(str, split1, split2, key1,
key2))
as T(c1, c2)
```

- 测试结果

c1(VARCHAR)	c2(VARCHAR)
v1	v2
null	null
v1	null
null	null
null	null
v1	null
v1	v2
v1	v2
v1	v2
v1	v2

8.11.6 类型转换函数

8.11.6.1 CAST

本文为您介绍如何使用实时计算类型转换函数CAST。

语法

```
CAST(A AS type)
```

入参

参数	数据类型
A	请参见#unique_342。

功能描述

将A值转换为给定类型。

示例

- **测试数据**

var1(VARCHAR)	var2(INT)
1000	30

- **测试语句**

```
SELECT CAST(var1 AS INT) as aa
FROM T1
```

- **测试结果**

aa(INT)
1000

8.11.7 聚合函数

8.11.7.1 AVG

本文为您介绍如何使用实时计算聚合函数AVG。Flink SQL中使用AVG函数返回指定表达式中所有值的平均值。

语法

```
AVG(A)
```

入参

参数	数据类型
A	TINYINT、SMALLINT、INT、BIGINT 、FLOAT 、DECIMAL 或DOUBLE

功能描述

返回列的平均数。

示例

- **测试数据**

var1(INT)	var2(INT)
4	30
6	30

- 测试语句

```
SELECT AVG(var1) as aa
FROM T1
```

- 测试结果

aa(INT)
5

8.11.7.2 CONCAT_AGG

本文为您介绍如何使用实时计算聚合函数CONCAT_AGG。Flink SQL中使用CONCAT_AGG函数将对应字段的所有字符串连接成新的字符串。

语法

```
CONCAT_AGG([linedelimiter,] value )
```

入参

参数	数据类型
linedelimiter	目前只支持字符串常量。可选。

功能描述

连接对应字段的字符串，默认连接符`\n`，连接完成后新生成的字符串。返回值VARCHAR类型。

示例

- 测试数据

c(VARCHAR)
Hi
Hi
Hi
Hi
Hi
Hi
Hi
Hi
Hi
Hi

· 测试语句

```
SELECT
concat_agg(c) as var1,
concat_agg('-', c) as var2
FROM MyTable
GROUP BY c
```

· 测试结果

var1(VARCHAR)	var2(VARCHAR)
Hi\nHi\nHi\nHi\nHi\nHi\nHi\nHi\nHi\nHi	Hi-Hi-Hi-Hi-Hi-Hi-Hi-Hi-Hi-Hi

8.11.7.3 COUNT

本文为您介绍如何使用实时计算聚合函数COUNT。Flink SQL中使用COUNT函数返回输入列的数量。

语法

```
COUNT(A)
```

入参

参数	数据类型
A	· 支持TINYINT，SMALLINT，INT，BIGINT，FLOAT，DECIMAL，DOUBLE，BOOLEAN，VARCHAR类型。 · 不支持DATE，TIME，TIMESTAMP，VARBINARY类型。

功能描述

返回输入列的个数。

示例

· 测试数据

var1(VARCHAR)
1000
100
10
1

- 测试语句

```
SELECT COUNT(var1) as aa
FROM T1
```

- 测试结果

aa(BIGINT)
4

8.11.7.4 FIRST_VALUE

本文为您介绍如何使用实时计算聚合函数FIRST_VALUE。Flink SQL中使用FIRST_VALUE函数返回数据流的第1条非null数据。

语法

```
T FIRST_VALUE( T value )
T FIRST_VALUE( T value, BIGINT order )
```

入参

参数	数据类型
value	任意参数类型，但输入参数只能为同一种类型。
order	BIGINT

功能描述

获取数据流的第1条非null数据。根据order判定FIRST_VALUE所在的行，取order值最小的记录作为FIRST_VALUE。

示例1

- 测试数据

a(BIGINT)	b(INT)	c(VARCHAR)
1L	1	“Hello”
2L	2	“Hello”
3L	3	“Hello”
4L	4	“Hello”
5L	5	“Hello”
6L	6	“Hello”
7L	7	“Hello World”

a(BIGINT)	b(INT)	c(VARCHAR)
8L	8	“Hello World”
20L	20	“Hello World”

- 测试语句

```
SELECT c,
 FIRST_VALUE(b)
OVER (
PARTITION BY c
ORDER BY PROCTIME() RANGE UNBOUNDED PRECEDING
) AS var1
FROM T1
```

- 测试结果

c(VARCHAR)	var1(BIGINT)
“Hello”	1
“Hello”	1
“Hello”	1
“Hello”	1
“Hello”	1
“Hello”	1
“Hello World”	7
“Hello World”	7
“Hello World”	7

示例2

- 测试数据

a(BIGINT)	b(INT)	c(VARCHAR)	order (BIGINT)
1L	1	“Hello”	7
2L	2	“Hello”	7
3L	3	“Hello”	7
4L	4	“Hello”	7
5L	5	“Hello”	6
6L	6	“Hello”	4
7L	7	“Hello World”	3
8L	8	“Hello World”	2

a(BIGINT)	b(INT)	c(VARCHAR)	order (BIGINT)
20L	20	“Hello World”	1

- 测试语句

```
SELECT c,
 FIRST_VALUE(b,order)
OVER (
PARTITION BY c
ORDER BY PROCTIME() RANGE UNBOUNDED PRECEDING
) AS var1
FROM T1
```

- 测试结果

c(VARCHAR)	var1(INT)
“Hello”	1
“Hello”	1
“Hello”	1
“Hello”	1
“Hello”	5
“Hello”	6
“Hello World”	7
“Hello World”	8
“Hello World”	20

8.11.7.5 LAST_VALUE

本文为您介绍如何使用实时计算聚合函数LAST_VALUE。Flink SQL中使用LAST_VALUE函数返回指定数据流的最后1条非null数据。

语法

```
T LAST_VALUE(T value)
T LAST_VALUE(T value, BIGINT order)
```

入参

参数	数据类型
value	任意参数类型，但输入参数只能为同一种类型。
order	BIGINT

功能描述

获取数据流的最后1条非null数据。根据order判定LAST_VALUE所在的行，取order值最大的记录作为LAST_VALUE。

示例1

- 测试数据

a(BIGINT)	b(INT)	c(VARCHAR)
1L	1	“Hello”
2L	2	“Hello”
3L	3	“Hello”
4L	4	“Hello”
5L	5	“Hello”
6L	6	“Hello”
7L	7	“Hello World”
8L	8	“Hello World”
20L	20	“Hello World”

- 测试语句

```
SELECT c,
 LAST_VALUE(b)
OVER (
PARTITION BY c
ORDER BY PROCTIME() RANGE UNBOUNDED PRECEDING
) AS var1
FROM T1
```

- 测试结果

c(VARCHAR)	var1(INT)
“Hello”	1
“Hello”	2
“Hello”	3
“Hello”	4
“Hello”	5
“Hello”	6
“Hello World”	7
“Hello World”	8

c(VARCHAR)	var1(INT)
“Hello World”	20

示例2

- **测试数据**

a(BIGINT)	b(INT)	c(VARCHAR)	order (BIGINT)
1L	1	“Hello”	5
2L	2	“Hello”	5
3L	3	“Hello”	6
4L	4	“Hello”	5
5L	5	“Hello”	6
6L	6	“Hello”	5
7L	7	“Hello World”	4
8L	8	“Hello World”	8
20L	20	“Hello World”	9

- **测试语句**

```
SELECT c,
 LAST_VALUE(b,order)
OVER (
PARTITION BY c
ORDER BY PROCTIME() RANGE UNBOUNDED PRECEDING
) AS var1
FROM T1
```

- **测试结果**

c(VARCHAR)	var1(INT)
“Hello”	1
“Hello”	1
“Hello”	3
“Hello”	3
“Hello”	3
“Hello”	3
“Hello World”	7
“Hello World”	8
“Hello World”	20

8.11.7.6 MAX

本文为您介绍如何使用实时计算聚合函数MAX。Flink SQL中使用MAX函数返回所有输入值的最大值。

语法

```
MAX(A)
```

入参

参数	数据类型
A	TINYINT，SMALLINT，INT，BIGINT，FLOAT，DECI 说明： 不支持类型： DATE，TIME，TIMESTAMP，VARBINARY。

功能描述

返回所有输入值的最大值。

示例

- 测试数据

var1(INT)
4
8

- 测试语句

```
SELECT MAX(var1) as aa
FROM T1
```

- 测试结果

aa(INT)
8

8.11.7.7 MIN

本文为您介绍如何使用实时计算聚合函数MIN。Flink SQL中使用MIN函数返回所有输入值的最小值。

语法

```
MIX(A)
```

入参

参数	数据类型
A	TINYINT，SMALLINT，INT，BIGINT，FLOAT，DECI 说明： 不支持类型： DATE，TIME，TIMESTAMP，VARBINARY。

功能描述

返回所有输入值的最小值。

示例

- 测试数据

var1(INT)
4
8

- 测试语句

```
SELECT MIX(var1) as aa
FROM T1
```

- 测试结果

aa(INT)
4

8.11.7.8 SUM

本文为您介绍如何使用实时计算聚合函数SUM。Flink SQL中使用SUM函数来返回所有输入值的数值之和。

语法

```
SUM(A)
```

入参

参数	数据类型
A	TINYINT，SMALLINT，INT，BIGINT，FLOAT，DECIMAL，DOUBLE。

功能描述

返回所有输入值的数值之和。

示例

- 测试数据

var1(INT)
4
4

- 测试语句

```
SELECT sum(var1) as aa
FROM T1
```

- 测试结果

aa(INT)
8

8.11.7.9 VAR_POP

Flink SQL中使用VAR_POP函数返回指定表达式中所有值的总体统计方差。

语法

```
T VAR_POP(T value)
```

入参

参数	数据类型
value	数值型，可为BIGINT, DOUBLE等类型。

功能描述

返回所有输入值的方差。

示例

- 测试数据

a(BIGINT)	c(VARCHAR)
2900	Hi
2500	Hi
2600	Hi
3100	Hello
11000	Hello

- 测试语句

```
SELECT
VAR_POP(a) as `result`,
 c
FROM MyTable
GROUP BY c
```

- 测试结果

result(BIGINT)	c
28889	Hi
15602500	Hello

8.11.7.10 STDDEV_POP

本文为您介绍如何使用实时计算聚合函数STDDEV_POP。Flink SQL中使用STDDEV_POP函数来返回数值的总体标准差。

语法

```
T STDDEV_POP(T value)
```

入参

参数	数据类型
value	BIGINT或DOUBLE

功能描述

返回数值的总体标准差。

示例

- **测试数据**

a(DOUBLE)	c(VARCHAR)
0	Hi
1	Hi
2	Hi
3	Hi
4	Hi
5	Hi
6	Hi
7	Hi
8	Hi
9	Hi

- **测试语句**

```
SELECT c, STDDEV_POP(a) as dou1
FROM MyTable
GROUP BY c
```

- **测试结果**

c(VARCHAR)	dou1(DOUBLE)
Hi	2.8722813232690143

8.11.8 其他函数

8.11.8.1 UUID

本文为您介绍如何使用实时计算UUID。Flink SQL中使用UUID函数返回通用唯一标识字符。

语法

```
VARCHAR UUID()
```

功能描述

返回通用唯一标识字符。

示例

- 测试语句

```
SELECT uuid() as `result`
FROM T1
```

- 测试结果

result(VARCHAR)
a364e414-e68b-4e5c-9166-65b3a153e257

8.11.8.2 DISTINCT

DISTINCT用于SELECT语句中，表示对查询的结果的去重后进行输出。

DISTINCT语法

```
SELECT DISTINCT expressions
FROM tables
  ...
```

- DISTINCT必须放到开始位置。
- expressions是一个或多个expression，可以是具体的column，也可以是function等任何合法表达式。

DISTINCT示例

- SQL语句

  Flink SQL中DISTINCT的示例如下：

```
CREATE TABLE distinct_tab_source(
    FirstName  VARCHAR,
    LastName VARCHAR
)WITH(
   type='random'
) ;
CREATE TABLE distinct_tab_sink(
```

```
    FirstName  VARCHAR,
    LastName VARCHAR
)WITH(
    type = 'print'
) ;
INSERT INTO distinct_tab_sink
SELECT DISTINCT FirstName, LastName -- 以FirstName和LastName两个列进行去重
FROM distinct_tab_source;
```

- 测试数据

FirstName	LastName
SUNS	HENGRAN
SUN	JINCHENG
SUN	SHENGRAN
SUN	SHENGRAN

- 查询结果

说明:

- 测试数据有4条记录。`DISTINCT FirstName, LastName`进行去重后将重复数据`SUN, SHENGRAN`去重输出3条结果记录。
- `SUNS,HENGRAN`和`SUN,SHENGRAN`两条记录并没有被去重，说明`DISTINCT FirstName, LastName`是对两个字段分别处理的，而不是拼接到一起再进行去重。

- DISTINCT的等效写法

在SQL中利用`GROUP BY`语句也可以到达和`DISTINCT`类似的去重效果。`GROUP BY`语法如下：

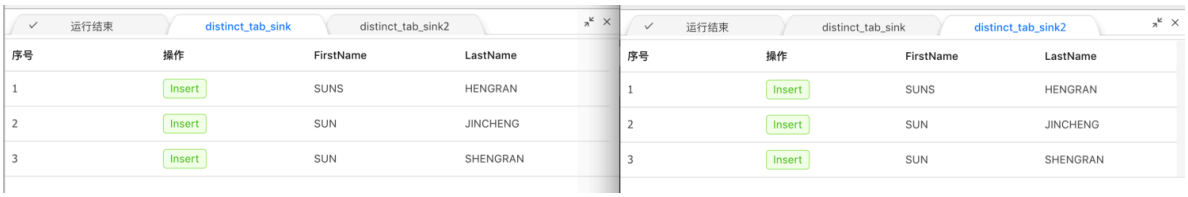
```
SELECT expressions
FROM tables
GROUP BY expressions
;
```

通过多路输出的方式编写的和DISTINCT示例等效的SQL如下：

```
CREATE TABLE distinct_tab_source(
    FirstName  VARCHAR,
    LastName VARCHAR
)WITH(
   type='random'
```

```
) ;
CREATE TABLE distinct_tab_sink(
    FirstName  VARCHAR,
    LastName VARCHAR
)WITH(
    type = 'print'
) ;
CREATE TABLE distinct_tab_sink2(
    FirstName  VARCHAR,
    LastName VARCHAR
)WITH(
    type = 'print'
) ;
INSERT INTO distinct_tab_sink
    SELECT DISTINCT FirstName, LastName -- 以FirstName和LastName两个列进行去重
        FROM distinct_tab_source;
INSERT INTO distinct_tab_sink2
    SELECT FirstName, LastName
        FROM distinct_tab_source
         GROUP BY FirstName, LastName; -- 以FirstName和LastName两个列进行去重
```

使用相同的测试数据产生的输出结果如下，和DISTINCT示例的输出结果保持一致。说明两种方式的语义一致。



DISTINCT在聚合函数COUNT中的作用

DISTINCT能使聚合函数COUNT统计去重后的计数。

```
COUNT(DISTINCT expression)
```

说明:

expression目前只支持一个表达式。

COUNT DISTINCT语法示例

- SQL语法

```
CREATE TABLE distinct_tab_source(
    FirstName  VARCHAR,
    LastName VARCHAR
)WITH(
   type='random'
) ;
CREATE TABLE distinct_tab_sink(
    cnt BIGINT  ,
    distinct_cnt BIGINT
)WITH(
    type = 'print'
```

```
) ;
INSERT INTO distinct_tab_sink
    SELECT
      COUNT(FirstName), -- 不去重
      COUNT(DISTINCT FirstName)-- 按FirstName去重
    FROM distinct_tab_source;
```

- 测试数据

FirstName	LastName
SUNS	HENGRAN
SUN	JINCHENG
SUN	SHENGRAN
SUN	SHENGRAN

- 测试结果

8.12 自定义函数（UDX）

8.12.1 UDX概述

本文为您介绍如何为实时计算自定义函数搭建环境和使用。

说明:
仅独享模式支持自定义函数。

UDX分类

实时计算支持以下3类自定义函数：

UDX分类	描述
UDF（User Defined Function）	用户自定义标量值函数（User Defined Scalar Function）。其输入与输出是一对一的关系，即读入一行数据，写出一条输出值。
UDAF（User Defined Aggregation Function）	自定义聚合函数，其输入与输出是多对一的关系， 即将多条输入记录聚合成一条输出值。可以与SQL中的GROUP BY语句联用。具体语法请参见#unique_359。

UDX分类	描述
UDTF（User Defined Table-valued Function）	自定义表值函数，调用一次函数输出多行或多列数据。

UDX DEMO

阿里云团队为您提供UDX DEMO便于您快速进行业务开发。以下实时计算UDX DEMO中包含UDF、UDAF和UDTF的实现，供您参考：

> 说明：
> - DEMO示例中已为您配置对应版本的开发环境，您无需进行环境搭建。
> - DEMO为Maven项目，您可使用IntelliJ IDEA进行开发。开发方法请参见使用IntelliJ IDEA开发自定义函数。

- 实时计算3.0版本

  blink_udx_3x
- 实时计算2.0版本

  blink_udx_2x
- 实时计算1.0版本

  blink_udx_1x

环境搭建

以上DEMO主要使用的依赖JAR包如下，如您需要单独使用，可自行下载：

- 基于实时计算3.2.1以下版本：
  - flink-streaming-java_2.11
  - flink-table_2.11
  - flink-core-blink-2.2.4
- 基于实时计算3.2.1及以上版本：

  请根据需求自行添加开源版本所支持的POM依赖包。 下载查看完整依赖包示例。

  > 说明：
  > 如果您需要依赖Snapshot版本，可以自行添加Snapshot版本所支持的POM依赖包。

注册使用

1. 登录实时计算控制台。

2. 单击顶部菜单中的开发，进入开发页面。
3. 在左侧的导航栏中单击资源引用。
4. 在资源引用页签的右上角，单击新建资源。
5. 在上传资源页面输入资源配置信息。

参数名称	说明
上传方式	当前仅支持本地上传。
资源选择	单击选择资源，选择需要引用的资源。
资源名称	输入资源名称。
资源备注	输入资源备注信息。
资源类型	选择引用资源类型，JAR、DICTIONARY或PYTHON。

6. 在资源引用页签中，将鼠标悬停在对应作业的右侧的更多上。
7. 选择下拉列表中的引用，将资源中的代码输入实时计算作业编辑窗口。
8. 在作业的编辑窗口的顶部，输入自定义函数声明，示例如下。

```
CREATE FUNCTION stringLengthUdf AS 'com.hjc.test.blink.sql.udx.
StringLengthUdf';
```

参数与返回值类型

实时计算支持定义Java UDX时，使用Java类型作为参数和返回值。下面为实时计算类型和Java类型的映射关系。

实时计算数类型	Java类型
TINYINT	java.lang.Byte
SAMLLINT	java.lang.Short
INT	java.lang.Integer
BIGINT	java.lang.Long
FLOAT	java.lang.Float
DOUBLE	java.lang.Double
DECIMAL	java.math.BigDecimal
BOOLEAN	java.lang.Boolean
DATE	java.sql.Date
TIME	java.sql.Time

实时计算数类型	Java类型
TIMESTAMP	java.sql.Timestamp
CHAR	java.lang.Character
STRING	java.lang.String
VARBINARY	java.lang.byte[]
ARRAY	暂不支持
MAP	暂不支持

自定义函数参数获取

在UDX中提供了可选的open(FunctionContext context)方法，FunctionContext具备参数传递功能，自定义配置项可以通过此对象来传递。

假设需要在作业中添加如下2个参数。

```
testKey1=lincoln
test.key2=todd
```

以UDTF为例，在open方法中通过context.getJobParameter即可获取。示例如下。

```
public void open(FunctionContext context) throws Exception {
      String key1 = context.getJobParameter("testKey1", "empty");
      String key2 = context.getJobParameter("test.key2", "empty");
      System.err.println(String.format("end open: key1:%s, key2:%s",
key1, key2));
}
```

说明:

具体的作业参数可参见作业参数。

8.12.2 自定义标量函数（UDF）

本文为您介绍如何为实时计算自定义标量函数（UDF）搭建开发环境、编写业务代码以及上线。

说明:

目前阿里云实时计算共享模式暂不支持自定义函数，仅独享模式支持自定义函数。

定义

用户定义的标量函数（UDF）将0个、1个或多个标量值映射到一个新的标量值。

开发环境搭建

请您参见#unique_93/unique_93_Connect_42_section_ck2_gcm_cgb。

编写业务逻辑代码

UDF需要在ScalarFunction类中实现eval方法。open方法和close方法可选。以Java为例，示例代码如下。

```
package com.hjc.test.blink.sql.udx;

import org.apache.flink.table.functions.FunctionContext;
import org.apache.flink.table.functions.ScalarFunction;

public class StringLengthUdf extends ScalarFunction {
    // 可选，open方法可以不编写。
    // 如果编写open方法需要声明'import org.apache.flink.table.functions.
FunctionContext;'。
    @Override
    public void open(FunctionContext context) {
        }
    public long eval(String a) {
        return a == null ? 0 : a.length();
    }
    public long eval(String b, String c) {
        return eval(b) + eval(c);
    }
    //可选，close方法可不写。
    @Override
    public void close() {
        }
}
```

上线

找到您指定的Class编写SQL语句后，点击上线，进入运维页面点击启动即可。

```
-- udf str.length()
CREATE FUNCTION stringLengthUdf AS 'com.hjc.test.blink.sql.udx.
StringLengthUdf';
create table sls_stream(
 a int,
 b int,
 c varchar
) with (
 type='sls',
 endPoint='yourEndpoint',
 accessKeyId='yourAccessId',
 accessKeySecret='yourAccessSecret',
 startTime = '2017-07-04 00:00:00',
 project='yourProjectName',
 logStore='yourLogStoreName',
 consumerGroup='consumerGroupTest1'
);
create table rds_output(
 id int,
 len bigint,
 content VARCHAR
) with (
 type='rds',
 url='yourDatabaseURL',
 tableName='yourDatabaseTableName',
 userName='yourDatabaseUserName',
 password='yourDatabasePassword'
```

```
);
insert into rds_output
select
 a,
 stringLengthUdf(c),
 c as content
from sls_stream
```

常见问题

Q：为什么实现了一个随机数生成函数，在运行时产生的值却是一样的?

A：如果自定义函数是无参的，并且没有声明是非确定性函数，编译期间可能会被优化掉变成一个常量值。所以如果想让函数变成非确定性函数，不被优化成常量，可以在自定义函数中override `isDeterministic()` 方法， 返回 `false` 即可。

8.12.3 自定义聚合函数（UDAF）

本文为您介绍如何为实时计算自定义聚合函数（UDAF）搭建开发环境、编写业务代码以及上线。

 说明:

目前阿里云实时计算共享模式暂不支持自定义函数，仅独享模式支持自定义函数。

定义

自定义聚合函数（UDAF）将多条记录聚合成1条记录。

UDAF抽象类内部方法

AggregateFunction的核心接口方法，如下所示。

 说明:

虽然UDAF可以用Java或者Scala实现，但是建议您使用Java，因为Scala的数据类型有时会造成不必要的性能损失。

- createAccumulator和getValue方法

```
/*
* @param <T> UDAF的输出结果的类型。
* @param <ACC> UDAF的accumulator的类型。accumulator是UDAF计算中用来存放计算中间结果的数据类型。您可以需要根据需要自行设计每个UDAF的accumulator。
*/
public abstract class AggregateFunction<T, ACC> extends UserDefine
dFunction {
/*
* 初始化AggregateFunction的accumulator。
* 系统在第一个做aggregate计算之前调用一次这个方法。
*/
public ACC createAccumulator();
/*
* 系统在每次aggregate计算完成后调用这个方法。
*/
```

```
public T getValue(ACC accumulator);
}
```

说明:

- createAccumulator和getValue可以定义在AggregateFunction抽象类内。
- UDAF必须包含1个accumulate方法。

- accumulator方法

```
public void accumulate(ACC accumulator, ...[用户指定的输入参数]...);
```

说明:

- 您需要实现一个accumulate方法，来描述如何计算用户的输入的数据，并更新到accumulator中。
- accumulate方法的第一个参数必须是使用AggregateFunction的ACC类型的accumulator。在系统运行过程中，底层runtime代码会把历史状态accumulator，和您指定的上游数据（支持任意数量，任意类型的数据）做为参数，一起发送给accumulate计算。

- retract和merge方法

createAccumulator、getValue 和 accumulate3个方法一起使用，就能设计出一个最基本的UDAF。但是实时计算一些特殊的场景需要您提供retract和merge两个方法才能完成。

在实时计算的场景里，很多时候的计算都是对无限流的一个提前的观测值（early firing）。既然有early firing，就会有对发出的结果的修改，这个操作叫做撤回（retract）。SQL翻译优化器会帮助 您自动判断哪些情况下会产生撤回的数据，哪些操作需要处理带有撤回标记的数据。但是您需要实现一个retract方法来处理撤回的数据。

```
public void retract(ACC accumulator, ...[您指定的输入参数]...);
```

说明:

- retract方法是accumulate方法的逆操作。例如，count UDAF，在accumulate的时，每来一条数据要加1，在retract的时候就是要减1。
- 类似于accumulate方法，retract方法的第1个参数必须是使用AggregateFunction的ACC类型的accumulator。在系统运行过程中，底层runtime代码会把历史状态

accumulator，和您指定的上游数据（任意数量， 任意类型的数据）一起发送给retract计算。

在实时计算中一些场景需要merge，例如，session window。 由于实时计算具有out of order的特性，后输入的数据有可能位于2个原本分开的session中间，这样就把2个session合为1个session。此时，需要使用merge方法把多个accumulator合为1个accumulator。

```
public void merge(ACC accumulator, Iterable<ACC> its);
```

说明:

- merge方法的第1个参数，必须是使用AggregateFunction的ACC类型的accumulator，而且第1个accumulator是merge方法完成之后，状态所存放的地方。
- merge方法的第2个参数是1个ACC type的accumulator遍历迭代器，里面有可能存在1个或者多个accumulator。

开发环境搭建

开发环境搭建请参见#unique_93/unique_93_Connect_42_section_ck2_gcm_cgb。

编写业务逻辑代码

以Java为例，举例代码如下。

```
import org.apache.flink.table.functions.AggregateFunction;

public class CountUdaf extends AggregateFunction<Long, CountUdaf.
CountAccum> {
    //定义存放count UDAF状态的accumulator的数据的结构。
    public static class CountAccum {
        public long total;
    }

    //初始化count UDAF的accumulator。
    public CountAccum createAccumulator() {
        CountAccum acc = new CountAccum();
        acc.total = 0;
        return acc;
    }

    //getValue提供了，如何通过存放状态的accumulator，计算count UDAF的结果的方
法。
    public Long getValue(CountAccum accumulator) {
        return accumulator.total;
    }

    //accumulate提供了，如何根据输入的数据，更新count UDAF存放状态的
accumulator。
    public void accumulate(CountAccum accumulator, Object iValue) {
        accumulator.total++;
    }

    public void merge(CountAccum accumulator, Iterable<CountAccum> its
) {
```

```
            for (CountAccum other : its) {
                accumulator.total += other.total;
            }
        }
}
```

说明:

AggregateFunction的子类支持open和close方法作为可选方法，可参考UDF或UDTF的写法。

上线和启动

自定义聚合函数（UDAF）的上线和启动步骤，请参见#unique_7、#unique_8和#unique_9。

UDAF DEMO

```
-- UDAF 计算count
CREATE FUNCTION countUdaf AS 'com.hjc.test.blink.sql.udx.CountUdaf';
create table sls_stream(
a int,
b bigint,
c varchar
) with (
type='sls',
endPoint='yourEndpoint',
accessKeyId='yourAccessId',
accessKeySecret='yourAccessSecret',
startTime = '2017-07-04 00:00:00',
project='yourPorjectName',
logStore='stream-test2',
consumerGroup='consumerGroupTest3'
);

create table rds_output(
len1 bigint,
len2 bigint
) with (
type='rds',
url='yourDatabaseURL',
tableName='yourDatabaseTableName',
userName='yourDatabaseUserName',
password='yourDatabasePassword'
);

insert into rds_output
select
count(a),
countUdaf(a)
from sls_stream
```

8.12.4 自定义表值函数（UDTF）

本文为您介绍如何为实时计算自定义表值函数（UDTF）搭建开发环境、编写业务代码以及上线。

说明:

目前阿里云实时计算共享模式暂不支持自定义函数，仅独享模式支持自定义函数。

定义

与自定义的标量函数类似，自定义的表值函数（UDTF）将0个、1个或多个标量值作为输入参数。与标量函数不同，表值函数可以返回任意数量的行作为输出，而不仅是1个值。返回的行可以由1个或多个列组成。

开发环境搭建

参见#unique_93/unique_93_Connect_42_section_ck2_gcm_cgb。

编写业务逻辑代码

UDTF需要在TableFunction类中实现eval方法。open方法和close方法可选。以Java为例，示例代码如下。

```
package com.hjc.test.blink.sql.udx;

import org.apache.flink.table.functions.FunctionContext;
import org.apache.flink.table.functions.TableFunction;

public class SplitUdtf extends TableFunction<String> {

    // 可选，  open方法可不编写。若编写，需要添加声明'import org.apache.flink.
table.functions.FunctionContext;'。
    @Override
    public void open(FunctionContext context) {
        // ... ...
        }

    public void eval(String str) {
        String[] split = str.split("\\|");
        for (String s : split) {
            collect(s);
        }
    }

    // 可选，close方法可不编写。
    @Override
    public void close() {
        // ... ...
        }
}
```

多行返回

UDTF可以通过多次调用collect()实现将1行的数据转为多行返回。

多列返回

UDTF不仅可以做到1行转多行，还可以1列转多列。如果您需要UDTF返回多列，只需要将返回值声明成Tuple或Row。

- 返回值为Tuple

  实时计算支持使用Tuple1到Tuple25，分别定义1个字段到25个字段。用Tuple3来返回3个字段的UDTF示例如下。

```
import org.apache.flink.api.java.tuple.Tuple3;
import org.apache.flink.table.functions.TableFunction;

// 使用Tuple作为返回值，一定要显式声明Tuple的泛型类型，如此例的String、Long和Integer。
public class ParseUdtf extends TableFunction<Tuple3<String, Long,
Integer>> {

public void eval(String str) {
String[] split = str.split(",");
// 以下代码仅作示例，实际业务需要添加更多的校验逻辑。
String first = split[0];
long second = Long.parseLong(split[1]);
int third = Integer.parseInt(split[2]);
Tuple3<String, Long, Integer> tuple3 = Tuple3.of(first, second,
third);
collect(tuple3);
}
}
```

   说明:

  使用Tuple时，字段值不能为null，且最多只能存在25个字段。

- 返回值为Row

  使用Row来实现返回3个字段的UDTF示例如下。

```
import org.apache.flink.table.types.DataType;
import org.apache.flink.table.types.DataTypes;
import org.apache.flink.table.functions.TableFunction;
import org.apache.flink.types.Row;

public class ParseUdtf extends TableFunction<Row> {

public void eval(String str) {
String[] split = str.split(",");
String first = split[0];
long second = Long.parseLong(split[1]);
int third = Integer.parseInt(split[2]);
Row row = new Row(3);
row.setField(0, first);
row.setField(1, second);
row.setField(2, third);
collect(row);
}

@Override
// 如果返回值是Row，则必须重载实现getResultType方法，显式地声明返回的字段类型。
public DataType getResultType(Object[] arguments, Class[] argTypes)
 {
return DataTypes.createRowType(DataTypes.STRING, DataTypes.LONG,
DataTypes.INT);
}
```

```
}
```

说明:

Row的字段值可以是null，但如果需要使用Row，必须重载实现`getResultType`方法。

SQL 语法

UDTF支持cross join和left join，在使用UDTF时需要添加 `lateral`和`table`关键字。以上述的`ParseUdtf`为例，需要先注册一个function名字。

```
CREATE FUNCTION parseUdtf AS 'com.alibaba.blink.sql.udtf.ParseUdtf';
```

cross join：左表的每一行数据都会关联上UDTF 产出的每一行数据，如果UDTF不产出任何数据，那么这1行不会输出。

```
select S.id, S.content, T.a, T.b, T.c
from input_stream as S,
lateral table(parseUdtf(content)) as T(a, b, c);
```

left join：左表的每一行数据都会关联上UDTF 产出的每一行数据，如果UDTF不产出任何数据，那么这1行的UDTF的字段会用null值填充。

说明:

left join UDTF 语句后面必须接 `on true`参数。

```
select S.id, S.content, T.a, T.b, T.c
from input_stream as S
left join lateral table(parseUdtf(content)) as T(a, b, c) on true;
```

上线和启动

自定义表值函数（UDTF）的上线和启动步骤，请参见#unique_7、#unique_8和#unique_9。

UDAF DEMO

```
-- UDTF str.split("\\|");
create function splitUdtf as 'com.hjc.test.blink.sql.udx.SplitUdtf';

create table sls_stream(
a INT,
b BIGINT,
c VARCHAR
) with (
type='sls',
endPoint='yourEndpoint',
accessKeyId='yourAccessKeyId',
accessKeySecret='yourAccessSecret',
startTime = '2017-07-04 00:00:00',
project='yourProjectName',
logStore='yourLogStoreName',
consumerGroup='consumerGroupTest2'
);
```

```
-- 将c字段传入splitUdtf，切分后得到多行1列的表T(s)。s表示字段名字。
create view v1 as
select a,b,c,s
from sls_stream,
lateral table(splitUdtf(c)) as T(s);

create table rds_output(
id INT,
len BIGINT,
content VARCHAR
) with (
type='rds',
url='yourDatabaseURL',
tableName='yourDatabaseTableName',
userName='yourDatabaseUserName',
password='yourDatabasePassword'
);

insert into rds_output
select
a,b,s
from v1
```

8.12.5 使用IntelliJ IDEA开发自定义函数

本文为您介绍如何使用IntelliJ IDEA开发实时计算自定义函数，包括搭建开发环境和将自定义函数引用到实时计算作业中。

说明:

- 本文档基于IntelliJ IDEA开发工具，请点击下载该工具。
- 目前阿里云实时计算共享模式暂不支持自定义函数，仅独享模式支持自定义函数。

下载及配置Maven

1. 下载Maven
   a. 打开Maven官网下载页面，下载*apache-maven-3.5.3-bin.tar.gz*。
   b. 解压下载的安装包到指定目录，例如：*/Users/xxx/Documents/maven*。
2. 配置环境变量
   a. 打开Terminal，输入`vim ~/.bash_profile`。
   b. 打开*.bash_profile*文件，在次文件中添加设置环境变量的命令。

```
export M2_HOME=/Users/xxx/Documents/maven/apache-maven-3.5.3
export PATH=$PATH:$M2_HOME/bin
```

   c. 添加之后保存并退出，执行以下命令使配置生效。`source ~/.bash_profile`

3. 查看配置是否生效

   输入`mvn -v`命令。如果打印如下信息，则说明配置生效。

```
Apache Maven 3.5.0 (ff8f5e7444045639af65f6095c62210b5713f426; 2017-
04-04T03:39:06+08:00)
Maven home: /Users/****/Documents/maven/apache-maven-3.5.0
Java version: 1.8.0_121, vendor: Oracle Corporation
Java home: /Library/Java/JavaVirtualMachines/jdk1.8.0_121.jdk/
Contents/Home/jre
Default locale: zh_CN, platform encoding: UTF-8
OS name: "mac os x", version: "10.12.6", arch: "x86_64", family: "
mac"
```

开发环境搭建

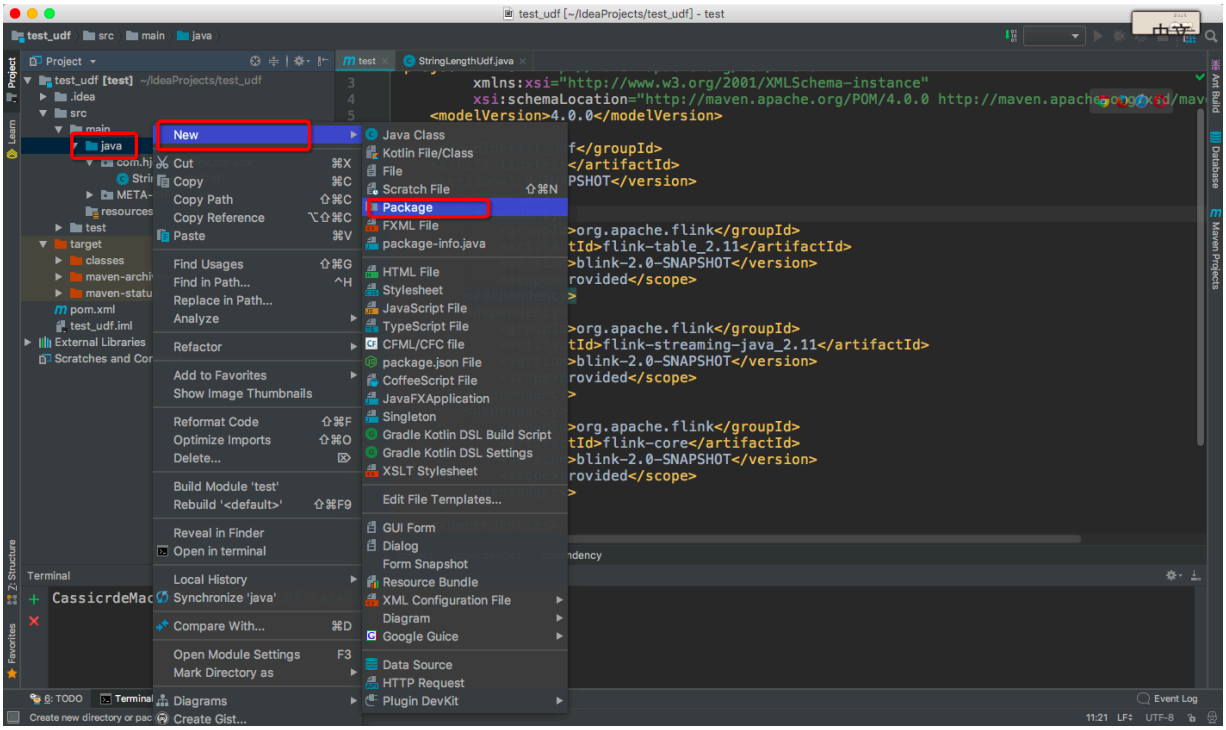
1. 在#unique_93/unique_93_Connect_42_section_ck2_gcm_cgb页面直接下载Demo（*RealtimeCompute-udxDemo.gz*文件）。
2. 在Linux环境下解压*RealtimeCompute-udxDemo.gz*。

```
tar xzvf RealtimeCompute-udxDemo.gz
```

3. 打开IntelliJ IDEA，点击Open打开以上Demo即可。

创建Package

操作如下图所示。

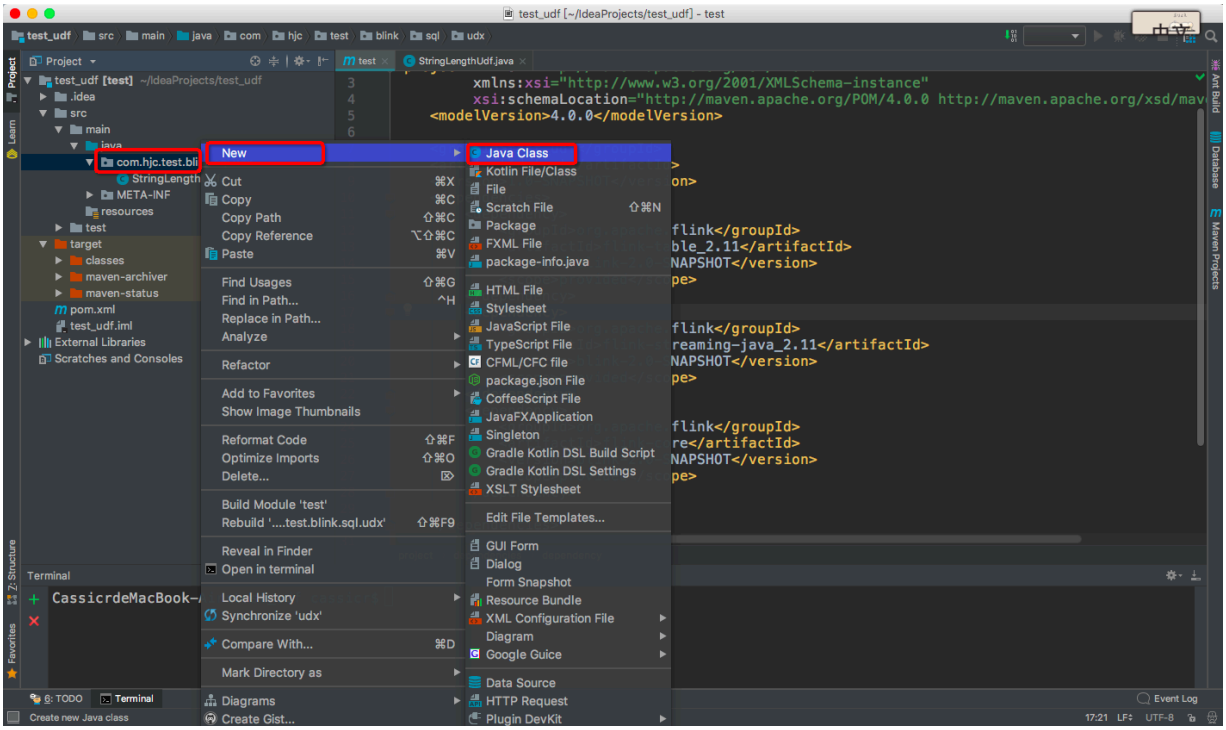


本文以`com.hjc.test.blink.sql.udx`为例，如下图所示。

```
package com.hjc.test.blink.sql.udx;
```

创建Class

操作如下图所示。



将代码粘贴进Class中测试

UDF示例代码如下所示。

```
package com.hjc.test.blink.sql.udx;

import org.apache.flink.table.functions.FunctionContext;
import org.apache.flink.table.functions.ScalarFunction;

public class StringLengthUdf extends ScalarFunction {
    // 可选，  open方法可以不写
    // 需要import org.apache.flink.table.functions.FunctionContext;
    @Override
    public void open(FunctionContext context) {
        }
    public long eval(String a) {
        return a == null ? 0 : a.length();
    }
    public long eval(String b, String c) {
        return eval(b) + eval(c);
    }
    //可选，close方法可以不写
    @Override
    public void close() {
        }
}
```

将项目写入JAR包

1. **在Terminal中输入**`mvn package`**或输入：**`mvn assembly:assembly`。

   说明：

   若需要将第三方依赖写入JAR包，请使用`mvn assembly:assembly`。

2. 编译后的JAR包为：`RealtimeCompute-udxDemo/target/RTCompute-udx-1.0-SNAPSHOT.jar`或`RealtimeCompute-udxDemo/target/RTCompute-udx-1.0-SNAPSHOT-jar-with-dependencies.jar`（将第三方依赖写入JAR包）。

将JAR包引用到实时计算作业中

步骤请参见#unique_363/unique_363_Connect_42_section_5q4_9pi_lny。