ALIBABA CLOUD

阿里云

实时计算（流计算）
Flink全托管

文档版本：20220317

阿里云

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
危险	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。	危险 重置操作将丢失用户配置数据。
警告	该类警示信息可能会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，恢复业务时间约十分钟。
注意	用于警示信息、补充说明等，是用户必须了解的内容。	注意 权重设置为0，该服务器不会再接受新请求。
说明	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。	单击**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。	在**结果确认**页面，单击**确定**。
`Courier字体`	命令或代码。	执行 `cd /d C:/window` 命令，进入Windows系统文件夹。
斜体	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。	`switch {active|stand}`

目录

1.产品公告 14
2.新功能发布记录 15
2.1. 2022-03-04版本 15
2.2. 2021-12-16版本 18
2.3. 2021-08-04版本 19
2.4. 2021-06-02版本 20
2.5. 2021-04-19版本 20
2.6. 2021-03-16版本 21
2.7. 2021-01-20版本 23
2.8. 2021-01-11版本 23
2.9. 2020-12-03版本 24
2.10. 2020-10-27版本 24
2.11. 2020-07-23版本 25
3.产品计费 26
3.1. 计费概述 26
3.2. 计量项和计费项 26
3.3. 计费方式 29
3.3.1. 包年包月 29
3.3.2. 按量付费 30
3.4. 到期或欠费后影响 31
3.5. 续费 32
3.6. 查看消费明细 33
3.7. 退款说明 35
4.快速入门 36
4.1. Flink SQL作业快速入门 36
4.2. Flink JAR作业快速入门 41

4.3. 数据实时入仓入湖快速入门 49
5.实例管理 60
5.1. 开通流程 60
5.2. 创建与管理项目空间 63
5.3. 资源变配 64
5.4. 修改虚拟交换机 65
5.5. 操作审计日志 66
6.权限管理 69
6.1. 阿里云账号角色授权 69
6.2. RAM用户授权 79
6.3. 作业操作账号授权 84
7.作业开发 87
7.1. 配置开发测试环境（Session集群） 87
7.2. Flink SQL开发指南 90
7.2.1. 概述 91
7.2.2. 模板中心 91
7.2.2.1. 代码模板 91
7.2.2.2. 数据同步模板 93
7.2.3. 作业开发 98
7.2.4. 作业启动 102
7.2.5. 作业暂停与停止 105
7.2.6. 作业调试 107
7.2.7. 管理自定义函数（UDF） 110
7.2.8. 管理Hive Metastore 112
7.2.9. 管理Hologres Catalog 117
7.2.10. 管理MySQL Catalog 123
7.2.11. 管理DLF Catalog 129
7.2.12. 管理自定义Connectors 138

7.3. Flink Datastream开发指南 141
7.3.1. 概述 141
7.3.2. 作业开发 142
7.3.3. 作业提交 145
7.3.4. 作业启动 150
7.3.5. 作业暂停与停止 153
7.4. Flink Python开发指南 154
7.4.1. 概述 154
7.4.2. 作业开发 154
7.4.3. 作业提交 157
7.4.4. 作业启动 161
7.4.5. 作业暂停与停止 163
7.4.6. 使用Python依赖 165
8.作业运维 170
8.1. 作业管理 170
8.1.1. 查看作业详情 170
8.1.2. 查看与修改作业状态 170
8.1.3. 修改作业配置 172
8.1.4. 管理作业版本 172
8.1.5. 变量配置 176
8.2. 作业调优 178
8.2.1. 查看作业性能 178
8.2.2. 配置自动调优 180
8.2.3. 配置细粒度资源 184
8.2.4. 高性能Flink SQL优化技巧 188
8.3. 作业监控 196
8.3.1. 配置监控告警（在Flink全托管开发控制台配置） 196
8.3.2. 配置监控告警（在ARMS控制台中配置） 202

8.3.3. 查看监控指标 203
8.3.4. 查看风险与异常作业 208
8.3.5. 作业诊断 210
8.4. 作业日志 211
8.4.1. 动态修改日志级别 211
8.4.2. 查看运行异常日志 213
8.4.3. 查看运行事件 217
8.4.4. 查看启动和运行日志 218
8.4.5. 配置历史作业实例日志输出 220
8.4.6. 查看历史作业实例日志 246
9.开发参考 250
9.1. 支持的上下游存储 250
9.2. Flink SQL参考 252
9.2.1. 概述 253
9.2.2. 关键字 254
9.2.3. 创建数据视图 255
9.2.4. DDL数据定义语句 255
9.2.4.1. 创建数据源表 255
9.2.4.1.1. 日志服务SLS源表 255
9.2.4.1.2. 消息队列Kafka源表 259
9.2.4.1.3. 数据总线DataHub源表 272
9.2.4.1.4. 全量MaxCompute源表 275
9.2.4.1.5. 增量MaxCompute源表 278
9.2.4.1.6. 消息队列RocketMQ版源表 282
9.2.4.1.7. 交互式分析Hologres源表 287
9.2.4.1.8. 全量Elasticsearch源表 293
9.2.4.1.9. Postgres的CDC源表（公测中） 295
9.2.4.1.10. MySQL的CDC源表 298

9.2.4.1.11. Hudi源表 306
9.2.4.1.12. Upsert Kafka源表 314
9.2.4.1.13. Datagen源表 316
9.2.4.1.14. 全量数据湖Iceberg源表 317
9.2.4.1.15. 模拟数据生成源表 323
9.2.4.2. 创建数据结果表 326
9.2.4.2.1. 日志服务SLS结果表 327
9.2.4.2.2. 消息队列Kafka结果表 328
9.2.4.2.3. 数据总线DataHub结果表 330
9.2.4.2.4. 表格存储Tablestore结果表 333
9.2.4.2.5. MaxCompute结果表 335
9.2.4.2.6. 消息队列RocketMQ版结果表 340
9.2.4.2.7. 云数据库HBase版结果表 343
9.2.4.2.8. 云数据库Redis版结果表 348
9.2.4.2.9. 交互式分析Hologres结果表 352
9.2.4.2.10. 云数据库MongoDB版结果表 356
9.2.4.2.11. Elasticsearch结果表 358
9.2.4.2.12. Phoenix5结果表 361
9.2.4.2.13. 云数据库RDS MySQL结果表 363
9.2.4.2.14. 云原生数据仓库AnalyticDB MySQL版3.0结果表 367
9.2.4.2.15. ClickHouse结果表 371
9.2.4.2.16. print结果表 377
9.2.4.2.17. Upsert Kafka结果表 378
9.2.4.2.18. Hudi结果表 381
9.2.4.2.19. Blackhole结果表 393
9.2.4.2.20. InfluxDB结果表 394
9.2.4.2.21. 数据湖Iceberg结果表 397
9.2.4.2.22. MySQL结果表 406

9.2.4.2.23. 云原生数据仓库AnalyticDB PostgreSQL版结果表 409
9.2.4.3. 创建数据维表 412
9.2.4.3.1. 云数据库HBase版维表 412
9.2.4.3.2. 表格存储Tablestore维表 416
9.2.4.3.3. MaxCompute维表 418
9.2.4.3.4. 云数据库Redis版维表 422
9.2.4.3.5. Elasticsearch维表 425
9.2.4.3.6. 交互式分析Hologres维表 428
9.2.4.3.7. 云数据库RDS MySQL维表 432
9.2.4.3.8. 云原生数据仓库AnalyticDB MySQL版3.0维表 436
9.2.4.3.9. FileSystem维表 440
9.2.4.3.10. MySQL维表 441
9.2.4.3.11. 模拟数据生成维表 446
9.2.4.3.12. 云原生数据仓库AnalyticDB PostgreSQL版维表 449
9.2.5. DML数据操作语句 452
9.2.5.1. INSERT INTO语句 452
9.2.5.2. CREATE TABLE AS(CTAS)语句 453
9.2.5.3. CREATE DATABASE AS(CDAS)语句 458
9.2.5.4. AUTO OPTIMIZE语句 462
9.2.6. DQL数据查询语句 464
9.2.7. 窗口函数 464
9.2.7.1. 概述 464
9.2.7.2. 滚动窗口 466
9.2.7.3. 滑动窗口 470
9.2.7.4. 会话窗口 474
9.2.7.5. OVER窗口 477
9.2.8. 内置函数 485
9.2.8.1. 概述 485

9.2.8.2. 标量函数 485
9.2.8.2.1. BITAND 485
9.2.8.2.2. BITNOT 486
9.2.8.2.3. BITOR 487
9.2.8.2.4. BITXOR 488
9.2.8.2.5. CONV 489
9.2.8.2.6. DATEDIFF 490
9.2.8.2.7. DATE_ADD 491
9.2.8.2.8. DATE_FORMAT 492
9.2.8.2.9. DATE_FORMAT_TZ 493
9.2.8.2.10. DATE_SUB 494
9.2.8.2.11. FROM_UNIXTIME 495
9.2.8.2.12. ISNAN 496
9.2.8.2.13. JSON_VALUE 497
9.2.8.2.14. KEYVALUE 498
9.2.8.2.15. MURMUR_HASH 499
9.2.8.2.16. MURMUR_HASH_64 501
9.2.8.2.17. PARSE_URL 502
9.2.8.2.18. REGEXP 504
9.2.8.2.19. REGEXP_REPLACE 505
9.2.8.2.20. SPLIT_INDEX 506
9.2.8.2.21. TO_DATE 507
9.2.8.2.22. TO_TIMESTAMP_TZ 508
9.2.8.2.23. UNIX_TIMESTAMP 509
9.2.8.3. 表值函数 510
9.2.8.3.1. GENERATE_SERIES 510
9.2.8.3.2. JSON_TUPLE 511
9.2.8.3.3. MULTI_KEYVALUE 512

9.2.8.3.4. STRING_SPLIT 514
9.2.8.4. 聚合函数 515
9.2.8.4.1. APPROX_COUNT_DISTINCT 515
9.2.8.4.2. FIRST_VALUE 516
9.2.8.4.3. LAST_VALUE 518
9.2.8.5. 机器学习函数CLUSTER_SERVING 519
9.2.9. 自定义函数 521
9.2.9.1. Java 521
9.2.9.1.1. 概述 521
9.2.9.1.2. 自定义标量函数（UDF） 524
9.2.9.1.3. 自定义聚合函数（UDAF） 526
9.2.9.1.4. 自定义表值函数（UDTF） 529
9.2.9.2. Python 532
9.2.9.2.1. 概述 532
9.2.9.2.2. 自定义标量函数（UDF） 534
9.2.9.2.3. 自定义聚合函数（UDAF） 535
9.2.9.2.4. 自定义表值函数（UDTF） 537
9.3. Flink State兼容性参考 539
9.3.1. 概述 539
9.3.2. SQL修改及兼容性 541
9.3.2.1. 概述 541
9.3.2.2. 修改Query 542
9.3.2.2.1. Alias 542
9.3.2.2.2. Group Aggregate 542
9.3.2.2.3. Over Aggregate 544
9.3.2.2.4. Group Window Aggregate 547
9.3.2.2.5. Window TVF 551
9.3.2.2.6. Deduplication 555

9.3.2.2.7. Top-N 558
9.3.2.2.8. Window Top-N 560
9.3.2.2.9. Join 563
9.3.2.2.10. Pattern Recognition 564
9.3.2.2.11. Where 564
9.3.2.2.12. Union All 564
9.3.2.2.13. Temporal Sort 565
9.3.2.2.14. Limit 566
9.3.2.3. 修改Source 566
9.3.2.4. 修改Sink 569
9.3.2.5. 其他限制 572
9.3.3. 状态迁移 573
9.4. Flink Datastream参考 575
9.4.1. MaxCompute DataStream Connector 575
9.4.2. DataHub DataStream Connector 581
9.4.3. Kafka DataStream Connector 584
9.4.4. MySQL CDC DataStream Connector 593
9.5. GeminiStateBackend参考 597
9.5.1. GeminiStateBackend介绍 597
9.5.2. GeminiStateBackend配置 598
10.常见问题 604
10.1. 基本概念 604
10.2. 计费常见问题 607
10.3. 操作指导 609
10.4. 分场景排错指引 619
10.5. 上下游存储 625
10.6. SQL常见问题 635
10.7. State常见问题 642

11.视频专区 647
11.1. 什么是阿里云实时计算Flink版 647
11.2. Flink全托管开通流程 647
11.3. Flink全托管产品使用 647
11.4. MySQL实时同步至Hologres 647
12.相关协议 648
12.1. Flink全托管服务等级协议（SLA） 648

1.产品公告

本文为您介绍Flink全托管公告内容，包括版本更新、功能更新和产品活动等。

RocketMQ接入点变更导致实时计算作业适配升级公告

- 升级公告

  因消息队列RocketMQ接入点地域化变更（详情请参见关于TCP内网接入点设置的公告），如果您已使用了VVR 3.0.1以下版本的RocketMQ Connector，则您需要将您的实时计算作业升级至VVR 3.0.1及以上版本，并将作业中EndPoint参数取值更改为新的RocketMQ接入点，EndPoint参数详情请参见：

  - Flink RocketMQ源表文档：消息队列RocketMQ版源表。
  - Flink RocketMQ结果表文档：消息队列RocketMQ版结果表。

- 注意事项
  - 旧的RocketMQ接入点在2021年11月后完全不可用，且使用旧的RocketMQ接入点的作业可能存在稳定性风险，因此请您尽快安排作业升级，务必在2021年11月之前完成升级工作。
  - RocketMQ产品承诺：2021年11月前不会下线旧的RocketMQ接入点，但旧的RocketMQ接入点无法保证作业稳定性。
  - 本次升级会导致作业的State无法兼容，请结合业务情况合理安排升级时间，尽量减少对业务的影响。
  - 2021年11月1日以后，实时计算Flink版产品侧不再对使用了旧的RocketMQ接入点的作业进行维护和支持。

Flink全托管公测转商业化通知

Flink全托管服务于2020年9月15日结束公测，进行商业化收费。为了保证您业务平稳过渡，请各位用户评估业务影响。针对在申请公测时间过长的用户，我们将免费服务期限统一为2020年9月30日 23:59:59，特此通知。

商业化产品定价详情请参见计量项和计费项，如果您有其他问题，请提交工单进行咨询。

2020年7月23日21:00-2020年7月23日24:00-升级公告

2020年7月23日21:00-2020年7月23日24:00，对**Flink全托管**开发控制台进行升级，升级期间开发控制台不可用，如果您有紧急情况，请您提交工单。

2.新功能发布记录

2.1. 2022-03-04版本

本文为您介绍2022年3月4日发布的Flink全托管版本的重大功能变更和主要缺陷修复。

概述

2022年3月4日正式对外发布VVR 4.0.12版本，该版本是基于Apache Flink 1.13。在本次新版本中，对于常见的Kafka->Flink->Hologres链路，我们支持JSON Schema变化自适应；在数据湖构建上，我们发布了企业级Hudi Connector；在开发效能提升上，我们提供了超过二十种常见的Flink SQL作业模板；在运维服务能力增强上我们提供了强大的作业诊断能力和不停止作业，可动态调整日志级别的能力；还包含了Clickhouse的企业级特性、新的Connector、新的数据入仓入湖语法特性等诸多强大且丰富的数据处理能力。另外，本次新版本还同步修改了若干已在Apache Flink社区修复的缺陷。

> **说明** 目前该版本已经在公共云上逐步上线，2022年3月15日前会发布完毕。如果您发现新功能未开放，请耐心等待。

新特性

特性	详情	相关文档
Hologres自适应JSON Schema结构变化	JSON作为流式数据处理中最为常见的Event格式之一，其Schema的变化对于实时流作业及其后端的存储引擎中的表而言，都希望是一个透明的过程。 在本次新版本中，我们对于该需求进行了以下增强： • 在消费JSON前，可以根据JSON的Schema进行表结构的设定。 • 在后续持续消费时，如果JSON的Schema发生变化，后端Hologres表的Schema也会根据此变化而变化。	• 管理Hologres Catalog • 交互式分析Hologres结果表

增强Iceberg和Hudi数据湖的构建能力	• 支持阿里云DLF作为Catalog。
	通过DLF Catalog，您可以访问Hudi或Iceberg以及其他DLF支持的引擎，帮助您快速构建实时数据湖。
	• 支持优化Iceberg表的小文件。
	通过AUTO OPTIMIZE语句，启动一个流式优化任务，自动地将Iceberg表的若干小文件重写为大文件。
	• 推出Flink全托管内置企业级Hudi Connector，降低运维复杂度。
	  ○ 支持使用Flink CDC打通数据库整库入湖，并自动同步表结构变更。
	  ○ 集成阿里云OSS和DLF等组件，完善数据在计算引擎间的连通性。
		• 管理DLF Catalog
		• AUTO OPTIMIZE语句
		• Hudi结果表

特性	详情	相关文档
提升日志查看和设置的易用性	• 新增日志分页显示。 在作业探查页签中，新增日志分页的能力，避免作业长期运行会导致日志过大页面无法打开的问题。 • 支持动态修改日志级别。 在不重启作业的前提下，可以在作业探查页签中，动态修改运行中的TM的日志级别，以满足您定位问题的需求。 • 支持查看失败的TM日志。 在作业探查页签中，支持展示JM还在运行情况下已经失败的TM的日志，方便您排查TM失败的原因。	• 查看启动和运行日志 • 动态修改日志级别 • 查看失效Task Managers日志
提供Flink+Clickhouse多个企业级特性	• 支持Exactly Once语义。 针对开源大数据平台E-MapReduce中的ClickHouse组件（非云Clickhouse产品）提供Exactly Once语义。 • 支持ClickHouse的Nested类型。 对于ClickHouse的Nested类型，可以将其映射为Flink的Array类型。 • 支持直接写ClickHouse分布式表的本地表。 通过对分布式表的本地表进行直接写入的方式，可以显著提高写ClickHouse分布式表的吞吐量。	ClickHouse结果表
优化作业诊断规则和界面	• 新增20多种诊断规则，全面分析作业的运行状态。 根据作业的实际情况，给出高、中、低三种风险等级提示。 • 优化诊断界面的，协助您更好地查看问题。	作业诊断
数据同步支持新增计算列	CTAS语句支持在Source表上新增计算列，并修改目标表的主键为新增列。 在进行数据入仓入湖时，CTAS语句允许指定新增计算列的位置，并把它作为目标表的物理列，实时地将计算列的结果同步到目标表中。同时，CTAS语句也支持修改目标表的主键，把新增列作为目标表的主键字段。	CREATE TABLE AS(CTAS)语句
更便捷地生成测试数据	新增支持模拟数据生成Connector。 通过模拟数据生成Connector，您可以更便捷地生成贴近业务含义的测试数据，满足您开发测试中验证业务逻辑的需要。	• 模拟数据生成源表 • 模拟数据生成维表

特性	详情	相关文档
新增模版中心，加速作业开发	• 提供20多种代码模板。 20多个Flink SQL常见通用场景的模版，帮助您快速了解如何使用Flink SQL构建作业代码。 • 提供MySQL到Hologres数据同步模版。 帮助您快速创建Flink CDC作业，完成数据同步入仓入湖。	• 代码模板 • 数据同步模板
更清晰地展示资源使用情况	在Flink开发控制台页面左下角，会展示当前项目空间下使用的CPU和Memory情况，方便您快速管理项目资源。	无
通过作业状态标签，快速判断作业运行情况	• 新增异常作业（Falied作业）和风险作业（检测存在运行问题的作业）标签，您可以查看异常和风险作业的个数，并筛选出对应作业。 • 风险作业可以配合诊断功能查看问题产生原因，并根据建议修复。	查看风险与异常作业
快速定位Checkpoint慢节点的日志	在快照历史中，新增对节点快照状态的排序能力，并支持在快照历史界面一键跳转到TM日志中，查看慢Checkpoint原因。	定位慢Checkpoint并查看对应Task Managers的日志
支持云原生数据仓库AnalyticDB PostgreSQL版结果表和维表	• Flink支持将数据写入云原生数据仓库AnalyticDB PostgreSQL版结果表 • Flink支持关联云原生数据仓库AnalyticDB PostgreSQL版，进行关联查询。	• 云原生数据仓库AnalyticDB PostgreSQL版结果表 • 云原生数据仓库AnalyticDB PostgreSQL版维表
提升企业级状态存储后端易用性	• 新增实时进行参数优化调整的能力，最大化降低人工调优的复杂度和成本，可以避免95%以上人工调参的需要。 • 单核吞吐能力提升10%~40%，帮助您轻松应对流量洪峰与低谷等变化场景。	• GeminiStateBackend介绍 • GeminiStateBackend配置 • State常见问题

性能优化

企业级状态存储后端在本次新版本中包含了大量优化，极大提升了双流或多流Join作业的性能，计算资源利用率平均可以提升50%，典型场景下可以提升100% ~ 200%，帮助您更平滑地运行有状态的流计算应用。

缺陷修复

- 优化Catalog服务，解决数据库或表数量较大时刷新不出来的问题。
- 修复Session集群没有显示Flink版本的问题。
- 修复Metric页面WaterMarkLag曲线显示问题。
- 优化Metric页面曲线翻页展示效果。
- 修复Flink CDC currentFetchEventTimeLag指标、类冲突等问题。
- 修复CTAS语法无法修改已有列的问题。

2.2. 2021-12-16版本

本文为您介绍2021-12-16号发布的Flink全托管版本的重大功能变更和主要缺陷修复。

VVP 2.8.0版本重大功能变更

类别	变更	详情
新增	支持作业探查功能 说明 使用该功能需要在作业模版或者单作业的高级设置中开启日志归档。	• 支持在VVP上查看Flink作业的启动日志、运行日志和历史的日志。 详情请参见查看历史作业实例日志和查看启动和运行日志。 • 支持对正在运行的作业TM和JM进行性能分析，包括火焰图、内存动态和线程动态。 详情请参见查看作业性能。
	支持图形化的方式注册Catalog	您可以使用图形化的方式注册Hive、Hologres、MySQL的Catalog。
	支持检测state兼容性	在修改作业选择历史state恢复的情况下，会协助您进行兼容性检测，判断业务影响。
	支持变量替换功能	可以设置通用变量（例如AK等）后，在SQL文本中直接引用。 详情请参见变量配置。
优化	优化SQL细粒度配置	• 支持细粒度断chain分析展示。 • 支持Operator级别的资源配置。
	优化告警信息	在告警信息中增加作业名称，帮助您判断受影响的作业。
	优化UI界面	包括图形化的方式注册Catalog等。

VVR 3.0.7&VVR 4.0.11版本重大功能变更

类别	变更
VVR 3.0.7	支持作业探查功能。
	修复Apache Flink1.12版本已知问题。
	发布Flink CDC商业功能： • 支持Schema evolution。 • 支持整库同步的Flink SQL语义。 详情请参见CREATE TABLE AS(CTAS)语句、CREATE DATABASE AS(CDAS)语句和数据实时入仓入湖快速入门。
	支持Gemini Statebackend将state存储到OSS上。

VVR 4.0.11 类别	变更
	优化作业资源管理方案。
	优化DataStream Connector。 详情请参见MySQL CDC DataStream Connector。
	修复Gemini Backend异常。
	修复Apache Flink社区1.13版本已知问题。

2.3. 2021-08-04版本

本文为您介绍2021-08-04号发布的Flink全托管版本的重大功能变更和主要缺陷修复。

VVP 2.5.7版本重大功能变更

- 作业停止后会清理该作业生成的所有Checkpoint，因此如果您需要恢复作业的状态，可以采取以下任何一种方式：
  - 暂停作业后再恢复作业。
  - 停止后从Savepoint恢复作业。
- 新增
  - 开发类功能
    - SQL作业支持配置细粒度资源，即可以配置CPU和内存到Share Slot上。详情请参见配置细粒度资源。
    - 上线作业时，支持生成开发版本，您可以进行版本比较、回滚历史版本等操作。详情请参见管理作业版本。
    - 支持Hologres元数据，可以通过注册Hologres Catalog直接访问下游Hologres表。详情请参见管理Hologres Catalog。
    - 引擎版本命名格式变更为vvr-<vvr版本号>-flink-<flink版本号>。
    - 支持提交多个不同引擎版本的SQL作业。目前仅支持vvr-3.0.3-flink-1.12和vvr-4.0.8-flink-1.13两个版本。

      说明 从VVR 3.0.3版本（对应Flink 1.12版本）开始，VVP支持同时运行多个不同引擎版本的SQL作业。如果您的作业已使用了Flink 1.12及更早版本的引擎，您需要按照以下情况进行处理：
      - Flink 1.12版本：停止后启动作业，系统将自动将引擎升级为vvr-3.0.3-flink-1.12版本。
      - Flink 1.11或Flink 1.10版本：手动将作业引擎版本升级到vvr-3.0.3-flink-1.12或vvr-4.0.8-flink-1.13版本后重启作业，否则会在启动作业时超时报错。
  - 运维类功能
    - 支持自定义设置报警频率。
    - 支持设置Webhook报警。
- 优化
  - 优化上传JAR包的方式，提高上传速度。

- 下载到SLS的日志中新增作业名称标签，方便定位作业问题。

VVR 4.0.8版本重大功能变更

- 支持Holo Catalog。
- 支持细粒度资源配置。
- 云数据库RDS MySQL Connector JDBC版本升级，兼容更多字符集。
- MySQL的CDC Connector支持多并发读取全量数据、无锁算法和断点续传功能。
- 云原生数据仓库AnalyticDB MySQL版3.0 Connector支持ON Depulicate Key Update语义。

VVR 4.0.8版本主要缺陷修复

修复Kafka currentEmitEventTimeLag在某些Partition没有数据时显示Log.MAX的异常。

2.4. 2021-06-02版本

本文为您介绍2021-06-02号发布的Flink全托管版本的重大功能变更和主要缺陷修复。

VVP 2.4.2版本主要缺陷修复

- 修复运行事件部分日志展示不全的问题。
- 修复DataHub源表无法调试的问题。
- 修复VVP界面无法保存配置修改的问题。
- 修复无法跳转到ARMS界面的问题。
- 修复语法校验报错不明显，提示UnknownStatusException异常的问题。
- 修复部分文案和曲线描述不准确的问题。

VVR 3.0.2版本重大功能变更

- 支持获取SLS的Tag字段，详情请参见日志服务SLS源表。
- 新增ClickHouse的Connector，详情请参见ClickHouse结果表。
- 新增CLUSTER_SERVING机器学习函数，详情请参见机器学习函数CLUSTER_SERVING。

2.5. 2021-04-19版本

本文为您介绍2021-04-19号发布的Flink全托管版本的重大功能变更和主要缺陷修复。

VVP 2.4.1版本重大功能变更

- 支持批作业的创建、提交和运维。
- 支持文件夹功能。
- 支持多人协同开发编辑锁功能。
- 支持Python UDF功能，详情请参见概述。
- 新增新手入门场景，帮助您快速进行作业开发。

VVR 3.0.1版本重大功能变更

- 支持VVP新手入门功能定制Connector。
- Flink CDC Connector支持设置延时等指标。
- 消息队列RocketMQ Connector升级底层客户端依赖版本，因此所有的Connector=mq的DDL需要根据

RocketMQ接入点的变更，调整Endpoint参数取值至RocketMQ的TCP内网接入点，详情请参见关于TCP内网接入点设置的公告。

VVR 3.0.1版本主要缺陷修复

- 内置函数支持RAW类型，修复作业升级产生的UDF不兼容问题。
- 修复SLS Connector Taskmanager.log日志中明文显示AccessKey信息的问题。

2.6. 2021-03-16版本

本文为您介绍2021-03-16号发布的Flink全托管版本的重大功能变更和主要缺陷修复。

VVP 2.4.0版本重大功能变更

- 支持Flink 1.12版本。
- 开发控制台界面全新升级，作业上线后，您可以在开发界面继续对作业进行修正而不影响上线作业状态。
- Session模式作业支持设置Checkpoint和Savepoint。
- 优化用户管理界面，可以添加用户名并显示用户名。
- 支持Pyflink作业的提交。详情请参见Flink Python开发指南。

VVP 2.4.0版本主要缺陷修复

修复作业信息另存后，作业脚本丢失的缺陷。

VVR 3.0.0版本重大功能变更

- 支持在SQL JOIN中使用Hints语法。
- Kafka的Avro格式支持用户自定义Schema。
- 支持Kafka、DataHub、SLS和MQ获取元消息列。
- 优化Mysql-CDC功能：支持多种方式启动位点，包括binlog position、timestamp、earliest-offset、latest-offset和initial。
- 新增监控指标：numBytesIn、numBytesInPerSecond、sourceIdleTime、pendingRecords、watermarkLag、numBytesOut、numBytesOutPerSecond、numRecordsOut和numRecordsIn。
- 新增Connector：FileSystem维表。
- 新增内置函数：
  - 标量函数

函数名称				
BITAND	DATEDIFF	FROM_UNIXTIME	MURMUR_HASH_64	TO_DATE
BITNOT	DATE_ADD	ISNAN	PARSE_URL	TO_TIMESTAMP_TZ
BITOR	DATE_FORMAT	JSON_VALUE	REGEXP_EXTRACT	UNIX_TIMESTAMP
BITXOR	DATE_FORMAT_TZ	KEYVALUE	REGEXP_REPLACE	
CONV	DATE_SUB	MURMUR_HASH	SPLIT_INDEX	

- 表值函数

函数名称	
GENERATE_SERIES	JSON_TUPLE
MULTI_KEYVALUE	STRING_SPLIT

- 聚合函数

函数名称		
APPROX_COUNT_DISTINCT	FIRST_VALUE	LAST_VALUE

VVR 3.0.0版本主要缺陷修复

- 修复Savepoint触发后导致Checkpoint文件丢失的缺陷。
- 修复Postgres-CDC WAL日志无限增长导致磁盘空间不足的问题。
- 修复日志中隐藏用户名和密码等敏感信息的缺陷。

注意事项

- SQL编辑器中SQL脚本名称和作业名称变更

  原VVP平台没有做开发和运维分离，即SQL脚本和上线作业间不存在关联关系，因此SQL脚本和上线作业的命名方式未做一致性要求。新版VVP支持开发和运维分离：

  - SQL脚本名和线上作业名一一对应，原SQL脚本名中如果存在中文，升级后将会把中文用随机数替代，原SQL脚本名称将在脚本配置的Label中展示。建议将脚本重命名为线上对应名称。
  - 如果存在重复的SQL脚本名称，将在SQL脚本名后加随机数。您也可以采用重命名的方式更正SQL脚本名称。

- SQL作业Flink版本升级变更

  VVP升级后，内置的SQL编译器版本将升级到Flink 1.12。但是Flink 1.11和Flink 1.12的类不兼容，之前历史的Flink 1.11版本的SQL作业将会运行不成功。因此，对于VVP升级前，基于历史版本已创建并且已运行的SQL作业，您有两个选择：

  - 升级作业的Flink版本到最新的Flink 1.12版本。

    注意

    - 升级后不能进行降级。一旦该SQL作业的Flink版本被更新到最新的Flink 1.12，就不能再进行降级。
    - 升级操作有失败的风险。建议您复制一个新的SQL作业尝试成功后，再在原有SQL作业上操作。

- 不升级作业的Flink版本并继续使用原有版本。

  注意

  - 原有的未上线SQL作业，需要选择Flink 1.12版本运行作业。
  - 原有的已上线的SQL作业将继续使用Flink 1.11版本运行。修改部署目标（所修改的部署目标要有Flink 1.11版本）、资源配置、日志配置、K8s配置、Flink镜像仓库名称、URL地址、Tag这些操作可以继续使用Flink 1.11运行作业。除此之外，其他修改需要将作业版本升级到Flink 1.12。

- 升级策略变更

  新版本将不再支持设置升级策略，全部都使用None模式。

2.7. 2021-01-20版本

本文为您介绍2021-01-20号发布的Flink全托管版本的重大功能变更和主要缺陷修复。

VVP主要缺陷修复

- 修复开发界面作业状态和实际运行作业状态显示不一致的问题。
- 修复UDF更新后语法检查异常的问题。
- 修复查看历史作业日志跳转失败的问题。
- 修复作业从Per-Job模式切换到Session模式时，无法查看Metic的问题。

VVR 2.1.4版本重大功能变更

支持Upset Kafka Connector，详情请参见Upsert Kafka结果表。

2.8. 2021-01-11版本

本文为您介绍2021-01-11号发布的Flink全托管版本的重大功能变更和主要缺陷修复。

VVP重大功能变更

- 支持智能诊断，对运行中的作业可以自动诊断部分问题。
- 支持查看Session模式下作业的Metric指标。
- 支持根据JobID定位作业。
- 支持对Session集群进行缩容。
- 支持在VVP上通过RAM用户名称进行授权。
- 支持Session和Per-Job模式切换。
- 通过VVP上传的JAR包大小限制提高至200 MB。

  说明 如果JAR包超过200 MB，建议对JAR包进行拆分或者通过OSS上传。

VVP主要缺陷修复

修复语法检查较慢的问题。

VVR 2.1.3版本重大功能变更

- VVP支持使用Session模式提交JAR包。
- 自动调优Source-Delay-Based策略支持Kafka。
- 增加Connector Format类型，和Apache Flink社区完全一致。
- 新增包含延迟在内的Metric指标。详情请参见数据曲线。

VVR 2.1.3版本主要缺陷修复

- 修复CDC Connector不支持polarDB重命名操作引起异常的问题。
- 修复Gemini Statebackend部分缺陷导致NullPointException、Savepoint恢复异常的问题。

2.9. 2020-12-03版本

本文为您介绍2020-12-03号发布的Flink全托管版本的重大功能变更和主要缺陷修复。

VVP重大功能变更

- 新增SQL结果预览（SQL Preview）：DML语句可以在SQL编辑器内直接运行，具备基础的调试能力，提升代码开发效率。详情请参见作业调试。
- 新增用户自定义连接器（User-Defined Connector）：对于系统不支持的Source或者Sink，您可以快速扩展新连接器满足需求。详情请参见管理自定义Connectors。
- 新增Session集群（Session-Cluster）：您可自定义Session集群，Session集群可以复用JM资源节省小作业资源消耗，并支持较快地提交作业，适合SQL结果预览等运行时间较短的使用场景。详情请参见配置开发测试环境（Session集群）。

VVR 2.1.2重大功能变更

- 支持TTL设置，在其他参数中配置table.exec.state.ttl: 129600000（单位为毫秒）。
- 支持Print Connector，您可以进行线上调试。
- 支持RocketMQ源表声明计算列。
- 支持MaxCompute源表，详情请参见增量MaxCompute源表。

VVR 2.1.2主要缺陷修复

- 修复Hologres Connector DDL上设置了某个字段为Not Null时，引起作业运行Failover的缺陷。
- 修复RDS未满足batchSize='5000'默认值，不下发数据的缺陷。
- 修复Kafka Connector暂停恢复Failover。
- 修复OTS结果表因WITH参数大小写不正确验证报错的缺陷。

2.10. 2020-10-27版本

本文为您介绍2020-10-27号发布的Flink全托管版本的重大功能变更。

版本重大功能变更

- 支持多项目：您可以创建不同的项目，不同的项目之间资源隔离，您可以针对不同的项目设置不同的权限。详情请参见创建与管理项目空间。
- 支持Hive Metastore：可读取Hive中维护的元数据信息，便于统一管理元数据。详情请参见管理Hive Metastore。

> 说明
> - 仅支持Hive Metastore 2.3.6版本。
> - Hive Metastore暂不支持Kerberos方式认证。

- 支持快速配置监控告警：您可以直接在Flink全托管开发控制台上快速配置报警策略，不用跳转到应用实时监控服务ARMS进行。详情请参见配置监控告警（在Flink全托管开发控制台配置）。
- Kafka支持Kerberos方式认证。

2.11. 2020-07-23版本

本文为您介绍2020-07-23号发布的Flink全托管版本的重大功能变更。

版本重大功能变更

- 支持开源Flink1.11版本。
- 新增多种内置Connector：
  - 源表
    - 日志服务SLS源表
    - 消息队列Kafka源表
    - 数据总线DataHub源表
  - 结果表
    - 日志服务SLS结果表
    - 消息队列Kafka结果表
    - 数据总线DataHub结果表
    - 表格存储Tablestore结果表
    - 云数据库HBase版结果表
    - Elasticsearch结果表
  - 维表
    - 云数据库HBase版维表
    - 表格存储Tablestore维表
    - MaxCompute维表
    - 云数据库Redis版维表
    - Elasticsearch维表
- 支持Flink SQL作业开发，包括：
  - SQL编辑器
  - 元数据管理
  - UDF管理
- 支持单作业Autopilot自动调优。

3.产品计费

3.1. 计费概述

本文介绍Flink全托管的计量项和计费项、计费方式、欠费与停机、续费、查看消费明细、退款说明和计费常见问题相关内容。

相关项	说明	相关文档
计量项和计费项	为您介绍Flink全托管产品的计量项、计费项，以及相关计算公式，以便您对支付的费用组成有所了解。	计量项和计费项
计费方式	Flink全托管支持包年包月和按量付费两种付费模式。通过此部分内容您可以分别了解两种计费方式的特点。	• 包年包月 • 按量付费
欠费与停机	实例欠费会有停机风险，可能会对您的线上业务造成影响。通过此部分内容您可以了解到欠费与停机机制。	到期或欠费后影响
续费	包年包月类型的实例到期后会自动释放。如果您需要继续使用，需要在实例释放前进行续费操作，延长对应实例的使用时间。此部分内容主要为您介绍手动续费实例、自动续费和取消自动续费的具体操作。	续费
查看消费明细	如果您对Flink全托管的消费情况有疑问，可以通过用户中心查看费用账单以及消费明细。	查看消费明细
退款说明	主要为您介绍Flink全托管的退费规则及相关操作。	退款说明
计费常见问题	在使用Flink全托管时，如果您有计费相关的问题，可以优先查看该文档。	计费常见问题

3.2. 计量项和计费项

本文为您介绍Flink全托管产品的计量项、计费项，以及相关计算公式，以便您对支付的费用组成有所了解。

计量项

Flink全托管的基本计量单位为Compute Unit（CU），即计算资源，1 CU=1核CPU+4 GB内存。

> **说明** CU对应实时计算底层系统的CPU计算能力。

1个实时计算作业（Job）的CU使用量取决于此Job输入数据流的QPS、计算复杂程度，以及具体的输入数据分布情况。

实时计算1 CU的处理能力可以通过以下方式估算：

业务类型	处理能力
简单业务 例如，单流过滤、字符串变换等操作。	1 CU每秒可以处理10000条数据。
复杂业务 例如，JOIN、GROUP BY或窗口函数等操作。	1 CU每秒可以处理1000至5000条数据。

您可以根据业务规模以及上述计算能力，估算所需购买的资源数量。

> **说明**
> - 上述计算能力估值仅限于实时计算内部处理能力，不包括对外数据读取和写入部分。外部数据的读写效率会影响您对实时计算能力的评估，例如：
    - 如果实时计算需要从日志服务（LogService）读取数据，但LogService对于请求调用配额（Quota）存在一定限制，则实时计算整体的计算能力将被限制在LogService允许的范围内。
    - 如果实时计算引用的RDS数据存储存在连接数或者TPS限制，则实时计算吞吐能力将受限于RDS本身的流控限制。
- 如果作业中使用窗口函数，CU的使用量会比简单作业高，建议至少购买4 CU。

计费项

Flink全托管的费用由以下两部分组成。

计费项	资源说明
管控资源	创建实例时，系统为每个集群部署一个开发控制台，每个开发控制台及其必要组件需要大概2 CU的管控资源。
计算资源	实际可用于计算的资源，根据您包年包月预购的资源量或按量付费实际的使用量来收费，计量单位为CU。

> **说明** VVR 3.0.4及以下版本的每个Session集群运行后，都会额外消耗0.5 CU的资源。

Flink全托管开通流程中的费用仅包含Flink全托管，在Flink全托管开通和使用过程中还涉及其他云产品的费用，详情请参见下表。

相关产品	计费说明
VPC	收费详情请参见产品计费。
OSS	收费详情请参见计费概述。

相关产品	计费说明
SLB	在您开通Flink全托管集群后，会自动开通SLB服务，单击创建Flink全托管集群后就开始收费，详情请参见按量计费。 **注意** 负载均衡SLB可以让您以公网的形式访问Flink全托管开发控制台，该SLB一旦创建，请不要用于其他情况，并且不要删除或者修改配置，否则可能导致Flink全托管集群不可用。
ARMS	在您开通Flink全托管集群后，会自动开通ARMS服务，为您提供专家版Prometheus监控能力。ARMS收费详情请参见收费规则。 目前提供的标准的Metric模版中，除Operator相关Metric信息收费外，其他Metric均不收费。Operator相关指标主要存在于Apache Flink官方文档中IO和Connectors相关章节。详情请参见Flink 1.11、Flink 1.12和Flink 1.13。 **注意** 如果将ARMS关闭，则在Flink控制台无法通过数据曲线排查定位问题，也无法配置监控告警。

计费公式

计量方式	地域	单价（元）	公式
包年包月	华北2（北京）、华东2（上海）、华东1（杭州）、华南1（深圳）	180/CU/月	一个实例的总价=（管控资源CU数+ 计算资源CU数）*单价*购买时长
	华北3（张家口）	135/CU/月	
	中国（香港）	295/CU/月	
	亚太东南1（新加坡）	284/CU/月	
	德国（法兰克福）	318.25/CU/月	
	印度尼西亚（雅加达）	279/cu/月	
按量付费	华北2（北京）、华东2（上海）、华东1（杭州）、华南1（深圳）	0.38/CU/小时	一个实例的总价=（管控资源CU数+ 计算资源CU数）* 单价
	新加坡	0.58/CU/小时	

说明 计费账单中的用量表示该计费账单周期内，使用的资源CU数，单位为CU*分钟。例如一个账单周期（1小时）内，10 CU使用21分钟，20 CU使用39分钟，则总体用量为21x10+39x20=990。

3.3. 计费方式

3.3.1. 包年包月

Flink全托管支持包年包月和按量付费两种付费模式。本文为您介绍包年包月付费模式的详情。

包年包月简介

包年包月是一种先付费后使用的计费方式。

包年包月实例支持的购买时长如下。

购买时长	1个月	2个月	3个月	4个月	5个月	6个月	7个月	8个月	9个月	1年

包年包月实例具有以下特点：

- 购买时，您可以提前预留资源，同时享受更大的价格优惠，帮您更大程度节省支出。详情请参见开通流程。
- 在创建包年包月实例时，您需要先支付费用才能开始使用包年包月资源。费用组成以及计算方法请参见计量项和计费项。
- 创建包年包月实例后，您可以通过工作空间资源变配，对实例进行扩缩容操作。详情请参见资源变配。
- 如果您需要在当前计费周期的到期日后继续使用包年包月实例，可以通过续费操作来实现。详情请参见续费。
- 如果您不再需要使用包年包月的实例，请通过提交提交工单处理。

> **注意**
>
> 当前不支持包年包月和按量计费两种计费模式互相转换。

在使用包年包月实例时，如果您对计量项和计费项有疑问，请参见计量项和计费项。如果您对当前产生的费用有疑问，请通过消费明细确认，具体操作请参见查看消费明细。

计算公式

一个实例的总价=（管控资源CU数+ 计算资源CU数）*单价*购买时长

> **注意**
>
> - 管控资源CU数固定为2 CU。
> - 以上计算公式为开通一个Flink全托管实例的费用。一个账户可以开通多个实例，结算时，需要按照多个实例的总费用结算。
> - 该公式中购买时长以月为单位。具体结算费用需要以购买页为准。

例如，以北京地域为例，单价为180元/CU/月，创建一个实例需管控资源2 CU，计算资源8 CU，创建时长为3个月，则创建一个实例所需的费用为：（2 CU+8 CU）*180元/CU/月*3个月=5400元。

结算周期

包年包月实例的计费周期是按照您购买的时长（以UTC+8时间为准），一个计费周期的起点为开通或续费实例的时间 （精确到秒），终点为到期日次日的零点（00:00:00）。

> **说明** 如果您的账户欠费，则有停机风险，系统会提醒或通知您。如果未及时续费，到期7天后您的实例会被释放。避免对您的服务造成影响，请及时续费。更多实例欠费后的状态变化说明，请参见到期或欠费后影响。

3.3.2. 按量付费

Flink全托管支持包年包月和按量付费两种付费模式。本文为您介绍按量付费模式的详情。

按量付费简介

按量付费是一种先使用后付费的计费方式。具有以下特点：

- 购买时，您可以按需开通资源，无需提前购买大量资源。详情请参见开通流程。
- 创建按量付费实例后，您可以通过工作空间资源变配，对实例进行扩缩容操作。详情请参见资源变配。
- 因业务需求不再使用实例时，您可以随时释放实例，以节省费用。

> **注意**
>
> - 支持从按量计费模式转化为包年包月模式。您可以在管理控制台，对应实例**操作**列，选择**更多** > **转为包年包月模式**。
> - 无论您是否运行作业，只要您开通了Flink全托管按量付费模式，系统就会按照计费规则收费。在不运行作业的情况下，系统也会对管控资源（固定2 CU）计费。详情请参见计费公式。

在使用按量付费实例时，如果您对计量项和计费项有疑问，请参见计量项和计费项。如果您对当前产生的费用有疑问，请通过消费明细确认，具体操作请参见查看消费明细。

计算公式

一个实例的总价=（管控资源CU数+ 计算资源CU数）* 单价

> **注意**
>
> - 管控资源CU数固定为2 CU。
> - 以上计算公式为开通一个Flink全托管实例的费用。一个账户可以开通多个实例，结算时，需要按照多个实例的总费用结算。
> - 具体结算费用需要以购买页为准。

例如，以北京地域为例，创建一个实例需管控资源2 CU，计算资源8 CU，创建时长为200个小时，则创建一个实例所需的费用为：（2 CU+8 CU）*200个小时*0.38元/CU/小时=760元。

结算周期

按量付费的Flink全托管实例都是按照每小时整点计算一次费用（以UTC+8时间为准），结算完毕后进入新的结算周期。如果您账户的可用余额（含现金、相关代金券、优惠券、预售券等）不能结清已出账单时，系统即判定账号欠费。账号欠费时，如果在规定时间内未充值，您将不能正常使用资源，更多信息请参见到期或欠费后影响。

阿里云提供延停权益，即当按量付费的资源发生欠费后，提供一定额度或时长继续使用云服务的权益。延停期间正常计费。延停的权益额度不是欠费总额的上限。您延停的额度或时长根据您在阿里云的历史消费等因素，每个月自动计算并更新。更多信息，请参见延期免停权益。

说明 当欠费（可用额度<0）金额或时长，超出延停权益的范围，则按量付费的产品会进入停服处理流程。但是自阿里云暂停服务之日起7天内，阿里云仍将存储您的代码逻辑和作业状态，并持续计算存储费用。超过7天，您仍未支付所欠服务费用的，阿里云将停止为您继续提供服务，同时，您保存在阿里云的全部数据将会被删除并清空且不可恢复。

3.4. 到期或欠费后影响

当包年包月实例到期后或者按量付费实例欠费后均有停机风险，系统会提醒您及时续费，以避免对您的服务造成影响。本文介绍实例到期后或欠费后对服务的影响。

计费周期

- 包年包月

  包年包月实例的计费周期是按照您购买的时长（以UTC+8时间为准），一个计费周期的起点为开通或续费实例的时间 （精确到秒），终点为到期日次日的零点（00:00:00）。

- 按量付费

  按照每小时整点计算一次费用（以UTC+8时间为准），结算完毕后进入新的结算周期。

  当您账户的可用余额（含现金、相关代金券、优惠券、预售券等）不足以支付上一计费周期的费用时，系统即判定账号欠费。

  注意
  - 无论您是否运行作业，只要您开通了Flink全托管按量付费模式，系统就会按照计费规则收费。在不运行作业的情况下，系统也会对管控资源（固定2 CU）计费。详情请参见计费公式。
  - 按量付费模式下，实际使用时长不足一小时按照实际使用分钟计费，实际使用时长不足一分钟，按照一分钟计费。

无论是包年包月实例还是按量付费实例，计费会从您的阿里云账户中扣划账户余额或者使用代金券，同时产生费用账单供您后续查询。详情请参见查看消费明细。

到期后实例影响

对于包年包月的实例，在到期后规定时间段内未及时续费，对实例会有影响，具体说明如下：

到期后时间段	对实例的影响
到期后7天内	您无法访问实例，线上已有作业还可以继续运行。此时您可以为实例续费，续费成功后可以继续使用实例。详情请参见续费。
到期超过7天后	实例将会自动释放，数据将被删除，且删除后无法恢复。

欠费影响

对于按量付费的实例，如果您的账户未在规定时间内充值结清欠费账单，实例将会自动释放，数据将被删除，且删除后无法恢复。

> **说明** 阿里云提供延停权益，即当按量付费的资源发生欠费后，提供一定额度或时长继续使用云服务的权益。延停期间正常计费。延停的权益额度不是欠费总额的上限。您延停的额度或时长根据您在阿里云的历史消费等因素，每个月自动计算并更新。更多信息，请参见延期免停权益。

欠费情况下，只有当您为账号充值，有足够的余额后才能恢复服务。

3.5. 续费

包年包月类型的实例到期后会自动释放。如果您想继续使用，需要在实例释放前进行续费操作，延长对应实例的使用时间。本文为您介绍手动续费实例、自动续费和取消自动续费的具体操作。

前提条件

待续费的包年包月实例处于正常或冻结状态。

注意事项

包年包月实例有到期时间，如果到期未续费，会导致业务中断甚至数据丢失。按量付费实例没有到期时间，不涉及续费操作。

手动续费实例

1. 登录实时计算控制台。
2. 单击**Flink全托管**页签，选择目标实例。
3. 在**操作**列下，选择**其他 > 订单续费**。
4. 在**续费**页面，选中您需要的**续费时长**。

   续费以月为单位，至少续费一个月。
5. 选中**Flink全托管服务等级协议**。
6. 单击**确认**。

   支付成功后，即可完成续费。

开通自动续费

开通自动续费可以免去您定期手动续费的烦恼，且不会因忘记续费而导致业务中断。您可以通过以下两种方式为实例开通自动续费。

- 在创建实例页面开通自动续费

  登录实时计算控制台，您可以在创建实例页面开通自动续费功能，如下图所示。更多创建Flink全托管实例的信息，请参见开通流程。

- 在续费管理页面开通自动续费
  i. 登录实时计算控制台。
  ii. 在顶部菜单栏，选择**费用 > 续费管理**。
  iii. 在**续费管理**页面，选择到期时间范围、产品和地域，过滤待操作的实例。
  iv. 单击**手动续费**页签，选择一种方式开通自动续费。
    - 为一个实例开通：选中目标实例，单击**操作**列下的**开通自动续费**。

- 为多个实例开通：同时选中多个目标实例，在实例列表底部单击**开通自动续费**。

v. 选择自动续费时长，单击**开通自动续费**。

单击**自动续费**页签，实例出现在列表中即表示已成功开通自动续费。

取消自动续费

如果当前计费周期结束后不再需要自动续费实例，您可以提前取消自动续费。开启自动续费后，系统会在实例到期前第9天开始自动扣款。如果需要取消自动续费，请在自动扣款前操作。

1. 登录实时计算控制台。
2. 在顶部菜单栏，选择**费用 > 续费管理**。
3. 在**续费管理**页面，选择到期时间范围、产品和地域，过滤待操作的实例。
4. 单击**自动续费**页签，选择一种方式取消续费。

取消续费的方式	操作	说明
恢复手动续费	○ 为一个实例恢复：选中目标实例，在**操作**列下单击**恢复手动续费**。 ○ 为多个实例恢复：选中多个目标实例，在实例列表底部单击**恢复为手动续费**。	恢复后，需要在实例到期前手动续费，详情请参见手动续费实例。
设置到期不续费	○ 为一个实例设置：选中目标实例，在**操作**列下单击**不续费**。 ○ 为多个实例设置：选中多个目标实例，在实例列表底部单击**设置为不续费**。	选择该方式后，实例到期即停止服务，并且只会提醒一次。如果需要继续使用实例，您可以在到期前手动续费，或变更该设置。

5. 单击**确认**。
   单击**手动续费**或**到期不续费**页签，实例出现在列表中即表示已成功取消自动续费。

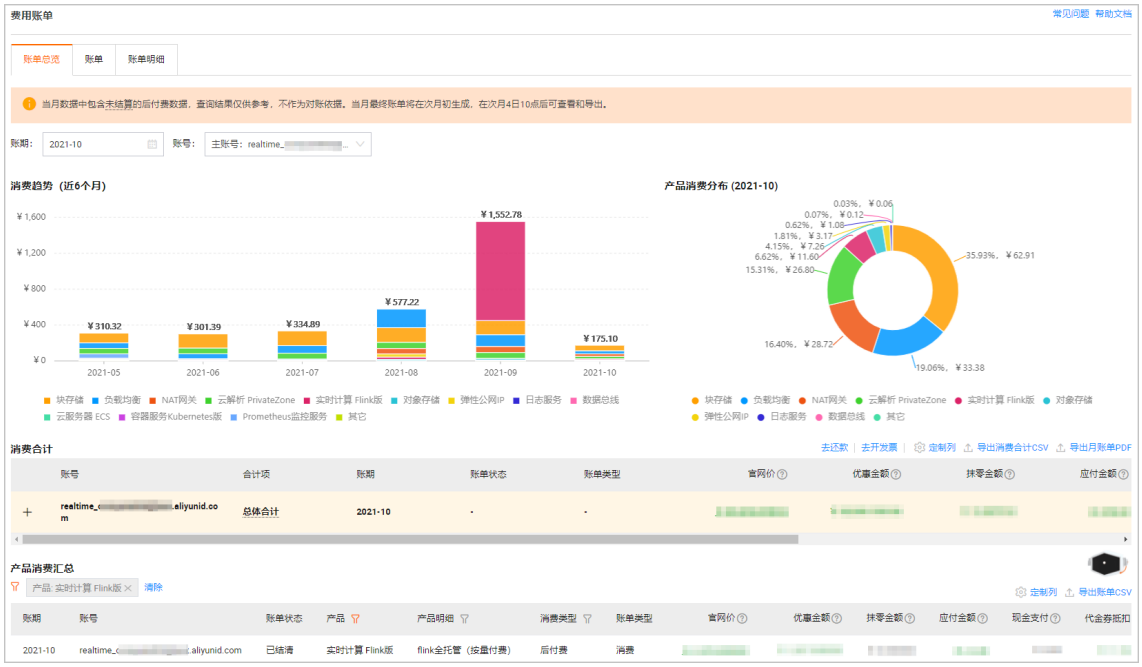
3.6. 查看消费明细

如果您对Flink全托管的消费情况有疑问，可以通过用户中心查看费用账单以及消费明细。本文为您介绍查看费用账单以及消费明细的具体操作方法。

操作步骤

1. 登录实时计算控制台。
2. 在顶部菜单栏，单击**费用**，进入用户中心。
3. 在左侧导航栏，单击**费用账单**。
4. 根据需要，单击相应的页签查看账单信息。
   - 单击**账单总览**页签，查看整体消费情况。

     账单总览按产品和账期汇总展示消费信息，每个产品一条消费汇总。



- 单击**账单**页签，查看消费流水信息。

  账单包括每一笔订单和每个计费周期的账单消费信息。

- 单击**账单明细**页签，查看消费详细信息。

  账单明细支持按照不同的统计项和统计周期进行筛选。消费明细包括产品的计费项、单价和用量情况等。

说明

- 您可以单击账单右上角的**定制列**，选择您希望展示的账单内容。
- 如果想更方便地分析您的账单，可以单击**导出账单CSV**，将您当前账单导出到本地查看。

3.7. 退款说明

本文介绍Flink全托管集群的退费规则及相关说明。

根据计费方式的特点，实时计算Flink支持的退款情况如下表所示。

计费方式	退款规则及说明
包年包月	包年包月的集群在购买时根据购买时长预先支付了费用，如果因业务需求变更不再使用集群，可以通过提交工单退订该资源，申请退款。 退款类型、退款规则及退款流程的详细信息，请参见退款规则及退款流程。
按量计费	按量付费的集群是在使用后才支付费用，因此不涉及退款。 如果因业务需求变更不再使用集群，可以随时释放集群。集群释放后，将不再进行计费。

4.快速入门

4.1. Flink SQL作业快速入门

Flink全托管支持通过SQL代码编辑和运行作业。本文带您快速体验Flink全托管产品的开通、SQL作业的创建和上线运行等操作。

前提条件

- 已准备阿里云账号及账户余额。
  - 账号注册操作步骤，请参见账号注册。
  - 阿里云账户余额不少于100.00元人民币或等值的代金券或优惠券。
- 已完成角色授权，详情请参见阿里云账号角色授权。

操作流程

1. 步骤一：创建Flink全托管实例

   开通一个北京地域按量付费的Flink全托管实例。

2. 步骤二：创建SQL作业并编写业务代码

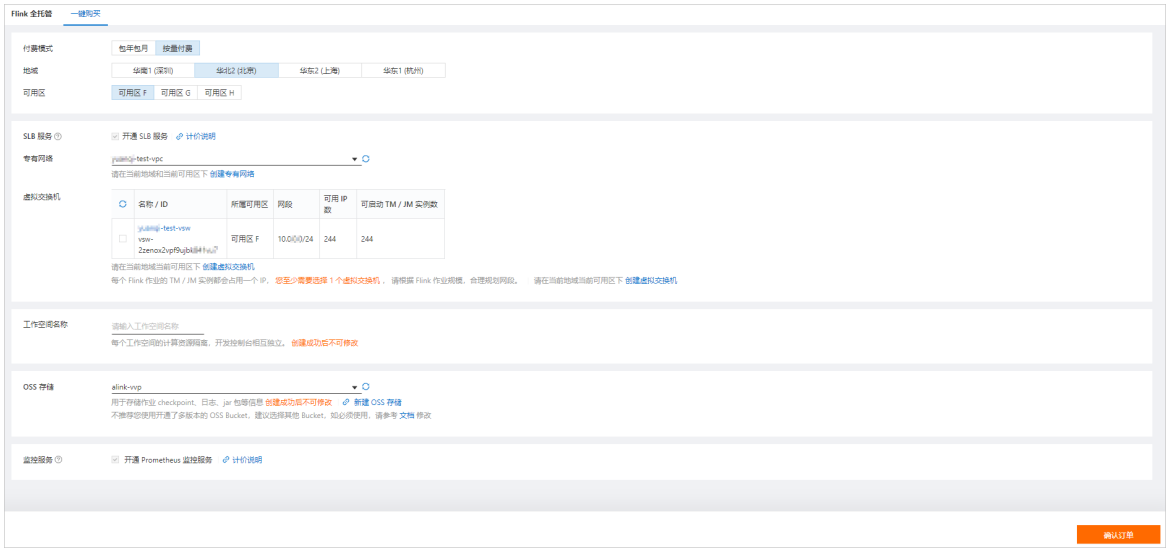
   在作业开发页面，创建一个SQL类型的流作业，并编写DDL和DML代码。

3. 步骤三：启动作业，查看Flink计算结果

   在作业运维页面，启动作业并查看作业运行状态。

步骤一：创建Flink全托管实例

1. 登录实时计算管理控制台。
2. 单击**Flink全托管**下的**购买产品**。
3. 在购买页面，填写配置信息。



类别	配置项	示例	说明
	付费模式	按量付费	支持**包年包月**和**按量付费**模式。

类别	配置项	示例	说明
基本配置			
	地域	北京	支持**华北2（北京）**、**华东2（上海）**、**华北1（杭州）**和**华南1（深圳）**。 说明 建议选择与上下游存储相同的**地域**。
	可用区	可用区F	建议选择与上下游存储相同的**可用区**。
网络配置	SLB服务	-	系统已默认选中并开通了SLB服务。
	专有网络	flink-test-vpc	建议选择与上下游存储相同的专有网络。
	虚拟交换机	flinktest-vsw-2ze4fyq366itq6xqp****	每个Flink作业的Task Manager和Job Manager实例都会占用一个 IP， 您可以选择1~5个虚拟交换机，根据Flink作业规模，合理规划网段。
工作空间配置	工作空间名称	flink-test	以字母开头，仅包含英文字母、数字、中划线（-），长度限制为1~60个字符。

类别	配置项	示例	说明
存储配置	OSS存储	flink-test-oss	用于存储作业的Checkpoint、日志和JAR包等信息。Flink全托管服务会在您选择的Bucket下创建以下目录来保存不同类型的数据： ○ *artifacts*：上传的JAR包会被存储到该目录。 ○ *flink-jobs*：Flink作业的HA信息和Checkpoint会被存储到该目录。 ○ *flink-savepoints*：在**Flink全托管**开发控制台上单击**Savepoint**，会触发**Savepoint**操作，最终的**Savepoint**文件会被存储到该目录。 ○ *logs*：如果您的作业中**日志模板**选择了**OSS**，则您的作业的日志会被存储到该目录。 ○ *sql-artifacts*：注册自定义函数和Connector所依赖的文件会被存储到该目录。 ○ *plan*：使用专家模式，配置的资源信息会被存储到该目录。 ○ *flink-sessionclusters*：Session集群的HA信息和Checkpoint信息会被存储到该目录。 **说明** ○ Flink全托管服务开通成功后，不可修改**OSS存储**。 ○ OSS存储需要与**Flink全托管**服务在同一地域。 ○ 选择Bucket时，请参见注意事项。
监控配置	监控服务	-	系统已默认选中并开通了Prometheus服务。

4. 单击**确认订单**并支付，即可开通成功。

> **说明** 支付完成后，单击**管理控制台**，即可在**Flink全托管**页签，看到正在创建的**工作空间**。通常，支付完成后5~10分钟即可完成**工作空间**的创建。

步骤二：创建SQL作业并编写业务代码

1. 创建SQL作业。
   i. 登录实时计算管理控制台。
   ii. 在**Flink全托管**页签，单击目标工作空间**操作**列下的**控制台**。
   iii. 在左侧导航栏，单击**作业开发**。
   iv. 单击**新建**。

v. 在**新建文件**对话框，填写作业配置信息。

作业参数	示例	说明
文件名称	flink-test	作业的名称。 **说明** 作业名称在当前项目中必须保持唯一。
文件类型	流作业/SQL	流作业和批作业均支持以下文件类型： ■ SQL ■ JAR ■ PYTHON **说明** VVP 2.4.1且VVR 3.0.1及以上版本支持批作业。
部署目标	vvp-workload	选择作业需要部署的集群名称。Flink全托管支持Per-Job集群和Session集群两种集群模式。两种集群模式的区别说明，请参见配置开发测试环境（Session集群）。
存储位置	作业开发	指定该作业的代码文件所属的文件夹。默认存放在**作业开发**目录。 您还可以在现有文件夹右侧，单击 图标，新建子文件夹。

vi. 单击**确认**。

2. 将以下作业代码拷贝到作业文本编辑区。

   创建一个产生随机数据的datagen源表，将datagen源表中的name字段中的数据插入blackhole结果表，代码示例如下所示。

```
CREATE TEMPORARY TABLE datagen_source(
  randstr VARCHAR
) WITH (
  'connector' = 'datagen'
);
CREATE TEMPORARY TABLE print_table(
  randstr  VARCHAR
) WITH (
  'connector' = 'print',
  'logger' = 'true'
);
INSERT INTO print_table
SELECT SUBSTRING(randstr,0,8) from datagen_source;
```

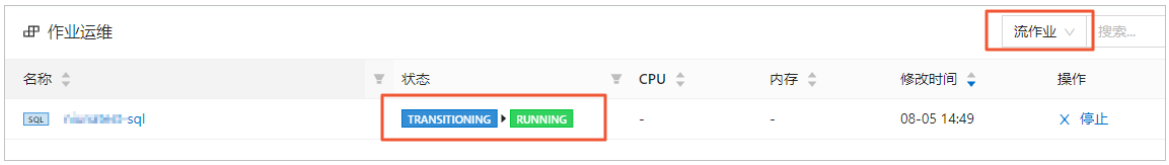
3. 单击**验证**，进行语法检查。
4. 验证通过后，单击**上线**。

步骤三：启动作业，查看Flink计算结果

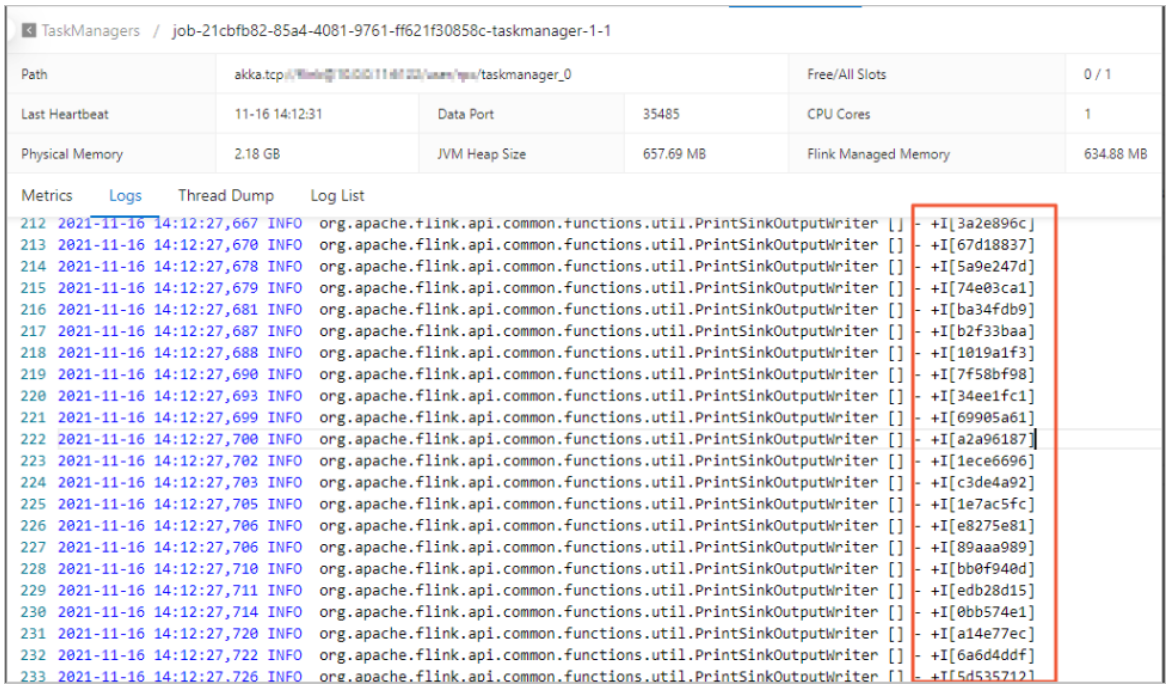
1. 在左侧导航栏，单击**作业运维**。
2. 单击目标作业名称**操作**列中的**启动**。

3. 单击**确认启动**。

   单击启动后，您可以看到作业从当前状态到期望状态的变化过程及最终结果。直到状态变为RUNNING，则代表作业运行正常。



4. 在作业运维页面，单击作业名称，在TaskManager页签的日志中查看Flink计算结果。



相关文档

- 创建项目或对项目资源进行变配，请参见创建与管理项目空间。
- 调试作业提高作业Job Manager资源利用率，请参见配置开发测试环境（Session集群）。

- 开启自动调优功能，请参见配置自动调优。
- 配置监控告警，请参见配置监控告警（在Flink全托管开发控制台配置）或配置监控告警（在ARMS控制台中配置）。

4.2. Flink JAR作业快速入门

Flink全托管支持通过JAR代码编辑和运行作业。本文带您快速体验Flink全托管产品的开通、JAR流作业和批作业的创建和上线运行等操作。

前提条件

- 已准备阿里云账号及账户余额。
  - 账号注册操作步骤，请参见账号注册。
  - 阿里云账户余额不少于100.00元人民币或等值的代金券或优惠券。
- 已完成角色授权，详情请参见阿里云账号角色授权。
- 已在本地准备好了测试JAR包和输入数据文件。
  - 单击FlinkQuickStart-1.0-SNAPSHOT.jar，下载测试JAR包。

    > **说明** 本文为您提供的是统计单词出现频率的测试JAR包。如果您有兴趣研究其源代码，请单击FlinkQuickStart.zip下载后进行编译。

  - 单击Shakespeare，下载输入数据文件Shakespeare。

操作流程

1. 步骤一：创建Flink全托管实例

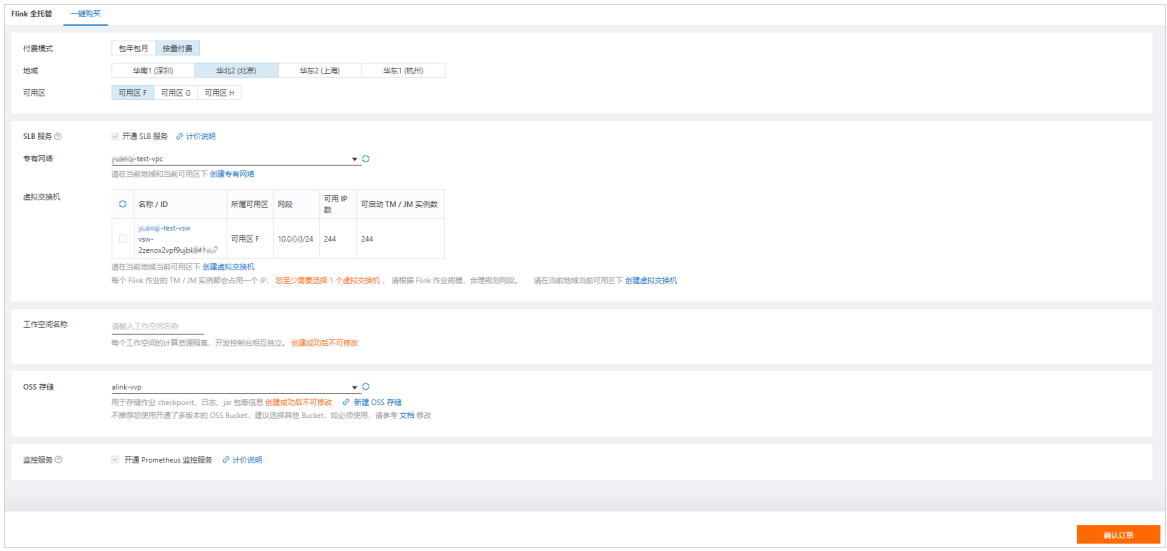
   开通一个北京地域按量付费的Flink全托管实例。
2. 步骤二：创建JAR作业，并配置作业信息

   Flink全托管支持创建JAR流作业和批作业。您可以根据需要，在作业开发页面创建对应的作业。
3. 步骤三：启动作业后，查看Flink计算结果

   在作业运维页面，启动作业后，查看Flink计算结果。

步骤一：创建Flink全托管实例

1. 登录实时计算管理控制台。
2. 单击**Flink全托管**下的**购买产品**。
3. 在购买页面，填写配置信息。



类别	配置项	示例	说明
基本配置	付费模式	按量付费	支持**包年包月**和**按量付费**模式。
	地域	北京	支持**华北2（北京）**、**华东2（上海）**、**华北1（杭州）**和**华南1（深圳）**。 **说明** 建议选择与上下游存储相同的**地域**。
	可用区	可用区F	建议选择与上下游存储相同的**可用区**。
网络配置	SLB服务	-	系统已默认选中并开通了SLB服务。
	专有网络	flink-test-vpc	建议选择与上下游存储相同的专有网络。
	虚拟交换机	flinktest-vsw-2ze4fyq366itq6xqp****	每个Flink作业的Task Manager和Job Manager实例都会占用一个 IP， 您可以选择1~5个虚拟交换机，根据Flink作业规模，合理规划网段。
工作空间配置	工作空间名称	flink-test	以字母开头，仅包含英文字母、数字、中划线（-），长度限制为1~60个字符。

类别	配置项	示例	说明
存储配置	OSS存储	flink-test-oss	用于存储作业的Checkpoint、日志和JAR包等信息。Flink全托管服务会在您选择的Bucket下创建以下目录来保存不同类型的数据： ○ *artifacts*：上传的JAR包会被存储到该目录。 ○ *flink-jobs*：Flink作业的HA信息和Checkpoint会被存储到该目录。 ○ *flink-savepoints*：在**Flink全托管**开发控制台上单击**Savepoint**，会触发**Savepoint**操作，最终的**Savepoint**文件会被存储到该目录。 ○ *logs*：如果您的作业中**日志模板**选择了**OSS**，则您的作业的日志会被存储到该目录。 ○ *sql-artifacts*：注册自定义函数和Connector所依赖的文件会被存储到该目录。 ○ *plan*：使用专家模式，配置的资源信息会被存储到该目录。 ○ *flink-sessionclusters*：Session集群的HA信息和Checkpoint信息会被存储到该目录。 **说明** ○ Flink全托管服务开通成功后，不可修改**OSS存储**。 ○ OSS存储需要与**Flink全托管**服务在同一地域。 ○ 选择Bucket时，请参见注意事项。
监控配置	监控服务	-	系统已默认选中并开通了Prometheus服务。

4. 单击**确认订单**并支付，即可开通成功。

> **说明** 支付完成后，单击**管理控制台**，即可在**Flink全托管**页签，看到正在创建的**工作空间**。通常，支付完成后5~10分钟即可完成**工作空间**的创建。

步骤二：创建JAR作业，并配置作业信息

1. 创建JAR流作业。
   i. 在左侧导航栏，单击**作业开发**。
   ii. 单击**新建**。

iii. 在**新建文件**对话框，填写作业配置信息。

作业参数	示例	说明
文件名称	flink-streaming-test-jar	作业的名称。 **说明** 作业名称在当前项目中必须保持唯一。
文件类型	流作业/JAR	流作业和批作业均支持以下文件类型： ■ SQL ■ JAR ■ PYTHON
部署目标	vvp-workload	选择作业需要部署的集群名称。Flink全托管支持Per-Job集群和Session集群两种集群模式。两种集群模式的区别说明，请参见配置开发测试环境（Session集群）。
存储位置	作业开发	指定该作业的代码文件所属的文件夹。默认存放在**作业开发**目录。 您还可以在现有文件夹右侧，单击 图标，新建子文件夹。

iv. 单击**确认**。

2. 在作业开发页面，填写基本配置信息。

   您可以直接填写以下配置信息，也可以单击**YAML**直接修改配置信息。配置参数解释如下表所示。

参数	示例	说明
部署目标	vvp-workload	您可以修改创建作业时已选择的部署目标。
JAR URI	oss://flink-test-oss/artifacts/namespaces/flink-test-default/FlinkQuickStart-1.0-SNAPSHOT.jar	单击FlinkQuickStart-1.0-SNAPSHOT.jar下载测试JAR包后，再单击右侧 图标选择文件，上传JAR包。
Entrypoint Class	org.example.WordCountStreaming	程序的入口类。如果您的JAR包未指定主类，请在此处输入您的Entrypoint Class类的标准路径。 **说明** 因为本文提供的测试JAR包中既包含了流作业代码，又包含批作业代码。所以，此处需要指定为流作业的程序入口。

参数	示例	说明
Entrypoint main args	--input oss://flink-test-oss/artifacts/namespaces/flink-test-default/Shakespeare	填写输入数据文件的OSS路径。 说明 ○ 本文输入数据文件和测试JAR包存放路径一致。统一放在OSS控制台，名称为flink-test-oss的Bucket下。 ○ 单击Shakespeare，下载输入数据文件Shakespeare。您还需要在开发控制台左侧**资源上传**页面，将Shakespeare输入数据文件上传到OSS的指定目录。上传的文件会固定被保存在*oss://flink-test-oss/artifacts/namespaces/flink-test-default*目录下。
附加依赖文件	无需填写	填写目标附加依赖文件的OSS路径或者URL。
并行度	1	作业并发数。

3. 单击**上线**。
4. 单击**确认**。
1. 创建JAR批作业。
   i. 在左侧导航栏，单击**作业开发**。
   ii. 单击**新建**。

iii. 在**新建文件**对话框，填写作业配置信息。

作业参数	示例	说明
文件名称	flink-batch-test-jar	作业的名称。 **说明** 作业名称在当前项目中必须保持唯一。
文件类型	批作业/JAR	流作业和批作业均支持以下文件类型： ■ SQL ■ JAR ■ PYTHON
部署目标	vvp-workload	选择作业需要部署的集群名称。Flink全托管支持Per-Job集群和Session集群两种集群模式。两种集群模式的区别说明，请参见配置开发测试环境（Session集群）。
存储位置	作业开发	指定该作业的代码文件所属的文件夹。默认存放在**作业开发**目录。 您还可以在现有文件夹右侧，单击 图标，新建子文件夹。

iv. 单击**确认**。

2. 在作业开发页面，填写基本配置信息。

   您可以直接填写以下配置信息，也可以单击**YAML**直接修改配置信息。配置参数解释如下表所示。

参数	示例	说明
部署目标	vvp-workload	您可以修改创建作业时已选择的部署目标。
JAR URI	oss://flink-test-oss/artifacts/namespaces/flink-test-default/FlinkQuickStart-1.0-SNAPSHOT.jar	单击FlinkQuickStart-1.0-SNAPSHOT.jar下载测试JAR包后，再单击右侧 图标选择文件，上传JAR包。
Entrypoint Class	org.example.WordCountBatch	程序的入口类。如果您的JAR包未指定主类，请在此处输入您的**Entrypoint Class**类的标准路径。 **说明** 因为本文提供的测试JAR包中既包含了流作业代码，又包含批作业代码。所以，此处需要指定为批作业的程序入口。

参数	示例	说明
Entrypoint main args	--input oss://flink-test-oss/artifacts/namespaces/flink-test-default/Shakespeare --output oss://flink-test-oss/artifacts/namespaces/flink-test-default/batch-quickstart-test-output.txt	填写输入数据文件和结果数据输出文件路径。 说明 ○ 本示例中输入数据文件、输出文件和测试JAR包存放路径一致。统一放在OSS控制台，名称为flink-test-oss的Bucket下。 ○ 本示例中，我们以将计算结果写入到OSS指定目录为例，为您展示如何配置该参数。在此您仅需要指定结果数据输出文件路径和名称，无需提前在指定目录创建。 ○ 单击Shakespeare，下载输入数据文件Shakespeare。您还需要在开发控制台左侧**资源上传**页面，将Shakespeare输入数据文件上传到OSS的指定目录。上传的文件会固定被保存在*oss://flink-test-oss/artifacts/namespaces/flink-test-default*目录下。
附加依赖文件	无需填写	填写目标附加依赖文件的OSS路径或者URL。
并行度	1	作业并发数。

3. 单击**上线**。
4. 单击**确认**。

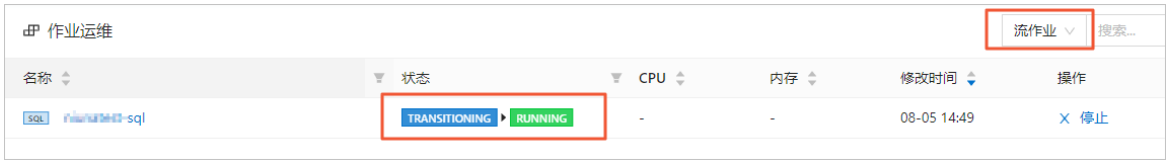
流作业

批作业

步骤三：启动作业后，查看Flink计算结果

1. 在左侧导航栏，单击**作业运维**。
2. 单击目标作业名称**操作**列中的**启动**。
3. 单击**确认启动**。

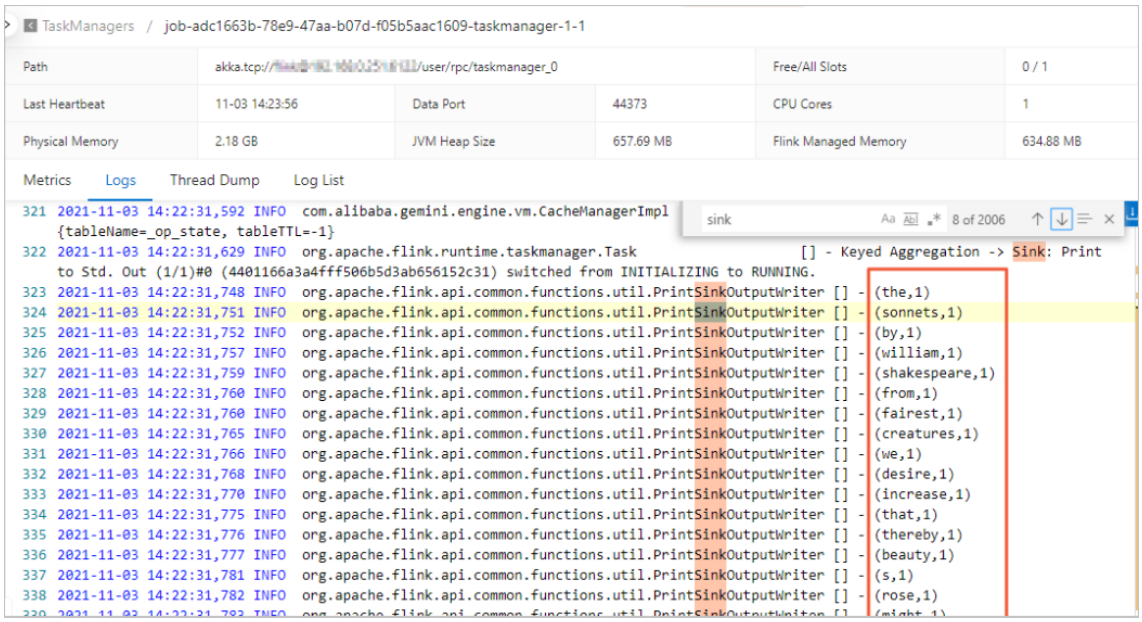
   单击启动后，您可以看到作业从当前状态到期望状态的变化过程及最终结果。直到状态变为RUNNING，则代表作业运行正常。



> **注意** 如果您需要启动批作业，则需要在**作业运维**页面，将作业类型切换为**批作业**，才可以看到您上线的批作业。系统默认展示的作业为**流作业**。

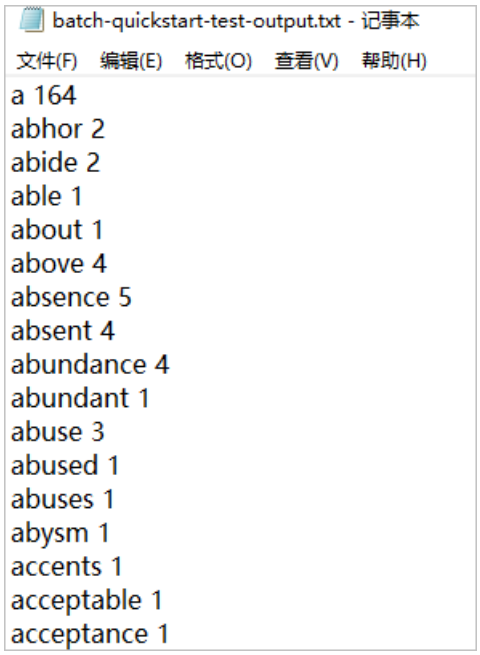
4. 查看Flink计算结果。
   - 流作业示例的计算结果：在**作业运维**页面，单击作业名称，在**TaskManager**页签的日志中查看Flink

计算结果。



- 批作业示例的计算结果：登录OSS管理控制台，在您配置的数据输出文件存放目录查看结果。

  本示例中的数据输出文件目录为*oss://flink-test-oss/artifacts/namespaces/flink-test-default/batch-quickstart-test-output.txt*。



相关文档

- 创建项目或对项目资源进行变配，请参见创建与管理项目空间。
- 调试作业提高作业Job Manager资源利用率，请参见配置开发测试环境（Session集群）。
- 开启自动调优功能，请参见配置自动调优。
- 配置监控告警，请参见配置监控告警（在Flink全托管开发控制台配置）或配置监控告警（在ARMS控制台中配置）。

4.3. 数据实时入仓入湖快速入门

Flink全托管产品提供丰富强大的数据实时入仓入湖能力。本文为您介绍如何在Flink全托管控制台上快速构建一个从MySQL到Hologres的数据同步作业。

背景信息

假设MySQL实例中有一个tpc_ds库，里面有24张表结构不相同的业务表。另外还有user_db1~user_db3三个库，由于进行了分库分表的设计，每个库中分别有3张表结构相同的表，共包含名称为user01~user09的9张表。在阿里云DMS控制台观察到MySQL中的库和表情况如下图所示。

此时，如果您希望开发一个数据同步的作业，将这些表和数据都同步到Hologres中，其中user分库分表能合并到Hologres的一张表中，则可以按照以下步骤进行：

- 步骤一：创建Catalog
- 步骤二：开发数据同步作业
- 步骤三：启动作业
- 步骤四：观察全量同步结果
- 步骤五：观察增量同步结果
- （可选）步骤六：作业资源配置

本文使用Flink全托管提供的CREATE TABLE AS(CTAS)语句和CREATE DATABASE AS(CDAS)语句来完成整库同步、分库分表合并同步，一键完成数据的全量和增量同步，以及实时的表结构变更同步。

前提条件

- 已准备阿里云账号及账户余额。
  - 账号注册操作步骤，请参见账号注册。
  - 阿里云账户余额不少于100.00元人民币或等值的代金券或优惠券。
- 已创建Flink全托管实例并完成角色授权，详情请参见开通流程和阿里云账号角色授权。
- 上下游存储
  - 已创建RDS MySQL实例，详情请参见创建RDS MySQL实例。
  - 已创建Hologres实例，详情请参见购买Hologres。

> **说明** RDS MySQL和Hologres需要与Flink全托管实例在相同地域相同VPC下，否则需要打通网络，详情请参见如何访问跨VPC里的存储资源？或Flink全托管集群如何访问公网？。

- 已准备好测试数据，并配置好白名单。详情请参见准备测试数据和配置IP白名单。

准备测试数据

1. 单击tpc_ds.sql、user_db1.sql、user_db2.sql和user_db3.sql下载测试数据到本地。
2. 在DMS数据管理控制台上，准备RDS MySQL的测试数据。
   i. 通过DMS登录RDS MySQL。

      详情请参见通过DMS登录RDS MySQL。

   ii. 在已登录的SQLConsole窗口，输入如下命令后单击**执行**。

      创建tpc_ds、user_db1、user_db2和user_db3四个数据库。

```
CREATE DATABASE tpc_ds;
CREATE DATABASE user_db1;
CREATE DATABASE user_db2;
CREATE DATABASE user_db3;
```

   iii. 在DMS控制台顶部菜单栏，单击**数据导入**。

   iv. 单击**批量数据导入**。

   v. 选择需要导入的数据库，上传对应的SQL文件，单击**提交申请**后，单击**执行变更**。

      同样的操作依次为tpc_ds、user_db1、user_db2和user_db3数据库导入对应的数据文件。

3. 在Hologres控制台创建my_user数据库，用于存放合并后的user表数据。

   操作步骤详情请参见创建数据库。

配置IP白名单

为了让Flink能访问MySQL和Hologres实例，您需要将Flink全托管实例的网段添加到在MySQL和Hologres的白名单中。

1. 获取Flink全托管实例的VPC网段。

i. 登录实时计算控制台。

ii. 在目标**工作空间**右侧**操作**列，选择**其他 > 工作空间详情**。

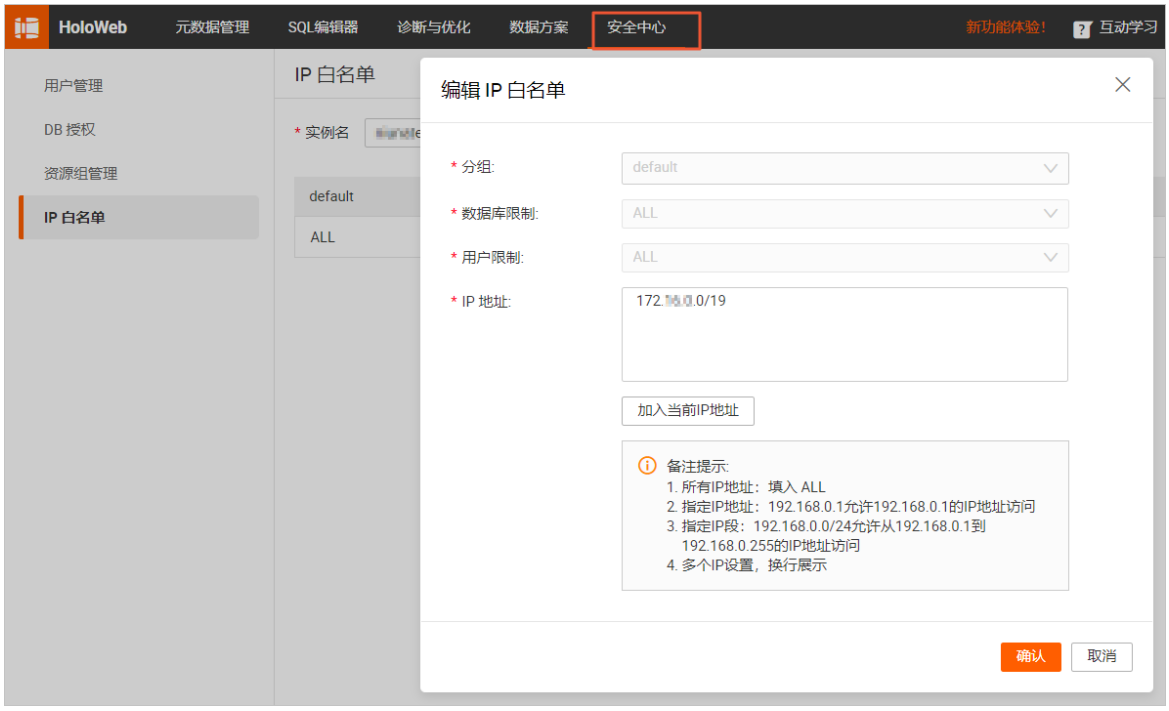
iii. 在**工作空间详情**对话框，查看**Flink全托管**虚拟交换机的**网段**信息。

2. 在RDS MySQ的IP白名单中，添加Flink全托管网段信息。

   操作步骤详情请参见设置IP白名单。

3. 在Hologres的IP白名单中，添加Flink全托管网段信息。

   操作步骤详情请参见IP白名单。

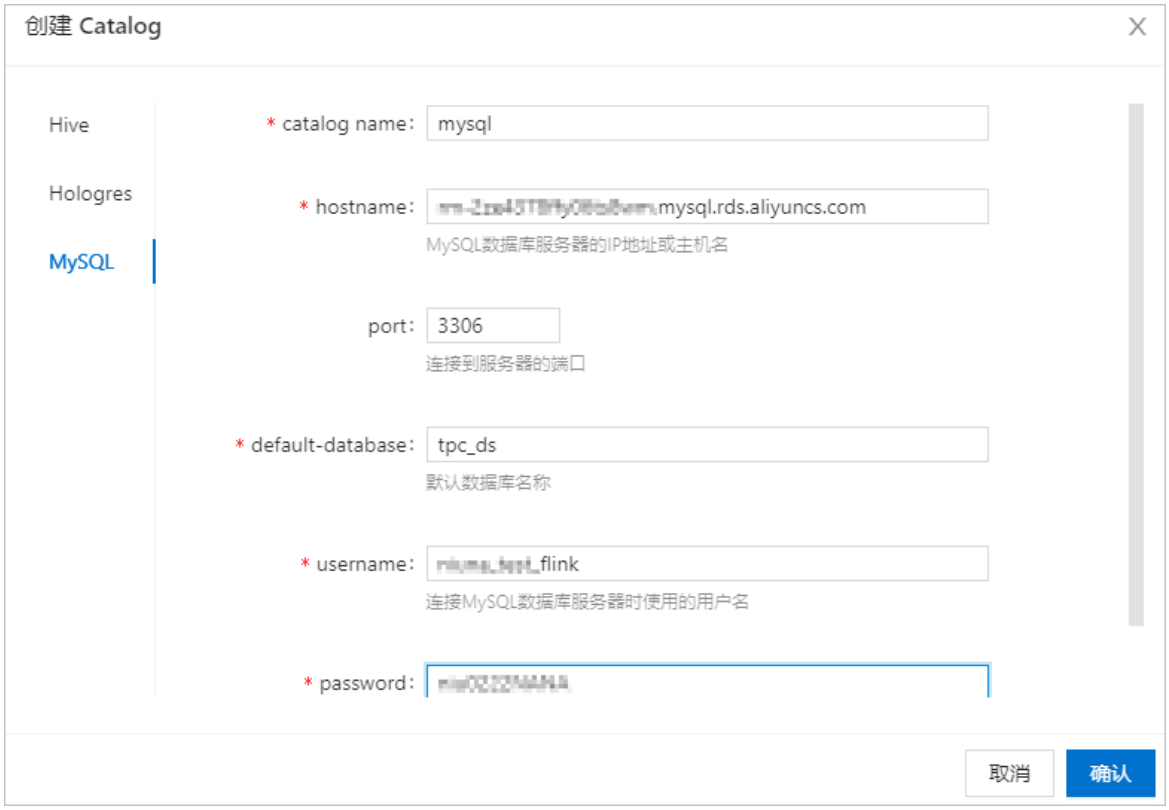


步骤一：创建Catalog

整库同步、分库分表合并同步、单表同步都需要依赖目标Catalog来创建目标表，也依赖源Catalog来获取源表列表和信息。因此，您需要通过控制台创建源Catalog和目标Catalog。本文将以源Catalog为MySQL Catalog和目标Catalog为Hologres Catalog为例，为您进行介绍。

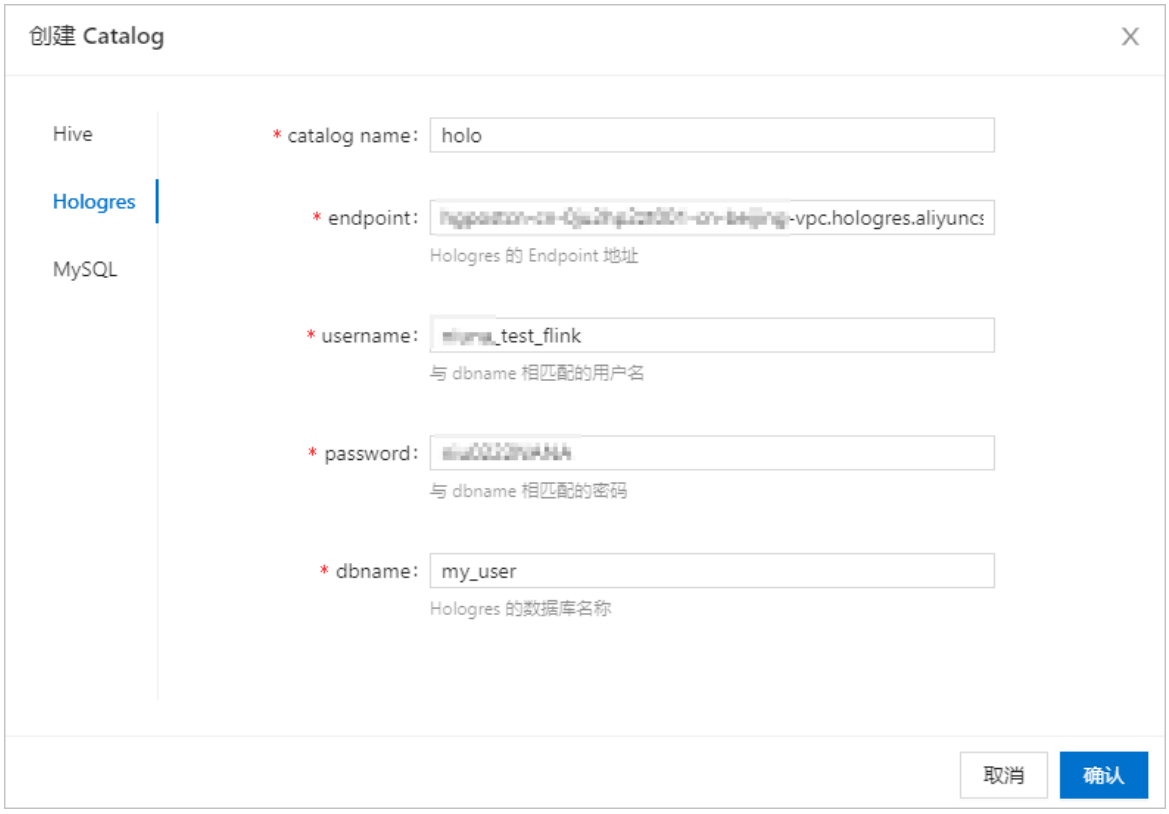
1. 创建名称为mysql的MySQL Catalog。

   操作步骤详情请参见配置MySQL Catalog。

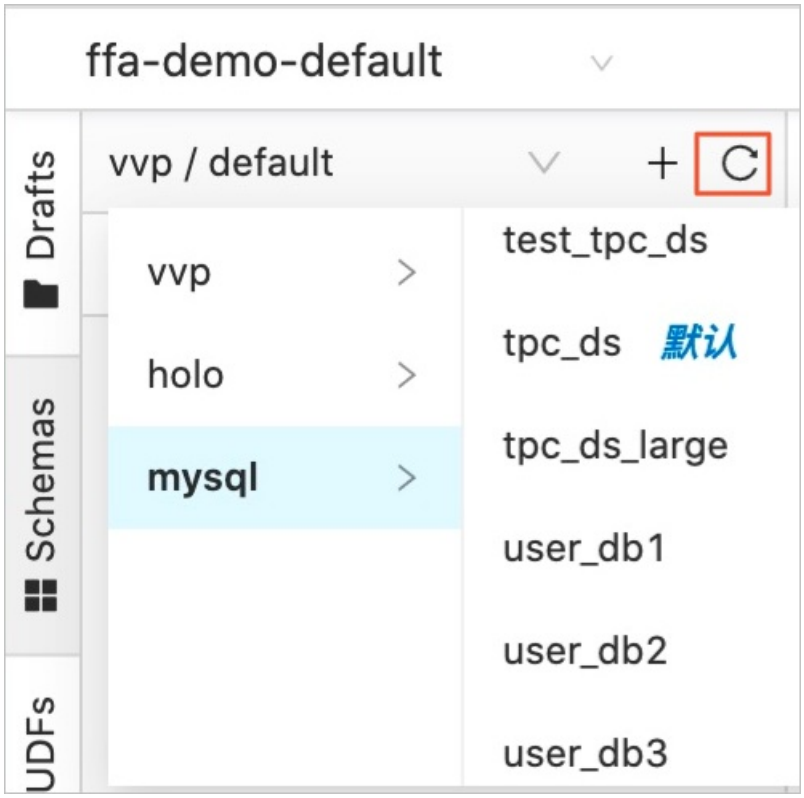


2. 创建名称为holo的Hologres Catalog。

   操作步骤详情请参见配置Hologres Catalog。



3. 在Schemas页签，确认已创建名为mysql和holo的Catalog。



步骤二：开发数据同步作业

1. 登录Flink全托管开发控制台，新建作业。
   i. 登录实时计算管理控制台。
   ii. 在**Flink全托管**页签，单击目标工作空间**操作**列下的**控制台**。
   iii. 在左侧导航栏，单击**作业开发**。
   iv. 单击**新建**。
   v. 在**新建文件**对话框，填写作业配置信息。

作业参数	示例	说明
文件名称	flink-test	作业的名称。 **说明** 作业名称在当前项目中必须保持唯一。
文件类型	流作业/SQL	数据同步作业仅支持流作业/SQL类型。
部署目标	vvp-workload	选择作业需要部署的集群名称。
存储位置	作业开发	指定该作业的代码文件所属的文件夹。默认存放在**作业开发**目录。 您还可以在现有文件夹右侧，单击 图标，新建子文件夹。

   vi. 单击**确认**。
2. 将以下作业代码拷贝到作业文本编辑区。

   将tpc_ds库中所有表同步至Hologres，并将user的分库分表合并同步到 Hologres的单表中。代码示例如下所示。

```
USE CATALOG holo;
BEGIN STATEMENT SET;
-- 同步TPCDS整库到Hologres的tpc_ds库中。
CREATE DATABASE IF NOT EXISTS tpc_ds
AS DATABASE mysql.tpc_ds INCLUDING ALL TABLES
/*+ OPTIONS('server-id'='8001-8004') */ ;
-- 同步user分库分表到Hologres的my_user.users表中。
CREATE TABLE IF NOT EXISTS my_user.users
AS TABLE mysql.`user_db[0-9]+`.`user[0-9]+`
/*+ OPTIONS('server-id'='8001-8004') */;
END;
```

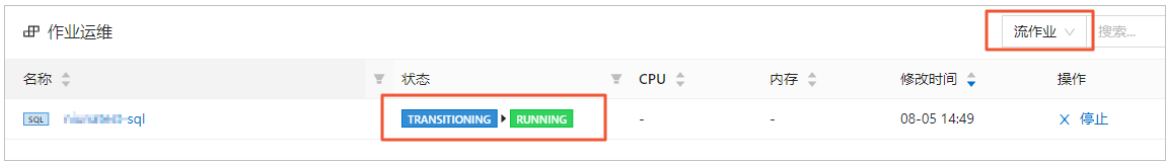
   将tpc_ds库中所有表同步至Hologres使用CDAS (CREATE DATABASE AS) 语法来实现，将user的分库分表合并同步到 Hologres的单表使用CTAS (CREATE TABLE AS) 语法来实现，最后再使用STATEMENT SET语法将这两条SQL语句合并在一个作业中提交。Flink会自动为Source进行优化，复用一个Source节点读取多张MySQL表的数据，这能显著降低MySQL的连接数和读取压力，提升稳定性。

> **说明** 如果只想同步库中的某些表，您也可以在CDAS语法中使用 INCLUDING TABLE或EXCLUDING TABLE 语法来指定具体需要同步的表。例如INCLUDING TABLE 'web.*'表示只同步中所有web开头的表。

步骤三：启动作业

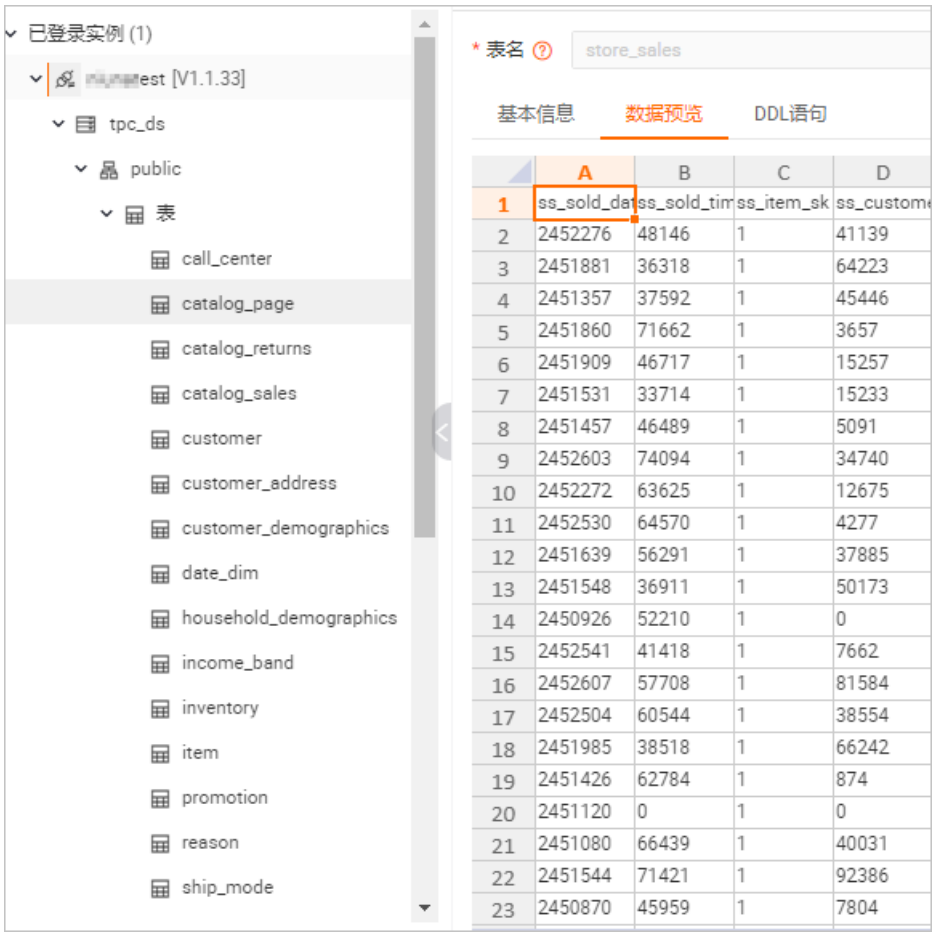
1. 在**作业开发**页面，单击**上线**。
2. 在**作业运维**页面，单击目标作业名称**操作**列中的**启动**。

   单击**确认启动**后，您可以看到作业从当前状态到期望状态的变化过程及最终结果。直到状态变为RUNNING，则代表作业运行正常。可以在作业运维页面观察作业的运行信息和状态。



步骤四：观察全量同步结果

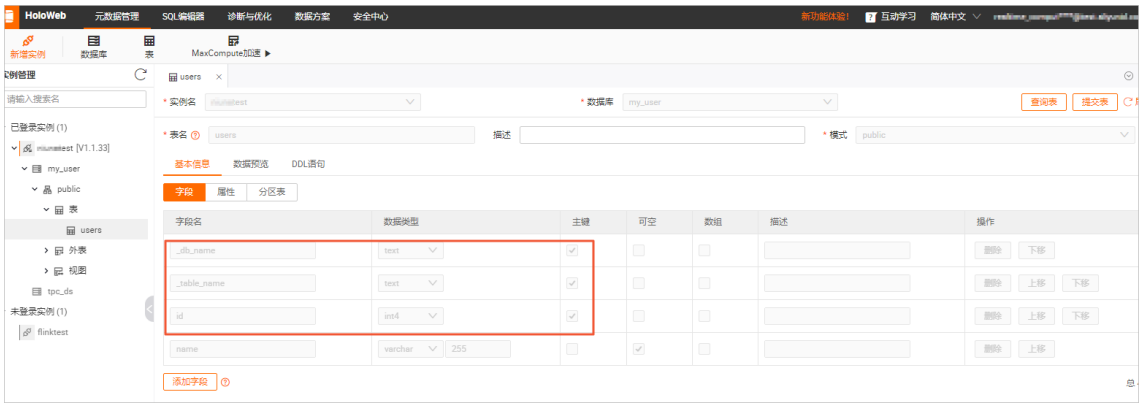
1. 登录Hologres管理控制台。
2. 在**元数据管理**页签，查看Hologres实例下的tpc_ds数据库中24张表和表数据。



3. 在**元数据管理**页签，查看my_user库下users表结构。

   同步后的表结构和数据如下图所示。

- 表结构



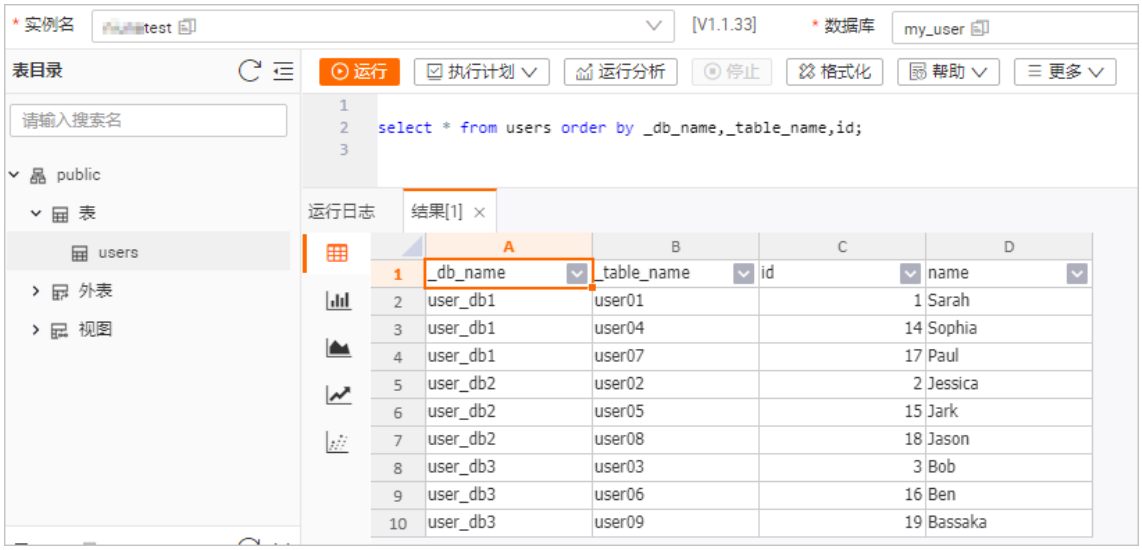
users表的表结构比MySQL源表中多了_db_name和_table_name两列，代表数据来源的库名和表名，且作为联合主键的一部分来保证分库分表合并后的数据唯一性。

- 表数据

在users表信息页面右上角，单击**查询表**后，输入如下命令，单击**运行**。

```
select * from users order by _db_name,_table_name,id;
```

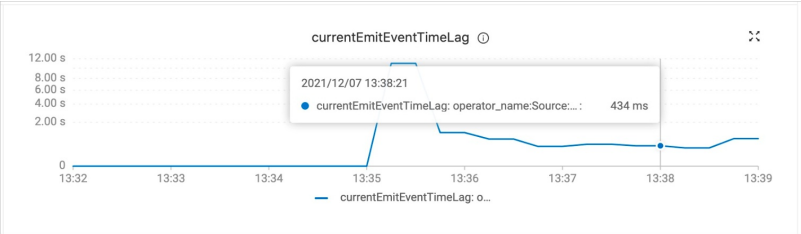
表数据结果如下图所示。



步骤五：观察增量同步结果

同步作业会在全量数据同步完以后自动切换到增量数据同步阶段，无需干预。您可以通过数据曲线页签的currentEmitEventTimeLag值来确定数据同步的阶段。

1. 登录实时计算管理控制台。
2. 在**Flink全托管**页签，单击对应工作空间**操作**列下的**控制台**。
3. 在左侧导航栏上，选择**应用 > 作业运维**。
4. 单击目标作业名称。
5. 单击**数据曲线**页签。
6. 观察currentEmitEventTimeLag曲线图，确定数据同步阶段。



- 值为0时，代表还在全量同步阶段。
- 值大于0时，代表已经进入增量同步阶段。

7. 验证实时同步数据变更和结构变更的能力。

   MySQL CDC数据源支持在增量同步阶段，实时同步表的数据变更以及表的结构变更。您可以在作业进入到增量同步阶段后，通过修改MySQL的user分表的表结构和数据，来验证实时同步数据变更和结构变更的能力。

   i. 通过DMS登录RDS MySQL。

      详情请参见通过DMS登录RDS MySQL。

   ii. 在user_db2数据库下，执行如下命令修改user02表的表结构，并插入和更新数据。

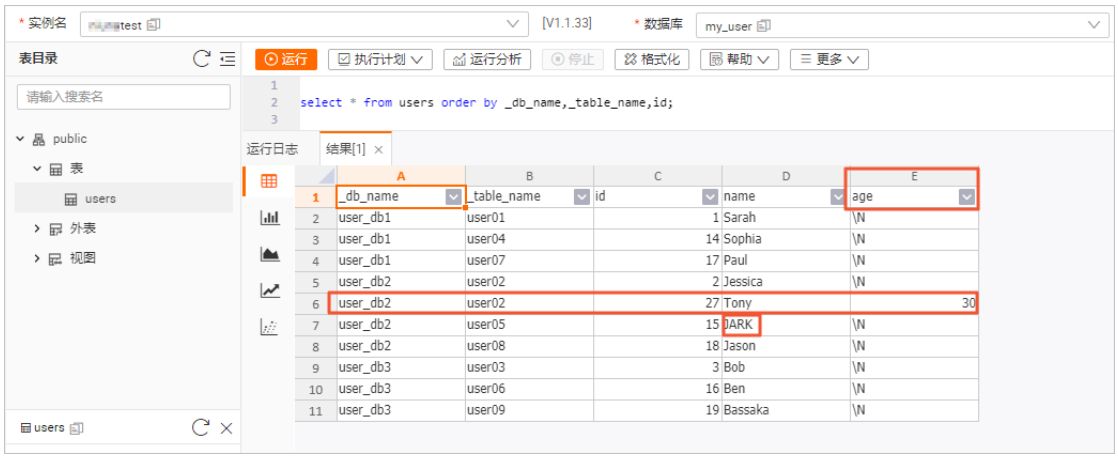
```
USE DATABASE `user_db2`;
ALTER TABLE `user02` ADD COLUMN `age` INT;   -- 添加age列。
INSERT INTO `user02` (id, name, age) VALUES (27, 'Tony', 30); -- 插入带有age的数据。
UPDATE `user05` SET name='JARK' WHERE id=15;  -- 更新另一张表，名字改成大写。
```

   iii. 在Hologres控制台，查看users表结构和数据的变化。

      在users表信息页面右上角，单击**查询表**后，输入如下命令，单击**运行**。

```
select * from users order by _db_name,_table_name,id;
```

      表数据结果如下图所示。



      虽然多张分表的Schema并不一致，但是在user02上的表结构变更，以及数据变更都能实时地同步到下游表中。在Hologres的users表中，看到了新增的age字段，插入的Tony数据以及更新成大写的JARK数据。

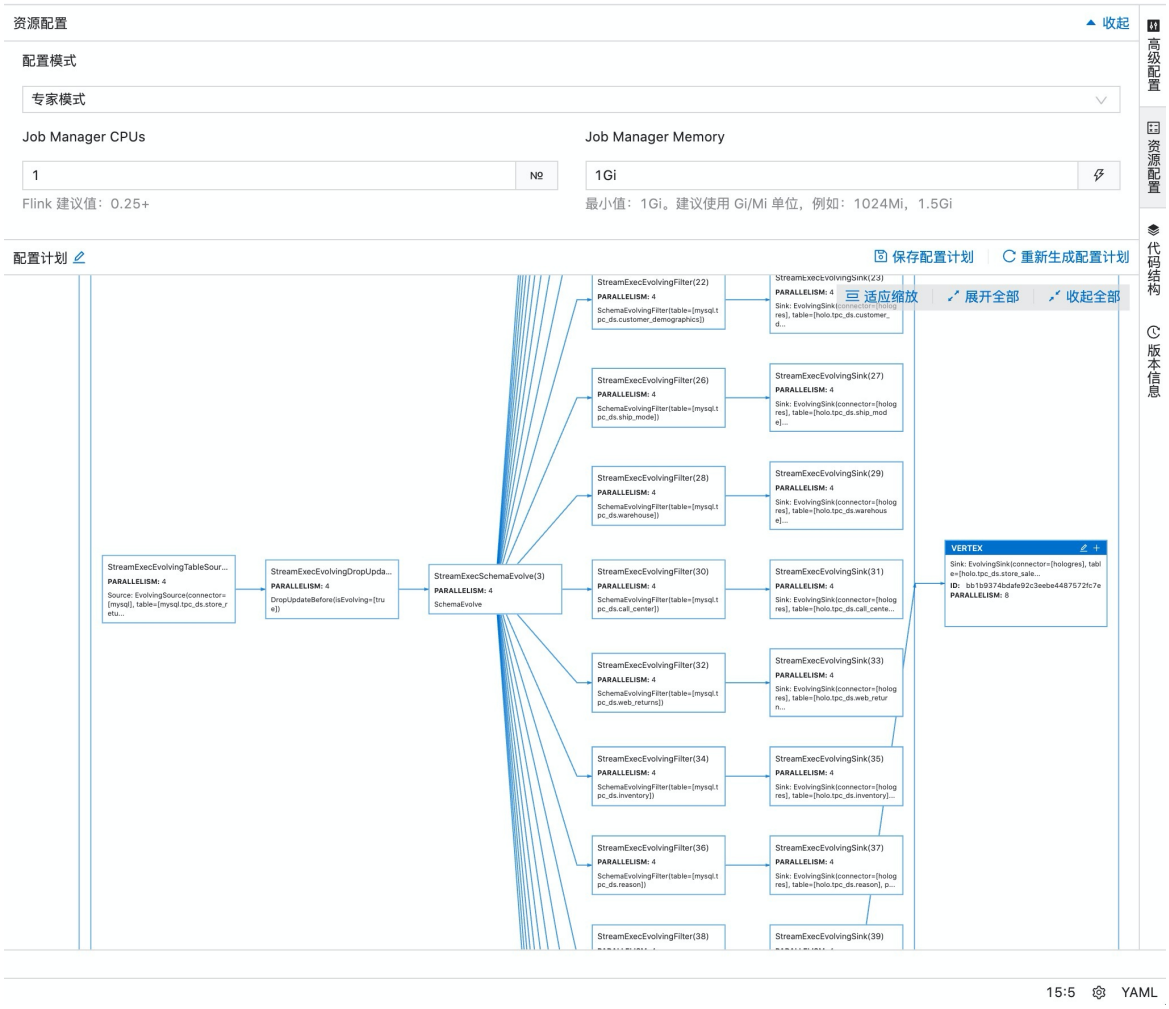
（可选）步骤六：作业资源配置

根据数据量的不同，我们往往需要调节不同节点的并发和资源，以达到更优的作业性能。您可以使用资源配置的基础模式简单配置作业并发度和CU数，也可以使用资源配置的专家模式细粒度地调整节点的并发和资源。

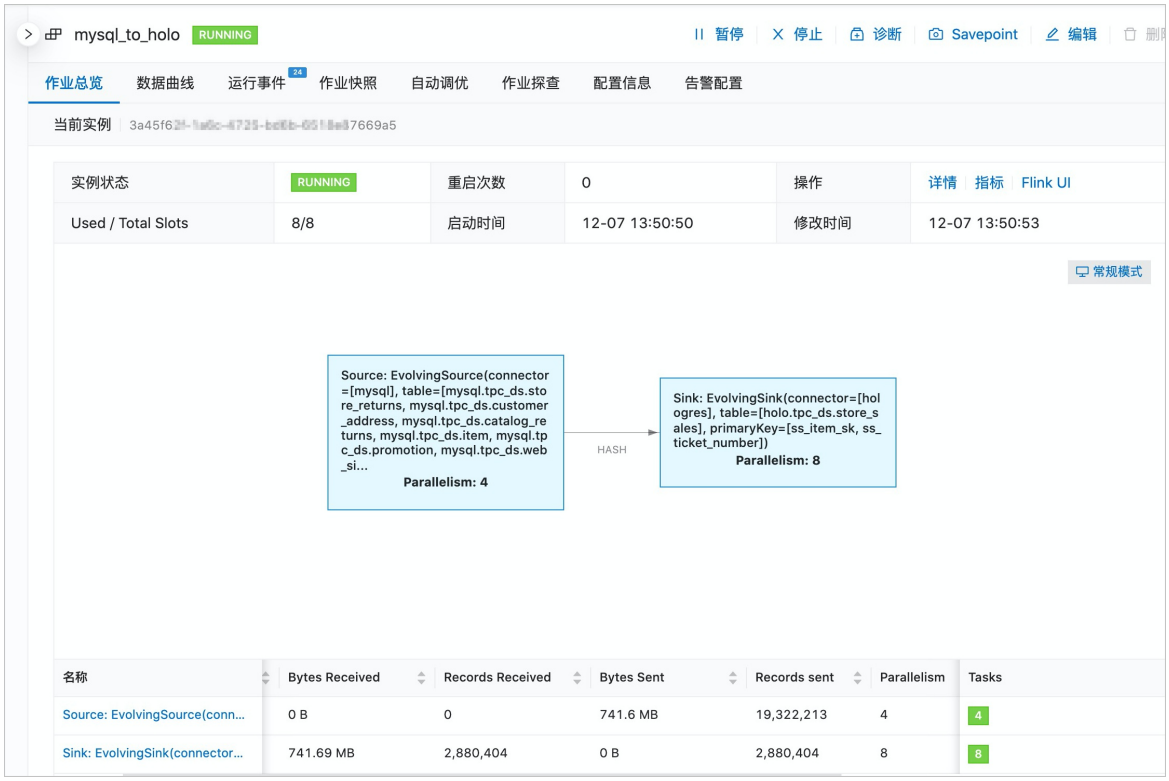
1. 在**作业运维**页面，作业详情页面右上角，单击**编辑**。
2. 在页面右侧，单击**资源配置**页签。
3. 资源模式选择为**专家模式**。
4. 单击**立刻获取**。
5. 单击**展开全部**。

   观察完整的拓扑图，通过完整的拓扑图能了解到整个数据的同步计划，即具体同步哪些表。
6. 手动设置每个节点的并发。

   设置作业为4并发；由于tpc_ds中的store_sales表数据量最大，可以单独设置holo.tpc_ds.store_sales Sink节点并发为8，提升 Hologres的写入性能。资源配置步骤详情请参见配置细粒度资源。经过调节后的作业资源配置计划如下图所示。



7. 单击**保存配置计划**
8. 单击**上线**。
9. 在**运维**页面，查看调整效果。



常见问题

- Q：更新了MySQL中的表结构，但是下游的表结构没有变化是怎么回事？

  A：表结构的变更同步并不识别具体的DDL，而是捕获前后两条数据之间的Schema变化。如果仅仅发生了DDL变更，但是上游无任何新增数据或者数据变更，则不会触发下游的数据变更。详细说明请参见表结构变更的同步策略。

- Q：Source出现 `finish split response timeout` 异常，是什么原因？

  A：该异常是因为task的CPU使用率过高导致来不及响应Coordinator的RPC请求。此时，您需要在资源配置页面增加Task Manager的CPU资源。

- Q：在全量阶段发生表结构变更有什么影响？

  A：如果在全量读取阶段发生了源表的表结构变更，则Flink会忽略该表的表结构变更，导致表结构变更并不会同步到下游表。

- Q：如果同步期间发生了不支持的表结构变更，导致作业同步失败，该怎么解决？

  如果发生了这种情况，需要重新同步该表的数据，即先停止作业，然后删掉下游表并重新启动同步作业，详情请参见CTAS使用限制和CDAS使用限制。

相关文档

- CREATE TABLE AS (CTAS) 语法功能介绍，请参见CREATE TABLE AS(CTAS)语句。
- CREATE DATABASE AS (CDAS) 语法功能介绍，请参见CREATE DATABASE AS(CDAS)语句。

Demo演示

MySQL实时同步至Hologres

5.实例管理

5.1. 开通流程

本文为您介绍通过阿里云账号开通Flink全托管的流程，以及开通过程中的注意事项。

前提条件

- 您已经拥有阿里云账号。如果您还没有阿里云账号，请先完成注册，详情请参见阿里云账号注册流程。
- 您已开通OSS和VPC：
  - **Flink全托管**产品采用存储计算分离架构，需要使用对象存储OSS存储您上传的JAR包、Flink作业的日志、Flink作业中生成的Savepoint或Checkpoint等信息。如果您还没有开通OSS，请先完成开通，详情请参见开通OSS服务。
  - 您还需要开通VPC，详情请参见创建专有网络。

  开通Flink全托管的过程中会自动开通SLB和ARMS，涉及到的云产品收费标准详情请参见计量项和计费项。
- 开通按量付费的服务前，请确保您的阿里云账户余额不少于100.00元人民币或等值的代金券或优惠券。

注意事项

选择OSS Bucket时，不推荐您选择开通了版本控制功能的Bucket。因为如果您的Bucket开通了版本控制功能，过多的删除标记会导致List慢等问题。因此，如果您开通了版本控制功能，则您需要配置生命周期规则，来定期清理删除标记和不必要的历史版本，从而减少存储成本并提升OSS性能。清理删除标记的操作详情，请参见使用生命周期管理文件版本。

> **说明** 在您的OSS Bucket未开启版本控制功能时，如果您需要设置生命周期规则，请勿对Flink的相关目录（包括artifacts、flink-jobs、flink-savepoints、sql-artifacts、plan和flink-sessionclusters等。）设置清除策略。因为如果设置了清除策略，则可能会把Flink相关文件清理掉，导致Flink作业运行异常。

开通流程

1. 登录实时计算控制台。
2. 单击**Flink全托管**下的**购买产品**。
3. 在**授权请求**页面，单击**前往RAM进行授权**。

4. 在**云资源访问授权**页面，单击页面下方的**同意授权**。

> **说明** 系统已默认勾选AliyunStreamAsiDefaultRole角色。

5. 在购买页面，填写配置信息。

类别	配置项	说明
基本配置	付费模式	支持**包年包月**和**按量付费**模式。
	购买时长	仅**包年包月**需要选择。
	自动续费	仅**包年包月**需要选择。**开启**自动续费功能后，后续您可以在阿里云控制台**续费管理**中修改续费策略。
	地域	支持**华北2（北京）**、**华东2（上海）**、**华北1（杭州）**、**华南1（深圳）**、**华北3（张家口）**、**中国（香港）**、**新加坡**、**德国（法兰克福）**和**印度尼西亚（雅加达）** **说明** 建议选择与上下游存储相同的**地域**。
	可用区	建议选择与上下游存储相同的**可用区**。

类别	配置项	说明
网络配置	专有网络	建议选择与上下游存储相同的**专有网络**。
	虚拟交换机	每个Flink作业的Task Manager和Job Manager实例都会占用一个IP， 您可以选择1~5个虚拟交换机，根据Flink作业规模，合理规划网段。
工作空间配置	工作空间名称	以字母开头，仅包含英文字母、数字、中划线（-），长度限制为1~60个字符。 **说明** ○ 同一个阿里云账号下的工作空间名称是唯一的，请勿填写重名名称，否则会提示您重新填写。 ○ **Flink全托管**服务开通成功后，不可修改**工作空间**名称。
	计算资源配额	仅**包年包月**模式需要填写，将按照此配额进行计费。
存储配置	OSS存储	Flink全托管服务会在您选择的Bucket下创建以下目录来保存不同类型的数据： ○ *artifacts*：上传的JAR包会被存储到该目录。 ○ *flink-jobs*：Flink作业的HA信息和Checkpoint会被存储到该目录。 ○ *flink-savepoints*：在**Flink全托管**开发控制台上单击**Savepoint**，会触发**Savepoint**操作，最终的**Savepoint**文件会被存储到该目录。 ○ *logs*：如果您的作业中**日志模板**选择了**OSS**，则您的作业的日志会被存储到该目录。 ○ *sql-artifacts*：注册自定义函数和Connector所依赖的文件会被存储到该目录。 ○ *plan*：使用专家模式，配置的资源信息会被存储到该目录。 ○ *flink-sessionclusters*：Session集群的HA信息和Checkpoint信息会被存储到该目录。 **说明** ○ Flink全托管服务开通成功后，不可修改**OSS存储**。 ○ OSS存储需要与**Flink全托管**服务在同一地域。 ○ 选择Bucket时，请参见注意事项。

6. **确认订单**并支付，即可开通成功。

说明

- 支付完成后，单击**管理控制台**，即可在**Flink全托管**页签，看到正在创建的**工作空间**。通常，支付完成后5~10分钟即可创建完成**工作空间**。
- Flink全托管开通后，您可以登录实时计算控制台查看控制台界面，详情请参见管理控制台介绍。此外，创建工作空间后默认生成一个名称为工作空间名称-default项目，您可以进入后进行作业开发。如果您想创建其他项目，详情请参见创建与管理项目空间。

5.2. 创建与管理项目空间

项目空间是Flink全托管管理作业的基本单元，您的所有配置、作业、权限均在单个项目空间下进行。您可以创建多个项目空间，为每个项目空间分配单独的资源和权限，实现多租户通过项目空间进行资源和权限的完全隔离。本文为您介绍如何创建与删除项目空间。

背景信息

当您购买了一个Flink全托管实例后，系统默认生成一个名称为工作空间名称-default项目，初始资源都归属于该项目空间。您可以在一个工作空间中创建项目空间、删除项目空间和项目空间资源分配。

前提条件

新建项目空间前，需要工作空间有可分配的资源。项目空间需要至少1CU，因此在新建项目空间前，需要保证工作空可分配的资源大于1 CU。

说明 如果工作空间剩余资源不足，您可以对工作空间先进行扩容，或者对工作空间名称-default项目空间先进行缩容。工作空间扩容操作请参见工作空间资源变配。

使用限制

仅阿里云账号有权限进行项目空间的创建与删除。

注意事项

- 删除项目空间前，请确认该项目空间下不存在运行中的作业。如果该项目空间中存在还在运行的作业，则会报错，提示您停止运行的作业后再删除项目空间。
- 项目空间删除后，您项目空间中的作业及数据将不能恢复，请谨慎操作。该项目空间的资源可被其他项目空间扩容使用。

创建项目空间

1. 登录实时计算控制台。
2. 在**Flink全托管**页签，选择目标工作空间**操作**列下的**其他 > 添加项目空间**。
3. 在**创建项目空间**对话框，填写项目空间信息。

参数	说明
项目空间名称	不可与已创建的项目空间重名。
资源分配	单个项目空间至少需要1CU，因此工作空间剩余可分配资源需要大于1CU。

4. 单击**确认**。

   项目空间创建完成后，您可以单击该项目空间名称，进入开发控制台，在该项目空间下进行作业开发。

删除项目空间

1. 登录实时计算控制台。
2. 在**Flink全托管**页签，单击目标工作空间左侧的 + 图标。
3. 单击目标项目空间**操作**列下的**删除**。
4. 单击**确定**。

5.3. 资源变配

Flink全托管支持对工作空间（实例）和项目空间的资源变配。本文为您介绍如何进行工作空间和项目空间的缩容或扩容，即工作空间和项目空间的资源变配。

注意事项

因为工作空间中支持创建多个项目空间，所以在对工作空间进行缩容前，需要先对项目空间进行缩容；对项目空间进行扩容前，需要实例中有可用的资源。

> **说明** 如果实例中剩余资源不足，您可以对实例先进行扩容，或者对工作空间名称-default项目空间先进行缩容。

工作空间资源变配

1. 登录实时计算控制台。
2. 单击**Flink全托管**页签。
3. 选择目标工作空间**操作**列下的**其他 > 工作空间资源变配**。

4. 在**资源变配**项，填写变配后的资源配额。

5. 选中**服务协议**后，单击**确认**并完成支付。

   完成支付后，即可在目标工作空间的**已使用/已购买CU**列，查看变配后的资源配额。

项目空间资源变配

针对已分配资源的项目空间，您可以增加或者减少其可用资源，即扩容或缩容。

1. 登录实时计算控制台。
2. 在**Flink全托管**页签，单击目标工作空间ID左侧的 + 图标。
3. 单击目标项目空间**操作**列下的**资源分配**。
4. 滑动指针，选择变配后的资源。
5. 单击**确认**。

5.4. 修改虚拟交换机

本文为您介绍如何修改虚拟交换机，解决虚拟交换机IP不足带来的作业启动失败的问题。

操作步骤

1. 登录实时计算控制台。
2. 单击**Flink全托管**页签。
3. 选择目标工作空间**操作**列下的**其他 > 修改虚拟交换机**。

4. 选中新增的交换机。

说明 如果列表中没有您需要的交换机，您可以单击列表上方提示信息中的**创建虚拟交换机**，完成**创建虚拟交换机**后，刷新此页面再选中新增的虚拟交换机。

5. 单击**确认**。

5.5. 操作审计日志

阿里云Flink全托管已与阿里云ActionTrail集成，您可以在ActionTrail中查看和检索用户行为日志，同时通过ActionTrail将日志投递到日志服务SLS或指定的OSS Bucket中，以满足实时审计、问题回溯分析等需要。

Flink全托管操作日志列表

事件名称	说明
AppendVswitchId	增加虚拟交换机
CheckAsiLeftResource	校验剩余资源数
CheckNameSpaceName	校验项目空间名称
CheckUserVSwitch	校验用户交换机
CheckUserVpc	校验VPC参数
CheckVirtualClusterName	校验集群名称
CreateNameSpace	创建项目空间
CreateServiceLinkedRole	创建关联角色
DeleteNameSpace	删除项目空间
DescribeAvailableCidr	获取可用的网段
GetClusterUsedCu	获取已使用的CU数

事件名称	说明
GetClusterState	获取集群状态
GetOrderDetails	获取订单详情
ListUserVpc	列出用户的VPC
ListUserVswitch	列出用户的交换机
ListSupportedZone	列出支持的可用区
ListOssInfo	列出OSS信息
OssParamsCheck	校验OSS参数
RefundAsiOrder	释放集群
ResourceSpec	预估Flink作业运行所需的资源
SearchOrdersV2	查询订单
UpdateNameSpace	更新项目空间配置

说明

- 请在资源所在地域，查看售卖类事件（新建、续费和变配）。
- 请在上海地域，查看非售卖类事件。

Flink全托管日志样例

Flink全托管集群添加虚拟交换机的日志示例如下。

```
{
  "ApiVersion": "2019-06-24",
  "RequestId": "83BB0682-9EBA-402F-8359-26B1676A9EE2",
  "EventType": "ApiCall",
  "UserIdentity": {
    "Type": "root-account",
    "InvokedBy": "",
    "AccountId": "1187xxxxxxxx4544",
    "UserName": "root",
    "PrincipalId": "1187xxxxxxxx4544",
    "AccessKeyId": "TMP.3Ker6WKYHexxxxxxpRuQaM",
    "Arn": ""
  },
  "AcsRegion": "cn-hangzhou",
  "EventName": "AppendVSwitchId",
  "IsBlack": false,
  "RequestParameters": {
    "AcsHost": "foasconsole-share.cn-shanghai.aliyuncs.com",
    "Is4Service": false,
    "ZoneId": "cn-beijing-f",
    "RequestId": "83BB0682-9EBA-402F-8359-26B1676A9EE2",
    "AcsProduct": "foasconsole-inner",
    "InstanceId": "sc_flinkserverless_public_cn-xxxxxxxx",
    "VSwitchIds": "[\\\"vsw-xxxxx\\\",\\\"vsw-xxxxx\\\"]",
    "AcceptLanguage": "zh-CN",
    "RegionId": "cn-shanghai",
    "HostId": "foasconsole-share.cn-shanghai.aliyuncs.com"
  },
  "EventSource": "foasconsole-share.cn-shanghai.aliyuncs.com",
  "ServiceName": "RealtimeCompute",
  "EventTime": "2020-12-01T11:54:59.366+0000",
  "ReferencedResources": {},
  "UserAgent": "realtime-compute.console.aliyun.com",
  "EventId": "83BB0682-9EBA-402F-8359-26B1676A9EE2",
  "ResponseElements": {
    "RequestId": "83BB0682-9EBA-402F-8359-26B1676A9EE2",
    "Result": true,
    "ErrCode": "000000"
  },
  "ErrorCode": "",
  "ErrorMessage": "",
  "EventVersion": "",
  "SourceIpAddress": "100.xx.xx.76"
}
```

6.权限管理

6.1. 阿里云账号角色授权

开通Flink全托管服务前，需要授予您的阿里云账号AliyunStreamAsiDefaultRole系统默认角色。本文为您介绍两种角色授权的方式。

背景信息

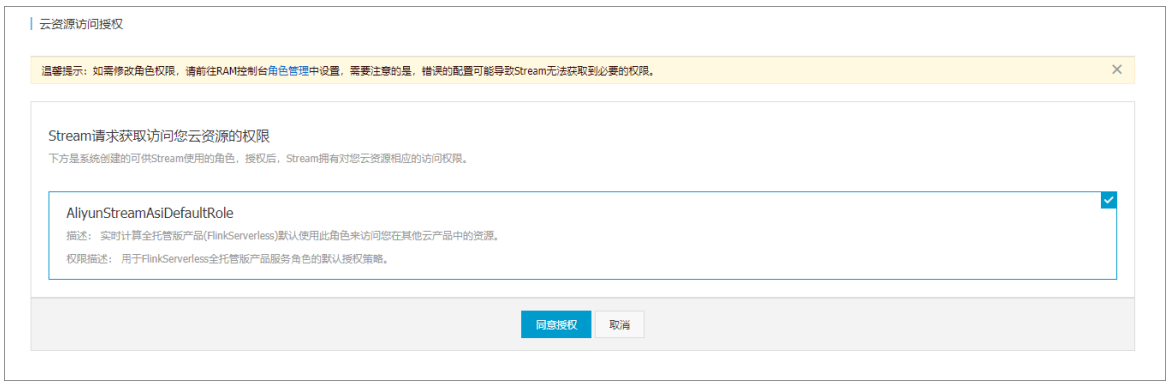
您的阿里云账号被正确授予AliyunStreamAsiDefaultRole角色后，Flink全托管服务才能正常地调用专有网络VPC、云服务器ECS、负载均衡SLB和应用实时监控服务ARMS等相关服务来启动Flink全托管的相关组件。

自动化授权

通常，第一次**购买产品**时，需要您进行自动化授权操作。

1. 登录实时计算控制台。
2. 单击**Flink全托管**下的**购买产品**。
3. 在**授权请求**页面，单击**前往RAM进行授权**。

4. 在**云资源访问授权**页面，单击页面下方的**同意授权**。



说明 系统已默认勾选AliyunStreamAsiDefaultRole角色。

手动授权（方式一）

如果您不小心删除了AliyunStreamAsiDefaultRole角色或者变更了授权策略导致Flink全托管服务不可用，请按照以下操作步骤重新授权。

1. 创建角色。
   i. 登录RAM控制台。
      - 云账号登录RAM控制台。
      - RAM用户登录RAM控制台。
   ii. 在左侧导航栏，选择**身份管理 > 角色**。
   iii. 在**角色**页面，单击**创建角色**。
   iv. 在**创建角色**面板，选择可信实体类型，单击**下一步**。

      可信实体类型如下：
      - 阿里云账号：受信云账号下的子用户可以通过扮演该RAM角色来访问您的云资源，受信云账号可以是当前云账号，也可以是其他云账号。
      - 阿里云服务：受信云服务可以通过扮演RAM角色来访问您的云资源。
      - 身份提供商：身份提供商功能，通过设置SSO可以实现从企业本地账号系统登录阿里云控制台，帮您解决企业的统一用户登录认证要求。

      详情请参见创建可信实体为阿里云账号的RAM角色、创建可信实体为阿里云服务的RAM角色或者创建可信实体为身份提供商的RAM角色。
   v. 输入相关角色信息，单击**完成**。

   > **说明** 角色名称为AliyunStreamAsiDefaultRole，如果AliyunStreamAsiDefaultRole角色已存在，则不必重复创建该角色。

2. 添加授权策略。
   i. 登录RAM控制台。
      - 云账号登录RAM控制台。
      - RAM用户登录RAM控制台。
   ii. 在左侧导航栏，选择**身份管理 > 角色**。
   iii. 在**角色**页面，找到AliyunStreamAsiDefaultRole角色，单击**操作**列下的**精确授权**。
   iv. 在**添加权限**面板，选择权限类型为**自定义策略**，并输入策略名称。

      需要为AliyunStreamAsiDefaultRole角色添加三个授权策略：
      - 策略一：自定义策略（AliyunStreamAsiDefaultRolePolicy0）

```
{
    "Version": "1",
    "Statement": [
        {
            "Action": "oss:ListBuckets",
            "Resource": "acs:oss:*:*:*",
            "Effect": "Allow"
        },
        {
            "Action": [
                "oss:AssociateEipAddress"
```

```
                "ecs:AssociateEipAddress",
                "ecs:AttachNetworkInterface",
                "ecs:AuthorizeSecurityGroup",
                "ecs:AuthorizeSecurityGroupEgress",
                "ecs:CreateNetworkInterface",
                "ecs:CreateNetworkInterfacePermission",
                "ecs:CreateSecurityGroup",
                "ecs:DeleteNetworkInterface",
                "ecs:DeleteNetworkInterfacePermission",
                "ecs:DeleteSecurityGroup",
                "ecs:DescribeNetworkInterfacePermissions",
                "ecs:DescribeNetworkInterfaces",
                "ecs:DescribeSecurityGroupAttribute",
                "ecs:DescribeSecurityGroupReferences",
                "ecs:DescribeSecurityGroups",
                "ecs:DetachNetworkInterface",
                "ecs:JoinSecurityGroup",
                "ecs:LeaveSecurityGroup",
                "ecs:ModifyNetworkInterfaceAttribute",
                "ecs:ModifySecurityGroupAttribute",
                "ecs:ModifySecurityGroupPolicy",
                "ecs:ModifySecurityGroupPolicy",
                "ecs:ModifySecurityGroupRule",
                "ecs:RevokeSecurityGroup",
                "ecs:RevokeSecurityGroupEgress",
                "ecs:UnassociateEipAddress"
            ],
            "Resource": "*",
            "Effect": "Allow"
        },
        {
            "Action": [
                "arms:ListDashboards"
            ],
            "Resource": "*",
            "Effect": "Allow"
        },
        {
            "Action": [
                "vpc:DescribeVpcAttribute",
                "vpc:DescribeVpcs",
                "vpc:DescribeVSwitchAttributes",
                "vpc:DescribeVSwitches",
                "vpc:DescribeRouteTableList",
                "vpc:DescribeRouteTables",
                "vpc:DescribeRouteEntryList",
                "vpc:DescribeRouterInterfaceAttribute",
                "vpc:DescribeRouterInterfaces",
                "vpc:DescribeVRouters"
            ],
            "Resource": "*",
            "Effect": "Allow"
        }
    ]
}
```

- 策略二：自定义策略（AliyunStreamAsiDefaultRolePolicy1）

```
{
    "Version": "1",
    "Statement": [
        {
            "Action": [
                "slb:AddBackendServers",
                "slb:AddListenerWhiteListItem",
                "slb:AddTags",
                "slb:AddVServerGroupBackendServers",
                "slb:CreateLoadBalancer",
                "slb:CreateLoadBalancerHTTPListener",
                "slb:CreateLoadBalancerHTTPSListener",
                "slb:CreateLoadBalancerTCPListener",
                "slb:CreateLoadBalancerUDPListener",
                "slb:CreateRules",
                "slb:CreateVServerGroup",
                "slb:DeleteLoadBalancer",
                "slb:DeleteLoadBalancerListener",
                "slb:DeleteRules",
                "slb:DeleteServerCertificate",
                "slb:DeleteVServerGroup",
                "slb:DescribeHealthStatus",
                "slb:DescribeListenerAccessControlAttribute",
                "slb:DescribeLoadBalancerAttribute",
                "slb:DescribeLoadBalancerHTTPListenerAttribute",
                "slb:DescribeLoadBalancerHTTPListenerAttributes",
                "slb:DescribeLoadBalancerHTTPSListenerAttribute",
                "slb:DescribeLoadBalancerTCPListenerAttribute",
                "slb:DescribeLoadBalancerUDPListenerAttribute",
                "slb:DescribeLoadBalancers",
                "slb:DescribeRegions",
                "slb:DescribeRules",
                "slb:DescribeServerCertificates",
                "slb:DescribeTags",
                "slb:DescribeVServerGroupAttribute",
                "slb:DescribeVServerGroups",
                "slb:ModifyLoadBalancerInstanceSpec",
                "slb:ModifyLoadBalancerInternetSpec",
                "slb:ModifyLoadBalancerPayType",
                "slb:RemoveBackendServers",
                "slb:RemoveListenerWhiteListItem",
                "slb:RemoveVServerGroupBackendServers",
                "slb:SetBackendServers",
                "slb:SetListenerAccessControlStatus",
                "slb:SetLoadBalancerHTTPListenerAttribute",
                "slb:SetLoadBalancerHTTPSListenerAttribute",
                "slb:SetLoadBalancerName",
                "slb:SetLoadBalancerStatus",
                "slb:SetLoadBalancerTCPListenerAttribute",
                "slb:SetLoadBalancerUDPListenerAttribute",
                "slb:SetRule",
```

```
                "slb:SetServerCertificateName",
                "slb:SetVServerGroupAttribute",
                "slb:StartLoadBalancerListener",
                "slb:StopLoadBalancerListener",
                "slb:UploadServerCertificate"
            ],
            "Resource": "*",
            "Effect": "Allow"
        }
    ]
}
```

- 策略三：自定义策略（FlinkServerlessPolicy）

```
{
    "Version": "1",
    "Statement": [
        {
            "Action": [
                "ram:*"
            ],
            "Resource": [
                "acs:ram:*:*:domain/*",
                "acs:ram:*:*:application/*"
            ],
            "Effect": "Allow"
        }
    ]
}
```

v. 单击**确定**。

> **说明** 创建好以上角色及策略后，即可正常开通及使用Flink全托管服务。

手动授权（方式二）

如果您不小心删除了AliyunStreamAsiDefaultRole角色或者变更了授权策略导致Flink全托管服务不可用，您可以按照以下操作步骤先删除资源编排服务ROS的资源栈、RAM角色和RAM权限策略后，登录实时计算控制台，重新授权。

1. 删除资源编排服务ROS的资源栈。
   i. 登录ROS控制台。
   ii. 在左侧导航栏，单击**资源栈**。
   iii. 在页面顶部，地域切换为**杭州**。
   iv. 在**资源栈名称**后，分别输入*FlinkServerlessStack*和*FlinkOnAckStack*，并单击 🔍 图标。

> **说明**
> - *FlinkServerlessStack*：Flink全托管的ROS资源栈统一名称。
> - *FlinkOnAckStack*：容器服务ACK的ROS资源栈统一名称。

v. 在对应资源栈**操作**列中，单击**删除**，在弹出的对话框中，单击**确定**。

2. 删除RAM角色。
   i. 登录RAM控制台。
      - 云账号登录RAM控制台。
      - RAM用户登录RAM控制台。
   ii. 在左侧导航栏，选择**身份管理 > 角色**。
   iii. 在搜索栏中，输入*AliyunStreamAsiDefaultRole*。
   iv. 在**操作**列中，单击**删除**，在弹出的对话框中，单击**确定**。

> **说明** 您需要单击*AliyunStreamAsiDefaultRole*名称，进入角色详情页面，**移除权限**后，此角色才能删除成功。

3. 删除RAM权限策略。
   i. 登录RAM控制台。
      - 云账号登录RAM控制台。
      - RAM用户登录RAM控制台。
   ii. 在左侧导航栏，单击**权限管理 > 权限策略**。
   iii. 在搜索栏中，分别输入*AliyunStreamAsiDefaultRolePolicy0*、*AliyunStreamAsiDefaultRolePolicy1*和*FlinkServerlessPolicy*。
   iv. 在**操作**列中，单击**删除**，在弹出的对话框中，单击**确定**。
4. 在实时计算控制台，重新授权。
   i. 登录实时计算控制台。
   ii. 单击**Flink全托管**下的**购买产品**。
   iii. 在**授权请求**页面，单击**前往RAM进行授权**。

iv. 在**云资源访问授权**页面，单击页面下方的**同意授权**。

权限说明

在使用Flink全托管的过程中，不同的使用场景下，您可能会涉及开通以下权限。当您需要开通以下权限时，请参见上文中的自动化授权、手动授权（方式一）或者手动授权（方式二）。

- ECS相关权限

  为了能从公网访问开发控制台，需要在您账号下开通弹性公网IP（EIP）。此外，为了连通VPC内的资源，需要在VPC下创建弹性网卡（ENI）。这些ENI会被添加到Flink Serverless专属安全组中，因此Flink服务需要有对EIP、安全组和ENI的操作权限。

  > **注意** EIP的公网流量会产生额外费用。

权限名称（Action）	权限说明
ecs:AssociateEipAddress	申请EIP地址，您可以通过公网访问Flink服务。
ecs:AttachNetworkInterface	允许Flink服务将您的弹性网卡绑定到Flink资源池。
ecs:AuthorizeSecurityGroup	Flink产品会创建一个新的安全组，该权限用于增加该安全组内一条入方向安全组规则。
ecs:AuthorizeSecurityGroupEgress	Flink产品会创建一个新的安全组，该权限用于增加该安全组内一条出方向安全组规则。
ecs:CreateNetworkInterface	允许Flink服务创建您VPC内的弹性网卡，支持Flink服务连接您的VPC。
ecs:CreateNetworkInterfacePermission	允许Flink服务授权弹性网卡权限。
ecs:CreateSecurityGroup	Flink产品会创建一个新的安全组，该权限用于创建安全组。
ecs:DeleteNetworkInterface	Flink任务结束后删除对应资源的弹性网卡。
ecs:DeleteNetworkInterfacePermission	允许Flink服务拥有您的弹性网卡解绑权限。
ecs:DeleteSecurityGroup	Flink产品会创建一个新的安全组，该权限用于删除该安全组。
ecs:DescribeNetworkInterfacePermissions	允许您的弹性网卡从Flink Serverless资源池解绑。

权限名称（Action）	权限说明
ecs:DescribeNetworkInterfaces	允许Flink服务查询弹性网卡。
ecs:DescribeSecurityGroupAttribute	允许Flink服务查询一个安全组的安全组规则。
ecs:DescribeSecurityGroupReferences	允许Flink服务查询安全组及安全组级别的授权行为。
ecs:DescribeSecurityGroups	允许Flink服务查询创建的安全组的基本信息。
ecs:DetachNetworkInterface	允许Flink服务将您的弹性网卡从Flink资源池解绑。
ecs:JoinSecurityGroup	允许Flink服务将弹性网卡加入指定安全组。
ecs:LeaveSecurityGroup	允许Flink服务将弹性网卡移除指定安全组。
ecs:ModifyNetworkInterfaceAttribute	允许Flink服务修改一个弹性网卡的名称、描述以及所属安全组等。
ecs:ModifySecurityGroupAttribute	允许Flink服务修改一个安全组的名称或者描述。
ecs:ModifySecurityGroupPolicy	允许Flink服务修改安全组内的连通策略。
ecs:ModifySecurityGroupRule	允许Flink服务修改安全组入方向规则的描述信息。
ecs:RevokeSecurityGroup	允许Flink删除一条安全组入方向的权限。
ecs:RevokeSecurityGroupEgress	允许Flink删除一条安全组出方向的权限。
ecs:UnassociateEipAddress	允许Flink服务释放EIP。

- 负载均衡SLB相关权限

  为了您能从公网访问开发控制台，需要在您账号下开通按量付费SLB服务，因此需要管理SLB的相关权限。

  注意　按量付费的SLB服务会产生额外费用。

权限名称（Action）	权限说明
slb:AddBackendServers	允许Flink服务修改负载均衡的Backend。
slb:AddListenerWhiteListItem	允许Flink服务添加监听访问控制白名单。
slb:AddTags	允许Flink服务为指定的负载均衡实例添加标签。
slb:AddVServerGroupBackendServers	允许Flink服务向指定的后端服务器组中添加后端服务器。
slb:CreateLoadBalancer	允许Flink服务创建负载均衡实例。
slb:CreateLoadBalancerHTTPListener	允许Flink服务创建HTTP监听。
slb:CreateLoadBalancerHTTPSListener	允许Flink服务创建HTTPS监听。

权限名称（Action）	权限说明
slb:CreateLoadBalancerTCPListener	允许Flink服务创建TCP监听。
slb:CreateLoadBalancerUDPListener	允许Flink服务创建UDP监听。
slb:CreateRules	允许Flink服务为指定HTTP或HTTPS监听创建转发规则。
slb:CreateVServerGroup	允许Flink服务添加后端服务器组并向指定的后端服务器组中添加后端服务器。
slb:DeleteLoadBalancer	允许Flink服务删除负载均衡实例。
slb:DeleteLoadBalancerListener	允许Flink服务删除后端监听。
slb:DeleteRules	允许Flink服务删除指定HTTP或HTTPS监听的转发规则。
slb:DeleteVServerGroup	允许Flink服务删除服务器组。
slb:DescribeHealthStatus	允许Flink服务查询后端服务器的健康状态。
slb:DescribeListenerAccessControlAttribute	允许Flink服务查询指定负载均衡监听的白名单配置。
slb:DescribeLoadBalancerAttribute	允许Flink服务查询指定负载均衡实例的详细信息。
slb:DescribeLoadBalancerHTTPListenerAttribute	允许Flink服务查询HTTP监听配置。
slb:DescribeLoadBalancerHTTPSListenerAttribute	允许Flink服务查询HTTPS监听配置。
slb:DescribeLoadBalancerTCPListenerAttribute	允许Flink服务查询TCP监听配置。
slb:DescribeLoadBalancerUDPListenerAttribute	允许Flink服务查询UDP监听配置。
slb:DescribeLoadBalancers	允许Flink服务查询已创建的负载均衡实例。
slb:DescribeRegions	允许Flink服务查询可用负载均衡实例的地域。
slb:DescribeRules	允许Flink服务查询指定监听已配置的转发规则。
slb:DescribeTags	允许Flink服务查询Tags列表。
slb:DescribeVServerGroupAttribute	允许Flink服务查询服务器组的详细信息。
slb:DescribeVServerGroups	允许Flink服务查询服务器组列表。
slb:ModifyLoadBalancerInstanceSpec	允许Flink服务修改负载均衡的示例规格。
slb:ModifyLoadBalancerInternetSpec	允许Flink服务修改公网负载均衡实例的计费方式（按流量计费或按带宽计费）。
slb:ModifyLoadBalancerPayType	允许Flink服务修改公网负载均衡实例的购买方式（按量付费或包年包月）。

权限名称（Action）	权限说明
slb:RemoveBackendServers	允许Flink服务移除后端服务器。
slb:RemoveListenerWhiteListItem	允许Flink服务删除监听白名单中的IP。
slb:RemoveVServerGroupBackendServers	允许Flink服务从指定的后端服务器组中移除后端服务器。
slb:SetBackendServers	允许Flink服务设置后端服务器权重。
slb:SetListenerAccessControlStatus	允许Flink服务设置是否开启指定监听的白名单访问控制。
slb:SetLoadBalancerHTTPListenerAttribute	允许Flink服务修改HTTP监听的配置。
slb:SetLoadBalancerHTTPSListenerAttribute	允许Flink服务修改HTTPS监听的配置。
slb:SetLoadBalancerName	允许Flink服务设置负载均衡实例的名称。
slb:SetLoadBalancerStatus	允许Flink服务设置负载均衡实例的状态。
slb:SetLoadBalancerTCPListenerAttribute	允许Flink服务修改TCP监听的配置。
slb:SetLoadBalancerUDPListenerAttribute	允许Flink服务修改UDP监听的配置。
slb:SetRule	允许Flink服务修改虚拟服务器的转发规则。
slb:SetServerCertificateName	允许Flink服务设置服务器证书名称。
slb:SetVServerGroupAttribute	允许Flink服务修改虚拟服务器组配置。
slb:StartLoadBalancerListener	允许Flink服务启动指定监听。
slb:StopLoadBalancerListener	允许Flink服务停止指定监听。

- OSS相关权限

  为了使您能够查看OSS Bucket列表，需要您开通OSS相关权限。

权限名称（Action）	权限说明
oss:ListBuckets	允许Flink服务查看OSS Bucket列表。

- ARMS相关权限

  Flink的指标会存入ARMS，因此会为您开通ARMS服务。

  > **注意** 您可以免费使用ARMS中Flink相关Metric的存储。

权限名称（Action）	权限说明
arms:ListDashboards	允许Flink服务查看ARMS Dashboard列表。

- VPC相关权限

  开通Flink全托管的过程中，需要VPC内资源的Describe权限。

权限名称（Action）	权限说明
vpc:DescribeVpcAttribute	允许Flink服务查询指定VPC的配置信息。
vpc:DescribeVpcs	允许Flink服务查询已经创建的VPC。
vpc:DescribeVSwitchAttributes	允许Flink服务查询指定交换机的信息。
vpc:DescribeVSwitches	允许Flink服务查询已创建的交换机。
vpc:DescribeRouteTableList	允许Flink服务查询路由表列表。
vpc:DescribeRouteTables	允许Flink服务查询指定路由表。
vpc:DescribeRouteEntryList	允许Flink服务查询路由表条目列表。
vpc:DescribeRouterInterfaceAttribute	允许Flink服务查询路由器接口配置。
vpc:DescribeRouterInterfaces	允许Flink服务查询路由器接口。
vpc:DescribeVRouters	允许Flink服务查询指定地域的路由器列表。

- RAM相关权限

  开通Flink全托管的过程中，需要RAM相关权限进行资源配置。

权限名称（Action）	权限说明
ram:*	可以增删改查domain和application这两个RAM资源。

6.2. RAM用户授权

如果RAM用户需要对Flink全托管进行购买或资源调整等操作，则需要阿里云账号为该RAM用户授权。只有当RAM用户被阿里云账号授予相关的权限后，RAM用户才可以正常访问实时计算控制台，进行购买或者资源调整等操作。本文为您介绍如何为RAM用户授权。

背景信息

Flink全托管为您提供了以下三种权限策略，您可以根据业务情况为RAM用户选择适合的权限策略。

类别	说明
AliyunStreamFullAccess（系统策略集合）	包括权限列表中所有的权限。
AliyunStreamReadOnlyAccess（系统策略集合）	只读访问流计算（Stream）服务，无法操作。仅包括权限列表中的DescribeInstances、QueryCreateInstancePrice、QueryRenewInstancePrice、QueryModifyInstancePrice、QueryConvertPostpayInstancePrice和DescribeNamespaces权限。

类别	说明
自定义策略	您可以为RAM用户授予一个或多个不同的权限，实现灵活的细粒度权限管理。

说明

- 系统策略：统一由阿里云创建，您只能使用不能修改，策略的版本更新由阿里云维护。
- 自定义策略：您可以自主创建、更新和删除，策略的版本更新由您自己维护。

前提条件

已创建RAM用户，如果您未创建RAM用户，请参见创建RAM用户。

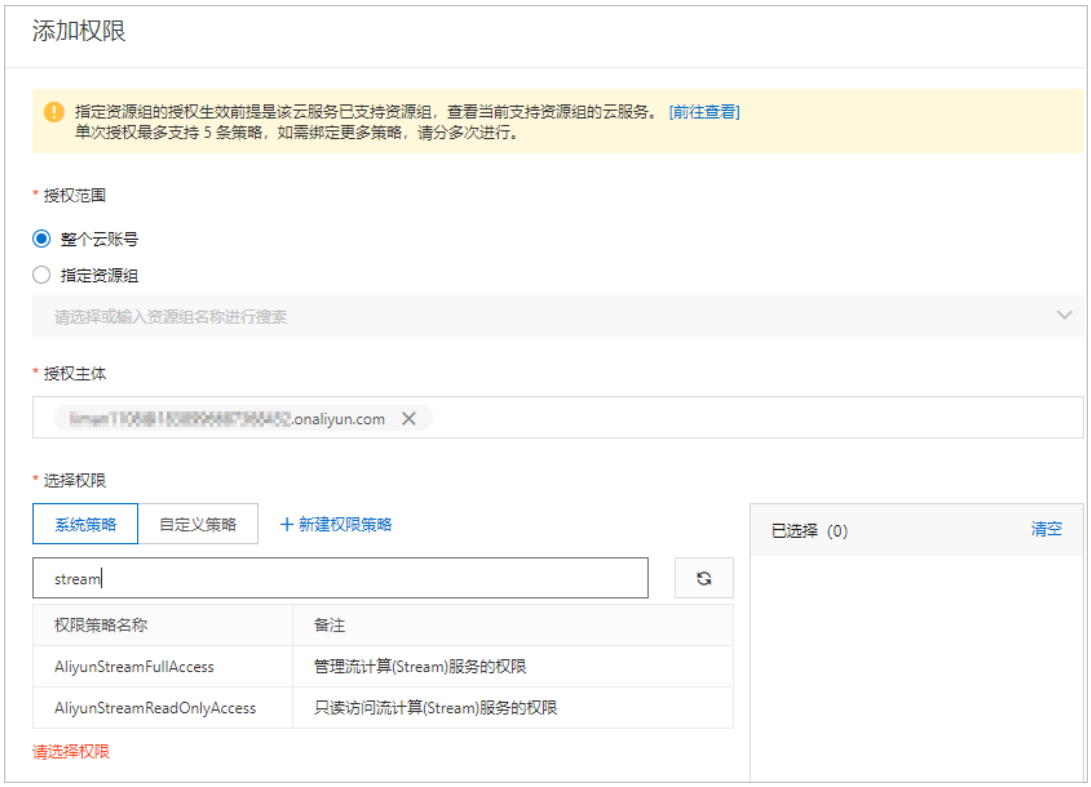
授权操作

1. 使用阿里云账号登录RAM控制台。
2. 在左侧导航栏，选择**身份管理 > 用户**。
3. 在**用户**页面，单击目标RAM用户**操作**列的**添加权限**。
4. 在**添加权限**面板，为RAM用户添加权限。
   i. 选择授权应用范围。请选择为**整个云账号**。
      - **整个云账号**：权限在当前阿里云账号内生效。
      - **指定资源组**：权限在指定的资源组内生效。
   ii. 输入被授权主体。

      被授权主体即需要授权的RAM用户，系统会自动填入当前的RAM用户，您也可以添加其他RAM用户。
   iii. 选择权限策略。

- 系统策略

  在**系统策略**页签，在文本框中输入*stream*后，单击对应的策略名称。



- 自定义策略

  自定义策略创建步骤，请参见创建自定义权限策略。

在使用自定义脚本时，需要将权限列表表格中的信息替换为您的实际权限信息。展示所有集群信息的策略脚本示例如下。

```
{
    "Version": "1",
    "Statement": [
        {
            "Action": "stream:DescribeVvpInstances",
            "Resource": "acs:stream:cn-beijing:1838996687368452:vvpinstance/*",
            "Effect": "Allow"
        }
    ]
}
```

> **说明** 权限策略中Action表示要执行的操作，Resource表示要操作的对象，Effect表示授权效果是允许还是禁止。权限策略语法和结构的详情请参见权限策略语法和结构。

5. 单击**确定**。
6. 单击**完成**。

权限列表

说明 您需要将策略内容中的以下参数替换为您的实际值。需要替换的参数如下：

- {#regionId}：目标Flink全托管实例所在的地域。
- {#accountId}：阿里云账号的uid。
- {#instanceId}：目标Flink全托管的实例ID。
- {#namespace}：目标工作空间的名称。

类别	权限	策略内容
Flink全托管实例	新购Flink全托管实例	`Action: "stream:CreateVvpInstance"` `Resource: "acs:stream:{#regionId}:{#accountId}:vvpinstance/*"`
	释放按量付费的Flink全托管实例	`Action: "stream:DeleteVvpInstance"` `Resource: "acs:stream:{#regionId}:{#accountId}:vvpinstance/{#instanceId}"`
	Flink全托管实例续费	`Action: "stream:RenewVvpInstance"` `Resource: "acs:stream:{#regionId}:{#accountId}:vvpinstance/{#InstanceId}"`
	Flink全托管实例扩缩容	`Action: "stream:ModifyVvpPrepayInstanceSpec"` `Resource: "acs:stream:{#regionId}:{#accountId}:vvpinstance/{#instanceId}"`
	变更Flink全托管实例付费模式	`Action: "stream:ConvertVvpInstance"` `Resource: "acs:stream:{#regionId}:{#accountId}:vvpinstance/{#InstanceId}"`
	查看Flink全托管实例	`Action: "stream:DescribeVvpInstances"` `Resource: "acs:stream:{#regionId}:{#accountId}:vvpinstance/*"`
	查询Flink全托管实例创建价格	`Action: "stream:QueryCreateVvpInstance"` `Resource: "acs:stream:{#regionId}:{#accountId}:vvpinstance/*"`

类别	权限	策略内容
	查询Flink全托管实例续费价格	`Action: "stream:QueryRenewVvpInstance"` `Resource: "acs:stream:{#regionId}:{#accountId}:vvpinstance/{#InstanceId}"`
	查询Flink全托管实例扩缩容价格	`Action: "stream:QueryModifyVvpPrepayInstanceSpec"` `Resource: "acs:stream:{#regionId}:{#accountId}:vvpinstance/{#InstanceId}"`
	查询Flink全托管按量付费实例转包年包月价格	`Action: "stream:QueryConvertVvpInstance"` `Resource: "acs:stream:{#regionId}:{#accountId}:vvpinstance/{#InstanceId}"`
工作空间	创建工作空间	`Action: "stream:CreateVvpNamespace"` `Resource: "acs:stream:{#regionId}:{#accountId}:vvpinstance/{#instanceId}/vvpnamespace/*"`
	删除工作空间	`Action: "stream:DeleteVvpNamespace"` `Resource: "acs:stream:{#regionId}:{#accountId}:vvpinstance/{#instanceId}/vvpnamespace/{#namespace}"`
	变更工作空间资源	`Action: "stream:ModifyVvpPrepayNamespaceSpec"` `Resource: "acs:stream:{#regionId}:{#accountId}:vvpinstance/{#instanceId}/vvpnamespace/{#namespace}"`

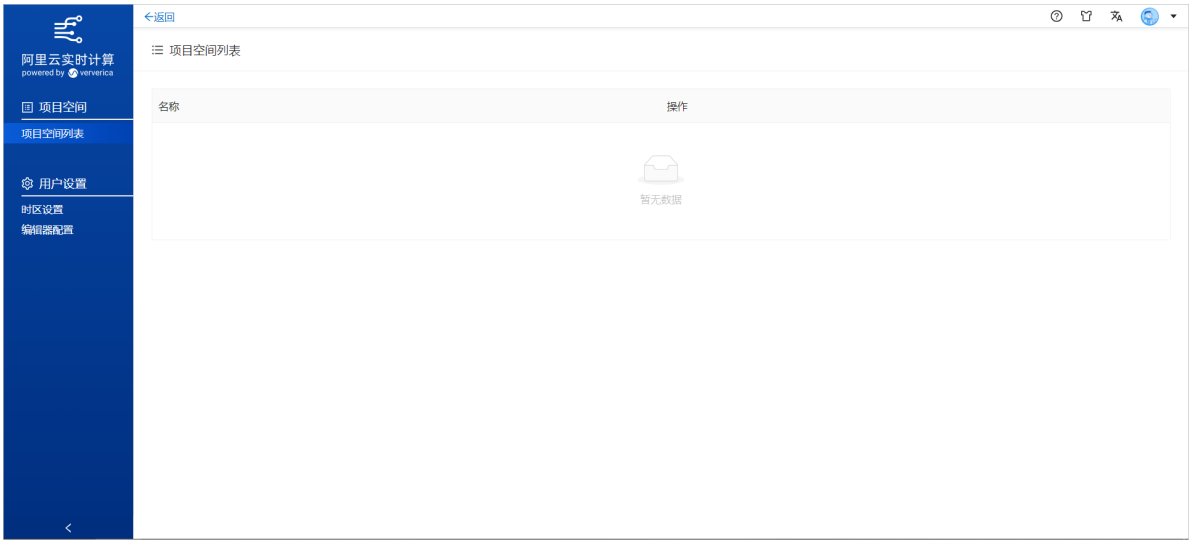
	查看工作空间	`Action: "stream:DescribeVvpNamespaces"`
		`Resource: "acs:stream:*:{#accountId}:vvpnamespace/{#InstanceId}"`
		`Resource: "acs:stream:{#regionId}:{#accountId}:vvpinstance/{#instanceId}/vvpnamespace/*"`

6.3. 作业操作账号授权

本文为您介绍如何为阿里云账号或RAM用户授权，以进行作业开发等相关操作。

背景信息

- Flink全托管产品支持给其他阿里云账号或RAM用户授权，来共同使用Flink全托管产品。如果其他阿里云账号或RAM用户没有被购买Flink全托管的阿里云账号授权，则在进入到Flink全托管开发控制台后，会看到项目空间列表为空，无法进行作业开发等相关操作。



- 阿里云账号或RAM用户被授予不同的角色，可以使用的功能会有差异。详情请参见下表。

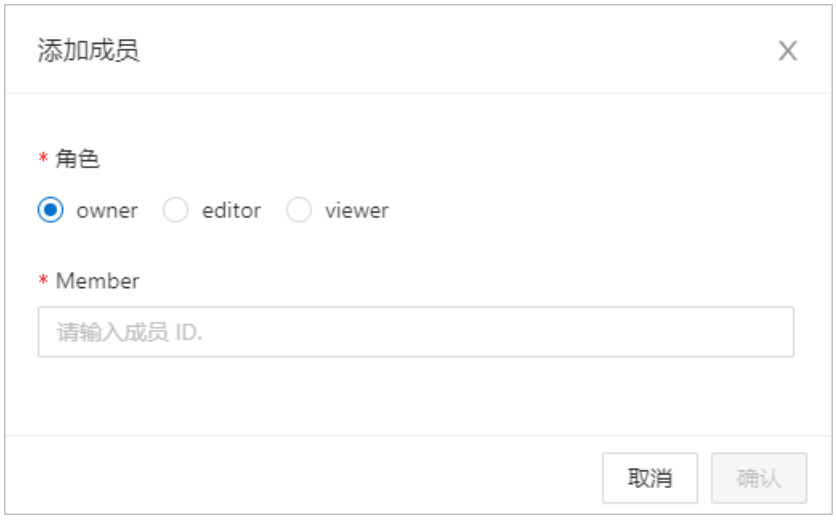
功能	owner	editor	viewer
查看作业列表	Y	Y	Y
启停作业	Y	Y	N
更改作业配置	Y	Y	N
查看资源	Y	Y	Y
上传资源	Y	Y	N
编写SQL	Y	Y	N
注册UDF	Y	Y	N
注册元数据	Y	Y	N
查看作业模板	Y	Y	Y
增删改作业模板	Y	N	N
管理成员	Y	N	N

账号授权

1. 创建阿里云账号或者RAM用户。
   - 阿里云账号的创建步骤，请参见阿里云账号注册流程。

- RAM用户的创建步骤，请参见创建RAM用户。

2. 在Flink全托管开发控制台给阿里云账号或者RAM用户授权。
   i. 使用阿里云账号登录实时计算管理控制台。
   ii. 在**Flink全托管**页签，单击目标工作空间**操作**列下的**控制台**。
   iii. 在左侧导航栏上，选择**系统管理 > 成员管理**。
   iv. 单击**添加角色绑定**。
   v. 选择**角色**，填写Member信息。



   - **角色**：不同角色的功能使用差异请参见背景信息。
   - **Member**：填写阿里云账号或者RAM用户账号信息。

用户	获取方法
阿里云账号	填写阿里云账号的账号ID或登录账号名称。在实时计算管理控制台页面右上角，鼠标悬停在头像上，单击**基本资料**后进行查看。
RAM用户	填写RAM用户的UID或登录账号名称。查看方法请参见查看用户组基本信息。

   vi. 单击**确认**。

3. 使用阿里云账号或者RAM用户登录Flink全托管开发控制台。

   RAM用户登录开发控制台的操作步骤，请参见RAM用户登录阿里云控制台。

> **注意** 如果该阿里云账号或RAM用户已经登录Flink全托管开发控制台，在被授权后，需要重新登录Flink全托管开发控制台，否则会看不到项目信息。

7.作业开发

7.1. 配置开发测试环境（Session集群）

Session集群适用于非生产环境的开发测试环境，您可以使用Session集群模式部署或调试作业，提高作业JM（Job Manager）资源利用率和提高作业启动速度。

背景信息

Flink全托管支持Per-Job集群和Session集群两种集群模式。两种集群模式有以下区别：

- Per-Job集群（默认）：作业之间资源隔离，每个作业都需要一个独立的JM，因为小任务JM的资源利用率较低，因此适用于占用资源比较大或持续稳定运行的作业。
- Session集群：多个作业可以复用相同的JM，可以提高JM资源利用率。因此适用于大量占用资源比较小或任务启停比较频繁的小并发作业，可以有效节约资源开销。

> **说明**
>
> - 每个项目空间可以配置多个Session集群，但只能选择一个Session集群开启SQL Preview 。该配置项的详细说明请参见本文参数解释中的**设置为SQL Previews集群**。
> - Session集群暂不支持开启自动调优功能。
> - 无论您是否使用Session集群，在创建Session集群时会消耗集群资源，其中消耗的资源和您创建集群时选择的资源配置有关。
> - VVR 3.0.4及以下版本的每个Session集群运行后，都会额外消耗0.5 CU的资源。

使用限制

Session集群有以下限制：

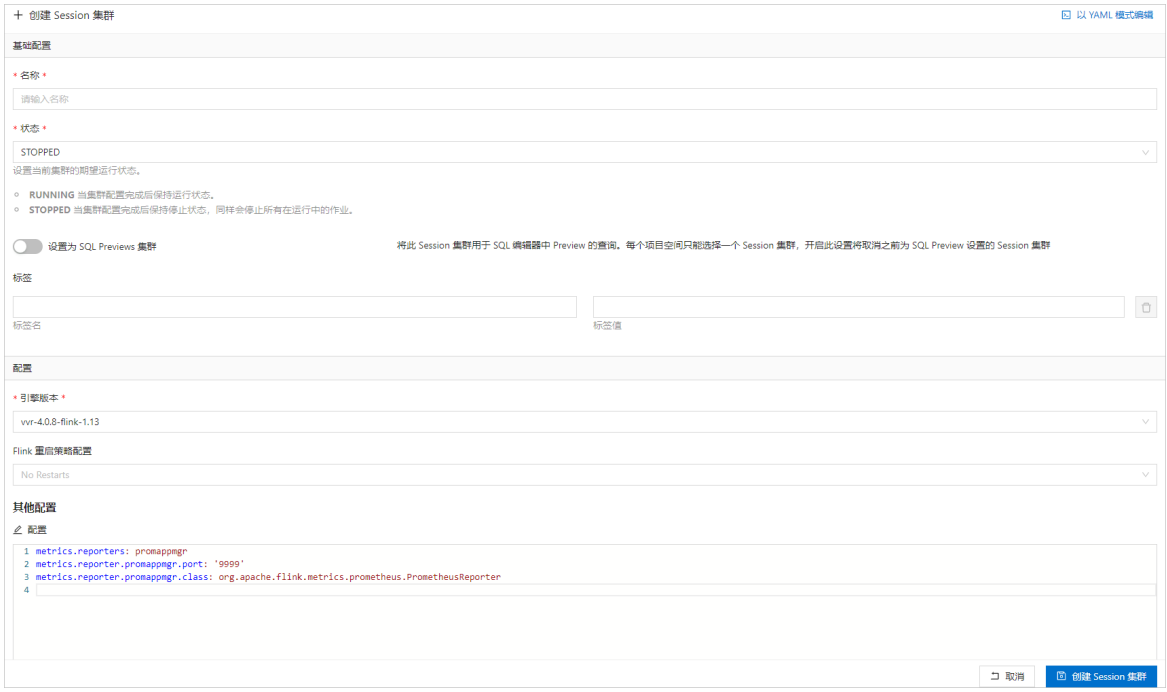
- 暂停Session集群后，不支持修改Flink计算引擎版本。
- 暂停Session集群后，不支持对TM个数或资源进行缩容。
- 不支持显示数据曲线。
- 不支持监控告警功能。

注意事项

- JobManager单点故障会对集群内的所有作业造成影响。
- TaskManager单点故障会对在其上有task运行的相关作业造成影响。
- 同一个TaskManager内部，不同Task之间如果没有进程隔离，则存在相互影响的潜在风险。
- 如果Session集群为默认配置，则有以下建议：
  - 对于单并发的小作业，建议整个集群的作业总数不超过100个。
  - 对于复杂作业，建议单作业最大并发数不超过512，64个并发的中等规模作业单集群不多于32个。否则可能会出现心跳超时等问题影响集群稳定性。此时，您需要增大心跳间隔和心跳超时时间。
  - 如果您需要同时运行更多的任务，则需要增加Session集群的资源配置。
- 不建议使用开启了SQL Preview的Session集群运行作业。因为如果产品后续发布新版本，则需要停止Session集群后，手动适配Flink计算引擎，否则本地调试会报错。

创建Session集群

1. 登录实时计算控制台。
2. 在**Flink全托管**页签，单击目标工作空间**操作**列下的**开发控制台**。
3. 在左侧导航栏上，选择**系统管理 > Session集群**。
4. 在页面右上角，单击**创建Session集群**。
5. 填写配置信息。



参数解释如下表所示。

模块	配置项	说明
基础配置	名称	集群名称。
	状态	设置当前集群的期望运行状态： ○ STOPPED：当集群配置完成后保持停止状态，同样会停止所有在运行中的作业。 ○ RUNNING：当集群配置完成后保持运行状态。
	设置为SQL Previews集群	将此Session集群设置为SQL Preview查询的资源集群。 **说明** 每个项目空间只能有一个Session集群开启SQL Preview，开启此设置将取消之前为SQL Preview设置的Session集群。
	标签名	您可以在标签选项中添加作业标签，便于在总览页面快速定位作业。
	标签值	无。

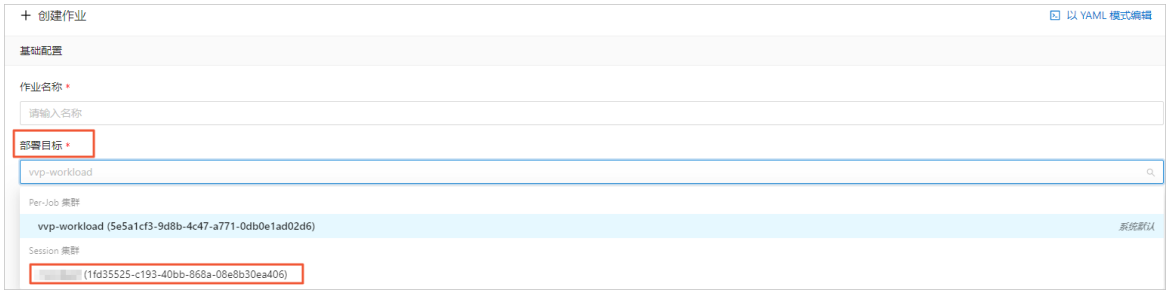
模块	配置项	说明
配置	引擎版本	当前作业使用的Flink的引擎版本。 说明 Python API作业需要选择vvr-2.1.4-flink 1.11及以上版本。
	Flink重启策略配置	该参数取值如下： ○ No Restarts：不重启。 ○ Fixed Delay：固定延迟。选择该选项后，您还需要填写尝试重启的次数和每次重启时间间隔。 ○ Failure Rate：故障率。选择该选项后，您还需要填写检测故障率的时间间隔，时间间隔内的最大失败次数和每次重启时间间隔。 说明 如果您没有配置该参数，则按Apache Flink默认的重启策略，即当有Task失败时，如果没有开启Checkpoint，JobManager进程不会重启。如果开启了Checkpoint，则JobManager进程会重启。
	其他配置	在此设置其他Flink配置。例如 `taskmanager.numberOfTaskSlots: 1` 。
资源配置	**Task Managers数量**	默认与并行度一致。
	Job Manager CPUs	默认值为1。
	Job Manager Memory	最小值为1 GiB，推荐值为2 GiB。单位建议使用GiB或MiB，例如，1024 MiB或1.5 GiB。 JobManager推荐配置包含JobManager资源以及心跳相关参数。具体内容如下： ○ 由于JobManager上承载TaskManager心跳、作业Task序列化和资源调度等功能。因此建议JobManager资源不小于默认配置，请根据集群负载情况进行处理。 ○ 为了集群稳定，避免JobManager主线程繁忙导致心跳超时。因此建议心跳间隔（heartbeat.interval）不小于10秒，同时心跳超时（heartbeat.timeout）不小于50秒。请根据TaskManager个数与作业量上涨情况而上调。
	Task Manager CPUs	默认值为2。

模块	配置项	说明
	Task Manager Memory	最小值为1 GiB，推荐值为8 GiB。单位建议使用GiB或MiB，例如，1024 MiB或1.5 GiB。 TaskManager推荐配置包含单个TaskManager的Slot个数（taskmanager.numberOfTaskSlots）及TaskManager资源大小。具体内容如下： ○ 对于单并发小作业，建议单Slot的CPU内存比为1:4，使用的资源不小于1核2 GiB。 ○ 对复杂作业，建议单Slot使用资源不小于1核4 GiB。在默认资源配置下，每个TaskManager可以配置2个Slot。 ○ TaskManager资源不宜过小，也不宜过大，推荐默认资源配置并将Slot数目设为2。 说明 ■ 如果单个TaskManager资源过小，则可能影响其上作业的稳定性，并且由于其Slot数目不多，无法有效平摊TaskManager的开销，降低了资源的利用效率。 ■ 如果单个TaskManager资源过大，则TaskManager上运行的作业数会很多，一旦TaskManager发生单点故障，影响面会很大。
日志配置	Root Log Level	TRACE、DEBUG、INFO、WARN和ERROR。
	Logger level	填写日志级别。
	Logging Profile	日志模板，可以选择系统模板，也可以选择用户配置。

说明 关于Flink与资源编排框架（例如Kubernetes、Yarn等）集成的相关选项详情，请参见Resource Orchestration Frameworks。

6. 单击**创建Session集群**。

   Session集群创建完成后，您就可以在创建作业的**部署目标**中选择此集群。



7.2. Flink SQL开发指南

7.2.1. 概述

Flink全托管开发控制台为您提供作业开发、作业启动、作业调试、管理自定义函数、配置Hive Metastore和管理自定义Connectors功能。

Flink SQL开发指南主要包含以下内容：

- 作业开发
- 作业启动
- 作业调试
- 管理自定义函数（UDF）
- 管理Hive Metastore
- 管理自定义Connectors

7.2.2. 模板中心

7.2.2.1. 代码模板

Flink全托管为您提供了21种代码模板，每种代码模板都为您提供了具体的使用场景、代码示例和使用指导。您可以通过代码模板快速地了解Flink产品功能和相关语法，实现自己的业务逻辑。本文为您介绍代码模板的使用场景和操作步骤。

背景信息

Flink全托管支持的模板和其使用场景详情如下表所示。

模板类别	模板名称	使用场景
基础模板	创建表	如果您需要创建表并保存在Catalog中，则可以使用DDL创建表。对表的操作和数据库类似。
	创建临时表	如果您只需要在当前会话或SQL脚本中使用某些表，则可以使用临时表。
	创建临时视图	如果您需要在Flink SQL开发过程中，重用代码，组织长查询或者SQL脚本来简化开发，则可以使用临时视图。
	INSERT INTO	如果您需要将查询结果写入到外部存储系统中的表，从而提供给下游应用程序，则可以使用INSERT INTO语法。
	STATEMENT SET	如果您需要将查询的结果输出到下游的两到多个外部系统中，则可以使用STATEMENT SET语法。
	Watermark	如果您希望Flink可以正确处理乱序数据，则需要使用Watermark。您可以在建表DDL中创建一个Watermark，即在一个已有字段上定义一个Watermark生成表达式，标记这个字段为时间属性字段。详情请参见Event Time and Watermarks。
	GROUP BY	如果您需要实时对数据进行聚合分析，则可以使用GROUP BY语句。
	滚动窗口聚合	如果您需要对时间序列数据进行分组，并对每个分组内的数据进行聚合分析，则可以使用滚动窗口。

模板类别	模板名称	使用场景
聚合分析	滑动窗口聚合	如果您需要每间隔一段时间更新一个窗口内的数据，则可以使用滑动窗口。
	累积窗口聚合	如果您需要提早看到窗口的计算结果，例如每分钟看到最新的窗口结果，则可以使用累积窗口。
	会话窗口聚合	如果您需要实时统计用户在一个活跃会话期间的数据，则可以使用会话窗口。
	Over窗口聚合	如果您需要对一个窗口内的每个元素进行统计分析，则需要使用OVER窗口。
	级联窗口聚合	如果您需要同时对同一个流的数据进行不同时间维度（例如1min、5min、30min、1h）的聚合，则可以使用级联窗口。
去重模板	去重	如果您需要去除数据流中的重复数据，则可以使用去重语法。
Top-N模板	Top-N	如果您需要根据业务要求计算出当前排名前几位或者后几位的数据，则可以使用Top-N语法。
	窗口Top-N	如果您需要在某个时间范围内计算出目前排名前几位或者后几位的数据，则可以使用窗口Top-N语法。
CEP模板	模式检测（CEP）	在数据流情景中，如果您需要搜索一组事件模式，则可以使用MATCH_RECOGNIZE语法。
Join模板	Regular Join	如果您需要关联查询其他流表的数据进行计算，则可以使用Regular Join语法。
	Interval Join	如果您需要关联某张表在指定时间范围内的数据，则可以使用Interval Join语法。
	时态表Join	如果您需要流表中的每条数据都关联其他表中对应时间版本数据，则可以使用时态表Join。
	维表Join	如果您需要关联静态维表上的数据，则可以使用维表Join。

使用限制

仅实时计算引擎vvr-4.0.12-flink-1.13及以上版本支持使用代码模板。

注意事项

如果该模板中不包含INSERT INTO语句，则不支持上线该作业。

操作步骤

本文以Top-N模板为您介绍如何使用代码模板。

1. 登录实时计算管理控制台。
2. 在**Flink全托管**页签，单击对应工作空间**操作**列下的**控制台**。
3. 在左侧导航栏上，选择**应用** > **模板中心**。

4. 在**代码模板**页签，双击目标模板名称，或者单击选中模板后，在右下角单击**继续**。

5. 根据需要，修改**文件名称**、**部署目标**、**存储位置**和SQL代码。

6. 单击**确认**。

   单击**确认**后，就会进入**作业开发**页面。后面的操作和正常的作业开发完全相同，参数配置详情和后续操作请参见作业开发和作业启动。

7.2.2.2. 数据同步模板

Flink CDC支持数据的实时同步和表结构变更的自动同步。您可以使用数据同步模版帮您快速生成数据同步的Flink SQL作业代码。本文为您介绍如何使用多库多表同步或者分库分表合并模板实现数据同步。

背景信息

Flink全托管支持多库多表同步和分库分表合并两种数据同步方式，二者的区别请参见下表。

同步方式	详情

同步方式	详情
多库多表同步	如果您需要将一个或多个数据库中的一张或多张业务数据表实时复制到另一个数据库时，则可以使用多库多表同步模板。Flink CDC可以自动同步整库的全量和增量数据，同时还能实时将每张源表的表结构变更（加列等）实时同步到对应的目标表中。多库多表同步模板会将用户参数翻译成CDAS语句，有关CDAS语句的更多能力，详情请参见CREATE DATABASE AS(CDAS)语句。
分库分表合并	当同一份业务数据分散在不同的数据库或者业务表时，我们往往需要用到分库分表合并的能力，将表结构相似的分库分表的数据同步到一张目标表中。Flink CDC可以自动同步源中所有分库分表的全量和增量数据（包括新增的符合正则表达式的新表），同时还能实时将每张源表的表结构变更（加列等）实时同步到对应的目标表中。分库分表合并同步模板会将用户参数翻译成CTAS语句，有关CTAS语句的更多能力，详情请参见CREATE TABLE AS(CTAS)语句。

使用限制

目前仅支持从MySQL到Hologres的数据同步。

注意事项

在同步MySQL到Hologres的数据时，会同步表结构变更，详情请参见CREATE TABLE AS(CTAS)语句。同步时需要注意以下几点：

- 无需您手动在Hologres上创建表。

  Flink会自动在Hologres上创建表，但如果Hologres已经存在对应的表，Flink会根据路径信息确定映射到哪张表。

- Hologres不支持更改列的类型。

  如果当前字段的类型和Hologres下游表中对应的字段类型不同，则会写入失败，您需重修改列类型并重新同步。

关于Hologres结果表详情请参见交互式分析Hologres结果表。

操作步骤

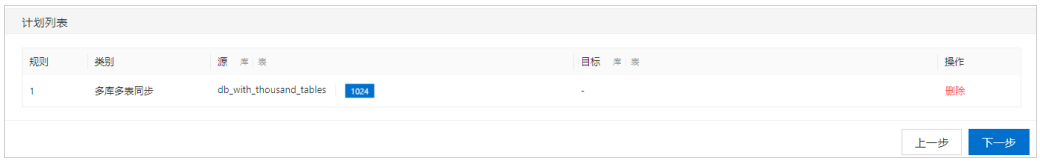
1. 登录实时计算管理控制台。
2. 在**Flink全托管**页签，单击对应工作空间**操作**列下的**控制台**。
3. 在左侧导航栏上，选择**应用 > 模板中心**。
4. 在**数据同步**页签，双击**MySQL到Hologres数据同步**模板名称，或者单击选中模板后，在右下角单击**继续**。
5. 配置数据同步信息。

i. 选择源和目标后，单击**下一步**。

类别	配置项	详情
源	MySQL Catalog	您可以直接选择已创建好的MySQL Catalog。如果您没有提前创建好源MySQL Catalog，则请单击**创建Catalog**进行创建。创建详情请参见管理MySQL Catalog。
	MySQL server-id	每个MySQL数据库客户端的唯一ID。取值范围为5400~6400。该参数也支持ID范围的格式，例如5400-5408。推荐在开启增量读取模式多并发读取数据时，设置该参数为ID范围，因为这样可以使得每个并发使用不同的ID。 **注意** 不要和其他访问数据库的作业或者服务使用同一个Server ID。
目标	Hologres Catalog	您可以直接选择已创建好的目标Hologres Catalog。如果您没有提前创建好目标Hologres Catalog，则请单击**创建Catalog**进行创建。详情请参见管理Hologres Catalog。

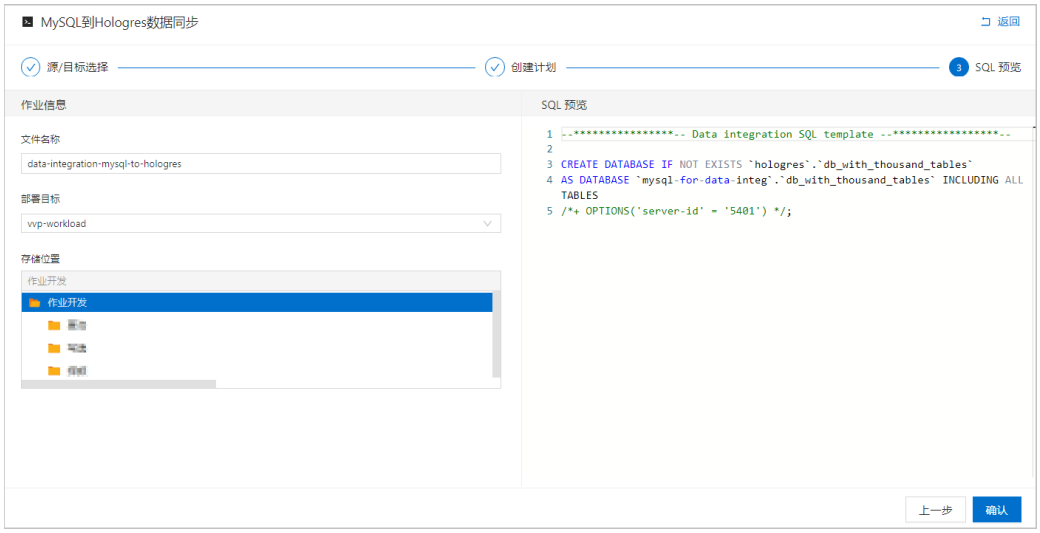
ii. 创建计划。

- 多库多表同步
  a. 在**多库多表同步**页签下，选中目标的源库和表名称。
  b. 单击**保存计划**，查看计划列表。



> **说明** 在进行多库多表数据同步时，该模板会在目标Catalog中创建与源Catalog同名的数据库和表。如果您需要修改数据库或表名称，则建议通过编写SQL的方式进行修改，详情请参见CREATE TABLE AS(CTAS)语句和CREATE DATABASE AS(CDAS)语句。

  c. 单击**下一步**。
  d. 查看系统根据您选择的目标库和表生成对应的SQL代码。



您可以根据需要修改**文件名称**、**部署目标**或**存储位置**等信息。

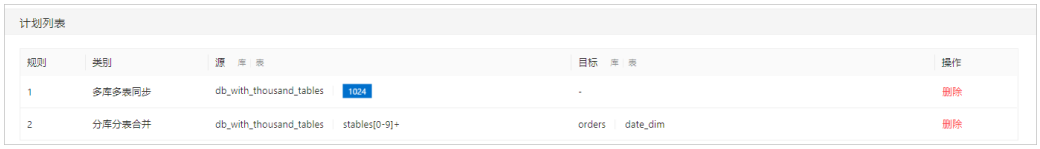
- 分库分表合并

  a. 在**分库分表合并**页签下，选中目标的源库和表名称，或者直接输入您创建的目标库和表名称。

> **说明** 推荐您使用正则表达式填写源库和源表信息。

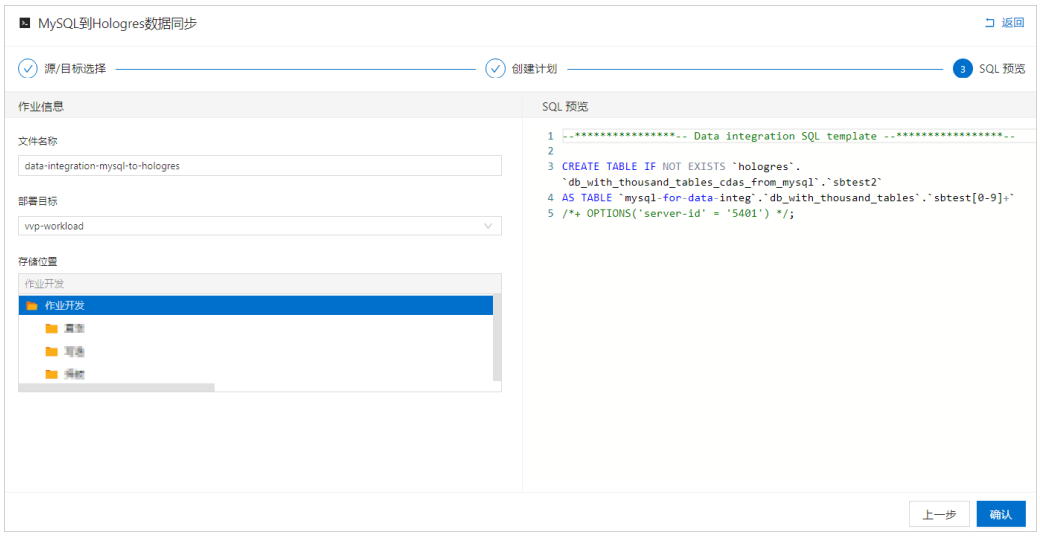
  b. 单击**保存计划**，查看计划列表。

在一个作业中，可以同时创建多个多库多表同步和分库分表合并的计划，示例如下图所示。



  c. 单击**下一步**。

  d. 查看系统根据您选择的目标库和表生成对应的SQL代码。



您可以根据需要修改**文件名称**、**部署目标**或**存储位置**等信息。

6. 单击**确认**。

单击**确认**后，就会进入作业开发页面。后面的操作和正常的作业开发完全相同，参数配置详情和后续操

作请参见作业开发和作业启动。

7.2.3. 作业开发

本文为您介绍Flink全托管SQL作业开发的限制说明和操作步骤。

使用限制

- SQL编辑器提交的SQL作业，仅支持开源Flink V1.11、Flink V1.12和Flink V1.13版本。
- SQL支持的上下游存储（Connector）列表，请参见支持的上下游存储。

操作步骤

为了方便您编写和管理Flink SQL作业，提高作业开发效率，Flink全托管产品为您提供Flink SQL的全套功能，包括元数据管理、UDF注册和SQL编辑器等。

1. 登录Flink全托管开发控制台，新建作业。
   i. 登录实时计算管理控制台。
   ii. 在**Flink全托管**页签，单击目标工作空间**操作**列下的**控制台**。
   iii. 在左侧导航栏，单击**作业开发**。
   iv. 单击**新建**。

v. 在**新建文件**对话框，填写作业配置信息。

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必须保持唯一。
文件类型	流作业和批作业均支持以下文件类型： ■ SQL ■ JAR ■ PYTHON 说明 VVP 2.4.1且VVR 3.0.1及以上版本支持批作业。
部署目标	选择作业需要部署的集群，支持以下两种集群模式： ■ **Per-Job集群**（默认）：适用于占用资源比较大或持续稳定运行的作业。因为作业之间资源隔离，每个作业都需要一个独立的JM，小任务JM的资源利用率较低。 ■ **Session集群**：适用于占用资源比较小或任务启停比较频繁的作业。因为多个作业可以复用相同的JM，可以提高JM资源利用率。 说明 如果您需要开启SQL Preview功能，必须选择**Session集群**，且已将其设置为SQL Previews集群，详情请参见作业调试和配置开发测试环境（Session集群）。
存储位置	指定该作业的代码文件所属的文件夹。 您还可以在现有文件夹右侧，单击 图标，新建子文件夹。

vi. 单击**确认**。

2. 在作业开发页面，编写DDL和DML代码。

   示例代码如下。

```
--创建源表datagen_source。
CREATE TEMPORARY TABLE datagen_source(
  name VARCHAR
) WITH (
  'connector' = 'datagen'
);
--创建结果表blackhole_sink。
CREATE TEMPORARY TABLE blackhole_sink(
  name  VARCHAR
) WITH (
  'connector' = 'blackhole'
);
--将源表数据插入到结果表。
INSERT INTO blackhole_sink
SELECT
  name
from datagen_source;
```

3. 在作业开发页面右侧，单击**高级配置**，填写配置信息。

   参数解释如下表所示。

类别	配置项	说明
基础配置	部署目标	您可以修改创建作业时已选择的部署目标。
	附加依赖文件	如果您需要添加更多依赖文件，请选择或者输入任意合法的文件地址。 **说明** Session集群不支持设置附加依赖文件，仅Per-Job集群支持设置附加依赖文件。
常规配置	引擎版本	当前作业使用的Flink的引擎版本。 **说明** 从VVR 3.0.3版本（对应Flink 1.12版本）开始，VVP支持同时运行多个不同引擎版本的SQL作业。如果您的作业已使用了Flink 1.12及更早版本的引擎，您需要按照以下情况进行处理： ○ Flink 1.12版本：停止后启动作业，系统将自动将引擎升级为vvr-3.0.3-flink-1.12版本。 ○ Flink 1.11或Flink 1.10版本：手动将作业引擎版本升级到vvr-3.0.3-flink-1.12或vvr-4.0.7-flink-1.13版本后重启作业，否则会在启动作业时超时报错。
	编辑标签	您可以为作业设置标签。后续可以通过搜索的方式，快速找到目标作业。
	创建实例的最大重试次数	创建实例失败后的重试次数。

行为配置类别	配置项	说明
	Stop with Drain	如果开启Stop with Drain功能，当作业被手动停止时，所有基于Event Time的窗口都会被触发。
Flink配置	Checkpoint间隔	定时执行Checkpoint的时间间隔。如果不填写，将会关闭Checkpoint。
	两次Checkpoint之间的最短时间间隔	两次Checkpoint之间的最短时间间隔，如果Checkpoint最大并行度是1，则该配置确保两个Checkpoint之间有一个最短时间间隔。
	开启Unaligned Checkpoint	开启Unaligned Checkpoint会大大降低反压情况下Checkpoint的总执行时间。但是也会导致增大单次Checkpoint的大小。
	Flink重启策略配置	当有Task失败时，如果没有开启Checkpoint，JobManager进程不会重启。如果开启了Checkpoint，则JobManager进程会重启。该参数取值如下： ○ Failure Rate：基于失败率重启。 ○ Fixed Delay：固定间隔重启。 ○ No Restarts（默认值）：不会重启。
	更多Flink配置	在此设置其他Flink配置。例如 `taskmanager.numberOfTaskSlots: 1` 。
日志配置	日志归档	默认已开启日志归档功能，并设置归档日志有效期为7天。开启日志归档后，您可以在作业探查页面查看历史作业实例的日志，详情请参见查看历史作业实例日志。 说明 ○ 在VVR 3.x版本，仅VVR 3.0.7及以上版本支持开启日志归档功能。 ○ 在VVR 4.x版本，仅VVR 4.0.11及以上版本支持开启日志归档功能。
	Root Log Level	日志级别从低到高顺序如下： i. TRACE：比DEBUG更细粒度的信息。 ii. DEBUG：系统运行状态的信息。 iii. INFO：重要或者您感兴趣的信息。 iv. WARN：系统可能出现潜在错误的信息。 v. ERROR：系统出现错误和异常的信息。
	Log Levels	填写日志名称和日志级别。
	Logging Profile	日志模板，可以选择系统模板，也可以选择用户配置。

4. 在作业开发页面右侧，单击**资源配置**，选择配置信息及资源信息。

支持以下三种配置模式：

- 基础模式：开源Flink的资源配置方式，您可以配置以下信息。

配置项	说明
并发度	作业全局并发数，默认值为1。
Job Manager CPUs	默认值为1。
Job Manager Memory	最小值为1 GiB。单位建议使用GiB或MiB，例如，1024 MiB或1.5 GiB。
Task Manager CPUs	默认值为1。
Task Manager Memory	最小值为1 GiB。单位建议使用GiB或MiB，例如，1024 MiB或1.5 GiB。

- 专家模式 （BETA）：Flink全托管引入的全新的资源配置模式，支持对作业所使用的资源进行细粒度的资源控制，以满足作业吞吐的要求。

  系统会自动根据您配置的资源需求，以Native K8s的模式运行作业，TM的规格和个数将会根据Slot的规格和作业并发度，由系统自动决定。

  > **说明** 仅SQL作业支持专家模式。

- 智能模式（BETA）：智能模式为专家模式的升级版。在智能模式下，作业将启用专家模式的资源配置，同时开启AutoPilot自动调优功能。

  在智能配置模式下，您无需配置相关资源，AutoPilot会自动为作业生成资源配置，并根据作业的运行情况帮您进行资源配置的调优，在确保作业处于健康的状态下，优化作业资源的使用。AutoPilot详情请参见配置自动调优。

5. 单击**保存**。
6. 单击**验证**。
7. 单击**上线**。

   完成作业开发和语法检查后，即可上线作业，将数据发布至生产环境。

7.2.4. 作业启动

完成作业开发和上线后，您需要在作业运维页面启动作业至运行阶段。此外，作业在暂停或者停止后需要恢复时，需要启动作业。

背景信息

作业启动的场景如下：

- 上线后启动：您可以选择全新启动（无状态），并可以指定读取数据的时间。
- 暂停后启动：暂停作业时，您可以按照需要选择在暂停前是否执行一次Savepoint。系统将按照以下情况进行处理：
  - 如果您选中了在暂停前执行一次Savepoint，则作业暂停后启动恢复时，系统会从最新的Savepoint状态恢复。

    > **说明** 如果Savepoint已经执行完成，但作业还没有开始停止，此时不会再进行Checkpoint。

- 如果您没有选中在暂停前执行一次Savepoint，则作业暂停后启动恢复时，系统会从最新的Checkpoint状态恢复。

- 停止后启动：停止作业时，您可以按照需要选择在停止前是否执行一次Savepoint，系统将按照以下情况进行处理：
  - 如果选中执行一次Savepoint（系统默认选中），那么作业会先进行Savepoint后进入停止流程。在作业停止后，系统会自动清除作业相关的Checkpoint信息。
  - 如果未选中执行一次Savepoint，那么作业会进入停止流程。在作业停止后，系统会自动清除作业相关的Checkpoint信息。

  作业停止后启动时，您可以选择全新启动（无状态），同时您也可以指定读取数据的时间；或者从所选Savepoint开始恢复作业。

使用限制

- 仅实时计算引擎VVR4.0.11及以上版本支持State兼容性检测。
- 仅实时计算引擎VVR4.0.8及以上版本的界面上支持设置启动位点。

  作业启动界面指定的读取数据时间优先级高于作业DDL代码里设置的startTime。即如果您在作业DDL和作业启动界面都指定了读取数据时间，则作业启动界面指定的读取数据时间生效。

> **说明**
>
> - 不是所有的Connector都支持startTime，详情请参见各Connector WITH参数是否包含startTime，例如日志服务SLS WITH参数。
> - 只有全新启动作业并指定的startTime，startTime才生效。如果基于Checkpoint或Savepoint启动作业，即使指定了startTime，startTime也不会生效。

注意事项

在启动作业时，如果您选择从原来的Checkpoint或Savepoint启动作业，则系统会进行State兼容性检测。兼容性检测详情请参见Flink State兼容性参考。

操作步骤

1. 登录实时计算管理控制台。
2. 在**Flink全托管**页签，单击对应工作空间**操作**列下的**控制台**。
3. 在左侧导航栏上，选择**应用 > 作业运维**。
4. 单击目标作业名称**操作**列中的**启动**。
5. 在**作业启动配置**对话框，选择启动方式和参数设置。

   不同场景的启动界面如下所示：

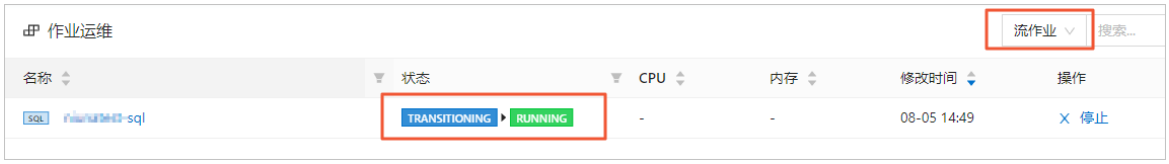
   - 上线后启动

- 暂停后启动

- 停止后启动

6. 单击**确认启动**。
7. 查看作业状态变化。

单击启动后，您可以看到作业从当前状态到期望状态的变化过程及最终结果。



作业的运行状态及含义如下表所示。

- **期望状态**

状态	说明
RUNNING	期望作业正常运行。
SUSPENDED	期望作业暂停运行。
CANCELLED	期望作业停止运行。

- **当前状态**

状态	说明
RUNNING	作业运行中。 运行时会有以下子状态： ■ ClusterUnreachable：VVP无法访问JM（JobManager），获取不到Job的状态。 ■ UNSTABLE（Low）：作业有可优化调优。 ■ UNSTABLE（Mid）：作业存在性能问题。 ■ UNSTABLE（High）：作业存在失败风险或数据正确性问题。
CANCELLED	作业停止运行。
SUSPENDED	作业暂停运行。
FINISHED	作业运行完成。
TRANSITIONING	作业处于向期望状态的过渡阶段。
FAILED	作业向期望状态转换失败，需要手动处理。

7.2.5. 作业暂停与停止

您可以在Flink全托管开发控制台上对状态为RUNNING的作业进行暂停或停止操作，本文为您介绍如何暂停与停止作业。

背景信息

暂停和停止操作在功能说明、使用场景和启动方案方面各有不同。您需要根据实际的业务需要，选择对应的操作。二者的差异情况如下表所示。

操作	功能说明	使用场景	启动方案
暂停	暂停作业时，您可以按照需要选择在暂停前是否执行一次Savepoint。	更改了SQL逻辑或者修改了资源配置后，需要经过暂停和启动的步骤，才能变更生效。	作业暂停后再启动时，默认从最新的Savepoint或Checkpoint状态恢复。详情请参见作业启动。
停止	停止作业时，您可以选中是否在停止前执行一次Savepoint。默认选中在停止前执行一次Savepoint。系统将按照以下情况进行处理： • 如果选中执行一次Savepoint，那么作业会先进行Savepoint后进入停止流程。在作业停止后，系统会自动清除作业相关的Checkpoint信息。 • 如果未选中执行一次Savepoint，那么作业会进入停止流程。在作业停止后，系统会自动清除作业相关的Checkpoint信息。	更改作业版本、增删WITH参数或增加作业参数后，需要经过停止和启动的步骤，才能使变更生效。另外，如果无法复用State，希望作业全新启动时，您需要停止后再启动作业。	作业停止后再启动时，您可以选择全新启动（无状态），同时您也可以指定读取数据的时间；或者从所选Savepoint恢复作业。详情请参见作业启动。 说明 如果您修改了SQL等逻辑导致State不兼容，则只能从无状态启动作业。

作业暂停

1. 登录实时计算管理控制台。
2. 在**Flink全托管**页签，单击对应工作空间**操作**列下的**控制台**。
3. 在左侧导航栏上，单击**作业运维**。
4. 单击目标作业**操作**列下的**暂停**。
5. 单击**确定**。

说明 单击**确定**前，您可以选中**在暂停前做一次Savepoint**。

作业停止

1. 登录实时计算管理控制台。
2. 在**Flink全托管**页签，单击对应工作空间**操作**列下的**控制台**。
3. 在左侧导航栏上，单击**作业运维**。

4. 单击目标作业**操作**列下的**停止**。
5. 单击**确定**。

说明 停止时默认选中**在停止前创建Savepoint**，您也可以取消选中。

7.2.6. 作业调试

您可以使用作业调试功能模拟作业运行、检查输出结果，验证SELECT或INSERT业务逻辑的正确性，提升开发效率，降低数据质量风险。

背景信息

使用SQL Preview功能时，您可以使用线上数据，也可以指定调试数据。调试可以包含多个SELECT或INSERT的复杂作业。此外，查询语句支持UPSERT，即可以执行count（*)等包含更新操作的语句。

前提条件

已创建用于SQL Preview的Session集群，因为系统在作业调试前会检测是否有可用于SQL Preview的Session集群，如果无Session集群时，则会提示并引导您配置用于SQL Preview的Session集群。配置Session集群详情请参见配置开发测试环境（Session集群）。

使用限制

- 仅SQL作业支持单击执行进行作业调试，Python作业和Datastream作业均不支持单击执行进行作业调试。
- VVR 4.0.8及以下版本的Session集群不支持调试CDC源表，因为CDC源表不是Append-only模式的。

操作步骤

1. 进入作业开发页面，新建SQL作业。
   i. 登录实时计算管理控制台。
   ii. 在**Flink全托管**页签，单击目标工作空间**操作**列下的**开发控制台**。
   iii. 在左侧导航栏，单击**作业开发**。
   iv. 单击**新建**。
   v. 填写完作业信息后，单击**确认**。

      作业类型必须选择SQL。
2. 分别注册源表和结果表。

   分别编写源表和结果表SQL代码后，单击**执行**。注册成功的表会出现在Schemas页签中的Tables列表中，您可以单击表名，查看表的字段和数据类型信息。以注册datagen源表、blackhole结果表和blackhole_cnt结果表为例，示例代码如下：

- datagen源表

```
CREATE TABLE `vvp`.`default`.`datagen` (
  `id`   VARCHAR(2147483647),
  `name` VARCHAR(2147483647)
) WITH (
  'connector' = 'datagen',
  'rows-per-second' = '5'
);
```

- blackhole结果表

```
CREATE TABLE `vvp`.`default`.`blackhole` (
  `id` VARCHAR(2147483647),
  `name` VARCHAR(2147483647)
) WITH (
  'connector' = 'blackhole'
);
```

- blackhole_cnt结果表

```
CREATE TABLE `vvp`.`default`.`blackhole_cnt` (
  `cnt` BIGINT
) WITH (
  'connector' = 'blackhole'
);
```

3. 编写调试DML语句。

   以从datagen中读取数据后，一个直接写入blackhole中，另一个取总数后写入blackhole_cnt中为例，代码如下。

```
BEGIN STATEMENT SET;
insert into blackhole
  select *
    from datagen;
insert into `blackhole_cnt`
  select count(*)
    from `datagen`;
END;
```

4. 单击**验证**。

   在SQL编辑页面下方，您可以看到SQL语法校验结果，只有通过校验的SQL才能被执行。

5. 单击**执行**。

6. 配置调试数据。

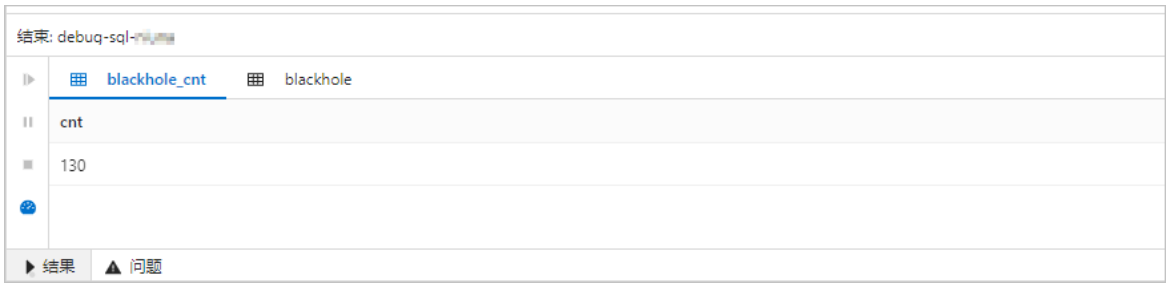
   如果您使用线上数据，直接单击**执行**即可。如果您需要使用调试数据，需要先上传调试数据。

参数解释如下表所示。

配置项	说明
下载调试数据模板	为了便于编辑，您可以直接下载调试数据模版，模版已适配源表的数据结构。
上传调试数据	如果您需要使用本地调试数据进行调试，您可以先下载好调试数据模板，本地编辑好数据后上传，并选中**使用调试数据**。 调试数据文件存在以下限制： ○ 上传文件仅支持CSV格式。 ○ CSV格式的文件必须含有表头，例如 id(INT)。 ○ 调试数据CSV文件最大支持1 MB或1千条记录。
预览数据	上传好调试数据后，单击源表名称左侧的 + 图标，可以预览数据和下载调试数据。
执行代码预览	调试功能会自动改变源表和结果表的DDL代码，但不会改变作业中的实际代码。您可以在下方预览代码详情。

7. 确定好调试数据后，单击**执行**。

   单击**执行**后，会在SQL编辑器下方显示调试结果。



> **说明** Flink全托管默认读取最多1000条数据后会自动暂停。

7.2.7. 管理自定义函数（UDF）

本文为您介绍如何管理Flink全托管自定义函数（UDF），包括注册、更新和删除。

注意事项

为了避免JAR包依赖冲突，在开发自定义函数时您需要注意以下几点：

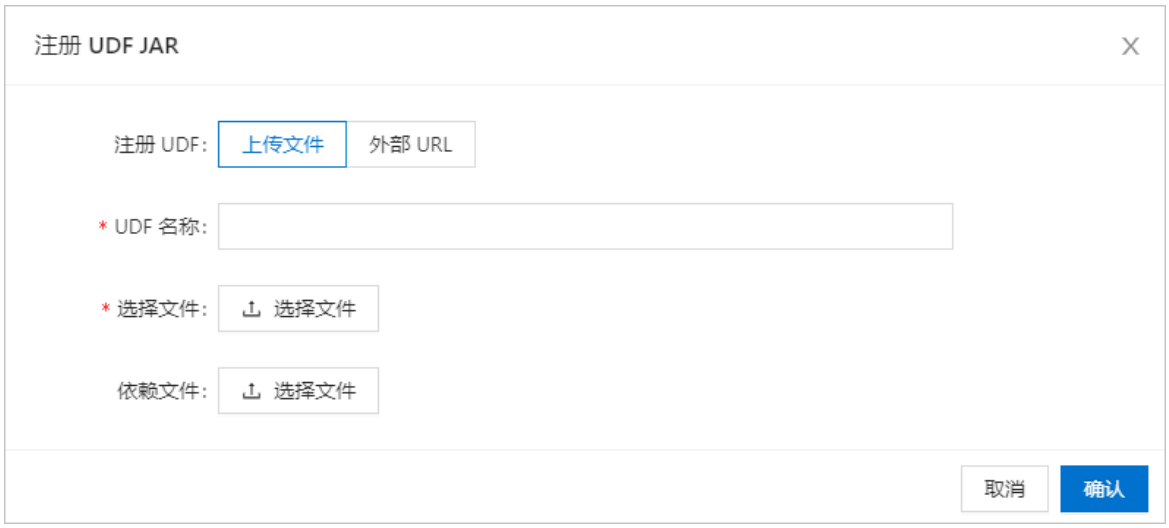
- 作业开发页面选择的Flink版本，请和Pom依赖中的Flink版本保持一致。
- Flink相关依赖，scope请使用provided，即在依赖中添加 `<scope>provided</scope>` 。
- 其他第三方依赖请采用Shade方式打包，Shade打包详情请参见Apache Maven Shade Plugin。

Flink 依赖冲突问题，详情请参见如何解决Flink依赖冲突问题？

注册UDF

如果您的SQL需要使用UDF，您需要先注册您的UDF，才能在SQL中使用。

1. 登录实时计算管理控制台。
2. 在**Flink全托管**页签，单击目标工作空间**操作**列下的**控制台**。
3. 在左侧导航栏上，单击**作业开发**。
4. 单击**UDFs**。
5. 在页面左上角，单击 + 图标。
6. 上传UDF JAR文件。



您可以通过以下任何一种方式上传UDF JAR文件：

- **上传文件**：单击**选择文件**项右侧的**选择文件**后，选择您的目标UDF Artifact文件。如果有依赖文件，单击**依赖文件**项右侧的**选择文件**，选择您的目标UDF Artifact所依赖的文件。

> **说明**
>
> - 您的UDF JAR文件会被上传到您选择的OSS Bucket中的*sql-artifacts*目录下。此外，Flink开发控制台会解析您UDF JAR文件中是否使用了Flink UDF、UDAF和UDTF接口的类，并自动提取类名，填充到Function Name字段中。
> - 对于Java类型的UDF，其依赖可以打包到UDF JAR包中，也可以通过依赖文件项进行上传；对于Python类型的UDF，其依赖推荐通过的单独上传依赖文件方式上传。

- **外部URL**：输入外部URL地址。

> **说明**
>
> - 外部URL需要使用OSS下的路径。如果您需要使用公网服务，则需要先打通实时计算Flink全托管和公网的网络链接。详情请参见Flink全托管集群如何访问公网？。
> - 在UDF Artifact文件或者其依赖文件比较大时，推荐通过外部URL的方式进行上传。需要注意的是，如果外部URL是OSS Bucket地址，其依赖文件必须位于*sql-artifacts/namespaces/{namespace}*目录下。

7. 单击**确认**。

   在SQL编辑器页面左侧**UDFs**列表，您可以看到所有注册成功的UDF。

更新UDF

如果您的UDF JAR文件中，新增了UDF或已注册了UDF的代码变更，您可以按照以下步骤更新UDF JAR文件。

1. 登录实时计算管理控制台。
2. 在**Flink全托管**页签，单击目标工作空间**操作**列下的**控制台**。
3. 在左侧导航栏上，单击**作业开发**。
4. 单击**UDFs**。
5. 在**UDF JARs**列表中，鼠标悬停在目标UDF名称上，单击 ⬆。
6. 上传UDF JAR文件。

   您可以通过以下任何一种方式上传UDF JAR文件：

   - **上传文件**：单击**选择文件**项右侧的**选择文件**后，选择您的目标UDF Artifact文件。如果有依赖文件，单击**依赖文件**项右侧的**选择文件**，选择您的目标UDF Artifact所依赖的文件。

- **外部URL**：输入外部URL地址。

> **注意**
> - 您上传的新UDF JAR文件中，必须包含当前已经注册UDF的所有类，不能出现某个已注册UDF的类消失的情况。
> - 只有重启作业或提交新作业时，才会使用新UDF JAR文件中的代码。如果引用该UDF JAR的作业正在运行，则依然会使用旧UDF JAR文件。

7. 单击**修改**。

删除UDF

如果您的UDF JAR文件不再使用，您可以按照以下步骤删除UDF。

1. 登录实时计算管理控制台。
2. 在**Flink全托管**页签，单击目标工作空间**操作**列下的**控制台**。
3. 在左侧导航栏上，单击**作业开发**。
4. 单击**UDFs**。
5. 在**UDFs**列表中，鼠标悬停在目标UDF名称上，单击 图标。

> **说明** 在删除UDF JAR文件前，请确定其注册的UDF没有被作业或SQL文件引用。

6. 选中**Drop以上所有Functions并且删除关联的文件**。

   如果您要删除该UDF JAR文件，则需要删除该UDF JAR中所有注册的UDF，避免有脏数据残留。
7. 单击**确认**。

7.2.8. 管理Hive Metastore

本文为您介绍如何在Flink全托管模式下配置Hive Metastore、查看Hive元数据、使用Hive Metastore和删除Hive Metastore。

背景信息

您可以将Hive Metastore配置文件和Hadoop依赖存放至对象存储OSS控制台指定目录后，再在Flink全托管开发控制台上配置Hive Metastore功能。配置成功后，您就可以在Flink全托管开发控制台上直接使用DML创建业务逻辑，获取Hive中表的元数据信息，无需再使用DDL语句声明相应的表信息。Hive Metastore可以作为Stream作业和Batch作业的源表或结果表。

前提条件

- 已在Hive Metastore侧开启了Hive Metastore服务。

  相关命令如下：
  - `hive --service metastore`：开启Hive Metastore服务。
  - `netstat -ln | grep 9083`：查询Hive Metastore服务是否已开启。

    其中9083是Hive Metastore的默认端口号。如果您在hive-site.xml配置了其他的端口号，则需要将9083改为对应的端口号。

- 已在Hive Metastore侧配置了白名单，使得Flink全托管可以访问Hive Metastore。

  Flink全托管网段的获取方法请参见设置白名单，在Hive Metastore侧配置白名单的方法请参见添加安全组规则。
- 已授予VVP和Flink用户对 `hive.metastore.warehouse.dir` 配置的目录的读权限，其中 `hive.metastore.warehouse.dir` 的目录请在hive-site.xml中查看。
- 如果Hive Metastore Catalog中存在外部表，需要授予VVP和Flink用户对每张外部表的数据目录的读权限。

  您可以通过**show create table ${tableName}**命令来查看外部表的数据目录，`LOCATION` 字段中的内容即为外部表的数据目录。

注意事项

使用Hive Metastore功能需要注意以下几点：

- 支持自建Hive Metastore。
- VVP 2.3.0及以上版本支持Hive Metastore，支持的版本情况具体如下：
  - VVP 2.3.0版本仅支持Hive Metastore 2.3.6版本。
  - VVP 2.3.0以上版本支持Hive Metastore 2.2.0~2.3.7版本。
- Hive Metastore不支持Kerberos鉴权。

  > **说明** 不开启kerberos鉴权时，在VVP端访问Hive的默认用户名为vvp，在Flink集群中访问Hive的默认用户名为flink。因此您要保证vvp和flink这两个用户都有访问Hive元数据和文件系统（例如HDFS）上的Hive表数据的权限。
- 一个Flink全托管实例仅支持一个Hive Metastore，多项目无法设置多Hive Metastore。
- Hive Metastore只读，即您不能通过Flink全托管开发控制台在Hive中创建物理表。

配置Hive Metastore服务

1. 连通Hadoop集群和Flink全托管的VPC。

   您可以使用云解析PrivateZone产品连通Hadoop集群和Flink全托管的VPC，详情请参见解析器（Resolver）。网络连通后，Flink全托管就可以使用Hadoop集群的配置文件访问Hadoop集群。
2. 在OSS控制台新建目录，并将Hive配置文件和Hadoop依赖上传至目标路径。
   i. 登录OSS管理控制台。
   ii. 在左侧导航栏，单击**Bucket列表**。
   iii. 单击目标Bucket名称。
   iv. 在左侧导航栏，单击**文件管理**。

v. 在*oss://${bucket}/artifacts/namespaces/${ns}/*路径下，新建${hms}目录。

在OSS上创建目录的操作详情，请参见创建目录。目标路径中的变量含义如下表所示：

目录	说明
${bucket}	您Flink全托管实例使用的Bucket名称。
${ns}	您要使用Hive Metastore功能的Flink全托管项目名称。
${hms}	您在Flink全托管上显示的Hive Metastore名称。

> **说明** 开通Flink全托管服务后，系统会自动在您指定的Bucket下创建*/artifacts/namespaces/${ns}/*目录来存储JAR包等数据。如果您在OSS控制台没有看到如上目录，则需要在Flink全托管开发控制台的**资源管理**页面，手动上传一个文件来触发目录创建。

vi. 在*oss://${bucket}/artifacts/namespaces/${ns}/${hms}*路径下，新建hive-conf-dir和hadoop-conf-dir目录。

其中hive-conf-dir和hadoop-conf-dir目录中存放的文件详情如下：

- *oss://${bucket}/artifacts/namespaces/${ns}/${hms}/hive-conf-dir/*用于存放Hive配置文件hive-site.xml。
- *oss://${bucket}/artifacts/namespaces/${ns}/${hms}/hadoop-conf-dir/*用于存放Hadoop配置文件，包括core-site.xml、hdfs-site.xml、yarn-site.xml和mapred-site.xml。

在OSS上新建目录的操作详情，请参见创建目录。创建成功后，您可以在左侧导航栏的**资源上传**页面中，查看新建的目录及文件，并复制OSS链接信息。

vii. 将您的Hive配置文件（hive-site.xml）上传到hive-conf-dir目录下。上传文件的操作请参见上传文件。

在上传hive-site.xml文件前，请您先检查下配置文件中hive.metastore.uris参数配置是否符合下列要求。

```
<property>
    <name>hive.metastore.uris</name>
    <value>thrift://xx.yy.zz.mm:9083</value>
    <description>Thrift URI for the remote metastore. Used by metastore client to c
onnect to remote metastore.</description>
 </property>
```

其中 `xx.yy.zz.mm` 为Hive的内网IP或者公网IP。

> **说明** 如果您将hive.metastore.uris填写为hostname，您需要配置域名解析服务。否则VVP远程访问Hive时，hive.metastore.uris参数值会被解析失败并报错 `UnknownHostException` 。配置域名解析服务详情请参见添加PrivateZone解析记录

viii. 将您的以下配置文件上传到hadoop-conf-dir目录。上传文件的操作请参见上传文件。
- hive-site.xml
- core-site.xml
- hdfs-site.xml
- mapred-site.xml
- 其他文件，例如Hive作业使用的压缩包。

3. 在Flink全托管开发控制台上配置Hive Metastore服务。
   i. 登录实时计算控制台。
   ii. 在**Flink全托管**页签，单击目标工作空间**操作**列下的**开发控制台**。
   iii. 在左侧导航栏，单击**作业开发**。
   iv. 在页面左上角，单击**新建**，文件类型选择SQL。
   v. 在文本编辑区域，输入创建Hive Metastore的命令。

```
CREATE CATALOG ${HMS Name} WITH (
    'type' = 'hive',
    'default-database' = 'default',
    'hive-version' = '<hive-version>',
    'hive-conf-dir' = '<hive-conf-dir>',
    'hadoop-conf-dir' = '<hadoop-conf-dir>'
);
```

参数	说明
${HMS Name}	Hive Metastore名称。
type	Connector类型，固定值为hive。
default-database	默认数据库名称。
hive-version	Hive Metastore版本号。 **说明** Flink全托管兼容的Hive Metastore版本为2.2.0~2.3.7，hive-version的配置情况如下： - 如果Hive Metastore版本为2.0.0~2.2.0，则hive-version需要指定为2.2.0。 - 如果Hive Metastore版本为2.3.0~2.3.7，则hive-version需要指定为2.3.6。
hive-conf-dir	存放Hive配置文件的目录。目录为*oss://${bucket}/artifacts/namespaces/${ns}/${hms}/hive-conf-dir/*。
hadoop-conf-dir	存放Hadoop依赖的目录。目录为*oss://${bucket}/artifacts/namespaces/${ns}/${hms}/hadoop-conf-dir/*。

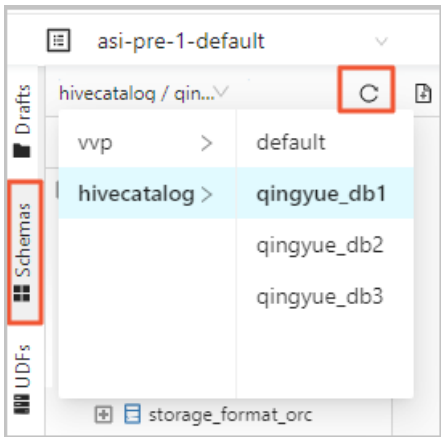
vi. 单击**运行**。

Hive Metastore服务配置完成后，您就可以在作业中引用Hive Metastore表信息，作为结果表和维表，无需声明表的DDL。Hive Metastore表名称格式为${hive-catlog-name}.${hive-db-name}.${hive-table-name}。

此外，如果您想停用Hive Metastore服务，详情请参见删除Hive Metastore服务。

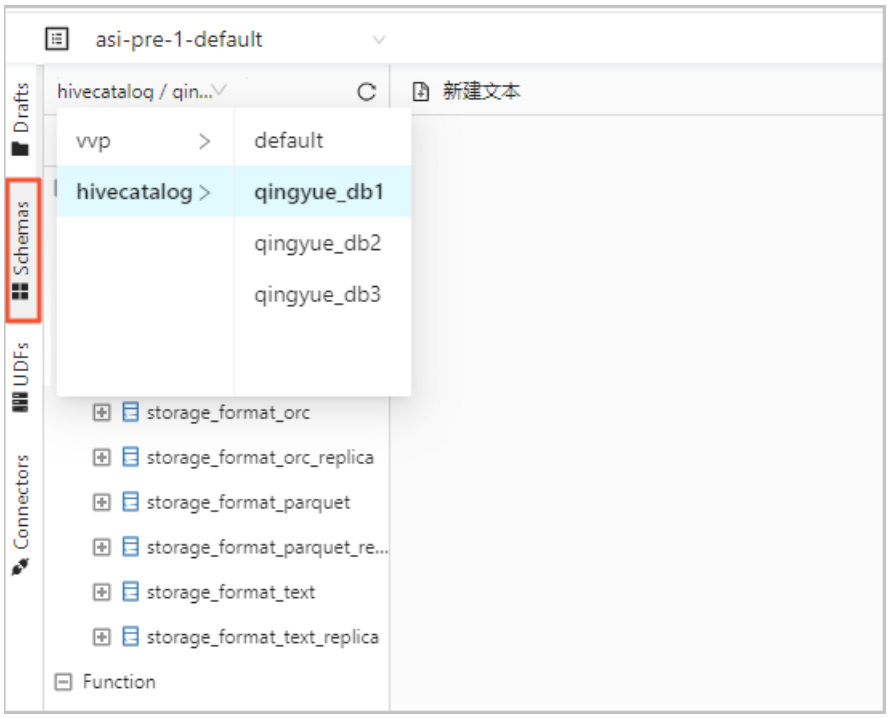
vii. 在左侧，单击**Schemas**页签。

viii. 单击C图标，刷新查看新建的Hive Catalog。



查看Hive元数据

1. 登录实时计算管理控制台。
2. 在**Flink全托管**页签，单击目标工作空间**操作**列下的**开发控制台**。
3. 在左侧导航栏，单击**作业开发**。
4. 单击**Schemas**页签。
5. 在顶部菜单栏下拉框中，切换到目标Hive Metastore。



6. 在Tables列查看不同数据库下的表和字段信息。

使用Hive Metastore

- 从Hive表中读取数据。

```
INSERT INTO ${other_sink_table}
SELECT ...
FROM `${catalog_name}`.`${db_name}`.`${table_name}`
```

- 写入结果数据至Hive表。

```
INSERT INTO `${catalog_name}`.`${db_name}`.`${table_name}`
SELECT ...
FROM ${other_source_table}
```

删除Hive Metastore服务

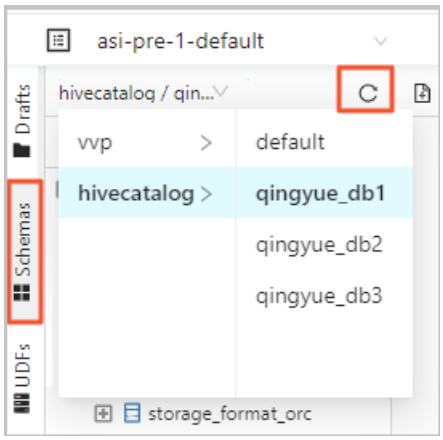
1. 登录实时计算控制台。
2. 在**Flink全托管**页签，单击目标工作空间**操作**列下的**开发控制台**。
3. 在左侧导航栏，单击**作业开发**。
4. 在页面左上角，单击**新建**，文件类型选择SQL。
5. 在文本编辑区域，输入以下命令。

   ```
   Drop CATALOG ${HMS Name}
   ```

   其中，HMS Name为您要删除的在Flink全托管开发控制台上显示的Hive Metastore名称。

   > **注意** 删除Hive Metastore服务不会影响已运行的作业，但对未上线或者作业需要暂停恢复的作业，均产生影响，请您谨慎操作。

6. 单击**运行**。
7. 在左侧，单击**Schemas**页签。
8. 单击C图标，刷新查看新建的Hive Catalog是否已被删除。



7.2.9. 管理Hologres Catalog

配置Hologres Catalog后，您就可以在Flink全托管控制台直接读取Hologres元数据，不用再手动注册Hologres表，可以提高作业开发的效率且保证数据的正确性。本文为您介绍如何在Flink全托管模式下配置、查看及删除Hologres Catalog。

背景信息

Catalog提供了元数据信息，例如数据库、表、分区、视图以及数据库或其他外部系统中存储的函数和信息。详情请参见Catalogs。

本文从以下方面为您介绍如何管理Hologres Catalog：

- 配置Hologres Catalog
- 查看Hologres Catalog
- 使用Hologres Catalog
- 删除Hologres Catalog

使用限制

- 仅Flink 1.13及以上版本支持配置Hologres Catalog。
- 不支持修改Catalog DDL。

配置Hologres Catalog

1. 登录实时计算管理控制台。
2. 在**Flink全托管**页签，单击目标工作空间**操作**列下的**控制台**。
3. 在左侧导航栏，单击**作业开发**。
4. 在页面左上角，单击**新建**，文件类型选择SQL。
5. 在文本编辑区域，输入配置Hologres Catalog的命令。

```
CREATE CATALOG <catalogname> WITH (
  'type' = 'hologres',
  'endpoint' = '<endpoint>',
  'username' = '<AccessKey>',
  'password' = '<AccessSecret>',
  'dbname' = '<dbname>'
);
```

catalogname为Hologres Catalog名称。仅支持小写字母a-z和数字0-9，不含大写字母、中划线（-）、下划线（_）等特殊字符。WITH参数详情如下表所示。

参数	说明	是否必填
type	类型，固定值为hologres。	是
endpoint	Hologres的Endpoint地址。 详情请参见实例配置。	是
username	阿里云账号的AccessKey。	是
password	阿里云账号的AccessSecret。	是
dbname	您访问Hologres的默认数据库名称。	是

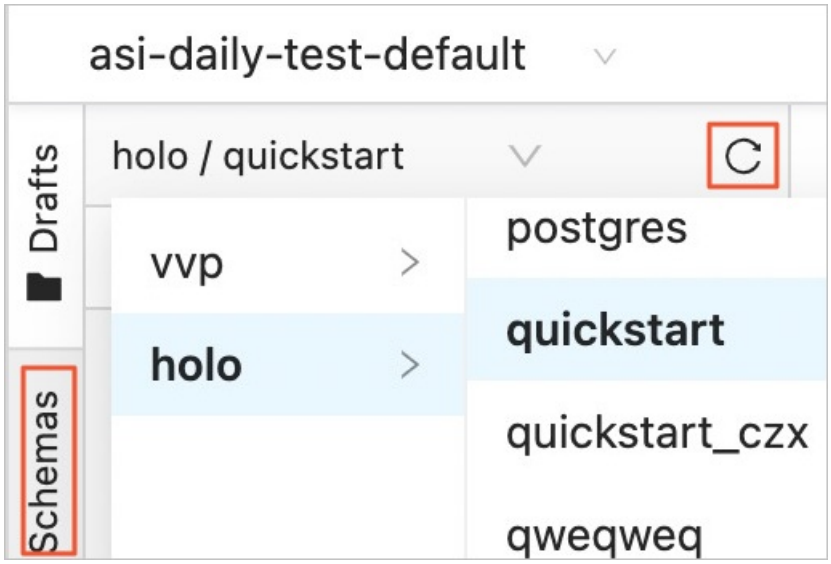
参数	说明	是否必填
ignore-non-persisted-options	在使用Hologres Catalog时创建带有不可持久化选项的表时，是否忽略非可持久化选项。参数取值如下： ○ true（默认值）：可以成功创建出表，而忽略所有非可持久化选项。 ○ false：会报创建表失败的错误。 **说明** Hologres Catalog表选项的可持久化意味着当再次从Catalog读取该表的相关信息时，可以重新获取您在DDL中定义的一致的信息。目前仅支持endpoint、username、password和dbname可持久化选项。	否

6. 单击**执行**。

   执行完会提示Query has been executed。如果您需要删除Hologres Catalog，详情请参见删除Hologres Catalog。

7. 在左侧，单击**Schemas**页签。

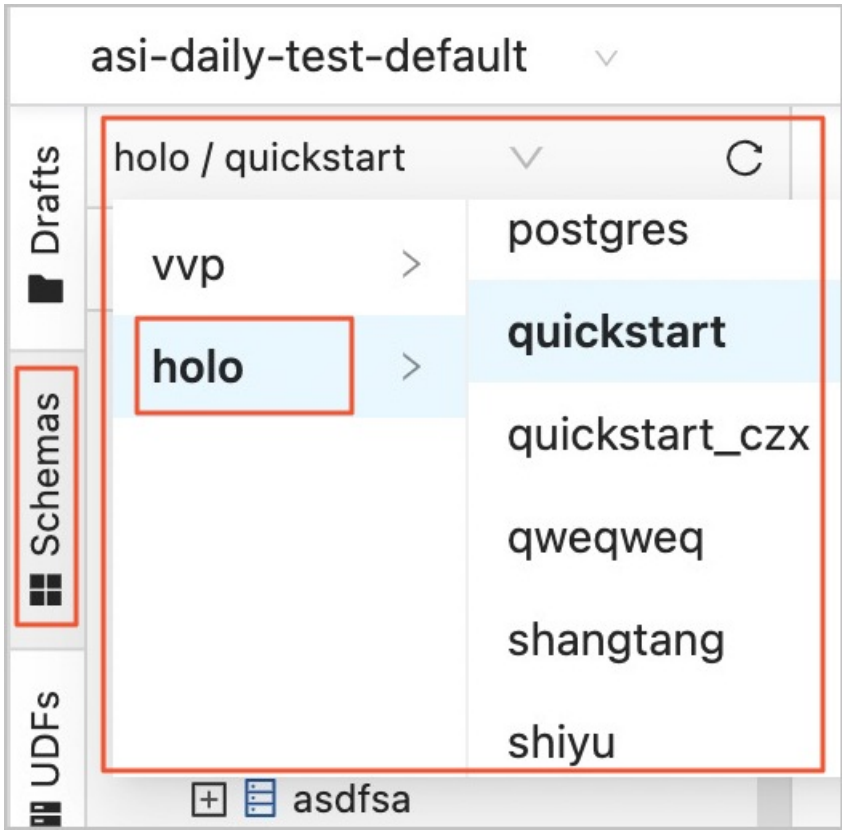
8. 单击C图标，刷新查看新建的Hologres Catalog。



查看Hologres Catalog

Hologres Catalog配置成功后，您可以通过以下步骤查看Hologres元数据。

1. 登录实时计算管理控制台。
2. 在**Flink全托管**页签，单击目标工作空间**操作**列下的**控制台**。
3. 在左侧导航栏，单击**作业开发**。
4. 单击**Schemas**页签。
5. 在顶部菜单栏下拉框中，切换到目标Hologres Catalog名称，本文以holo为例。



6. 查看不同Hologres Catalog下的数据库、表和函数信息。

> **说明** 如果Schema为public时，则表名称前面就省略了Schema的前缀，即直接显示tableName。

使用Hologres Catalog

说明

- 如果Schema为public时，在填写${schema_name.table_name}时，可以省略模式名称的前缀，直接填写tableName，即${table_name}。
- 通过Hologres Catalog读取出来的表支持消费update数据，读取出来的Hologres表的ignoredelete属性自动被设置为了false，mutatetype属性被自动设置为了insertorupdate。ignoredelete和mutatetype属性含义详情请参见宽表Merge和局部更新功能。

- 创建Hologres表。
  - 使用 `USE CATALOG HoloName` 命令直接引用Hologres数据服务。

```
USE CATALOG ${catalog_name};
CREATE TABLE `${db_name}`.`${schema_name.table_name}`(
  ...
 ) WITH (
   'connector' = 'hologres'
 );
```

    USE语法详情请参见USE语句。
  - DDL语句中直接引用Hologres数据服务信息。

```
CREATE TABLE `${catalog_name}`.`${db_name}`.`${schema_name.table_name}` (
  ...
) WITH (
  'connector' = 'hologres'
);
```

    说明

    - 在已注册的Hologres数据服务下创建表时，WITH参数中connector为必填参数，且取值为hologres。可省略endpoint等其他参数。
    - 目前不支持直接在Hologres表中添加或修改WITH参数，可以通过SQL hints的方式在INSERT语句中添加或修改WITH参数。WITH参数详情请参见Hologres源表WITH参数、Hologres结果表WITH参数和Hologres维表WITH参数

- 从Hologres表中读取数据。

```
INSERT INTO ${other_sink_table}
SELECT ...
FROM `${catalog_name}`.`${db_name}`.`${schema_name.table_name}`
```

- 写入结果数据至Hologres表。

```
INSERT INTO `${catalog_name}`.`${db_name}`.`${schema_name.table_name}`
SELECT ...
FROM ${other_source_table}
```

- 作为CTAS的目标端Catalog。

```
CREATE TABLE IF NOT `${catalog_name}`.`${db_name}`.`${schema_name.table_name}`
WITH (
  'connector' = 'hologres'
) AS TABLE ${other_source_table};
```

当从源端同步数据过程中，为保证数据能够写入Hologres，Hologres Catalog会在以下情况被迫改写目标端的Schema：

- 上游Schema使用DECIMAL类型的列作为主键。

  由于Hologres不支持DECIMAL类型作为主键，因此Hologre默认会改写该列类型为BIGINT。如果该改写不满足您的需求，则您也可以使用CTAS语法将引用的类转换为STRING类型，并重新建立主键。

- 上游Schema中包含类型TIME、TIMESTAMP或TIMESTAMP_LTZ且精度大于6。

  由于Hologres支持的时间类型精度为6，因此为了确保数据能够写入到Hologres中，Flink会隐式地丢弃高于Hologres所支持的最高精度的部分。

- 作为CDAS的目标端Catalog。

```
CREATE DATABASE IF NOT EXISTS `${catalog_name}`.`${db_name}`
WITH (
  'sink.parallelism' = '5' -- 设置每张结果表的并发数。
) AS DATABASE ${other_source_database};
```

在同步过程之中，您可以在WITH参数声明结果表的参数。当作业启动时，这些参数将被应用到需要同步的下游表中。具体可以调节的参数请参见交互式分析Hologres结果表。

删除Hologres Catalog

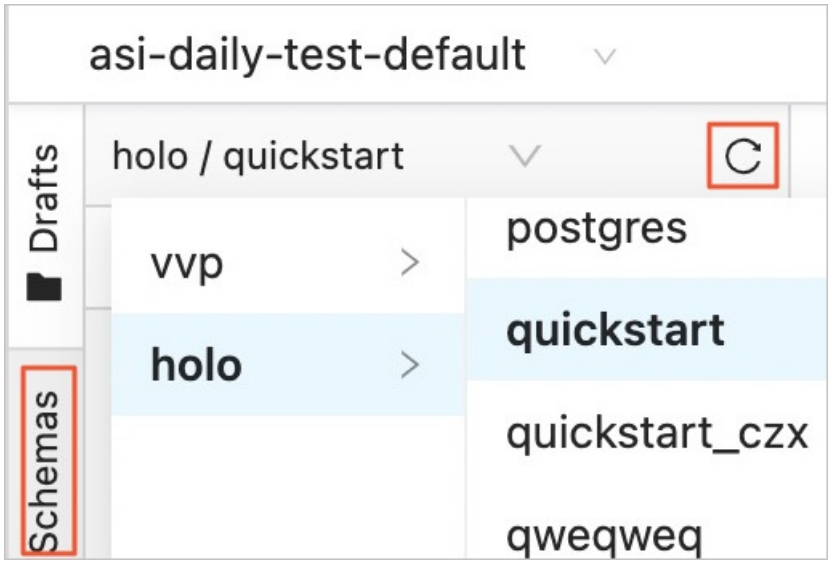
1. 登录实时计算管理控制台。
2. 在**Flink全托管**页签，单击目标工作空间**操作**列下的**控制台**。
3. 在左侧导航栏，单击**作业开发**。
4. 在页面左上角，单击**新建**，文件类型选择SQL。
5. 在文本编辑区域，输入以下命令。

```
DROP CATALOG ${catalog_name}
```

   其中，catalog_name为您要删除的在Flink全托管开发控制台上显示的Hologres Catalog名称。

   > **说明** 删除Hologres Catalog不会影响已运行的作业，但对未上线或者作业需要暂停恢复的作业，均产生影响，请您谨慎操作。

6. 单击**执行**。
7. 在左侧，单击**Schemas**页签。
8. 单击C图标，刷新查看新建的Hologres Catalog是否已被删除。



7.2.10. 管理MySQL Catalog

配置MySQL Catalog后，您就可以在Flink全托管控制台直接访问MySQL实例中的表。本文为您介绍如何在Flink全托管模式下配置、查看及删除MySQL Catalog。

背景信息

MySQL Catalog具有以下功能特点：

- 直接访问MySQL实例中的表，无需通过DDL语句手动注册MySQL表，提升开发效率和正确性。
- MySQL Catalog提供的表可以直接作为Flink SQL作业中的MySQL CDC源表、MySQL结果表和MySQL维表。
- 支持RDS MySQL、PolarDB MySQL或自建MySQL。
- 支持直接访问分库分表逻辑表。
- 支持配合CDAS和CTAS语法完成基于MySQL数据源的整库同步、分库分表合并同步、表结构变更同步。

本文将从以下方面为您介绍如何管理MySQL Catalog：

- 配置MySQL Catalog
- 查看MySQL Catalog
- 使用MySQL Catalog
- 删除MySQL Catalog

使用限制

- 仅Flink计算引擎vvr-4.0.11-flink-1.13及以上版本支持配置MySQL Catalog。
- 不支持修改Catalog DDL。
- 仅支持查询数据库和表，不支持创建数据库和表。
- 作为源表仅支持流读、不支持批读，支持作为维表和结果表。
- MySQL仅支持5.7和8.0.x版本。

> **说明** 如果MySQL Catalog提供的表被作为MySQL CDC源表时，则需要在RDS MySQL、PolarDB MySQL或者自建MySQL上开启Binlog等配置，详情请参见配置MySQL。

配置MySQL Catalog

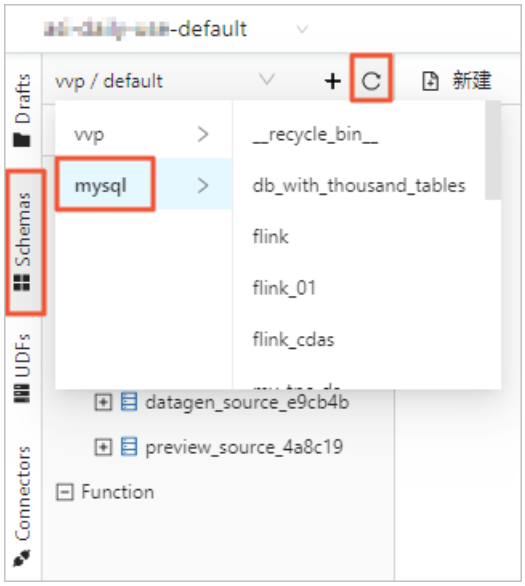
支持UI与SQL命令两种方式配置MySQL Catalog，推荐使用UI方式配置MySQL Catalog。

1. 登录实时计算管理控制台。
2. 在**Flink全托管**页签，单击目标工作空间**操作**列下的**控制台**。
3. 在左侧导航栏，单击**作业开发**。
4. 在左侧，单击**Schemas**页签。
5. 单击 + 图标，在菜单栏下拉框中选择**创建Catalog**
6. 在**创建Catalog**页面，选择**MySQL**。
7. 填写参数配置信息。

参数	说明	是否必填
catalogname	MySQL Catalog名称。	是
type	类型，固定值为mysql。	是
hostname	MySQL数据库的IP地址或者Hostname。	是
port	MySQL数据库服务的端口号，默认值为3306。	否
default-database	默认的MySQL数据库名称。	是
username	MySQL数据库服务的用户名。	是

参数	说明	是否必填
password	MySQL数据库服务的密码。	是

8. 单击**确认**。
9. 单击C图标，刷新查看新建的MySQL Catalog。



1. 登录实时计算管理控制台。
2. 在**Flink全托管**页签，单击目标工作空间**操作**列下的**控制台**。
3. 在左侧导航栏，单击**作业开发**。
4. 在页面左上角，单击**新建**，文件类型选择SQL。
5. 在文本编辑区域，输入配置MySQL Catalog的命令。

```
CREATE CATALOG <catalogname> WITH(
  'type' = 'mysql',
  'hostname' = '<hostname>',
  'port' = '<port>',
  'username' = '<username>',
  'password' = '<password>',
  'default-database' = '<dbname>'
);
```

参数	说明	是否必填
catalogname	MySQL Catalog名称。	是
type	类型，固定值为mysql。	是
hostname	MySQL数据库的IP地址或者Hostname。	是
port	MySQL数据库服务的端口号，默认值为3306。	否
default-database	默认的MySQL数据库名称。	是

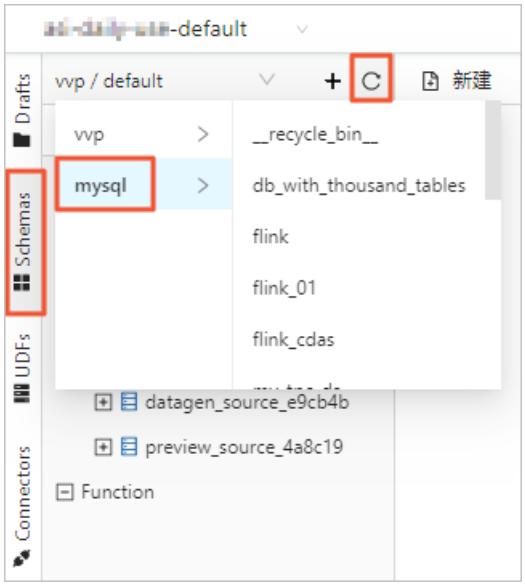
参数	说明	是否必填
username	MySQL数据库服务的用户名。	是
password	MySQL数据库服务的密码。	是

6. 单击**执行**。

   执行完会提示Query has been executed。

7. 在左侧，单击**Schemas**页签。

8. 单击C图标，刷新查看新建的MySQL Catalog。



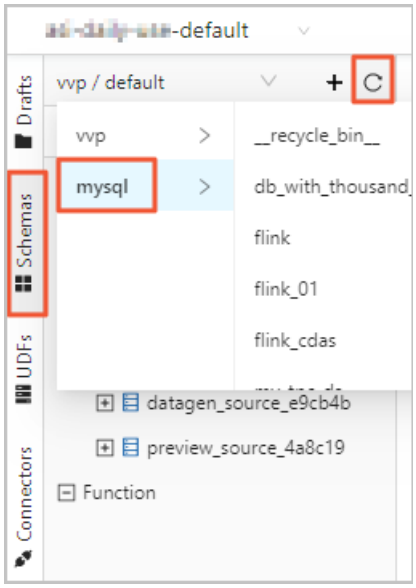
UI方式

SQL命令方式

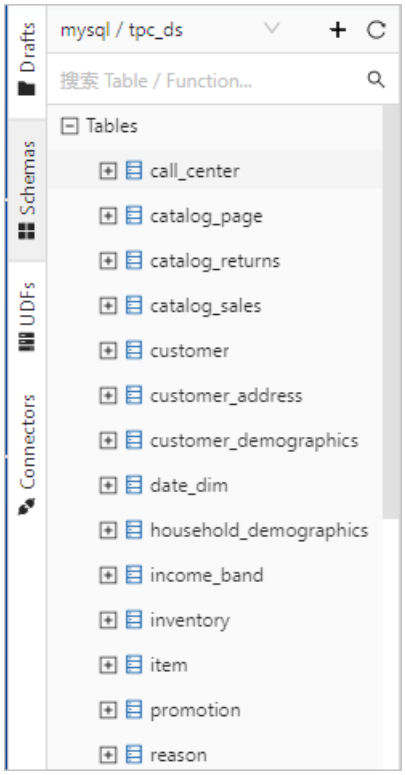
查看MySQL Catalog

MySQL Catalog配置成功后，您可以通过以下步骤查看MySQL元数据。

1. 登录实时计算管理控制台。
2. 在**Flink全托管**页签，单击目标工作空间**操作**列下的**控制台**。
3. 在左侧导航栏，单击**作业开发**。
4. 单击**Schemas**页签。
5. 在顶部菜单栏下拉框中，切换到目标MySQL Catalog名称，本文以mysql为例。



6. 查看目标MySQL Catalog下的数据库、表和函数信息。



使用MySQL Catalog

- 从MySQL源表中读取数据。

```
INSERT INTO ${other_sink_table}
SELECT ...
FROM `${mysql_catalog}`.`${db_name}`.`${table_name}` /*+ OPTIONS('server-id'='6000-6018')
*/;
```

> **说明** 如果将MySQL Catalog作为MySQL CDC源表，建议使用SQL Hints来为作业指定不同的server-id。如果源表需要多并发读取，server-id还需要配置成范围格式，范围中的server-id个数需要大于等于并发度。

- 读取MySQL分库分表逻辑表。

  MySQL Catalog支持使用正则表达式，将库名和表名作为逻辑表名，来读取分库分表的数据。例如，有一个分库分表的MySQL数据库，包括user01、user02和user99等多个表，分散在db01~db10等数据库中，且所有表的Schema都相互兼容，则可以通过如下正则表达式的库名表名来访问到所有user的分库分表。

```
SELECT ... FROM `db.*`.`user.*` /*+ OPTIONS('server-id'='6000-6018') */;
```

  分库分表的逻辑表会返回额外的_db_name (STRING) 和_table_name (STRING)两个系统字段，且这两个字段与原分表的主键会作为逻辑表的新联合主键以保证主键的唯一性。如果user01~user99的主键均为id，则user逻辑表的联合主键为(_db_name, _table_name, id)。MySQL Catalog支持结合正则表达式读取分库分表数据，具体示例请参见CREATE TABLE AS(CTAS)语句。

- 使用CTAS和CDAS实时同步MySQL数据变更和结构变更。

```
USE CATALOG `${target_catalog}`;
-- 单表同步，实时同步表级别的表结构变更和数据变更。
CREATE TABLE IF NOT EXISTS `${target_table_name}`
WITH (...)
AS TABLE `${mysql_catalog}`.`${db_name}`.`${table_name}`
/*+ OPTIONS('server-id'='6000-6018') */;
-- 整库同步，实时同步整库级别的表结构变更和数据变更。
CREATE DATABASE `${target_db_name}` WITH (...)
AS DATABASE `${mysql_catalog}`.`${db_name}` INCLUDING ALL TABLES
/*+ OPTIONS('server-id'='6000-6018') */;
```

  更多示例请参见CREATE TABLE AS(CTAS)语句或CREATE DATABASE AS(CDAS)语句。

- 从MySQL维表中读取数据。

```
INSERT INTO ${other_sink_table}
SELECT ...
FROM ${other_source_table} AS e
JOIN `${mysql_catalog}`.`${db_name}`.`${table_name}` FOR SYSTEM_TIME AS OF e.proctime AS
w
ON e.id = w.id;
```

- 写入结果数据至MySQL表中。

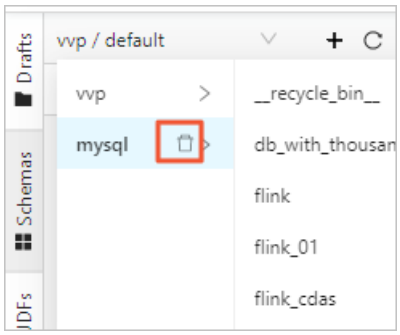
```
INSERT INTO `${mysql_catalog}`.`${db_name}`.`${table_name}`
SELECT ...
FROM ${other_source_table}
```

删除MySQL Catalog

支持UI与SQL命令两种方式删除MySQL Catalog，推荐使用UI方式删除MySQL Catalog。

1. 登录实时计算管理控制台。
2. 在**Flink全托管**页签，单击目标工作空间**操作**列下的**控制台**。
3. 在左侧导航栏，单击**作业开发**。

4. 在左侧，单击Schemas页签。
5. 在顶部菜单栏下拉框中，选择目标MySQL Catalog名称，单击 图标。



6. 在弹出的页面中，单击**删除**。
7. 单击 图标，刷新查看新建的MySQL Catalog是否已被删除。

1. 登录实时计算管理控制台。
2. 在**Flink全托管**页签，单击目标工作空间**操作**列下的**控制台**。
3. 在左侧导航栏，单击**作业开发**。
4. 在页面左上角，单击**新建**，文件类型选择SQL。
5. 在文本编辑区域，输入以下命令。

```
DROP CATALOG ${catalog_name}
```

其中，catalog_name为您要删除的在Flink全托管开发控制台上显示的MySQL Catalog名称。

> **说明** 删除MySQL Catalog不会影响已运行的作业，但会导致使用该Catalog下表的作业，在上线或重启时报无法找到该表的错误，请您谨慎操作。

6. 单击**执行**。
7. 在左侧，单击Schemas页签。
8. 单击 图标，刷新查看新建的MySQL Catalog是否已被删除。

UI方式

SQL命令方式

7.2.11. 管理DLF Catalog

配置DLF Catalog后，您就可以在Flink全托管控制台直接访问DLF实例中的表。本文为您介绍如何在Flink全托管模式下配置、查看及删除DLF Catalog。

背景信息

阿里云数据湖构建（Data Lake Formation，简称DLF）是阿里云官方提供的一款统一元数据管理的产品。您可以方便地借助DLF产品来管理创建的表格，这些表格的格式可以是流行的开源格式，例如Iceberg、Hudi、Delta、Parquet、ORC或Avro等。目前阿里云EMR常见的计算引擎（例如Spark、Flink、Hive或Presto等）都跟DLF有着较为完善的集成度。详情请参见产品简介。

本文将从以下方面为您介绍如何管理DLF Catalog：

- 配置DLF Catalog
- 查看DLF Catalog
- 使用DLF Catalog
- 删除DLF Catalog

前提条件

已开通阿里云数据湖构建服务。如果您没有开通，则可以在DLF产品首页，单击**立即开通**。

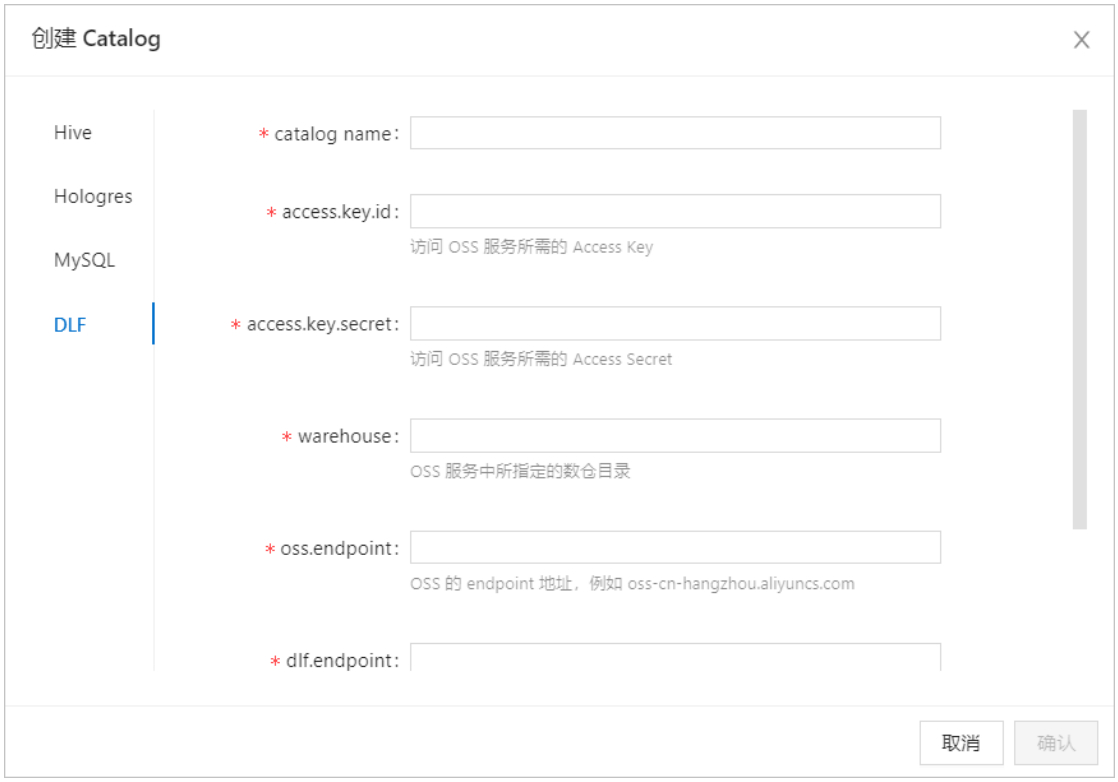
使用限制

- 仅Flink计算引擎vvr-4.0.12-flink-1.13及以上版本支持配置DLF Catalog。
- Flink在DLF Catalog下仅支持管理Iceberg和Hudi两种数据湖格式。

配置DLF Catalog

支持UI与SQL命令两种方式配置DLF Catalog，推荐使用UI方式配置DLF Catalog。

1. 登录实时计算管理控制台。
2. 进入创建Catalog页面。
   i. 在**Flink全托管**页签，单击目标工作空间**操作**列下的**控制台**。
   ii. 在左侧导航栏，单击**作业开发**。
   iii. 在左侧，单击**Schemas**页签。
   iv. 单击 + 图标，在菜单栏下拉框中选择**创建Catalog**。
3. 创建DLF Catalog。
   i. 在**创建Catalog**页面，选择**DLF**。
   ii. 填写参数配置信息。



参数	说明	是否必填	备注
catalogname	DLF Catalog名称。	是	请填写为自定义的英文名。
access.key.id	阿里云账号的AccessKey ID。	是	获取方法请参见获取AccessKey。
access.key.secret	阿里云账号的AccessKey Secret。	是	获取方法请参见获取AccessKey。
warehouse	DLF Catalog中表默认的OSS存放路径。	是	格式为*oss://<bucket>/<object>*。其中： ■ bucket：表示您创建的OSS Bucket名称。 ■ object：表示您存放数据的路径。 **说明** 请在OSS管理控制台上查看您的bucket和object名称。

参数	说明	是否必填	备注
oss.endpoint	阿里云对象存储服务OSS的Endpoint。	是	请详情参见访问域名和数据中心。 **说明** ■ 推荐您为oss.endpoint参数配置OSS的VPC Endpoint。例如，如果您选择的地域为cn-hangzhou地域，则oss.endpoint需要配置为oss-cn-hangzhou-internal.aliyuncs.com。 ■ 如果您需要跨VPC访问OSS，则请参见如何访问跨VPC里的存储资源？
dlf.endpoint	阿里云DLF服务的Endpoint。	是	详情请参见已开通的地域和访问域名。 **说明** ■ 推荐您为dlf.endpoint参数配置DLF的VPC Endpoint。例如，如果您选择的地域为cn-hangzhou地域，则dlf.endpoint参数需要配置为dlf-vpc.cn-hangzhou.aliyuncs.com。 ■ 如果您需要跨VPC访问DLF，则请参见如何访问跨VPC里的存储资源？。
dlf.region-id	阿里云DLF服务的地域名。	是	详情请参见已开通的地域和访问域名。 **说明** 请和dlf.endpoint选择的地域保持一致。

iii. 单击**确认**。

4. 单击C图标，刷新查看新建的DLF Catalog。

1. 登录实时计算管理控制台。
2. 在**Flink全托管**页签，单击目标工作空间**操作**列下的**控制台**。
3. 在左侧导航栏，单击**作业开发**。
4. 在页面左上角，单击**新建**，文件类型选择SQL。
5. 在文本编辑区域，输入配置DLF Catalog的命令。

```
CREATE CATALOG dlf WITH (
   'type' = 'dlf',
   'access.key.id' = '<YourAliyunAccessKeyId>',
   'access.key.secret' = '<YourAliyunAccessKeySecret>',
   'warehouse' = '<YourAliyunOSSLocation>',
   'oss.endpoint' = '<YourAliyunOSSEndpoint>',
   'dlf.region-id' = '<YourAliyunDLFRegionId>',
   'dlf.endpoint' = '<YourAliyunDLFEndpoint>'
);
```

参数	说明	是否必填	备注
catalogname	DLF Catalog名称。	是	请填写为自定义的英文名。
type	类型。	是	固定值为dlf。
access.key.id	阿里云账号的AccessKey ID。	是	获取方法请参见获取AccessKey。
access.key.secret	阿里云账号的AccessKey Secret。	是	获取方法请参见获取AccessKey。

参数	说明	是否必填	备注
warehouse	DLF Catalog中表默认的OSS存放路径。	是	格式为*oss://<bucket>/<object>*。其中： ○ bucket：表示您创建的OSS Bucket名称。 ○ object：表示您存放数据的路径。 说明 请在OSS管理控制台上查看您的bucket和object名称。
oss.endpoint	阿里云对象存储服务OSS的Endpoint。	是	请详情参见访问域名和数据中心。 说明 ○ 推荐您为oss.endpoint参数配置OSS的VPC Endpoint。例如，如果您选择的地域为cn-hangzhou地域，则oss.endpoint需要配置为oss-cn-hangzhou-internal.aliyuncs.com。 ○ 如果您需要跨VPC访问OSS，则请参见如何访问跨VPC里的存储资源？
dlf.endpoint	阿里云DLF服务的Endpoint。	是	详情请参见已开通的地域和访问域名。 说明 ○ 推荐您为dlf.endpoint参数配置DLF的VPC Endpoint。例如，如果您选择的地域为cn-hangzhou地域，则dlf.endpoint参数需要配置为dlf-vpc.cn-hangzhou.aliyuncs.com。 ○ 如果您需要跨VPC访问DLF，则请参见如何访问跨VPC里的存储资源？ 。

参数	说明	是否必填	备注
dlf.region-id	阿里云DLF服务的地域名。	是	详情请参见已开通的地域和访问域名。 说明 请和dlf.endpoint选择的地域保持一致。

6. 单击**执行**。

   执行完会提示Query has been executed。

7. 在左侧，单击**Schemas**页签。
8. 单击C图标，刷新查看新建的DLF Catalog。

UI方式

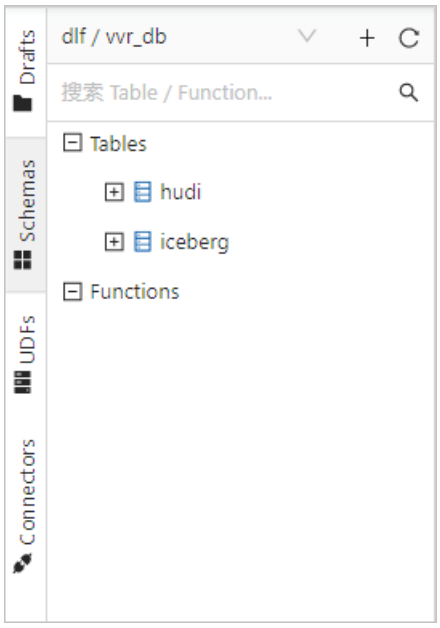
SQL命令方式

查看DLF Catalog

DLF Catalog配置成功后，您可以通过以下步骤查看DLF元数据。

1. 登录实时计算管理控制台。
2. 在**Flink全托管**页签，单击目标工作空间**操作**列下的**控制台**。
3. 在左侧导航栏，单击**作业开发**。
4. 单击**Schemas**页签。
5. 在顶部菜单栏下拉框中，切换到目标DLF Catalog名称，本文以dlf为例。

6. 查看目标DLF Catalog下的数据库、表和函数信息。



使用DLF Catalog

- 创建数据库和表
  i. 在**作业开发**页面，创建一个新的流SQL作业，编写如下代码。

```
CREATE DATABASE dlf.dlf_testdb;
CREATE TABLE dlf.dlf_testdb.iceberg (
  id    BIGINT,
  data  STRING,
  dt    STRING
) PARTITIONED BY (dt) WITH(
  'connector' = 'iceberg'
);
CREATE TABLE dlf.dlf_testdb.hudi (
  id    BIGINT PRIMARY KEY NOT ENFORCED,
  data  STRING,
  dt    STRING
) PARTITIONED BY (dt) WITH(
  'connector' = 'hudi'
);
```

ii. 依次选中三条DDL语句，单击**执行**，完成dlf数据库、dlf_testdb.iceberg表、dlf_testdb.hudi表的创建。

> **说明**
> - 您可以在您创建的Flink Session集群上执行数据库和表的创建操作，这样能更快地完成执行操作。
> - 创建完成后，在**作业开发**页面，单击Schemas，就可以查看到上述创建的数库和表。

- 写入数据
  i. 在**作业开发**页面，创建一个新的流SQL作业，编写如下代码。

```
INSERT INTO dlf.dlf_testdb.iceberg VALUES (1, 'AAA', '2022-02-01'), (2, 'BBB', '2022-02-01');
INSERT INTO dlf.dlf_testdb.hudi VALUES (1, 'AAA', '2022-02-01'), (2, 'BBB', '2022-02-01');
```

  ii. 单击**验证**，完成SQL语法的验证。
  iii. 单击**上线**，完成流作业的上线。
- 读取数据
  i. 在**作业开发**页面，创建一个新的流SQL作业，编写如下代码。

```
SELECT * FROM dlf.dlf_testdb.iceberg LIMIT 2;
SELECT * FROM dlf.dlf_testdb.hudi LIMIT 2;
```

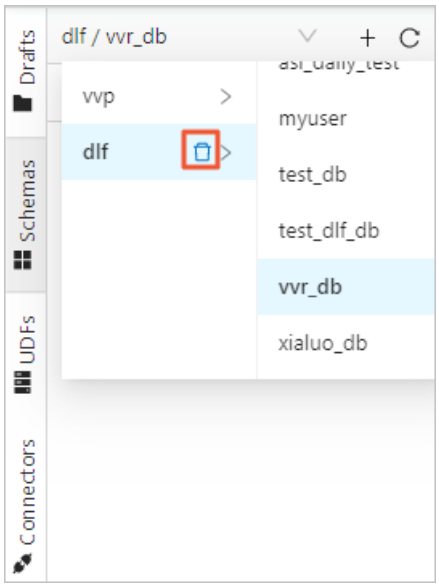
  ii. 单击**验证**，完成SQL语法的验证。
  iii. 单击**执行**，执行完后可以在控制台上查看返回的数据记录。

删除DLF Catalog

> **说明** 删除DLF Catalog不会影响已运行的作业，但会导致使用该Catalog下表的作业，在上线或重启时报无法找到该表的错误，请您谨慎操作。

支持UI与SQL命令两种方式删除DLF Catalog，推荐使用UI方式删除DLF Catalog。

1. 登录实时计算管理控制台。
2. 在**Flink全托管**页签，单击目标工作空间**操作**列下的**控制台**。
3. 在左侧导航栏，单击**作业开发**。
4. 在左侧，单击**Schemas**页签。
5. 在顶部菜单栏下拉框中，选择目标DLF Catalog名称，单击 图标。



6. 在弹出的页面中，单击**删除**。
7. 单击 图标，刷新查看新建的DLF Catalog是否已被删除。

1. 登录实时计算管理控制台。
2. 在**Flink全托管**页签，单击目标工作空间**操作**列下的**控制台**。
3. 在左侧导航栏，单击**作业开发**。
4. 在页面左上角，单击**新建**，文件类型选择SQL。
5. 在文本编辑区域，输入以下命令。

```
DROP CATALOG ${catalog_name}
```

   其中，catalog_name为您要删除的在Flink全托管开发控制台上显示的DLF Catalog名称。

6. 单击**执行**。
7. 在左侧，单击**Schemas**页签。
8. 单击 图标，刷新查看新建的DLF Catalog是否已被删除。

UI方式

SQL命令方式

7.2.12. 管理自定义Connectors

本文为您介绍如何管理Flink全托管自定义Connectors，包括创建、更新和删除。

背景信息

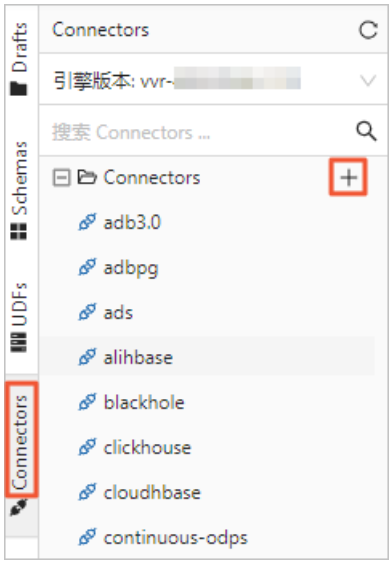
在使用Flink SQL开发作业时，需要使用SQL Connectors连接您的源表、结果表和维表。Flink全托管产品支持云上常用的多种类型的Connectors，但相对于目前大数据常用的技术栈而言，仍然只是较少的一部分，为了支持您可以使用更多类型的Connectors，我们支持您自定义Connectors后上传使用。

注意事项

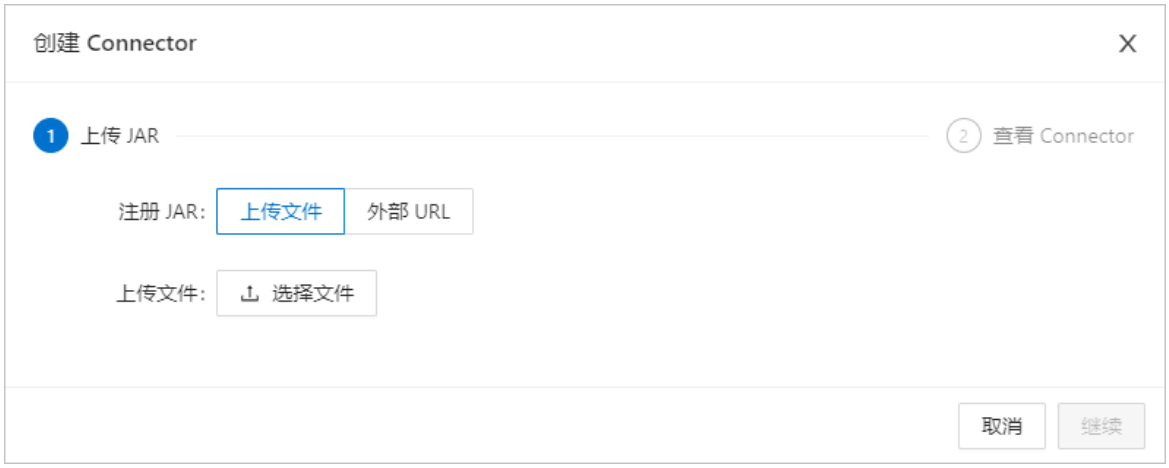
- 您需要根据社区定义的Connectors标准开发您自定义的Connectors，才能正确地被Flink全托管产品识别并使用，因此您需要明确的Connector Meta文件和申明Factory类。自定义Connector开发详情请参见User-defined Sources & Sinks。
- 同种类型的Connectors仅可以上传一次。
- 为了避免JAR包依赖冲突，您还需要注意以下几点：
  - Flink镜像和Pom依赖的Flink版本请保持一致。
  - 请不要上传Runtime层的JAR包，即在依赖中添加 `<scope>provided</scope>` 。
  - 其他第三方依赖请采用Shade方式打包，Shade打包详情参见Apache Maven Shade Plugin。

创建自定义Connectors

1. 登录实时计算控制台。
2. 在**Flink全托管**页签，单击目标工作空间**操作**列下的**控制台**。
3. 在左侧导航栏，单击**作业开发**。
4. 在**Connectors**页签，单击 + 图标。



5. 上传自定义Connectors JAR文件。



您可以通过以下任何一种方式上传自定义Connectors JAR文件：

- **上传文件**：单击**选择文件**后，选择您的目标Connectors JAR文件。
- **外部URL**：输入外部URL地址。如果外部URL是OSS Bucket地址，则自定义Connectors文件必须位于*sql-artifacts/namespaces/{namespace}*目录下。

6. 上传完成后，单击**继续**。

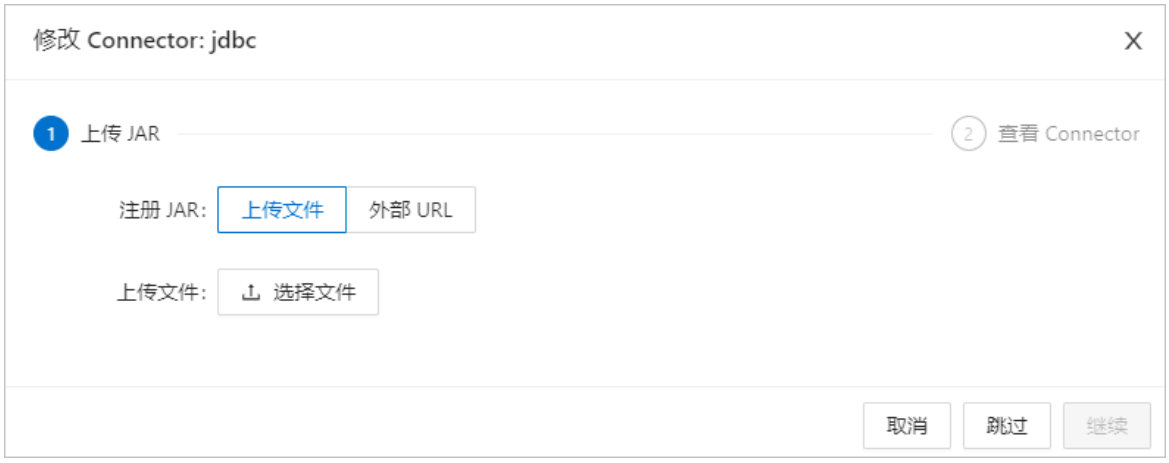
   系统会对您上传的自定义Connectors内容进行解析。如果解析成功，您可以继续下一步。如果解析失败，请确认您上传的自定义Connectors代码是否符合Flink社区标准。

7. 单击**完成**。

   创建完成的自定义Connectors会出现在Connectors列表中。

更新自定义Connectors

1. 登录实时计算控制台。
2. 在**Flink全托管**页签，单击目标工作空间**操作**列下的**控制台**。
3. 在左侧导航栏，单击**作业开发**。
4. 单击**Connectors**页签。
5. 在**Connectors**列表中，鼠标悬停在目标自定义Connectors名称上，单击 。
6. 上传自定义Connectors JAR文件。



您可以通过以下任何一种方式上传自定义Connectors JAR文件：

- **上传文件**：单击**选择文件**后，选择您的目标Connectors JAR文件。
- **外部URL**：输入外部URL地址。如果外部URL是OSS Bucket地址，则自定义Connectors文件必须位于*sql-artifacts/namespaces/{namespace}*目录下。

7. 上传完成后，单击**继续**。

   系统会对您上传的自定义Connectors内容进行解析。如果解析成功，您可以继续下一步。如果解析失败，请确认您上传的自定义Connectors代码是否符合Flink社区标准。

8. 单击**完成**。

   创建完成的自定义Connectors会出现在Connectors列表中。

删除自定义Connectors

如果您的自定义Connectors不再使用，您可以按照以下步骤删除自定义Connectors。

1. 登录实时计算控制台。
2. 在**Flink全托管**页签，单击目标工作空间**操作**列下的**控制台**。
3. 在左侧导航栏，单击**作业开发**。
4. 单击**Connectors**页签。
5. 在**Connectors**列表中，鼠标悬停在目标自定义Connectors名称上，单击 图标。
6. 单击**确认**。

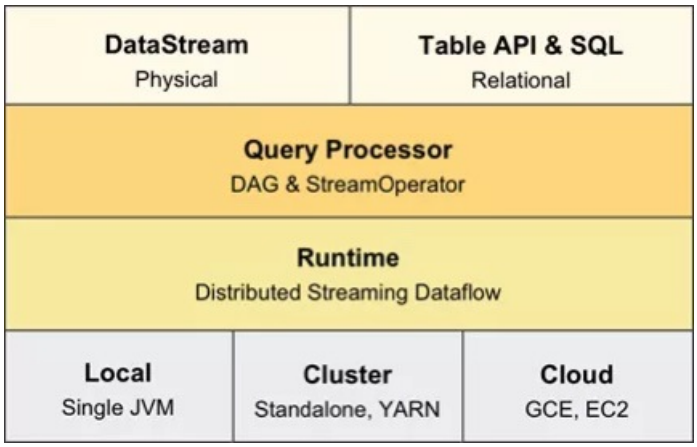
7.3. Flink Datastream开发指南

7.3.1. 概述

Flink全托管开发控制台为您提供Flink Datastream作业开发、提交、暂停和停止功能。

> **注意**
>
> - 如果Datastream作业访问的上下游存储提供了白名单机制，则您需要进行白名单配置。配置方法请参见设置白名单。
> - 目前实时计算产品支持的Datastream API完全兼容开源的Flink版本，详情请参见Flink DataStream API开发指南。
> - Table API作业的开发方式与Datastream作业相同。

Flink Datastream提供了阿里云Flink全托管产品的底层API调用功能，方便您灵活地使用Flink全托管。



Flink Datastream开发指南主要包含如下内容：

- 作业开发

  介绍Flink全托管DataStream API作业开发的限制说明和开发方法，详情请参见作业开发。

- 作业提交

  介绍如何提交Flink全托管DataStream API作业，详情请参见作业提交。

- 作业启动

  介绍如何启动Flink全托管DataStream API作业至集群运行，详情请参见作业启动。

- 作业暂停与停止

  介绍如何在Flink全托管开发控制台上对状态为RUNNING的作业进行暂停或停止操作，详情请参见作业暂停与停止。

7.3.2. 作业开发

本文为您介绍Flink全托管DataStream API开发的限制说明和开发方法。

使用限制

由于Flink全托管产品受部署环境、网络环境等因素的影响，所以开发Flink全托管DataStream作业，需要注意以下限制：

- 仅支持JAR形式的作业提交和运行。
- 支持一个主JAR包和多个附加依赖JAR包。
- 不支持在Main函数中读取本地配置。
- Flink全托管产品开发界面上配置的参数优先级均低于作业代码中的优先级。为了保证作业的正常运行，建议优先在开发界面配置Checkpoint相关参数，请勿在作业代码中配置Checkpoint相关参数。
- 产品运行环境使用的是JDK 1.8，作业开发也需要使用JDK 1.8。
- 支持开源Scala V2.11版本。

注意事项

为了避免JAR包依赖冲突，您需要注意以下几点：

- 作业开发页面选择的Flink版本，请和Pom依赖中的Flink版本保持一致。

- Flink相关依赖，scope请使用provided，即在依赖中添加 `<scope>provided</scope>` 。
- 其他第三方依赖请采用Shade方式打包，Shade打包详情请参见Apache Maven Shade Plugin。

Flink 依赖冲突问题，详情请参见如何解决Flink依赖冲突问题？

作业开发

您需要在线下完成作业开发后，再在Flink全托管开发控制台上提交作业到集群上运行。您在编写Flink全托管产品业务代码时，可以参见以下文档：

- Flink Datastream Connector依赖，请参见Connector依赖。
- Connector的使用方法，请参见Connector使用。
- Apache Flink简介，以及它的体系架构、应用程序和特性功能等，请参见Apache Flink介绍。
- Apache Flink V1.10业务代码开发，请参见Flink DataStream API开发指南和Flink Table API & SQL开发指南。
- Apache Flink的编码、Java语言、Scala语言、组件和格式等指南，请参见代码风格和质量指南。
- Apache Flink编码过程中遇到的问题及解决方法，请参见常见问题。

Connector使用

Maven中央库中已经放置了VVR Connector，以供您在作业开发时直接使用。您可以使用以下任意一种方式来使用Connector：

- （推荐）直接将Connector作为项目依赖打进作业JAR包。
  i. 在Maven项目的pom.xml文件中添加以下配置以引用SNAPSHOT仓库。

```
<repositories>
  <repository>
    <id>oss.sonatype.org-snapshot</id>
    <name>OSS Sonatype Snapshot Repository</name>
    <url>http://oss.sonatype.org/content/repositories/snapshots</url>
    <releases>
      <enabled>false</enabled>
    </releases>
    <snapshots>
      <enabled>true</enabled>
    </snapshots>
  </repository>
  <repository>
    <id>apache.snapshots</id>
    <name>Apache Development Snapshot Repository</name>
    <url>https://repository.apache.org/content/repositories/snapshots/</url>
    <releases>
      <enabled>false</enabled>
    </releases>
    <snapshots>
      <enabled>true</enabled>
    </snapshots>
  </repository>
</repositories>
```

  ii. 检查您的settings.xml配置文件中是否存在 `<mirrorOf>*</mirrorOf>` 配置。

如果存在 `<mirrorOf>*</mirrorOf>` 配置，则需要将此配置改为 `<mirrorOf>*,!oss.sonatype.org-snapshot,!apache.snapshots</mirrorOf>` 。修改的目的是为了避免SNAPSHOT仓库被覆盖，因为mirrorOf中只使用星号（*）会导致第一步中配置的两个repository被覆盖。

iii. 在作业的Maven POM文件中添加您需要的Connector作为项目依赖，示例如下。

```
<dependencies>
    <dependency>
        <groupId>com.alibaba.ververica</groupId>
        <artifactId>${connector.type}</artifactId>
        <version>${connector.version}</version>
    </dependency>
</dependencies>
```

每个Connector版本对应的Connector类型可能不同，建议您使用最新版本。Connector版本、VVR/Flink版本和Connector类型的对应关系请参见Connector列表。

注意

- 您需要在SNAPSHOT仓库（*oss.sonatype.org*）查找带SNAPSHOT的Connector版本， 在Maven中央库（*search.maven.org*）上会查找不到。
- 在使用多个Connector时，请注意META-INF目录需要Merge，即在pom.xml文件中添加如下代码。

```
<transformers>
    <!-- The service transformer is needed to merge META-INF/services files
-->
    <transformer implementation="org.apache.maven.plugins.shade.resource.Se
rvicesResourceTransformer"/>
    <transformer implementation="org.apache.maven.plugins.shade.resource.Ap
acheNoticeResourceTransformer">
        <projectName>Apache Flink</projectName>
        <encoding>UTF-8</encoding>
    </transformer>
</transformers>
```

- 上传Connector JAR包到Flink全托管开发控制台后，填写配置信息。
  i. 登录实时计算管理控制台。
  ii. 在**Flink全托管**页签，单击目标工作空间**操作**列下的**控制台**。
  iii. 在左侧导航栏，单击**资源上传**。
  iv. 单击**上传资源**，选择您要上传的目标Connector的JAR包。

  您可以上传您自己开发的Connector，也可以上传Flink全托管产品提供的Connector。Flink全托管产品提供的Connector官方JAR包的下载地址，请参见Connector列表。

  v. 在目标作业开发页面**附加依赖文件**项，选择目标Connector的JAR包。

Connector列表

Flink全托管仅支持提供了demo的Connector，具体内容如下表所示。

Connector版本	VVR/Flink版本	Connector类型

1.13-vvr-4.0.7	VVR 4.0.7（对应Flink 1.13）	• ververica-connector-datahub
		  ○ 文档请参见DataHub DataStream Connector。
		  ○ demo请参见ververica-connector-datahub-demo。
		• ververica-connector-kafka
		  ○ 文档请参见Kafka DataStream Connector。
		  ○ demo请参见ververica-connector-kafka-demo。
		• ververica-connector-odps或者ververica-connector-continuous-odps
		  ○ 文档请参见MaxCompute DataStream Connector。
		  ○ demo请参见ververica-connector-maxcompute-demo。
		• ververica-connector-mysql-cdc
		  文档请参见MySQL CDC DataStream Connector。

7.3.3. 作业提交

本文为您介绍如何提交Flink全托管DataStream API作业至集群运行。

上传JAR包

作业运行前，需要您按照以下步骤将JAR包、Python作业文件或Python依赖上传到Flink全托管开发控制台。

> **说明** 系统支持上传包的数量限制为200个，每个包大小限制为200 MB。如果JAR包超过200 MB，建议通过OSS上传，详情请参见如何在OSS控制台上传JAR包？

1. 登录实时计算管理控制台。
2. 在**Flink全托管**页签，单击目标工作空间**操作**列下的**控制台**。
3. 在左侧导航栏，单击**资源上传**。
4. 单击**上传资源**，选择您要上传的JAR包。

   如果您的作业是Python API类型，则需要上传PyFlink的官方JAR包。官方JAR的下载地址，请参见PyFlink V1.11和PyFlink V1.12。

   > **说明** VVP 2.4.0增加了单独的Python作业提交入口，推荐您通过该入口提交Python作业，详情请参见概述。

创建作业

1. 登录Flink全托管开发控制台，新建作业。
   i. 登录实时计算管理控制台。
   ii. 在**Flink全托管**页签，单击目标工作空间**操作**列下的**控制台**。
   iii. 在左侧导航栏，单击**作业开发**。
   iv. 单击**新建**。
   v. 在**新建文件**对话框，填写作业配置信息。

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必须保持唯一。
文件类型	流作业和批作业均支持以下文件类型： ■ SQL ■ JAR ■ PYTHON **说明** VVP 2.4.1且VVR 3.0.1及以上版本支持批作业。
部署目标	选择作业需要部署的集群，支持以下两种集群模式： ■ **Per-Job集群**（默认）：适用于占用资源比较大或持续稳定运行的作业。因为作业之间资源隔离，每个作业都需要一个独立的JM，小任务JM的资源利用率较低。 ■ **Session集群**：适用于占用资源比较小或任务启停比较频繁的作业。因为多个作业可以复用相同的JM，可以提高JM资源利用率。 **说明** 如果您需要开启SQL Preview功能，必须选择**Session集群**，且已将其设置为SQL Previews集群，详情请参见作业调试和配置开发测试环境（Session集群）。
存储位置	指定该作业的代码文件所属的文件夹。 您还可以在现有文件夹右侧，单击 图标，新建子文件夹。

   vi. 单击**确认**。
2. 在作业开发页面，填写基本配置信息。

   您可以直接填写以下配置信息，也可以单击**YAML**直接修改配置信息。配置参数解释如下表所示。

参数	说明
部署目标	您可以修改创建作业时已选择的部署目标。
JAR URI	请选择一个文件或者手动上传新文件，您可以拖拽文件到此区域或者单击右侧图标选择文件上传。 **说明** 如果您的作业是Python API类型，则需要填写PyFlink的官方JAR包，官方JAR的下载地址，请参见PyFlink V1.11和PyFlink V1.12。
Entrypoint Class	程序的入口类。如果您的JAR包未指定主类，请在此处输入您的**Entrypoint Class**类的标准路径。 **说明** 如果您的作业是Python API类型，Entrypoint class应该填写为org.apache.flink.client.python.PythonDriver。
Entrypoint main args	您可以在此处传入参数，在主方法里面去调用该参数。 **说明** 如果您的作业是Python API类型，需要首先上传您的Python作业文件。Python作业文件上传之后，默认会被上传到作业运行节点的*/flink/usrlib/*目录下。 假如您的Python作业文件名为word_count.py，则Entrypoint main args需要填写为 `-py /flink/usrlib/word_count.py` 。 Python作业文件的路径需要填写为完整路径，*/flink/usrlib/*不能省略，且不能更改。

参数	说明
附加依赖文件	○ （推荐）选择您已上传的目标附加依赖文件。 您需要提前通过Flink全托管开发控制台左侧**资源上传**或作业开发页面**附件依赖文件**右侧的上传附件依赖文件。上传的附件依赖文件会固定被保存在*oss://ossBucketName/artifacts/namespaces/namespaceName/* 目录*。 ○ 填写目标附加依赖文件的OSS路径。 您需要提前将附加依赖文件上传至当前实例对应的OSS Bucket，上传附加依赖文件的OSS Bucket必须为您开通Flink全托管时选择的OSS Bucket。 ○ 填写目标附加依赖文件的URL，目前仅支持以文件名结尾的URL，例如*http://xxxxxx/file*。 您需要提前将附加依赖文件上传至公开可访问的HTTP服务。 **说明** ○ Session集群不支持设置附加依赖文件，仅Per-Job集群支持设置附加依赖文件。 ○ 以上三种方式上传的附件依赖文件，在作业运行时，最终都会被加载到JM和TM所在Pod的*/flink/usrlib*目录下。 ○ Session模式的作业不支持配置附加依赖文件路径。 ○ 如果您的作业是Python API类型，则需要在这里选择您的Python作业文件，以及所用的依赖文件。Python依赖详情，请参见Python依赖管理。Python依赖上传之后，默认会被上传到作业运行节点的*/flink/usrlib/*目录下。
并行度	作业并发数。

3. 在作业开发页面右侧，单击**高级配置**，根据业务需要填写配置信息。

   参数解释如下表所示。

类别	配置项	说明

类别	配置项	说明
常规配置	引擎版本	仅支持vvr-4.0.7-flink-1.13和vvr-3.0.3-flink-1.12版本。 **说明** 从VVR 3.0.3版本（对应Flink 1.12版本）开始，VVP支持同时运行多个不同引擎版本的SQL作业。如果您的作业已使用了Flink 1.12及更早版本的引擎，您需要按照以下情况进行处理： ○ Flink 1.12版本：停止后启动作业，系统将自动将引擎升级为vvr-3.0.3-flink-1.12版本。 ○ Flink 1.11或Flink 1.10版本：手动将作业引擎版本升级到vvr-3.0.3-flink-1.12或vvr-4.0.7-flink-1.13版本后重启作业，否则会在启动作业时超时报错。
	编辑标签	您可以为作业设置标签。后续可以通过搜索的方式，快速找到目标作业。
行为配置	创建实例的最大重试次数	创建实例失败后的重试次数。
	Stop with Drain	如果开启Stop With Drain功能，当作业被手动停止时，所有基于Event Time的窗口都会被触发。
Flink配置	Checkpoint间隔	定时执行Checkpoint的时间间隔。如果不填写，将会关闭 Checkpoint。
	两次Checkpoint之间的最短时间间隔	两次Checkpoint之间的最短时间间隔，如果Checkpoint最大并行度是1，则该配置确保两个Checkpoint之间有一个最短时间间隔。
	开启Unaligned Checkpoint	开启Unaligned Checkpoint会大大降低反压情况下Checkpoint的总执行时间。但是也会导致增大单次Checkpoint的大小。
	Flink重启策略配置	当有Task失败时，如果没有开启Checkpoint，JobManager进程不会重启。如果开启了Checkpoint，则JobManager进程会重启。该参数取值如下： ○ Failure Rate：基于失败率（您填写时间间隔内的最大失败次数）重启。 ○ Fixed Delay：固定间隔重启。 ○ No Restarts（默认值）：不会重启。
	更多Flink配置	在此设置其他Flink配置。例如 `taskmanager.numberOfTaskSlots: 1` 。

类别	配置项	说明
资源配置	Job Manager CPUs	默认值为1。
	Job Manager Memory	最小值为1 GiB。单位建议使用GiB或MiB，例如，1024 MiB或1.5 GiB。
	Task Manager CPUs	默认值为1。
	Task Manager Memory	最小值为1 GiB。单位建议使用GiB或MiB，例如，1024 MiB或1.5 GiB。
日志配置	日志归档	默认已开启日志归档功能，并设置归档日志有效期为7天。开启日志归档后，您可以在作业探查页面查看历史作业实例的日志，详情请参见查看历史作业实例日志。 说明 ○ 在VVR 3.x版本，仅VVR 3.0.7及以上版本支持开启日志归档功能。 ○ 在VVR 4.x版本，仅VVR 4.0.11及以上版本支持开启日志归档功能。
	Root Log Level	日志级别从低到高顺序如下： i. TRACE：比DEBUG更细粒度的信息。 ii. DEBUG：系统运行状态的信息。 iii. INFO：重要或者您感兴趣的信息。 iv. WARN：系统可能出现潜在错误的信息。 v. ERROR：系统出现错误和异常的信息。
	Log Levels	填写日志名称和日志级别。
	Logging Profile	日志模板，可以选择系统模板，也可以选择用户配置。

4. 在作业开发页面右上角，单击**上线**。

7.3.4. 作业启动

完成作业开发和上线后，您需要在作业运维页面启动作业至运行阶段。此外，作业在暂停或者停止后需要恢复时，需要启动作业。

背景信息

作业启动的场景如下：

- 上线后启动：您只能选择全新启动（无状态）。
- 暂停后启动：暂停作业时，您可以选择是否执行一次Checkpoint或Savepoint后结束作业运行。作业暂停后启动时，从最新状态恢复。

- 停止后启动：停止作业时，系统会自动清除作业相关的Checkpoint信息。作业停止后启动时，您可以选择全新启动（无状态），或者从所选Savepoint开始恢复作业。

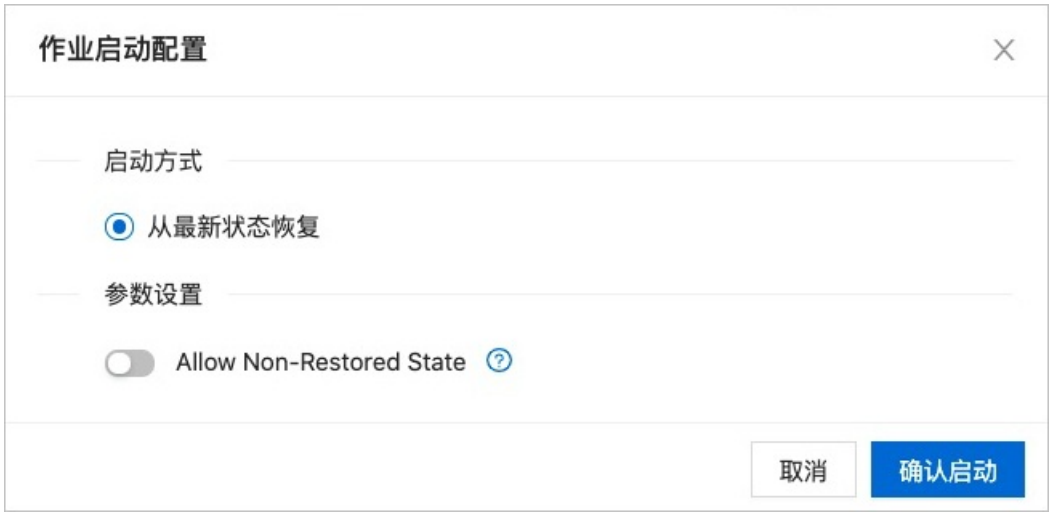
操作步骤

1. 登录实时计算管理控制台。
2. 在**Flink全托管**页签，单击对应工作空间**操作**列下的**控制台**。
3. 在左侧导航栏上，选择**应用 > 作业运维**。
4. 单击目标作业名称**操作**列中的**启动**。
5. 在**作业启动配置**对话框，选择启动方式和参数设置。

   不同场景的启动界面如下所示：

   - 上线后启动

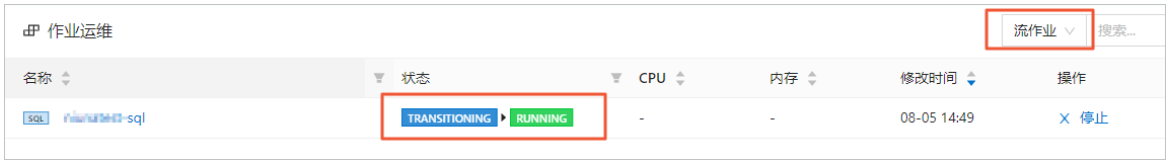
   - 暂停后启动



   - 停止后启动

6. 单击**确认启动**。
7. 查看作业状态变化。

   单击启动后，您可以看到作业从当前状态到期望状态的变化过程及最终结果。



   作业的运行状态及含义如下表所示。

   - **期望状态**

状态	说明
RUNNING	期望作业正常运行。
SUSPENDED	期望作业暂停运行。
CANCELLED	期望作业停止运行。

   - **当前状态**

状态	说明
RUNNING	作业正常运行。
CANCELLED	作业停止运行。
SUSPENDED	作业暂停运行。
FINISHED	作业运行完成。
TRANSITIONING	作业处于向期望状态的过渡阶段。
FAILED	作业向期望状态转换失败，需要手动处理。
ClusterUnreachable	VVP无法访问JM（JobManager），获取不到Job的状态。

常见问题

报错：Invalid versionName string

7.3.5. 作业暂停与停止

您可以在Flink全托管开发控制台上对状态为RUNNING的作业进行暂停或停止操作，本文为您介绍如何暂停与停止作业。

背景信息

暂停和停止操作在功能说明、使用场景和启动方案方面各有不同。您需要根据实际的业务需要，选择对应的操作。二者的差异情况如下表所示。

操作	功能说明	使用场景	启动方案
暂停	暂停作业时，您可以按照需要选择在暂停前是否执行一次Savepoint。	更改了SQL逻辑或者修改了资源配置后，需要经过暂停和启动的步骤，才能变更生效。	作业暂停后再启动时，默认从最新的Savepoint或Checkpoint状态恢复。详情请参见作业启动。
停止	停止作业时，您可以选中是否在停止前执行一次Savepoint。默认选中在停止前执行一次Savepoint。系统将按照以下情况进行处理： • 如果选中执行一次Savepoint，那么作业会先进行Savepoint后进入停止流程。在作业停止后，系统会自动清除作业相关的Checkpoint信息。 • 如果未选中执行一次Savepoint，那么作业会进入停止流程。在作业停止后，系统会自动清除作业相关的Checkpoint信息。	更改作业版本、增删WITH参数或增加作业参数后，需要经过停止和启动的步骤，才能使变更生效。另外，如果无法复用State，希望作业全新启动时，您需要停止后再启动作业。	作业停止后再启动时，您可以选择全新启动（无状态），同时您也可以指定读取数据的时间；或者从所选Savepoint恢复作业。详情请参见作业启动。 **说明** 如果您修改了SQL等逻辑导致State不兼容，则只能从无状态启动作业。

作业暂停

1. 登录实时计算管理控制台。
2. 在**Flink全托管**页签，单击对应工作空间**操作**列下的**控制台**。
3. 在左侧导航栏上，单击**作业运维**。
4. 单击目标作业**操作**列下的**暂停**。
5. 单击**确定**。

说明　单击**确定**前，您可以选中**在暂停前做一次Savepoint**。

作业停止

1. 登录实时计算管理控制台。
2. 在**Flink全托管**页签，单击对应工作空间**操作**列下的**控制台**。
3. 在左侧导航栏上，单击**作业运维**。
4. 单击目标作业**操作**列下的**停止**。
5. 单击**确定**。

说明　停止时默认选中**在停止前创建Savepoint**，您也可以取消选中。

7.4. Flink Python开发指南

7.4.1. 概述

Flink全托管开发控制台为您提供Flink Python作业开发、提交、暂停和停止、依赖管理等功能。

Flink Python开发指南主要包含如下内容：

- 作业开发
- 作业提交
- 作业启动
- 作业暂停与停止
- 使用Python依赖

7.4.2. 作业开发

本文为您介绍Flink全托管Python API作业开发的背景信息、使用限制、开发方法和Connector使用等。

背景信息

Flink全托管集群已安装下列软件包。

软件包	版本
apache-beam	2.23.0
avro-python3	1.9.1
certifi	2020.12.5
chardet	4.0.0
cloudpickle	1.2.2
crcmod	1.7
cython	0.29.16
dill	0.3.1.1
docopt	0.6.2
fastavro	0.23.6
future	0.18.2
grpcio	1.29.0
hdfs	2.6.0
httplib2	0.17.4
idna	2.10
jsonpickle	1.2
mock	2.0.0
numpy	1.19.5
oauth2client	3.0.0
pandas	0.25.3
pbr	5.5.1
pip	20.1.1
protobuf	3.15.3
py4j	0.10.8.1

软件包	版本
pyarrow	0.17.1
pyasn1-modules	0.2.8
pyasn1	0.4.8
pydot	1.4.2
pymongo	3.11.3
pyparsing	2.4.7
python-dateutil	2.8.0
pytz	2021.1
requests	2.25.1
rsa	4.7.2
setuptools	47.1.0
six	1.15.0
typing-extensions	3.7.4.3
urllib3	1.26.3
wheel	0.36.2

使用限制

由于Flink全托管产品受部署环境、网络环境等因素的影响，所以开发Python作业，需要注意以下限制：

- 仅支持开源Flink V1.12及以上版本。
- Flink全托管集群已预装了Python 3.7.9，且Python环境中已预装了Pandas、NumPy、PyArrow等常用的Python库。因此需要您在Python 3.7版本开发代码。
- Flink全托管运行环境使用的是JDK1.8，如果Python作业中依赖第三方JAR包，请确保JAR包兼容JDK1.8。
- 仅支持开源Scala V2.11版本，如果Python作业中依赖第三方JAR包，请确保使用Scala V2.11对应的JAR包依赖。

作业开发

您需要在线下完成Python API作业开发后，再在Flink全托管开发控制台上提交作业到集群上运行。您可以参见以下文档开发Flink全托管产品业务代码：

- Apache Flink V1.12业务代码开发，请参见Flink Python API开发指南。
- Apache Flink编码过程中遇到的问题及解决方法，请参见常见问题。

作业调试

您可以在Python自定义函数的代码实现中，通过logging的方式，输出日志信息，方便后期问题定位，示例如下。

```
@udf(result_type=DataTypes.BIGINT())
def add(i, j):
  logging.info("hello world")
  return i + j
```

日志输出后，您可以在TaskManager的日志文件查看。

Connector使用

Flink全托管所支持的Connector列表，请参见支持的上下游存储。Connector使用方法如下：

1. 登录实时计算控制台。
2. 在**Flink全托管**页签，单击目标工作空间**操作**列下的**控制台**。
3. 在左侧导航栏，单击**资源上传**。
4. 单击**上传资源**，选择您要上传的目标Connector的JAR包。

   您可以上传您自己开发的Connector，也可以上传Flink全托管产品提供的Connector。Flink全托管产品提供的Connector官方JAR包的下载地址，请参见Connector列表。
5. 在目标作业开发右侧**高级配置**面板中的**附加依赖文件**项，选择目标Connector的JAR包。
6. 在目标作业开发右侧**高级配置**面板中的**更多Flink配置**项，添加相关配置。

   假如您的作业需要依赖多个Connector JAR包，且名字分别为connector-1.jar和connector-2.jar，则配置信息如下。

   ```
   pipeline.classpaths: 'file:///flink/usrlib/connector-1.jar;file:///flink/usrlib/connector-2.jar'
   ```

7.4.3. 作业提交

本文为您介绍如何提交Flink全托管Python作业至集群。

上传资源

Python API作业运行前，需要您按照以下步骤将Python作业文件或Python依赖上传到Flink全托管开发控制台。

1. 登录实时计算管理控制台。
2. 在**Flink全托管**页签，单击目标工作空间**操作**列下的**控制台**。
3. 在左侧导航栏，单击**资源上传**。
4. 单击**上传资源**，选择您要上传的Python作业文件或Python依赖。

作业提交

1. 登录Flink全托管开发控制台，新建作业。
   i. 登录实时计算管理控制台。
   ii. 在**Flink全托管**页签，单击目标工作空间**操作**列下的**控制台**。
   iii. 在左侧导航栏，单击**作业开发**。

iv. 单击**新建**。

v. 在**新建文件**对话框，填写作业配置信息。

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必须保持唯一。
文件类型	流作业和批作业均支持以下文件类型： ■ SQL ■ JAR ■ PYTHON **说明** VVP 2.4.1且VVR 3.0.1及以上版本支持批作业。
部署目标	选择作业需要部署的集群。 **说明** Python作业仅支持**Per-Job集群**。
存储位置	指定该作业的代码文件所属的文件夹。 您还可以在现有文件夹右侧，单击 图标，新建子文件夹。

vi. 单击**确认**。

2. 在作业开发页面，填写基本配置信息。

   您可以直接填写以下配置信息，也可以单击**YAML**直接修改配置信息。配置参数解释如下表所示。

参数	说明
部署目标	您可以修改创建作业时已选择的部署目标。
Python文件地址	Python作业文件，可以为.py文件或者.zip文件。
Entry Module	程序的入口类。如果Python作业文件为.py文件，则该项不需要填写；如果Python作业文件为.zip文件，则需要在此处输入您的Entry Module，例如example.word_count。
Entrypoint main args	作业参数。
Python Libraries	第三方Python包。第三方Python包会被添加到Python worker进程的PYTHONPATH中，从而在Python自定义函数中可以直接访问。如何使用第三方Python包，详情请参见使用第三方Python包。

参数	说明

Python Archives	存档文件，目前仅支持ZIP格式的文件，例如 .zip、.jar、.whl和.egg等。
	存档文件会被解压到Python worker进程的工作目录下。如果存档文件所在的压缩包名称为mydata.zip，则在Python自定义函数中可以编写以下代码来访问mydata.zip存档文件。
	`def map():`
	`  with open("mydata.zip/mydata/data.txt") as f:`
	`  ...`
	请参见使用自定义的Python虚拟环境和使用数据文件，了解更多关于Python Archives的信息。

附加依赖文件	上传JAR包、数据文件等，上传的依赖文件默认会被上传到作业运行节点的*/flink/usrlib/*目录下。 **说明** Session集群不支持设置附加依赖文件，仅Per-Job集群支持设置附加依赖文件。
并行度	作业并发数。

3. 在作业开发页面右侧，单击**高级配置**，根据业务需要填写配置信息。

   参数解释如下表所示。

类别	配置项	说明
基础配置	部署目标	您可以修改创建作业时已选择的部署目标。
	附加依赖文件	如果您需要添加更多依赖文件，请选择或者输入任意合法的文件地址。 **说明** Session集群不支持设置附加依赖文件，仅Per-Job集群支持设置附加依赖文件。

类别	配置项	说明
常规配置	引擎版本	仅支持vvr-4.0.7-flink-1.13和vvr-3.0.3-flink-1.12版本。 **说明** 从VVR 3.0.3版本（对应Flink 1.12版本）开始，VVP支持同时运行多个不同引擎版本的SQL作业。如果您的作业已使用了Flink 1.12及更早版本的引擎，您需要按照以下情况进行处理： ○ Flink 1.12版本：停止后启动作业，系统将自动将引擎升级为vvr-3.0.3-flink-1.12版本。 ○ Flink 1.11或Flink 1.10版本：手动将作业引擎版本升级到vvr-3.0.3-flink-1.12或vvr-4.0.7-flink-1.13版本后重启作业，否则会在启动作业时超时报错。
	编辑标签	您可以为作业设置标签。后续可以通过搜索的方式，快速找到目标作业。
行为配置	创建实例的最大重试次数	创建实例失败后的重试次数。
	Stop with Drain	如果开启Stop With Drain功能，当作业被手动停止时，所有基于Event Time的窗口都会被触发。
Flink配置	Checkpoint间隔	定时执行Checkpoint的时间间隔。如果不填写，将会关闭Checkpoint。
	两次Checkpoint之间的最短时间间隔	两次Checkpoint之间的最短时间间隔，如果Checkpoint最大并行度是1，则该配置确保两个Checkpoint之间有一个最短时间间隔。
	开启Unaligned Checkpoint	开启Unaligned Checkpoint会大大减少反压情况下Checkpoint的总执行时间。但是也会导致增大单次Checkpoint的大小。
	Flink重启策略配置	当有Task失败时，如果没有开启Checkpoint，JobManager进程不会重启。如果开启了Checkpoint，则JobManager进程会重启。该参数取值如下： ○ Failure Rate：基于失败率重启。 ○ Fixed Delay：固定间隔重启。 ○ No Restarts（默认值）：不会重启。
	更多Flink配置	在此设置其他Flink配置。例如 `taskmanager.numberOfTaskSlots: 1` 。

类别	配置项	说明
日志配置	日志归档	默认已开启日志归档功能，并设置归档日志有效期为7天。开启日志归档后，您可以在作业探查页面查看历史作业实例的日志，详情请参见查看历史作业实例日志。 说明 ○ 在VVR 3.x版本，仅VVR 3.0.7及以上版本支持开启日志归档功能。 ○ 在VVR 4.x版本，仅VVR 4.0.11及以上版本支持开启日志归档功能。
	Root Log Level	日志级别从低到高顺序如下： i. TRACE：比DEBUG更细粒度的信息。 ii. DEBUG：系统运行状态的信息。 iii. INFO：重要或者您感兴趣的信息。 iv. WARN：系统可能出现潜在错误的信息。 v. ERROR：系统出现错误和异常的信息。
	Log Levels	填写日志名称和日志级别。
	Logging Profile	日志模板，可以选择系统模板，也可以选择用户配置。

4. 在作业开发页面右上角，单击**上线**。

7.4.4. 作业启动

完成作业开发和上线后，您需要在作业运维页面启动作业至运行阶段。此外，作业在暂停或者停止后需要恢复时，需要启动作业。

背景信息

作业启动的场景如下：

- 上线后启动：您只能选择全新启动（无状态）。
- 暂停后启动：暂停作业时，系统会默认执行一次Savepoint后结束作业运行。作业暂停后启动时，从最新状态恢复。
- 停止后启动：停止作业时，系统会自动清除作业相关的Checkpoint信息。作业停止后启动时，您可以选择全新启动（无状态），或者从所选Savepoint开始恢复作业。

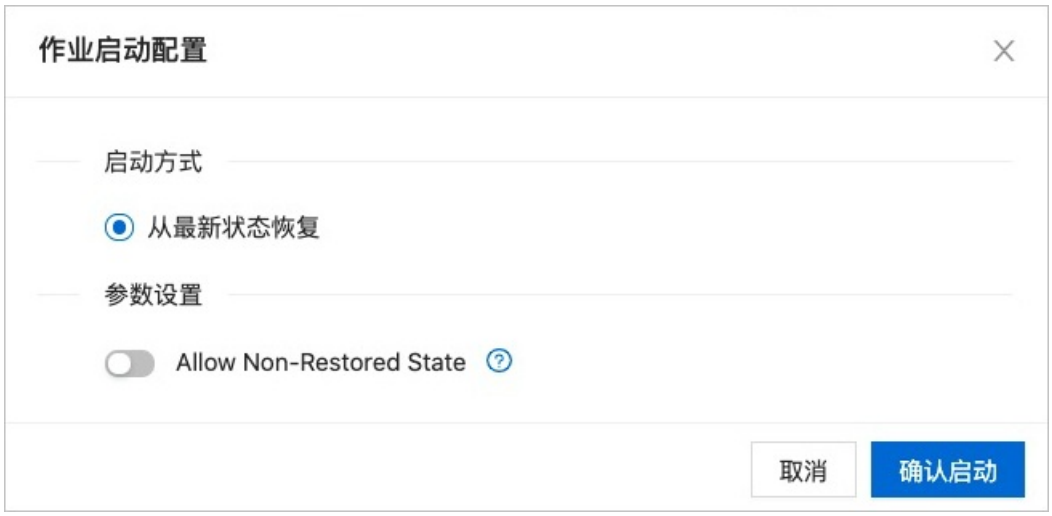
操作步骤

1. 登录实时计算管理控制台。
2. 在**Flink全托管**页签，单击对应工作空间**操作**列下的**控制台**。
3. 在左侧导航栏上，选择**应用 > 作业运维**。
4. 单击目标作业名称**操作**列中的**启动**。
5. 在**作业启动配置**对话框，选择启动方式和参数设置。

不同场景的启动界面如下所示：

- 上线后启动

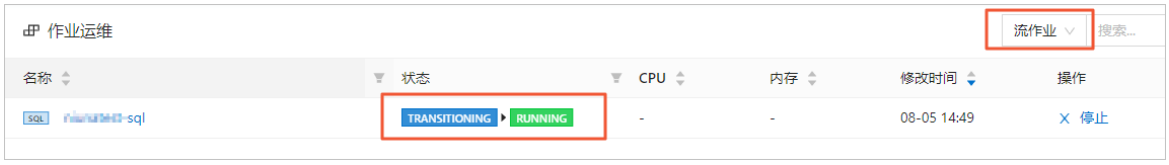
- 暂停后启动



- 停止后启动

6. 单击**确认启动**。
7. 查看作业状态变化。

   单击启动后，您可以看到作业从当前状态到期望状态的变化过程及最终结果。



作业的运行状态及含义如下表所示。

- **期望状态**

状态	说明
RUNNING	期望作业正常运行。
SUSPENDED	期望作业暂停运行。
CANCELLED	期望作业停止运行。

- **当前状态**

状态	说明
RUNNING	作业正常运行。
CANCELLED	作业停止运行。
SUSPENDED	作业暂停运行。
FINISHED	作业运行完成。
TRANSITIONING	作业处于向期望状态的过渡阶段。
FAILED	作业向期望状态转换失败，需要手动处理。
ClusterUnreachable	VVP无法访问JM（JobManager），获取不到Job的状态。

常见问题

报错：Invalid versionName string

7.4.5. 作业暂停与停止

您可以在Flink全托管开发控制台上对状态为RUNNING的作业进行暂停或停止操作，本文为您介绍如何暂停与停止作业。

背景信息

暂停和停止操作在功能说明、使用场景和启动方案方面各有不同。您需要根据实际的业务需要，选择对应的操作。二者的差异情况如下表所示。

操作	功能说明	使用场景	启动方案

操作	功能说明	使用场景	启动方案
暂停	暂停作业时，您可以按照需要选择在暂停前是否执行一次Savepoint。	更改了SQL逻辑或者修改了资源配置后，需要经过暂停和启动的步骤，才能变更生效。	作业暂停后再启动时，默认从最新的Savepoint或Checkpoint状态恢复。详情请参见作业启动。
停止	停止作业时，您可以选中是否在停止前执行一次Savepoint。默认选中在停止前执行一次Savepoint。系统将按照以下情况进行处理： • 如果选中执行一次Savepoint，那么作业会先进行Savepoint后进入停止流程。在作业停止后，系统会自动清除作业相关的Checkpoint信息。 • 如果未选中执行一次Savepoint，那么作业会进入停止流程。在作业停止后，系统会自动清除作业相关的Checkpoint信息。	更改作业版本、增删WITH参数或增加作业参数后，需要经过停止和启动的步骤，才能使变更生效。另外，如果无法复用State，希望作业全新启动时，您需要停止后再启动作业。	作业停止后再启动时，您可以选择全新启动（无状态），同时您也可以指定读取数据的时间；或者从所选Savepoint恢复作业。详情请参见作业启动。 说明 如果您修改了SQL等逻辑导致State不兼容，则只能从无状态启动作业。

作业暂停

1. 登录实时计算管理控制台。
2. 在**Flink全托管**页签，单击对应工作空间**操作**列下的**控制台**。
3. 在左侧导航栏上，单击**作业运维**。
4. 单击目标作业**操作**列下的**暂停**。
5. 单击**确定**。

说明 单击**确定**前，您可以选中**在暂停前做一次Savepoint**。

作业停止

1. 登录实时计算管理控制台。
2. 在**Flink全托管**页签，单击对应工作空间**操作**列下的**控制台**。
3. 在左侧导航栏上，单击**作业运维**。
4. 单击目标作业**操作**列下的**停止**。

5. 单击**确定**。

说明 停止时默认选中**在停止前创建Savepoint**，您也可以取消选中。

7.4.6. 使用Python依赖

您可以在Flink Python作业中使用自定义的Python虚拟环境、第三方Python包、JAR包和数据文件等，本文为您介绍如何在Python作业中使用这些依赖。

背景信息

本文通过以下场景为您介绍如何使用Python依赖：

- 使用自定义的Python虚拟环境
- 使用第三方Python包
- 使用JAR包
- 使用数据文件

使用自定义的Python虚拟环境

当前仅支持Python 3.7的Python虚拟环境，下文为您介绍如何准备Python 3.7的虚拟环境。

1. 准备Python 3.7的虚拟环境。

i. 在本地准备setup-pyflink-virtual-env.sh脚本，其内容如下。

```
set -e
# 下载Python 3.7 miniconda.sh脚本。
wget "https://repo.continuum.io/miniconda/Miniconda3-py37_4.9.2-Linux-x86_64.sh" -O
"miniconda.sh"
# 为Python 3.7 miniconda.sh脚本添加执行权限。
chmod +x miniconda.sh
# 创建Python的虚拟环境。
./miniconda.sh -b -p venv
# 激活Conda Python虚拟环境。
source venv/bin/activate ""
# 安装PyFlink依赖。
# update the PyFlink version if needed
pip install "apache-flink==1.12.1"
# 关闭Conda Python虚拟环境。
conda deactivate
# 删除缓存的包。
rm -rf venv/pkgs
# 将准备好的Conda Python虚拟环境打包。
zip -r venv.zip venv
```

ii. 在本地准备build.sh脚本，其内容如下。

```
#!/bin/bash
set -e -x
yum install -y zip wget
cd /root/
bash /build/setup-pyflink-virtual-env.sh
mv venv.zip /build/
```

iii. 在CMD命令行，执行如下命令。

```
docker run -it --rm -v $PWD:/build  -w /build quay.io/pypa/manylinux2014_x86_64 ./b
uild.sh
```

执行完该命令后，会生成一个名字为venv.zip的文件，即为Python 3.7的虚拟环境。

您也可以修改上述脚本，在虚拟环境中安装所需的第三方Python包。

2. 在Python作业中使用Python 3.7虚拟环境。

i. 登录实时计算控制台。

ii. 在**Flink全托管**页签，单击目标工作空间**操作**列下的**开发控制台**。

iii. 在左侧导航栏，单击**资源上传**，上传venv.zip文件。

> **说明** Flink全托管开发控制台上可以上传文件大小的限制为200 MB，而Python虚拟环境的大小通常会超过该限制，因此，您需要通过OSS管理控制台上传文件。

iv. 在目标作业开发页面的**Python Archives**项，选择venv.zip文件。

v. 单击右侧的**高级配置**，在**更多Flink配置**项，添加配置信息。

```
python.executable: venv.zip/venv/bin/python
```

vi. 单击**保存**。

使用第三方Python包

下面将从以下两个场景为您介绍如何使用第三方Python包：

- 使用可直接Import的第三方Python包

  如果您的第三方Python包是Zip Safe的，即不需要安装即可直接在Python作业中使用。操作步骤如下：

  i. 下载可直接Import的第三方Python包。

     a. 在浏览器上打开PyPI页面。

     b. 在搜索框中输入目标第三方Python包名称，例如apache-flink 1.12.2。

     c. 在搜索结果中，单击目标结果名称。

     d. 在左侧导航栏，单击Download files。

     e. 单击文件名中包含cp37-cp37m-manylinux1的包名称进行下载。

  ii. 登录实时计算控制台。

  iii. 在**Flink全托管**页签，单击目标工作空间**操作**列下的**开发控制台**。

  iv. 在左侧导航栏，单击**资源上传**，上传第三方Python包。

  v. 在目标作业开发页面的Python Libraries项，选择所上传的第三方Python包。

  vi. 单击**保存**。

- 使用需要编译的第三方Python包

  如果您的第三方Python包是tar.gz格式的压缩包，或从其他地方下载的源码包，且压缩包的根目录下存在setup.py文件，则这种类型的第三方Python包通常需要先编译才能被使用。您需要先在与Flink全托管兼容的环境下编译第三方Python包，然后才可在Python作业中调用第三方Python包。

  推荐使用quay.io/pypa/manylinux2014_x86_64镜像容器中的Python 3.7来编译第三方Python包，使用该容器编译生成的包兼容绝大多数Linux环境，关于该镜像容器的更多信息请参见manylinux。

  > **说明** Python 3.7的安装路径为 /opt/python/cp37-cp37m/bin/python3。

  下面以opencv-python-headless第三方Python包为例，介绍一下如何编译和使用该第三方Python包。

  i. 编译第三方Python包。

     a. 在本地准备requirements.txt文件，其内容如下。

     ```
     opencv-python-headless
     ```

     b. 在本地准备build.sh脚本，其内容如下。

     ```
     #!/bin/bash
     set -e -x
     yum install -y zip
     PYBIN=/opt/python/cp37-cp37m/bin
     "${PYBIN}/pip" install --target __pypackages__ -r requirements.txt --no-deps
     cd __pypackages__ && zip -r deps.zip . && mv deps.zip ../ && cd ..
     rm -rf __pypackages__
     ```

c. 在CMD命令行，执行如下命令。

```
docker run -it --rm -v $PWD:/build  -w /build quay.io/pypa/manylinux2014_x86_64 /
bin/bash build.sh
```

该命令执行完后，会生成一个名字为deps.zip的文件，该文件为编译之后的第三方Python包。

您也可以修改requirements.txt，安装其他所需的第三方Python包。此外，requirements.txt文件中可以指定多个Python依赖。

ii. 在Python作业中使用第三方Python包deps.zip。

a. 登录实时计算控制台。

b. 在**Flink全托管**页签，单击目标工作空间**操作**列下的**开发控制台**。

c. 在左侧导航栏，单击**资源上传**，上传deps.zip。

> **说明** Flink全托管产品限制可上传的最大文件大小为200 MB，而Python虚拟环境的大小通常会超过该限制，需要通过OSS控制台进行文件上传。

d. 在目标作业开发页面的**Python Libraries**项，选择deps.zip。

e. 单击**保存**。

使用JAR包

如果您的Flink Python作业中使用了Java类，例如作业中使用了Connector或者Java自定义函数时，可以通过如下方式来指定Connector或者Java自定义函数的JAR包。

1. 登录实时计算控制台。
2. 在**Flink全托管**页签，单击目标工作空间**操作**列下的**开发控制台**。
3. 在左侧导航栏，单击**资源上传**，上传需要使用的JAR包。
4. 在目标作业开发页面的附加依赖文件项，选择需要使用的JAR包。
5. 单击右侧的**高级配置**，在**更多Flink配置**项，添加配置信息。

   假如需要依赖多个JAR包，且名字分别为jar1.jar和jar2.jar，配置内容如下。

```
pipeline.classpaths: 'file:///flink/usrlib/jar1.jar;file:///flink/usrlib/jar2.jar'
```

6. 单击**保存**。

使用数据文件

> **说明** Flink全托管暂不支持通过上传数据文件的方式来进行python作业调试。

下面将从两个场景为您介绍如何使用数据文件：

- 通过Python Archives选项方式

  如果您的数据文件的数量比较多时，您可以将数据文件打包成一个ZIP包，然后通过如下方式在Python作业中使用。操作步骤如下：

  i. 登录实时计算控制台。

  ii. 在**Flink全托管**页签，单击目标工作空间**操作**列下的**开发控制台**。

  iii. 在左侧导航栏，单击**资源上传**，上传目标数据文件ZIP包。

iv. 在目标作业开发页面的Python Archives项，选择需要使用的数据文件ZIP包。

v. 在Python自定义函数中，可以通过如下方式访问数据文件。假如数据文件所在压缩包名称为mydata.zip。

```
def map():
    with open("mydata.zip/mydata/data.txt") as f:
    ...
```

- 通过附加依赖文件选项

如果您的数据文件数量比较少时，可以通过如下方式在Python作业中使用。操作步骤如下：

i. 登录实时计算控制台。

ii. 在**Flink全托管**页签，单击目标工作空间**操作**列下的**开发控制台**。

iii. 在左侧导航栏，单击**资源上传**，上传目标数据文件。

iv. 在目标作业开发页面的附加依赖文件项，选择需要的数据文件。

v. 在Python自定义函数中，可以通过如下方式访问数据文件。以数据文件名称为data.txt为例的代码如下。

```
def map():
    with open("/flink/usrlib/data.txt") as f:
    ...
```

8.作业运维

8.1. 作业管理

8.1.1. 查看作业详情

本文为您介绍如何查看作业总览信息，包括实例状态、启动时间和重启次数等。

操作步骤

1. 登录实时计算控制台。
2. 在**Flink全托管**页签，单击目标工作空间**操作**列下的**开发控制台**。
3. 在左侧导航栏上，选择**应用 > 作业运维**。
4. 单击目标作业名称。
5. 单击**作业总览**页签。

> **说明** 单击**指标**，您可以查看更多指标，包括Flink的常用指标。您也可以单击**Flink UI**去查看作业运行日志。

8.1.2. 查看与修改作业状态

本文为您介绍如何查看和修改作业状态，以及不同状态所代表的含义。

查看作业状态

1. 登录实时计算管理控制台。
2. 在**Flink全托管**页签，单击目标工作空间**操作**列下的**开发控制台**。
3. 在左侧导航栏上，选择**应用 > 作业运维**。
4. 单击目标作业名称。
5. 在**作业总览**页签，查看作业状态信息。

   页面左上角会显示两种作业状态：

   - 期望状态

状态	说明
RUNNING	按作业配置，启动作业。
CANCELLED	停止任何正在运行的作业。

状态	说明
SUSPENDED	先对当前运行的作业创建Savepoint，再停止作业。

- 当前状态

状态	说明
RUNNING	按作业配置定义，启动作业。
CANCELLED	停止任何正在运行的作业。
SUSPENDED	先对当前运行的作业创建Savepoint，再停止作业。
FINISHED	作业运行完成。
TRANSITIONING	作业处于向期望状态的过度阶段。
FAILED	作业向期望状态转换失败，需要手动处理。
ClusterUnreachable	VVP无法访问JM（JobManager），获取不到Job的状态。

说明 单击**启动**后，作业的期望状态会立刻变成**RUNNING**，当前状态会变成**TRANSITIONING**，并显示作业启动进度。前端开发平台VVP（Ververica Platform）会保证Flink作业最终达到期望状态。

如果Flink作业状态正在从某个状态变成期望状态的过程中，而作业的最终期望状态发生了改变，作业会先到达最初的期望状态，再转换为最终的期望状态。

修改作业状态

1. 登录实时计算控制台。
2. 在**Flink全托管**页签，单击目标工作空间**操作**列下的**开发控制台**。
3. 在左侧导航栏上，选择**应用 > 作业运维**。
4. 单击目标作业名称。
5. 单击**作业总览**页签，进入作业详情页面。
6. 根据需要单击以下按钮，修改作业状态。

其中，每个按钮含义如下表所示。

操作名称	说明

操作名称	说明
暂停	暂停作业，先对当前运行的作业创建Savepoint，再停止作业。暂停作业需要具备以下两个条件： ○ Savepoint地址配置完成。如果Savepoint地址没有配置，暂停操作将直接停止作业，导致作业状态丢失。 ○ Savepoint操作成功。
停止	停止作业。
诊断	系统会根据收集到的瞬时作业信息（例如Failover等）和常见的作业诊断规则，给您一些简单指导意见。
Savepoint	创建一个当前时刻的Savepoint。 **说明** 请先完成Savepoint地址配置，否则Savepoint会失败。
编辑	修改作业基础配置或高级配置。
删除	删除作业。只能删除当前状态不为**RUNNING**的作业。

8.1.3. 修改作业配置

作业上线运行后，您还可以修改作业配置，作业配置包含作业上线阶段的所有配置内容。本文为您介绍如何修改作业配置。

操作步骤

1. 登录实时计算控制台。
2. 在**Flink全托管**页签，单击目标工作空间**操作**列下的**开发控制台**。
3. 在左侧导航栏上，选择**应用 > 作业运维**。
4. 单击目标作业名称。
5. 修改作业配置信息。

   您可以通过以下任何一种方式修改作业配置：

   - **以模板模式编辑**

     在页面右上角，单击**编辑**后，进入作业开发页面，单击右侧的**高级配置**，修改作业配置信息。

   - **以YAML模式编辑**

     在页面右上角，单击**编辑**后，单击页面底部的**YAML**，修改配置信息。
6. 单击**保存**。

8.1.4. 管理作业版本

在作业开发页面，单击上线后，就会自动生成一个新版本。您可以对比当前编辑版本和历史版本的差异，实现回滚历史版本作业信息的需求。

背景信息

本文从以下方面为您介绍如何管理作业版本：

- 对比作业版本
- 回滚作业版本
- 锁定作业版本
- 删除作业版本

对比作业版本

仅支持SQL代码和作业配置内容的对比，有差异的内容就会被标记出来。

1. 登录实时计算管理控制台。
2. 在**Flink全托管**页签，单击目标工作空间**操作**列下的**控制台**。
3. 在左侧导航栏，单击**作业开发**。
4. 双击目标作业名称。
5. 在页面右侧，单击**版本信息**。

6. 在目标版本号对应的**操作**列，单击**对比**。
7. 在左侧**选择版本**中选择目标对比版本。

回滚作业版本

您可以将作业上线后产生的目标版本的作业信息回滚到当前编辑版本，当前编辑版本为您现在正在编辑的作业。

1. 登录实时计算管理控制台。
2. 在**Flink全托管**页签，单击目标工作空间**操作**列下的**控制台**。
3. 在左侧导航栏，单击**作业开发**。
4. 双击目标作业名称。
5. 在页面右侧，单击**版本信息**。

6. 在目标版本号对应的**操作**列，选择**更多** > **回滚**。
7. 在弹出的对话框，单击**回滚**。

锁定作业版本

系统最多支持50个作业版本， 超过50个后，系统会自动删除最旧的版本。您可以通过锁定指定版本，保证该版本不会被自动删除。

1. 登录实时计算管理控制台。
2. 在**Flink全托管**页签，单击目标工作空间**操作**列下的**控制台**。
3. 在左侧导航栏，单击**作业开发**。
4. 双击目标作业名称。
5. 在页面右侧，单击**版本信息**。

6. 在目标版本号对应的**操作**列，选择**更多 > 锁定**。

   锁定成功后，页面上方会弹出锁定成功的信息，且作业版本号前面会显示 🔒 图标。

> **说明** 不能删除锁定的作业版本，您需要先解锁再删除。如果您需要解锁，则需要选择**更多 > 解锁**。

删除作业版本

您可以删除一些您不需要的作业版本。

1. 登录实时计算管理控制台。
2. 在**Flink全托管**页签，单击目标工作空间**操作**列下的**控制台**。
3. 在左侧导航栏，单击**作业开发**。
4. 双击目标作业名称。
5. 在页面右侧，单击**版本信息**。

6. 在目标版本号对应的**操作**列，选择**更多** > **删除**。
7. 在弹出的对话框，单击**删除**。

8.1.5. 变量配置

在配置变量后，您可以在单作业的SQL编辑器或更多Flink配置、作业模版中的其他配置中去调用已配置的变量。本文为您介绍如何配置和调用变量。

背景信息

配置变量可以帮您解决以下两个问题：

- 避免DDL中出现明文AccessKey带来的安全风险。
- 动态修改变量值，避免直接修改SQL，提高变量替换的效率。

使用限制

- 仅支持新增和删除变量，不支持修改和查看变量值。
- 仅支持在创建临时表（即CREATE TEMPORARY TABLE）时的WITH参数中调用变量，如果在持久表（即CREATE TABLE）的WITH参数中调用变量，则这张表会无法正常使用。

注意事项

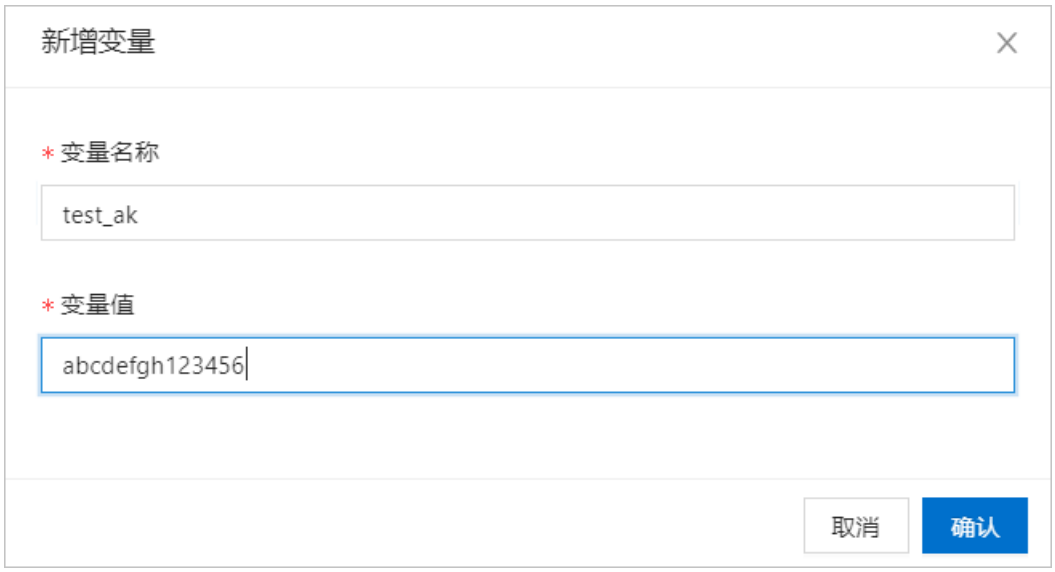
请谨慎删除已创建的变量，因为可能会导致开发中的SQL作业无法上线，或需要重新提交已上线的作业。但不影响已经上线作业的运行。

配置变量

1. 登录实时计算控制台。
2. 在**Flink全托管**页签，单击目标工作空间**操作**列下的**开发控制台**。

3. 在左侧导航栏上，选择**系统管理 > 变量配置**。
4. 单击**新增变量**。
5. 填写**变量名称**和**变量值**。

> **注意** 变量值的字母大小写敏感，请注意变量值的字母大小写。



6. 单击**确认**。

调用变量

配置变量后，您可以在单作业的SQL编辑器中去调用已配置的变量。

1. 在开发控制台左侧导航栏，单击**作业开发**。
2. 在目标作业的SQL编辑器中调用。

   变量在调用时的格式为 `${secret_values.变量名}` 。代码示例如下。

```
CREATE TEMPORARY table datahub_source(
  name VARCHAR
) WITH (
  'connector' = 'datagen'
);
CREATE TEMPORARY table datahub_sink(
  name  VARCHAR
) WITH (
  'connector'='datahub',
  'endpoint'='<yourEndpoint>',
  'project'='<yourProject>',
  'topic'='<yourTopic>',
  'accessId'='<yourAccessId>',
  'accessKey'='${secret_values.test_ak}'
);
INSERT INTO datahub_sink
SELECT
  LOWER(name)
from datahub_source;
```

8.2. 作业调优

8.2.1. 查看作业性能

您可以查看运行实例的作业性能，包括Job Manager的CPU、Task Managers的CPU、内存和线程的使用情况。本文为您介绍如何查看作业Job Manager和Task Managers的性能。

背景信息

查看Job Manager和Task Managers的性能可以协助您定位代码问题，例如作业代码是否存在问题、个别类是否初始化比较慢、个别类是否占用资源比较多等问题。Flink全托管支持查看的性能详情请参见下表。

维度	类别	详情
Job Manager	火焰图	• CPU耗时：通常顶层占据宽度比较大的函数消耗的CPU比较多，表示该类函数可能存在性能问题。 • 内存分配：查看不同类分配的内存情况。 • 锁耗时：查看锁竞争、死锁导致的性能下降问题。 • ITimer：取相同时间间隔下，查看每个间隔内所有线程之间CPU的消耗情况。 说明 关于火焰图的详情请参见火焰图。
Task Managers	火焰图	
	内存动态	JVM不同内存区域的内存变化。
	线程动态	查看线程的情况，并支持选择线程并进行分析采样。

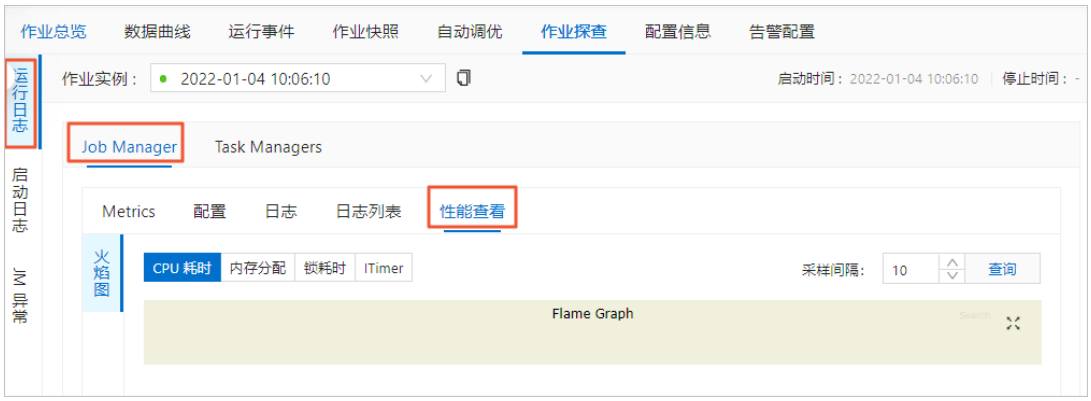
使用限制

- 仅Flink计算引擎VVR 4.0.11及以上版本支持查看作业性能。
- 仅支持查看运行实例的作业性能，历史实例不支持查看作业性能。

操作步骤

1. 登录实时计算控制台。
2. 在**Flink全托管**页签，单击目标工作空间**操作**列下的**开发控制台**。
3. 在左侧导航栏上，选择**应用 > 作业运维**。
4. 单击目标作业名称。
5. 单击**作业探查**。
6. 查看Job Manager或TaskManager的性能。
   - Job Manager性能

a. 在**运行日志**页签，单击**Job Manager**页签下的**性能查看**。

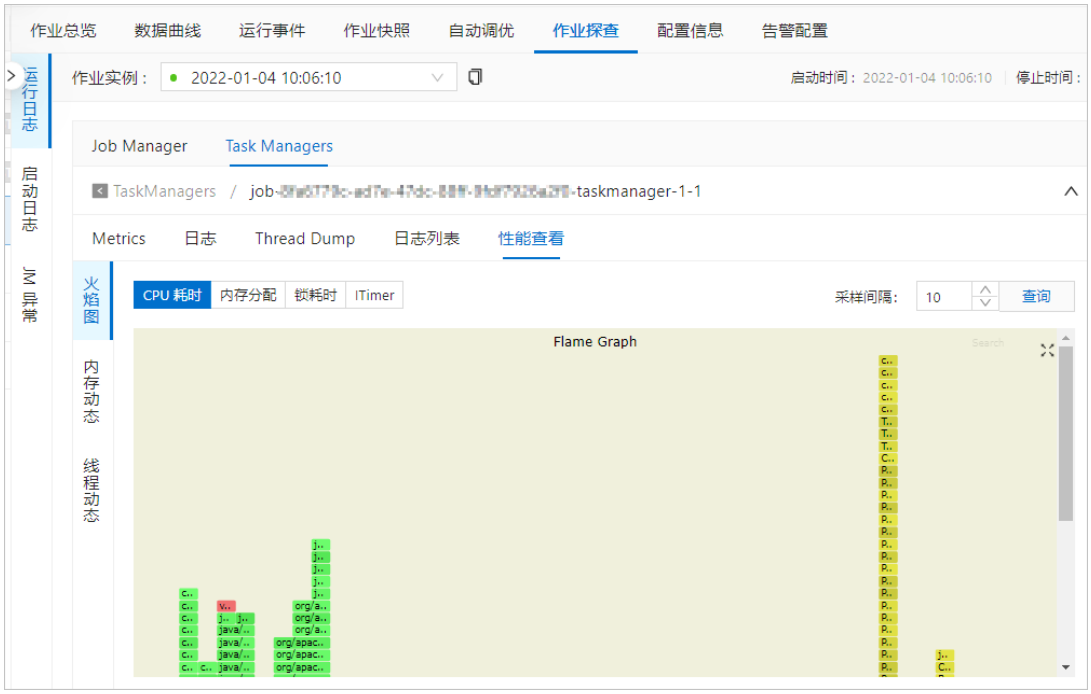


b. 在**火焰图**页签，查看Job Manager性能信息。

- Task Managers性能

a. 在**运行日志**页签，单击**Task Managers**页签下的**Path, ID**。

b. 单击**性能查看**。



c. 查看Task Managers性能信息。

8.2.2. 配置自动调优

Flink全托管已默认开启自动调优（Monitoring模式），本文为您介绍如何配置自动调优，以及配置过程中的注意事项。

背景信息

通常，您需要花费大量的时间进行作业调优。例如，新上线一个作业时，需要考虑如何配置该作业的资源、并发个数、Task Manager个数及大小等。此外，作业运行过程中，还需要考虑如何调整作业资源，使作业处于最高资源利用率；作业出现反压或延时增大的情况时，需要考虑如何调整作业配置等。

Flink全托管的自动调优功能，可以在各个算子和流作业上下游性能都达标和稳定的前提下，帮您更合理地调整作业并行度和资源配置，全局优化您的作业，解决作业吞吐量不足、全链路存在反压和资源浪费等各种性能调优问题。

使用限制

- 自动调优不支持Session集群部署的作业。
- 自动调优无法解决流作业性能瓶颈。

  因为调优策略对作业的处理模式是基于一定的假设的。例如，流量平滑变化、不能有数据倾斜、每个算子的吞吐能力能够随并发度的升高而线性拓展。当业务逻辑严重偏离以上假设时，作业可能会存在异常。例如：

  - 无法触发修改并发度的操作、作业不能达到正常状态和作业持续重启等。
  - 自定义标量函数UDF、自定义聚合函数UDAF或自定义表值函数UDTF性能问题。

  如果出现作业性能瓶颈，您需要将自动调优调整为Monitoring，进行手动调优。
- 自动调优无法识别外部系统的问题。

  外部系统故障或访问变慢时，会导致作业并行度增大，加重外部系统的压力，导致外部系统雪崩。常见的外部系统问题如下：

  - 数据总线DataHub分区不足或消息队列MQ吞吐量不足。
  - Sink性能问题。
  - 云数据库RDS死锁。

  如果出现外部系统问题，您需要自行解决。

注意事项

- 自动调优触发后需要重启作业，因此会导致作业短暂停止处理数据。
- 连续两次自动调优触发间隔为10分钟，您可以通过cooldown.minutes参数来调节触发自动调优的时间间隔。
- 如果您使用了DataStream API或Table API接口编写作业，请确认作业代码中未配置作业并行度，否则自动调优将无法调整作业资源，即自动调优配置无法生效。

默认调优行为

系统默认从并发度和内存两个方面为您进行自动调优：

- 自动调优会调整作业的并发度来满足作业流量变化所需要的吞吐。自动调优会监控消费源头数据的延迟变化情况、TaskManager（TM） CPU实际使用率和各个算子处理数据能力来调整作业的并发度。详情如下：

- 作业延迟Delay指标正常（不超过60s），不主动调高作业的并发度。
- 作业延迟Delay指标超过默认阈值60s，分以下两种情况来调整并发度：
  - 延迟正在下降，不进行并发度调整。
  - 延迟增加并且连续上升3分钟（默认值），调整作业并发度到当前实际TPS的两倍处理能力，但不超过设置最大的资源（默认值为64 CU）。
- 作业不存在延迟指标。
  - 作业某Vertex节点连续6分钟实际处理数据时间占比超过80%，调大作业并发度使得slot-utilization值降低到50%，但不超过设置最大的资源（默认为64 CU）。
  - 所有TM的平均利用率连续6分钟超过80%，调高并发度使TM的CPU使用率降低到50%。
- 所有TM的最大CPU使用率连续24小时低于20%，Vertex的实际处理数据时间低于20%时，调低作业的并发度使CPU和Vertex实际处理的时间占比提高到50%。

- 自动调优也会监控作业的内存使用和Failover情况，来调整作业的内存配置。详情如下：
  - 在JobManager GC频繁或者发生OOM异常时，会调高JM的内存，默认最大调整到16 GiB。
  - 在TM GC频繁或者发生OOM异常、HeartBeartTimout异常时，会调高TM的内存，默认最大调整到16 GiB。
  - 在TM内存使用率超过95%时，会调大TM的内存。
  - 在TM的实际内存使用率连续24小时低于30%时，降低TM内存的配置。

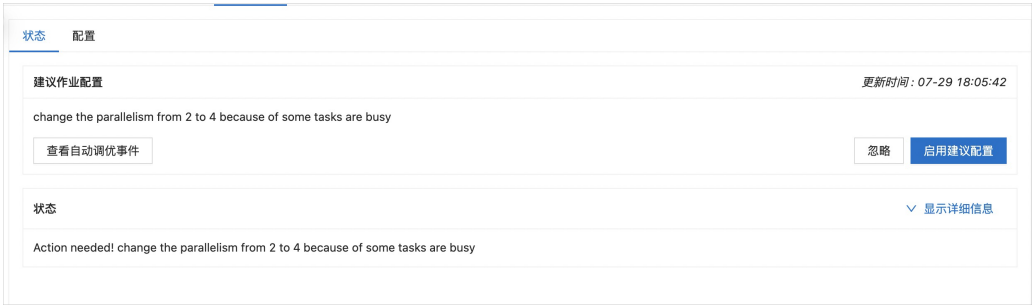
配置自动调优

1. 进入作业运维界面。
   i. 登录实时计算管理控制台。
   ii. 在**Flink全托管**页签，单击目标工作空间**操作**列下的**控制台**。
   iii. 在左侧导航栏上，选择**应用 > 作业运维**。
2. 配置自动调优。
   i. 单击目标作业名称。
   ii. 单击**自动调优**页签。
   iii. 单击**配置**页签。
   iv. 默认已开启自动调优功能（**Monitoring**模式），您可以选择关闭或者以Active自动启动的方式使配置生效。

- Monitoring（默认值）

  开启自动调优功能，系统会根据您选择的调优策略，给出该作业推荐的调优配置，需要您手动进行配置生效。您可以采取以下任意一种生效方式：

  - 在**配置**页签，将模式调整为Active。
  - 在**状态**页签，根据提示信息下进行操作：
    - 如果有**启用建议配置**按钮，单击后会自动重启让配置生效。



    - 如果没有**启用建议配置**按钮，则需要您根据提示信息在作业开发页面**高级配置**中，手动修改对应的配置后，上线重启作业。

- Active

  开启自动调优功能，系统会根据您选择的调优策略，给出该作业推荐的调优配置，并自动启动作业使配置生效，您无需操作。

- Disabled

  不开启自动调优功能。

v. （可选）选择Monitoring或Active模式后，填写自定义配置信息。

配置项	配置含义	默认值
cooldown.minutes	冷却时间是指作业被自动调优两次自动启动之间的最小时间间隔。	10分钟
resources.cu.max	使用的最大资源限制。	64 CU
parallelism.scale-up.interval	调大并发度时最小触发时间间隔。	6分钟
parallelism.scale-down.interval	调低并发度时最小触发时间间隔。	24小时
mem.scale-down.interval	调低内存时最小触发时间间隔。	24小时
parallelism.scale.max	并发度向上调整时，最大并发限制。	-1，表示最大并发没有限制。
parallelism.scale.min	并发度向下调整时，最小并发限制。	1，表示最小并发为1。
delay-detector.scale-up.threshold	可以容忍的最大延迟阈值。	1分钟
slot-usage-detector.scale-up.threshold	当Vertex处理数据时间占比大于该值时，触发调大并发度的操作。	0.8
slot-usage-detector.scale-down.threshold	当Vertex处理数据时间占比大于该值时，触发调小并发度的操作。	0.2
tm-cpu-usage-detector.scale-up.threshold	调大并发度时CPU使用率阈值。	0.8
tm-cpu-usage-detector.scale-down.threshold	调小并发度时CPU使用率阈值。	0.2
resources.memory-scale-up.max	调整单个Task Manager和Job Manager的内存时，能调整到的最大值。	16 GiB

3. 单击**保存**。

VVP 2.5.x和VVP 2.4.x配置项映射关系

VVP 2.5.x的配置项和VVP 2.4.x版本的大部分配置参数项兼容，配置项映射关系详情如下表所示。

VVP 2.4.x	VVP 2.5.x
cpu-based.scale-up.threshold	tm-cpu-usage-detector.scale-up.threshold
cpu-based.scale-down.threshold	tm-cpu-usage-detector.scale-down.threshold
cpu-based.scale-up.window-size.min	tm-cpu-usage-detector.scale-up.sample-interval
cpu-based.scale-down.window-size.min	tm-cpu-usage-detector.scale-down.sample-interval

VVP 2.4.x	VVP 2.5.x
source-delay-based.threshold	delay-detector.scale-up.threshold
source-delay-based.scale-up.window-size.min	delay-detector.scale-up.sample-interval
slot-utilization-based.threshold	slot-usage-detector.scale-down.threshold
slot-utilization-based.scale-down.window-size.min	slot-usage-detector.scale-down.sample-interval
slot-usage-detector.scale-up.threshold	slot-usage-detector.scale-up.threshold
slot-usage-detector.scale-up.sample-interval	slot-usage-detector.scale-up.sample-interval
memory-utilization-based.memory-usage-max.threshold	tm-memory-usage-detector.scale-up.threshold
memory-utilization-based.memory-usage-min.threshold	tm-memory-usage-detector.scale-down.threshold
memory-utilization-based.memory-usage.target-utilization	memory-utilization-based.memory-usage.target-utilization
memory-utilization-based.scale-up.window-size.min	不支持
memory-utilization-based.scale-down.window-size.min	tm-memory-usage-detector.scale-down.sample-interval
memory-utilization-based.gc-ratio.threshold	不支持
memory-utilization-based.gc-time-longest-ms.threshold	不支持
memory-utilization-based.gc-time-ms-per-second.threshold	jm-gc-detector.gc-time-ms-per-second.threshold tm-gc-detector.gc-time-ms-per-second.threshold
memory-utilization-based.memory-scale-up.max	不支持
memory-utilization-based.memory-scale-up.ratio	memory-utilization-based.memory-scale-up.ratio
job-exception-based.oom-exception.memory-scale-up.ratio	job-exception-based.oom-exception.memory-scale-up.ratio
job-exception-based.oom-exception.memory-scale-up.max	不支持
job-exception-based.oom-exception.include-tm-timeout	不支持

8.2.3. 配置细粒度资源

Flink全托管支持全新的资源配置模式，即您可以在专家模式下，对作业的Task所需的资源进行细粒度的资源控制，以满足不同场景下的作业运行需求。本文将为您介绍专家模式的资源配置方式。

背景信息

目前，Flink全托管支持以下三种资源配置模式：

- 基础模式：开源Flink的资源配置方式，您可以配置并发度、JM和TM的内存和CPU。
- 专家模式（BETA）：Flink全托管引入的全新的资源配置模式，支持对作业所使用的资源进行细粒度的资源控制，以满足作业吞吐的要求。

  系统会自动根据您配置的资源需求，以Native K8s的模式运行作业，TM的规格和个数将会根据Slot的规格和作业并发度，由系统自动决定。
- 智能模式（BETA）：智能模式为专家模式的升级版。在智能模式下，作业将启用专家模式的资源配置，同时开启AutoPilot自动调优功能。

  在智能配置模式下，您无需配置相关资源，AutoPilot会自动为作业生成资源配置，并根据作业的运行情况帮您进行资源配置的调优，在确保作业处于健康的状态下，优化作业资源的使用。AutoPilot详情请参见配置自动调优。

使用限制

- 仅SQL作业支持专家模式。
- 仅VVP 2.5.4版本及以上版本支持在专家模式下对作业的Task所需的资源进行细粒度的资源控制。

资源配置建议

- 建议Source节点并发度和分区数成比例，即并发度数能整除分区数。例如Kafka有16个分区，则并发度建议设置为16、8或4，这样可以避免数据倾斜。同时Source节点的并发度不宜设置太小，避免一个Source需要读取太多数据，导致出现入口瓶颈，影响作业吞吐。
- 建议按需配置除Source外的其他节点的并发度。流量大的节点，并发设置大一些；流量小的节点，并发设置小一些。
- 建议在有明确异常或者需求时，再调整Heap Memory和Off-heap Memory的大小，例如作业出现OOM或严重GC等。因为在作业正常运行时，调整Heap Memory和Off-heap Memory的大小，不会明显改变作业的吞吐量。

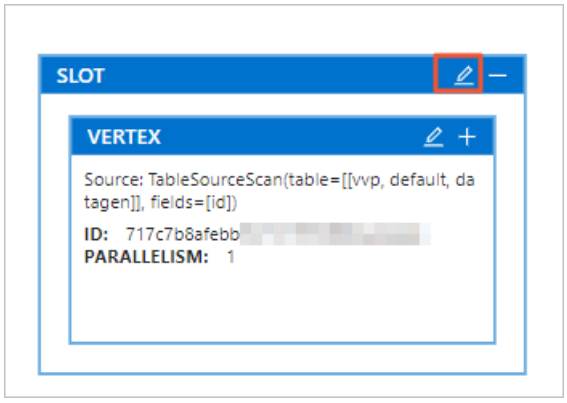
操作步骤

1. 进入资源配置入口。
   i. 登录实时计算管理控制台。
   ii. 在**Flink全托管**页签，单击目标工作空间**操作**列下的**控制台**。
   iii. 在左侧导航栏上，选择**应用 > 作业运维**。
   iv. 单击目标作业名称。
   v. 在作业详情页面右上角，单击**编辑**。
   vi. 在页面右侧，单击**资源配置**页签。
   vii. 资源模式选择为**专家模式**。
2. 配置JM的资源信息。

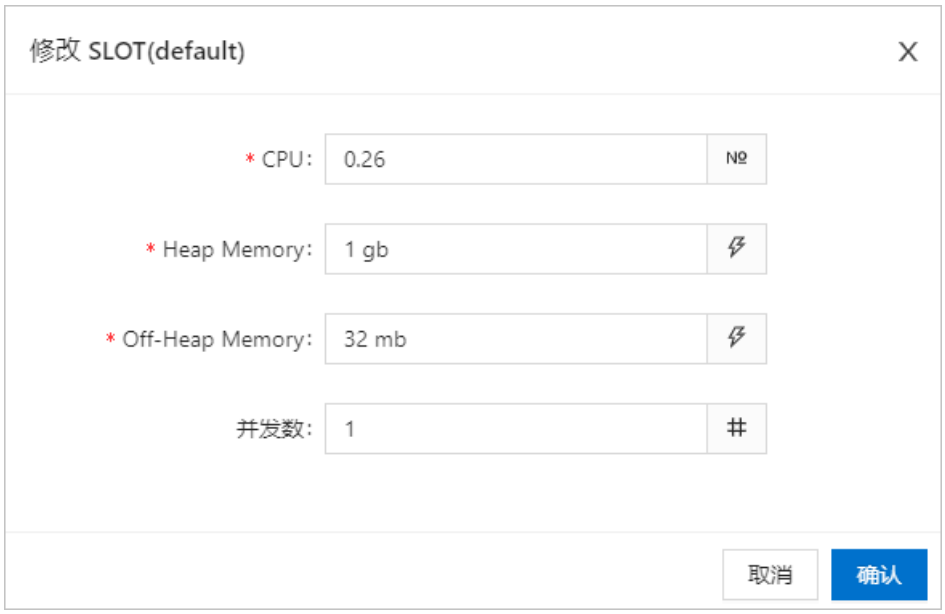
   您可以填写JM对应的CPU和内存。

配置项	说明
Job Manager CPUs	默认值为1。
Job Manager Memory	最小值为1 GB。单位建议使用GB或MB，例如，1024 MB或1.5 GB。

3. 配置SLOT资源。
   i. 单击**立刻获取**。
   ii. 单击SLOT框上的图标。



iii. 修改SLOT配置信息。



说明

- 此处设置的并发数为整个作业全局的并发数。设置完成后，系统将自动设置每个VERTEX为相同的并发数。您也可以单击VERTEX框的图标，按需为某个VERTEX设置并发数。
- 系统会根据作业的计算逻辑按需自动生成Statebackend、Python和Operator所需的内存，无需您手动进行配置。
- 系统默认会按照一个SLOT所需的全部内存大小（即Heap Memory、Off-heap Memory、Statebackend、Python和Operator内存之和），以1:4的比例，生成一个SLOT所需的CPU。

4. 修改资源配置生成策略。

i. 在作业开发页面右侧，**高级配置**面板**更多Flink配置**中，添加如下参数信息。

配置项	说明
table.exec.split-slot-sharing-group-per-vertex	是否为每个VERTEX分配单独的SLOT，默认为false。
table.exec.slot-sharing-group.prefer-heap-memory	一个SLOT默认的堆内存（Heap Memory）的大小，默认值为1 GB。
table.exec.slot-sharing-group.prefer-off-heap-memory	一个SLOT默认的堆外内存（Off-heap Memory）大小，默认值为32 MB。
table.exec.state-backend.prefer-managed-memory	一个State Backend所需的管理内存，默认值为512 MB。
table.exec.resource.default-parallelism	作业的初始并发度，默认值为1。

ii. 在**资源配置**面板中，单击**重新生成资源计划**。

iii. 单击**上线**。

8.2.4. 高性能Flink SQL优化技巧

本文将从作业配置和Flink SQL优化两方面为您介绍如何提升Flink SQL作业性能。

作业配置优化推荐方案

- 资源优化技巧

  VVP中限制了JobManager和TaskManager的CPU的实际使用大小，配置了多少个CPU，最大就只能使用多少个CPU。因此在资源优化时，建议：

  - 作业并发大时：在作业的**高级配置**的**资源配置**中，增加JobManager的资源，提高CPU和内存的大小，例如：
    - Job Manager CPUs设置为4。
    - Job Manager Memory设置为8 GiB。
  - 作业拓扑较复杂时，在作业的**高级配置**的**资源配置**中，增加TaskManager的资源，提高CPU和内存的大小，例如：
    - Task Manager CPUs设置为2。
    - Task Manager Memory设置为4 GiB。
  - 不建议修改taskmanager.numberOfTaskSlots，保持默认值1。

- 提升吞吐和解决数据热点的推荐配置

  在作业**高级配置**的**更多Flink配置**里添加以下代码，详情请参见Group Aggregate优化技巧。

```
execution.checkpointing.interval: 180s
state.backend: com.alibaba.flink.statebackend.GeminiStateBackendFactory
table.exec.state.ttl: 129600000
table.exec.mini-batch.enabled: true
table.exec.mini-batch.allow-latency: 5s
table.optimizer.distinct-agg.split.enabled: true
env.java.opts.taskmanager: -Xms4096m -XX:+UseParNewGC -XX:+UseCMSInitiatingOccupancyOnly
-XX:CMSInitiatingOccupancyFraction=70 -verbose:gc -XX:+HeapDumpOnOutOfMemoryError -XX:+Us
eConcMarkSweepGC -XX:+UseCMSCompactAtFullCollection -XX:CMSMaxAbortablePrecleanTime=1000
-XX:+CMSClassUnloadingEnabled -XX:+PrintGCDetails -XX:+PrintGCDateStamps -XX:SurvivorRati
o=5 -XX:ParallelGCThreads=4
```

参数解释如下表所示。

参数	说明
execution.checkpointing.interval	Checkpoint间隔时间，单位为毫秒。
state.backend	State backend的配置。
table.exec.state.ttl	State数据的生命周期，单位为毫秒。
table.exec.mini-batch.enabled	是否开启minibatch。
table.exec.mini-batch.allow-latency	批量输出数据的时间间隔。
table.optimizer.distinct-agg.split.enabled	是否开启PartialFinal优化，解决COUNT DISTINCT热点问题。
env.java.opts.taskmanager	启动Java选项。 **说明** 当源表为SLS时，不能设置该参数，否则会造成性能瓶颈。

Group Aggregate优化技巧

- 开启MiniBatch（提升吞吐）

  MiniBatch是缓存一定的数据后再触发处理，以减少对State的访问，从而提升吞吐并减少数据的输出量。

  MiniBatch主要基于事件消息来触发微批处理，事件消息会按您指定的时间间隔在源头插入。

  - 适用场景

    微批处理通过增加延迟换取高吞吐，如果您有超低延迟的要求，不建议开启微批处理。通常对于聚合场景，微批处理可以显著地提升系统性能，建议开启。

- 开启方式

  MiniBatch默认关闭，您需要在作业**高级配置**中的**更多Flink配置**填写以下代码。

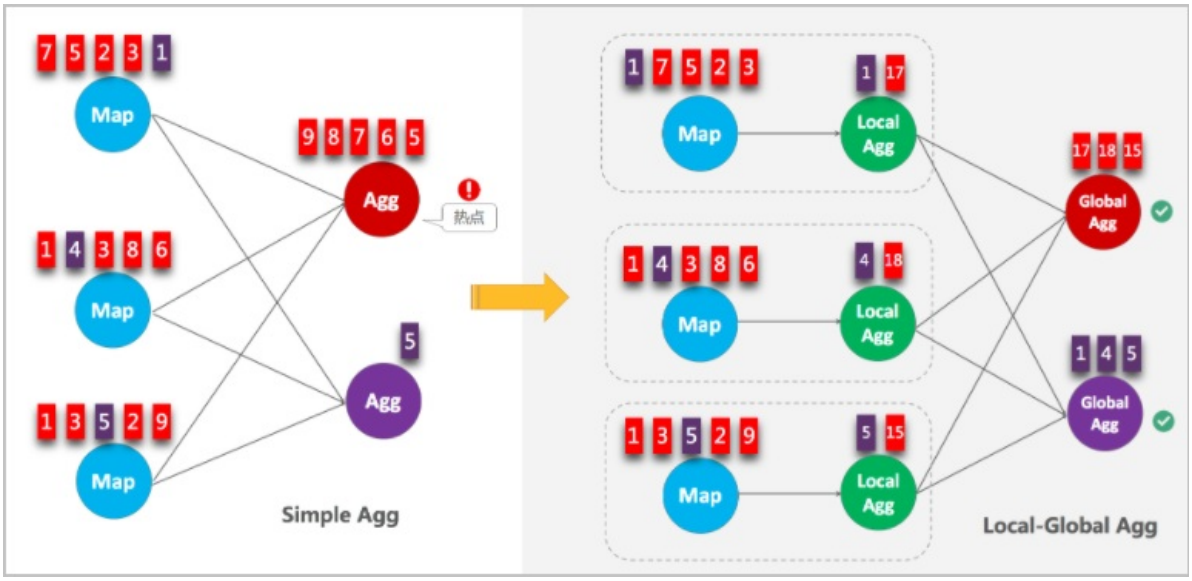
```
table.exec.mini-batch.enabled: true
table.exec.mini-batch.allow-latency: 5s
```

  参数解释如下表所示。

参数	说明
table.exec.mini-batch.enabled	是否开启mini-batch。
table.exec.mini-batch.allow-latency	批量输出数据的时间间隔。

- 开启LocalGlobal（解决常见数据热点问题）

  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。



  LocalGlobal优化将原先的Aggregate分成Local和Global两阶段聚合，即MapReduce模型中的Combine和Reduce两阶段处理模式。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）。

  - 适用场景

    提升普通聚合（例如SUM、COUNT、MAX、MIN和AVG）的性能，以及这些场景下的数据热点问题。

  - 使用限制

    LocalGlobal是默认开启的，但是有以下限制：

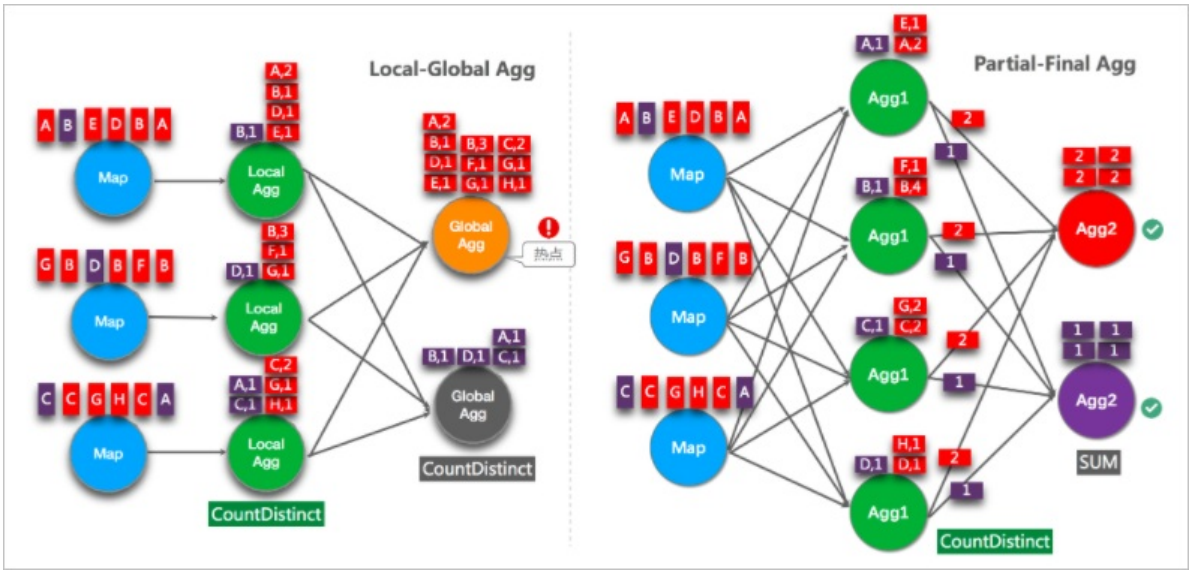
    - 在minibatch开启的前提下才能生效。
    - 需要使用AggregateFunction实现Merge。

  - 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）

  为了解决COUNT DISTINCT的热点问题，通常需要手动改写为两层聚合（增加按Distinct Key取模的打散层）。目前，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。



  LocalGlobal优化针对普通聚合（例如SUM、COUNT、MAX、MIN和AVG）有较好的效果，对于COUNT DISTINCT收效不明显，因为COUNT DISTINCT在Local聚合时，对于DISTINCT KEY的去重率不高，导致在Global节点仍然存在热点问题。

  - 适用场景

    使用COUNT DISTINCT，但无法满足聚合节点性能要求。

    > **说明**
    > - 不能在包含UDAF的Flink SQL中使用PartialFinal优化方法。
    > - 数据量较少的情况，不建议使用PartialFinal优化方法，浪费资源。因为PartialFinal优化会自动打散成两层聚合，引入额外的网络Shuffle。

  - 开启方式

    默认不开启。如果您需要开启，则需要在作业**高级配置**中的**更多Flink配置**填写以下代码。

```
table.optimizer.distinct-agg.split.enabled: true
```

  - 判断是否生效

    观察最终生成的拓扑图，是否由原来一层的聚合变成了两层的聚合。

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

  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能够使性能提升1倍。

  - 适用场景

    对于同一个字段上计算不同条件下的COUNT DISTINCT结果的场景，性能提升很大。

- 原始写法

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

- 优化写法

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

Flink SQL优化推荐方案

- TopN优化技巧
- 高效去重方案
- 高效的内置函数

TopN优化技巧

- TopN算法

  当TopN的输入是非更新流（例如Source），TopN只有1种算法AppendRank。当TopN的输入是更新流时（例如经过了AGG或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。

    使用该算法需要具备2个条件：

    - 输入流有PK（Primary Key）信息，例如ORDER BY AVG。
    - 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。

      如果您要获取到优化Plan，则您需要在使用ORDER BY SUM DESC时，添加SUM为正数的过滤条件，确保total_fee为正数。

```
insert
  into print_test
SELECT
  cate_id,
  seller_id,
  stat_date,
  pay_ord_amt  --不输出rownum字段，能减小结果表的输出量。
FROM (
    SELECT
      *,
      ROW_NUMBER () OVER (
        PARTITION BY 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
      ) as rownum  --根据上游sum结果排序。
    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，因此TopN能使用优化算法，只获取前100个数据。
          sum (total_fee) filter (
            where
              total_fee >= 0
          ) as pay_ord_amt
        FROM
          random_test
        WHERE
          total_fee >= 0
        GROUP
          BY cate_name,
          seller_id,
          stat_date,
          cate_id
      ) a
    ) WHERE
      rownum <= 100;
```

- UnaryUpdateRank：性能仅次于UpdateFastRank的算法。使用该算法需要具备的条件是输入流中存在PK信息。
- 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，则可使用UnaryUpdateRank或UpdateFastRank算法进行优化。

- TopN优化方法
  - 无排名优化

    TopN的输出结果不需要显示rownum值，仅需在最终前端显示时进行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Top-N。

- 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命中率的计算公式如下。

  ```
  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，例如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PartitionBy的Key维度特别大时，可以适当加大TopN的Cache Size，相对应地也建议适当加大TopN节点的Heap Memory。

  ```
  blink.topn.cache.size: 200000
  ```

  默认10000条，调整TopN cache到200000，那么理论命中率能达到200000*50/100/100000 = 100%。

- 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的最终结果会由于State TTL产生错乱。

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，还要灵活地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，必须是一个时间属性的字段（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：

  i. 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。
     - 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。

ii. 对排名进行过滤，只取第一条，达到了去重的目的。

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
```

  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
```

  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

在使用内置函数时，您需要注意以下几点：

- 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，请尽量使用内置函数替换自定义函数。实时计算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：`KEYVALUE(content, keyValueSplit, keySplit, keyName)`，当keyValueSplit和KeySplit是单字符，例如，冒号（:）、逗号（,）时，系统会使用优化算法，在二进制数据上直接寻找所需的keyName值，而不会将整个content进行切分，性能约提升30%。

- 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'` 。
- 如果需要匹配下划线（_），请注意要完成转义 `LIKE '%seller/_id%' ESCAPE '/'` 。下划线（_）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，导致作业阻塞，具体情况请参见Regex execution is too slow，因此建议使用LIKE。正则函数包括：

  - REGEXP
  - REGEXP_REPLACE

8.3. 作业监控

8.3.1. 配置监控告警（在Flink全托管开发控制台配置）

Flink全托管支持配置作业运行的监控告警，作业运行过程中，如果触发了监控规则，系统会给您发送告警消息，便于您及时发现并处理异常。本文为您介绍如何在Flink全托管开发控制台配置监控告警。

前提条件

已开通应用实时监控服务ARMS，详情请参见开通和升级ARMS。

背景信息

您可以在Flink全托管开发控制台，快速配置监控告警，操作简单、易用，效率高。您也可以在ARMS产品中配置监控告警时，需要指定集群、类型、大盘和PromQL等信息，但此方式操作复杂、上手成本高、告警配置效率低，详情请参见配置监控告警（在ARMS控制台中配置）。

创建自定义规则

1. 登录实时计算控制台。
2. 在**Flink全托管**页签，单击目标工作空间**操作**列下的**开发控制台**。
3. 在左侧导航栏上，单击**作业运维**。
4. 单击目标作业名称。
5. 单击**告警配置**页签。
6. 单击**告警规则**页签。
7. 选择**添加告警规则** > **自定义规则**。

   如果您已在Flink开发控制台系统管理中配置了告警模板，您也可以单击告警模板名称，继续后续的步骤，加快配置监控告警的效率。告警模板创建步骤请参见创建告警模板。
8. 填写告警规则信息。

类别	参数	描述
	规则名称	以字母开头，仅包含小写英文字母、数字、下划线（_），长度限制为3~64个字符。
	描述	规则的备注信息。

类别	参数	描述
具体规则	内容	配置触发报警的条件内容，配置完成后会每间隔指定时间，对指定的指标值与阈值进行计算比较，计算结果满足则自动报警。 其中： ○ 指标： ■ restart count in 1 minute：1分钟内Job Manager重启次数。 ■ checkpoint count in 5 minutes：5分钟内Checkpoint成功次数。 ■ Emit Delay：业务延时，即数据发生时间与数据离开Source算子的时间差值，单位为秒。 ■ IN RPS：每秒输入输出记录数。 ■ OUT RPS：每秒输入输出记录数。 ○ 时间差：在指定时间内的指标值。 ○ 运算符：支持>=和<=两种。 ○ 阈值：与指标进行比较的值。
	生效时间	告警监控的生效时间，可以指定只在白天（9点~18点）生效，默认全天生效。
	告警频率	支持1分钟~24小时。
通知方式	通知方式	支持以下通知方式： ○ DingTalk：钉钉 ○ Email：邮件 ○ SMS：短信 ○ Webhook：网络服务地址。 **说明** 您可以在联系人中配置联系人的电话、邮箱和钉钉信息。
	联系人组	您可以添加或者编辑联系人组和联系人。在**编辑联系人组和联系人**对话框，您可以新建Webhook和设置钉钉机器人报警，详情请参见常见问题。

9. 单击**确认**。

   保存后的告警规则默认已启用，且出现在告警规则列表中，您可以选择停止、编辑或删除。

创建告警模板

1. 登录实时计算控制台。
2. 在**Flink全托管**页签，单击目标工作空间**操作**列下的**开发控制台**。
3. 在左侧导航栏上，单击**作业运维**。

4. 单击目标作业名称。
5. 单击**告警配置**页签。
6. 单击**告警规则**页签。
7. 选择**添加告警规则 > 规则模板 > 添加规则模板**。
8. 填写规则信息。

类别	参数	描述
	规则名称	以字母开头，仅包含小写英文字母、数字、下划线（_），长度限制为3~64个字符。
	描述	规则的备注信息。

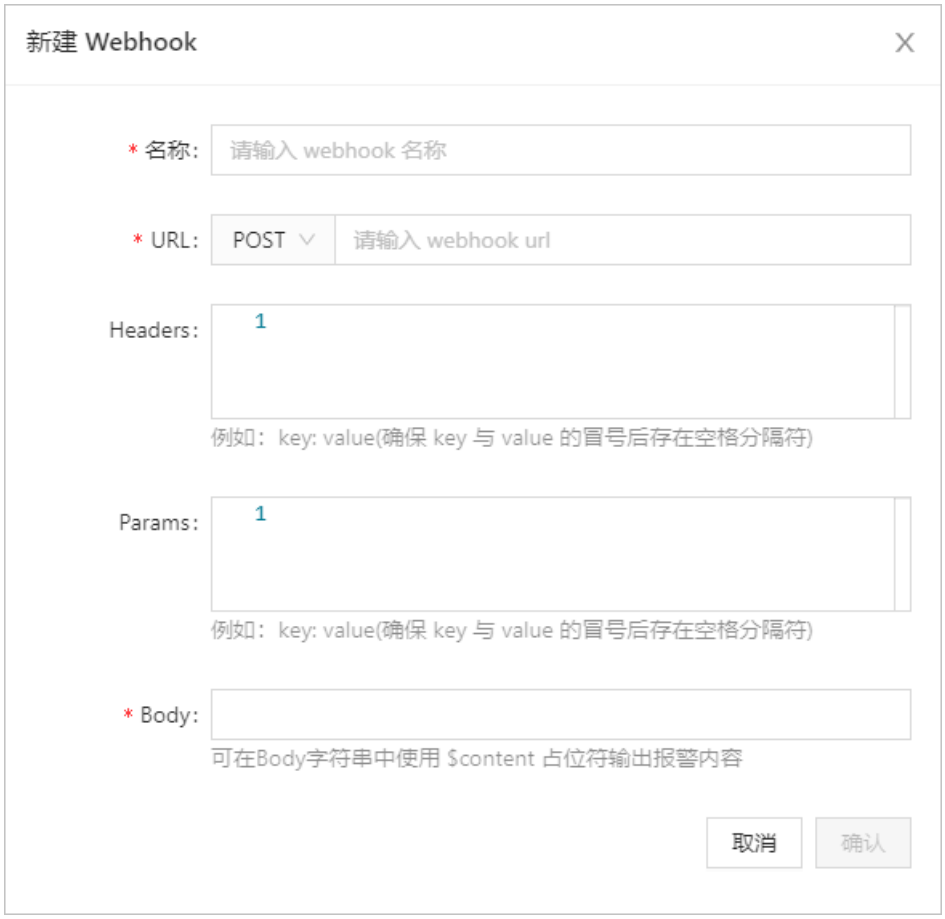
类别	参数	描述
具体规则	内容	配置触发报警的条件内容，配置完成后会每间隔指定时间，对指定的指标值与阈值进行计算比较，计算结果满足则自动报警。 其中： ○ 指标： ■ restart count in 1 minute：1分钟内Job Manager重启次数。 ■ checkpoint count in 5 minutes：5分钟内Checkpoint成功次数。 ■ Emit Delay：业务延时，即数据发生时间与数据离开Source算子的时间差值，单位为秒。 ■ IN RPS：每秒输入输出记录数。 ■ OUT RPS：每秒输入输出记录数。 ○ 时间差：在指定时间内的指标值。 ○ 运算符：支持>=和<=两种。 ○ 阈值：与指标进行比较的值。
	生效时间	告警监控的生效时间，可以指定只在白天（9点~18点）生效，默认全天生效。
	告警频率	支持1分钟~24小时。
通知方式	通知方式	支持以下通知方式： ○ DingTalk：钉钉 ○ Email：邮件 ○ SMS：短信 ○ Webhook：网络服务地址。 **说明** 您可以在联系人中配置联系人的电话、邮箱和钉钉信息。
	联系人组	您可以添加或者编辑联系人组和联系人。在**编辑联系人组和联系人**对话框，您可以新建Webhook和设置钉钉机器人报警，详情请参见常见问题。

9. 单击**确认**。

   创建的告警模板会出现在告警模板列表中，您可以选择编辑或删除。

常见问题

- 如何新建Webhook?
  i. 单击**编辑联系人组和联系人**。
  ii. 在**Webhook**页签，单击**新建Webhook**。
  iii. 在**新建Webhook**文本框，输入Webhook信息。



参数	说明
名称	必填，Webhook名称。
URL	必填，网络服务地址。
Headers	可选，请求头，用来存放cookie和token信息。格式为key: value。 **说明** 请确保key与value的冒号后存在空格分隔符。
Params	可选，请求参数，格式为key: value。 **说明** 请确保key与value的冒号后存在空格分隔符。
Body	必填，请求体，用来存放post的参数和参数数据。 您可以在Body字符串中使用$content占位符输出报警内容。

iv. 单击**确认**。

- 如何设置钉钉机器人报警？
  i. 单击**编辑联系人组和联系人**。
  ii. 在**联系人**页签，单击**新建联系人**。
  iii. 在**钉钉机器人**文本框，输入钉钉机器人的地址。

  您首先需要添加一个自定义钉钉机器人，然后获取他的Webhook地址，详情请参见添加自定义钉钉机器人并获取Webhook地址。

  > **注意** **安全设置**请至少选中**自定义关键词**，且关键词至少设置为**报警**才能收到告警信息。

  iv. 单击**提交**。

8.3.2. 配置监控告警（在ARMS控制台中配置）

Flink全托管支持配置作业运行的监控告警，作业运行过程中，如果触发了监控规则，系统会给您发送告警消息，便于您及时发现并处理异常。本文为您介绍如何在应用实时监控服务ARMS控制台上配置Prometheus监控报警。

前提条件

已开通应用实时监控服务ARMS，详情请参见开通和升级ARMS。

背景信息

在ARMS产品中配置监控告警时，需要指定集群、类型、大盘和PromQL等信息，但此方式操作复杂、上手成本高、告警配置效率低。您可以在Flink全托管开发控制台，快速配置监控告警，操作简单、易用，效率高，详情请参见配置监控告警（在Flink全托管开发控制台配置）。

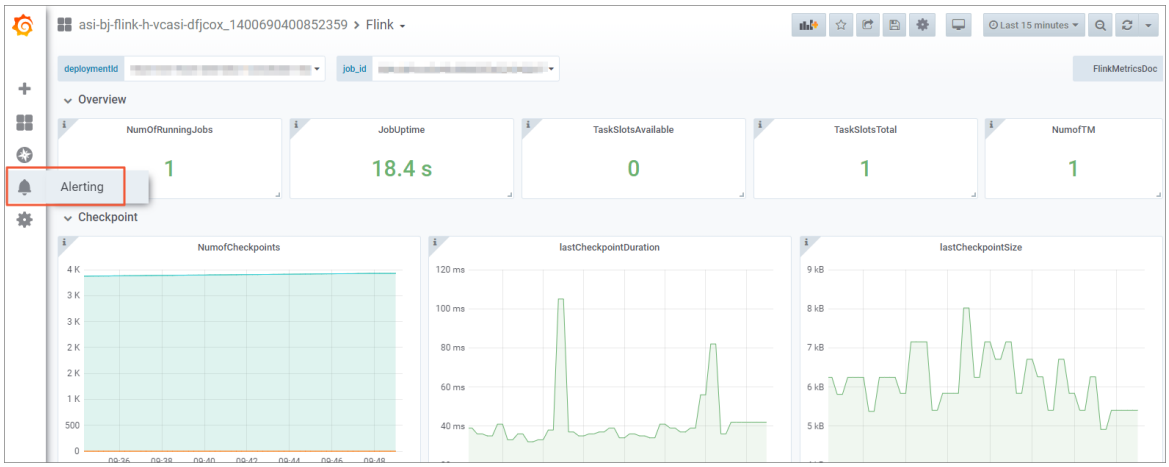
操作步骤

1. 登录实时计算控制台。
2. 在**Flink全托管**页签，单击对应工作空间**操作**列下的**开发控制台**。
3. 在左侧导航栏上，单击**作业运维**。
4. 单击目标作业名称。

   > **说明** 请选择作业**期望状态**和**当前状态**都为**RUNNING**的作业。

5. 在**作业总览**页面右上角，单击**指标**。

6. 在左侧导航栏上，单击**Alerting**。



7. 在Prometheus控制台，创建报警。

   详情请参见创建报警。

8.3.3. 查看监控指标

Flink全托管支持通过监控指标（Metrics）来帮您分析作业数据是否正常。本文为您介绍如何查看监控指标、Flink Source和Sink支持的指标含义，以及各Connector支持上报的Metrics信息情况。

背景信息

除了在Flink全托管控制台上查看监控指标的方式外，您也可以使用以下任意一种方式：

- 使用自建的Prometheus服务监控Flink Metrics。

  但是您需要在网络连通的情况下，先在Flink全托管控制台目标作业开发页面右侧**高级配置**面板**更多Flink配置**中添加如下代码。

```
metrics.reporter.promgatewayappmgr.class: org.apache.flink.metrics.prometheus.PrometheusPushGatewayReporter
metrics.reporter.promgatewayappmgr.host: ${your pushgateway host}
metrics.reporter.promgatewayappmgr.port: ${your pushgateway port}
```

> **说明** 该代码中，${your pushgateway host}和${your pushgateway port}需要替换为您的pushgateway的host和port。

- 通过ARMS API的方式获取Flink全托管的Metrics后，将Metrics集成在您自己的平台上。

  ARMS API详情请参见API概览；Operator相关的Metric详情，请参见Operator Metrics。

注意事项

- Source端的指标表现并不是判断问题根源的充分条件，只反映了Source当前的工作状况，您仍然需要其他辅助指标或工具来判断问题的根源。常见问题的指标表现如下。

场景	详情
作业中某些算子存在反压	判断反压最直接的方式是使用Flink UI提供的反压检查功能，而不是通过指标判断。反压的存在会导致Source端数据发送至下游算子的速率下降，您可能会观察到sourceIdleTime周期性上升，currentFetchEventTime和currentEmitEventTimeLag不断增长。极限情况下（某些算子卡死）sourceIdleTime会持续上升。
Source是性能瓶颈点	如果仅是Source的吞吐量不足，那么您的作业中不应该检测到反压，sourceIdleTime会维持在较低的值（Source一直在工作），currentFetchEventTimeLag和currentEmitEventTimeLag接近，但是两个指标均较高。
上游数据倾斜或某个分区为空	数据倾斜或空分区会导致某一个或者几个Source并行度处于闲置状态，您会观察到这些Source的sourceIdleTime指标较大。

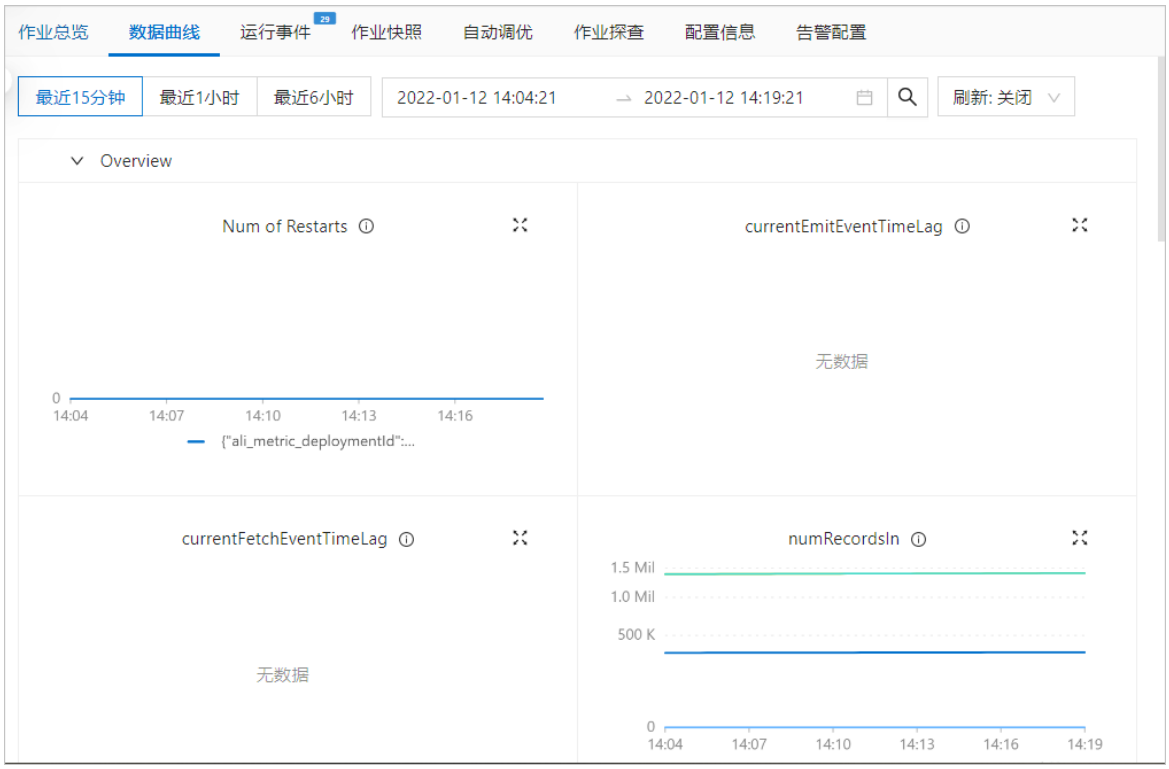
- 如果您发现作业的延迟较高，则可以使用以下指标帮您分析Flink当前的处理能力，以及数据在外部系统中的滞留情况。

指标	详情
sourceIdleTime	该指标反映Source是否有闲置，如果该指标较大，说明您的数据在外部系统中的产生速率较低。
currentFetchEventTimeLag和currentEmitEventTimeLag	均反映了Flink当前处理的延迟，您可以通过两个指标的差值（即数据在Source中停留的时间）分析Source当前的处理能力。 ○ 如果两个延迟非常接近，说明Source从外部系统中拉取数据的能力（网络I/O、并发数）不足。 ○ 如果两个延迟差值较大，说明Source的数据处理能力（数据解析、并发、反压）不足。
pendingBytes和pendingRecords	如果您使用的Connector汇报了pendingBytes和pendingRecords指标，那么您可以通过这两个指标分析数据在外部系统的滞留数据量。

操作步骤

1. 登录实时计算控制台。
2. 在**Flink全托管**页签，单击目标工作空间**操作**列下的**开发控制台**。
3. 在左侧导航栏上，选择**应用 > 作业运维**。
4. 单击目标作业名称。

5. 单击**数据曲线**。



6. 查看具体的数据曲线信息。

   Source和Sink支持的指标和各Connector是否支持上报Metrics信息详情，请参见指标说明。

指标说明

Flink全托管VVR 2.1.3及以上版本支持的Metrics指标相关信息如下表所示。

- Source

名称	类型	单位	描述
numBytesIn	Counter	Bytes	进入Source的总数据量。
numBytesInPerSecond	Meter	Bytes/秒	进入Source的数据速率。
numRecordsIn	Counter	条	进入Source的数据条目总数。
numRecordsInPerSecond	Meter	条/秒	进入Source的数据条目速率。
numRecordsInErrors	Counter	条	Source处理失败的数据条目总数。

名称	类型	单位	描述
currentFetchEventTimeLag	Gauge	毫秒	计算方法：数据被Source读取的时刻（FetchTime）- 数据事件时间（EventTime）。 该指标反映了数据在外部系统的滞留情况。 以Kafka为例，该指标为数据被Kafka Consumer拉取的时刻（FetchTime）与数据写入Kafka的时间（EventTime）差值。
currentEmitEventTimeLag	Gauge	毫秒	计算方法：数据离开Source的时刻（EmitTime）- 数据事件时间（EventTime）。 该指标反映了数据在外部系统和 Source 中的滞留情况。 以Kafka为例，该指标为数据离开Kafka Source时刻（FetchTime）与数据写入Kafka 的时间（EventTime）差值。
watermarkLag	Gauge	毫秒	计算方法：当前时间 - 数据的水印时间（Watermark）。 该指标反映了Watermark的延迟程度。
sourceIdleTime	Gauge	毫秒	计算方法：当前时间 - 最后一次处理数据的时间。 该指标反映了Source的闲置情况。
pendingBytes	Gauge	Bytes	外部系统中尚未被Source拉取的数据量。 **注意** 并不是所有的Source都汇报该指标，需要外部系统和Source的支持。
pendingRecords	Gauge	Records	外部系统中尚未被Source拉取的数据条数。 **注意** 并不是所有的Source都汇报该指标，需要外部系统和Source的支持。

各Connector是否支持上报Metrics信息汇总表

“√”表示支持上报该Metrics，“x”表示暂未支持上报该Metrics。

Connector	numBytesIn	numBytesInPerSecond	numRecordsIn	numRecordsInPerSecond	numRecordsInErrors	currentFetchEventTimeLag	currentEmitEventTimeLag	watermarkLag	sourceIdleTime	pendingBytes	pendingRecords
Kafka	√	√	√	√	x	√	√	√	√	x	√
MaxCompute	√	√	√	√	x	x	x	x	x	x	x
增量MaxCompute	√	√	√	√	x	x	x	x	x	x	x
MQ	√	√	√	√	√	√	√	√	√	x	x
SLS	x	x	√	√	x	√	√	√	x	x	x
DataHub	x	x	√	√	x	√	√	√	x	x	x
ElasticSearch	x	x	√	√	x	x	x	x	x	x	√
Hologres	x	x	√	√	x	x	x	x	x	x	x
云数据库HBase	x	x	x	x	x	x	x	x	x	x	x
Tablestore	x	x	x	x	x	x	x	x	x	x	x
Phoenix	x	x	x	x	x	x	x	x	x	x	x
Redis	x	x	x	x	x	x	x	x	x	x	x

- Sink指标说明

名称	类型	单位	描述
numBytesOut	Counter	Bytes	Sink的输出总数据量。
numBytesOutPerSecond	Meter	Bytes/秒	Sink的数据输出速率。
numRecordsOut	Counter	条	Sink的输出数据条目总数。
numRecordsOutPerSecond	Meter	条/秒	Sink的输出数据条目速率。
numRecordsOutErrors	Counter	条	Sink处理失败的数据条目总数。

名称	类型	单位	描述
currentSendTime	Gauge	毫秒	将最近一条数据输出到外部系统耗费的时间。

各Connector是否支持上报Metrics信息汇总表

“√”表示支持上报该Metrics，“x”表示暂未支持上报该Metrics。

Connector	numBytesOut	numBytesOutPerSecond	numRecordsOut	numRecordsOutPerSecond	numRecordsOutErrors	currentSendTime
Kafka	√	√	√	√	√	√
MaxCompute	x	x	x	√	x	√
增量MaxCompute	x	x	x	√	x	√
MQ	√	√	x	x	x	√
SLS	x	x	√	√	x	√
DataHub	√	√	√	√	x	√
ElasticSearch	x	x	x	x	x	x
Hologres	x	x	√	√	x	√
云数据库HBase	√	√	√	√	x	√
Tablestore	x	x	√	√	x	√
Phoenix	x	x	√	√	x	√
Redis	x	x	√	√	x	√

注意 并不是所有的Source都汇报该指标，需要外部系统和Source的支持。更多指标信息请参见Standardize Connector Metrics。

8.3.4. 查看风险与异常作业

在运维页面的作业总览大盘上，您可以快速查看正常、风险和异常作业的数量和列表，实时地了解作业的健康程度。本文为您介绍如何查看风险和异常作业的数量和列表。

背景信息

关于异常作业和风险作业的定义详情，如下表所示。

类别	定义
异常作业	系统会每1分钟刷新一次作业状态，如果作业状态是Failed，则该作业会被定义为异常作业。
风险作业	系统会每1分钟刷新一次作业状态，如果作业状态为Unstable、Failing或ClusterUnreachable，则该作业会被定义为风险作业。三种状态的含义为： • Unstable：一小时失败三次。 • Failing：十分钟失败三次。 • ClusterUnreachable：存在JM丢失情况。 系统根据风险的严重程度，将风险作业划分为以下三个等级： • 高（High）：作业存在可能引起运行异常或数据准确性的风险。 • 中（Mid）：作业存在性能瓶颈的风险。 • 低（Low）：作业存在资源利用率不高的风险，您可以使用参数进行优化或者减少资源。

注意事项

在过滤出风险和异常作业列表后，您可以单击目标作业名称进入作业详情页面，再单击诊断去查看具体的风险或异常原因。根据系统提供的建议去进行相应的修复或者调优，以使作业恢复正常。作业诊断详情请参见作业诊断。

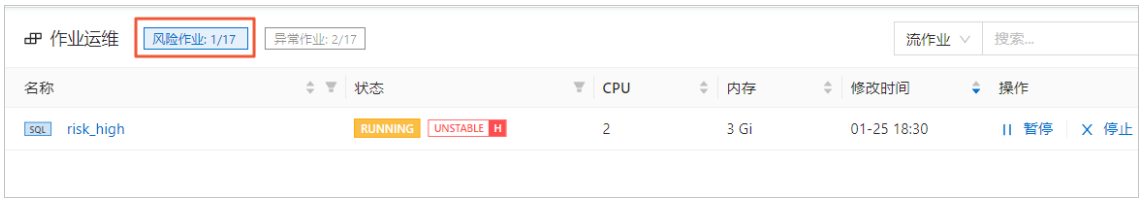
操作步骤

1. 登录实时计算控制台。
2. 在**Flink全托管**页签，单击目标工作空间**操作**列下的**控制台**。
3. 在左侧导航栏上，选择**应用 > 作业运维**。
4. 您可以查看以下信息。
   - 查看风险和异常作业的数量

     在页面顶部，直接查看风险作业和异常作业的数量。

   - 查看风险作业列表

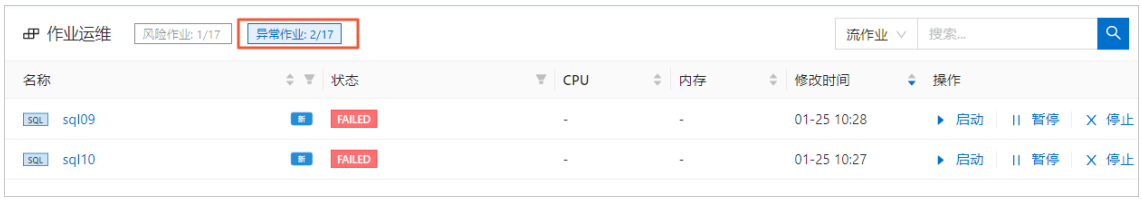
在页面顶部，单击**风险作业**。



> **说明** 如果您需要取消过滤，展示全量作业，则可以再次单击**风险作业**。

- 查看异常作业列表

  在页面顶部，单击**异常作业**。



> **说明** 如果您需要取消过滤，展示全量作业，则可以再次单击**异常作业**。

8.3.5. 作业诊断

如果作业状态为风险或者异常，则您可以通过作业诊断功能，自动诊断作业风险或者异常的可能原因，再根据系统提供的修复建议进行处理，即可使作业快速恢复正常。本文为您介绍如何使用作业诊断功能。

背景信息

Flink全托管支持对针对运行中的作业进行风险或者异常诊断，再根据诊断结果的综合信息给出对应的健康分。您可以根据诊断的健康分判断作业的健康程度，并根据优化建议进行修复，使作业快速恢复正常。Flink全托管支持的诊断项及处理建议详情如下表所示。

诊断阶段	分类	诊断项	风险等级	处理建议
启动阶段	稳定性	检查Session集群是否异常。	中	请您提交工单寻求技术支持协助。
		检查集群剩余资源是否充足。	中	请调小作业资源配置或对集群进行扩容。
运行阶段	稳定性	检查机器硬件是否异常。	低	异常处理中，作业会自动恢复，无需您额外处理。
		检查作业TM内存配置是否合理。	中	请调整相应节点的内存配置。
		检查作业是否开启HA（High Available）。	中	请重新上线作业并手动重启（暂停后再启动）作业。
		检查Checkpoint功能是否异常。	中	请您提交工单寻求技术支持协助。

诊断阶段	分类	诊断项	风险等级	处理建议
		检查作业所在服务器是否负载过高。	中	请手动重启（暂停后再启动）作业，系统将会为其调度至负载较低的服务器。
停止阶段	性能问题	检查作业是否停止异常。	高	请您提交工单寻求技术支持协助。
		检查作业版本是否过低。	中	请暂停作业后，修改作业版本，然后启动作业。

操作步骤

1. 登录实时计算控制台。
2. 在**Flink全托管**页签，单击目标工作空间**操作**列下的**控制台**。
3. 在左侧导航栏上，选择**应用 > 作业运维**。
4. 单击目标作业名称。
5. 在作业详情页面右上角，单击**诊断**。

6. 在左侧，查看诊断结果。

相关文档

快速查看风险或异常作业，请参见查看风险与异常作业。

8.4. 作业日志

8.4.1. 动态修改日志级别

当INFO级别的日志无法满足您定位问题的需求时，您可以修改打印日志的级别为DEBUG。本文为您介绍如何动态修改日志级别及使用限制和注意事项。

背景信息

系统支持的日志级别从低到高的顺序与下面表格内容中的前后顺序一致。

级别	详情
TRACE	比DEBUG更细粒度的信息。
DEBUG	系统运行状态的信息。
INFO	重要或者您感兴趣的信息。
WARN	系统可能会出现的潜在错误信息。

级别	详情
ERROR	系统出现错误和异常的信息。

使用限制

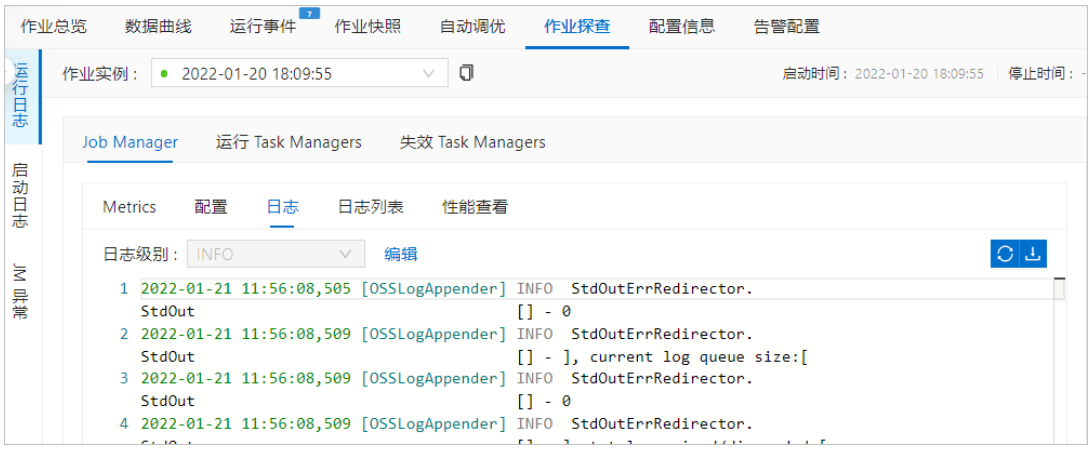
- 只支持设置为INFO或DEBUG，不支持设置为其他级别，以避免级别设置太高导致在日志中定位不到原因的问题。
- 支持修改整个JM的日志级别。
- 支持修改单个TM日志级别，不能一次性修改所有TM日志级别。如果需要修改所有TM日志级别，则需要您进行单独配置。

注意事项

- 日志修改为DEBUG级别后，会打印DEBUG、INFO、WARN、ERROR级别的日志。
- 日志修改为INFO级别后，只会打印INFO、WARN、ERROR级别的日志。
- 修改日志级别后，只会影响新输出的日志，之前已输出的日志不会受到影响。
- DEBUG日志的开启会对作业性能造成少许影响。如果您没有定位问题的必要，建议将日志级别设置为INFO。

操作步骤

1. 登录实时计算控制台。
2. 在**Flink全托管**页签，单击目标工作空间**操作**列下的**开发控制台**。
3. 在左侧导航栏上，选择**应用 > 作业运维**。
4. 单击目标作业名称。
5. 单击**作业探查**。
6. 在左侧**运行日志**页签，选择正在运行的作业实例。
7. 修改Job Manager或Task Managers日志级别。
   - 修改Job Manager日志级别
     a. 在**Job Manager**页签，单击**日志**。



     b. 单击**编辑**。

c. 单击**确定**。

d. 在下拉框中选择目标日志级别。

e. 单击**确认**。

- 修改Task Managers日志级别

a. 在**运行Task Managers**页签，单击**Path, ID**。

b. 单击**日志**。

c. 单击**编辑**。

d. 单击**确定**。

e. 在下拉框中选择目标日志级别。

f. 修改日志级别后，单击**确认**。

8.4.2. 查看运行异常日志

在作业启动或者运行异常时，您可以查看运行异常日志。本文为您介绍如何查看Job Manager异常、失效Task Managers日志和慢Checkpoint对应的Task Managers日志。

背景信息

类型	详情
Job Manager异常	如果作业出现Failover，导致作业运行失败，则需要查看Job Manager异常。关于如何查看Job Manager异常的详情，请参见查看Job Manager异常。 说明 如果Job Manager都没启动起来，不属于Job Manager异常，需要在启动日志中去查看。
失效Task Managers日志	运行状态为正常的作业，不会有失败TM。但是对于风险等级较高的，可能有失败的TM，但TM又被拉起了，这时可以查看失败的TM归档日志去定位可能的原因。关于如何查看失败的TM的日志详情，请参见查看失效Task Managers日志。 说明 只有开启了日志归档才能查看失效Task Managers日志，并且只能查看日志归档中配置的保留天数之内的失效Task Managers日志。
慢Checkpoint对应的Task Managers日志	作业运行过程中，如果有的Checkpoint运行耗时较长，您需要去查看其具体的原因，则可以去查看其关联的TM的日志。关于如何查看慢Checckpoint关联的TM的日志详情，请参见定位慢Checkpoint并查看对应Task Managers的日志。

前提条件

作业实例为运行状态。

注意事项

支持日志分页，每页显示1 MB，大概会显示8000~9000行日志，基本可以帮您排查错误。如果在第一页还未定位到问题，请注意切换日志查看页码。

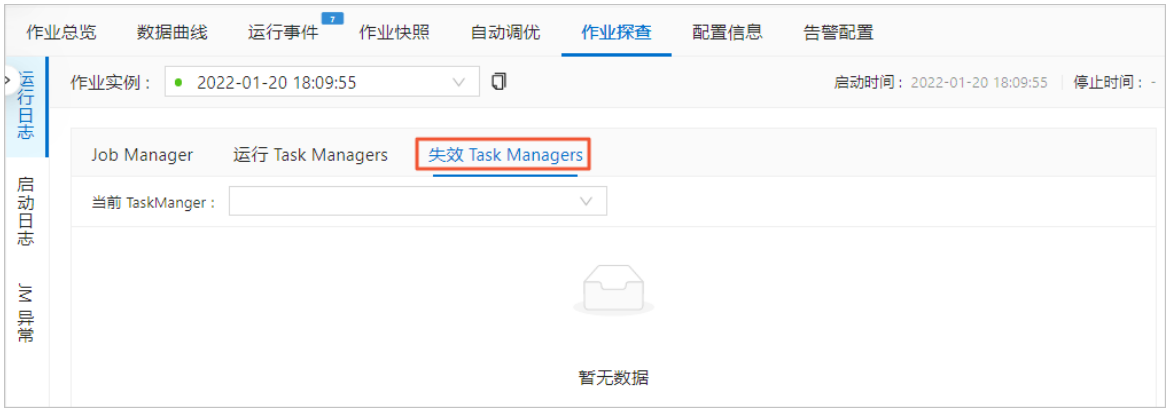
查看Job Manager异常

1. 登录实时计算控制台。
2. 在**Flink全托管**页签，单击目标工作空间**操作**列下的**开发控制台**。
3. 在左侧导航栏上，选择**应用 > 作业运维**。
4. 单击目标作业名称。
5. 单击**作业探查**。
6. 在左侧**JM异常**页签，查看异常信息。

查看失效Task Managers日志

1. 登录实时计算控制台。
2. 在**Flink全托管**页签，单击目标工作空间**操作**列下的**开发控制台**。
3. 在左侧导航栏上，选择**应用 > 作业运维**。
4. 单击目标作业名称。
5. 单击**作业探查**。
6. 在左侧**运行日志**页签，选择正在运行的作业实例。

   左侧绿色小点表示此实例为运行的实例。

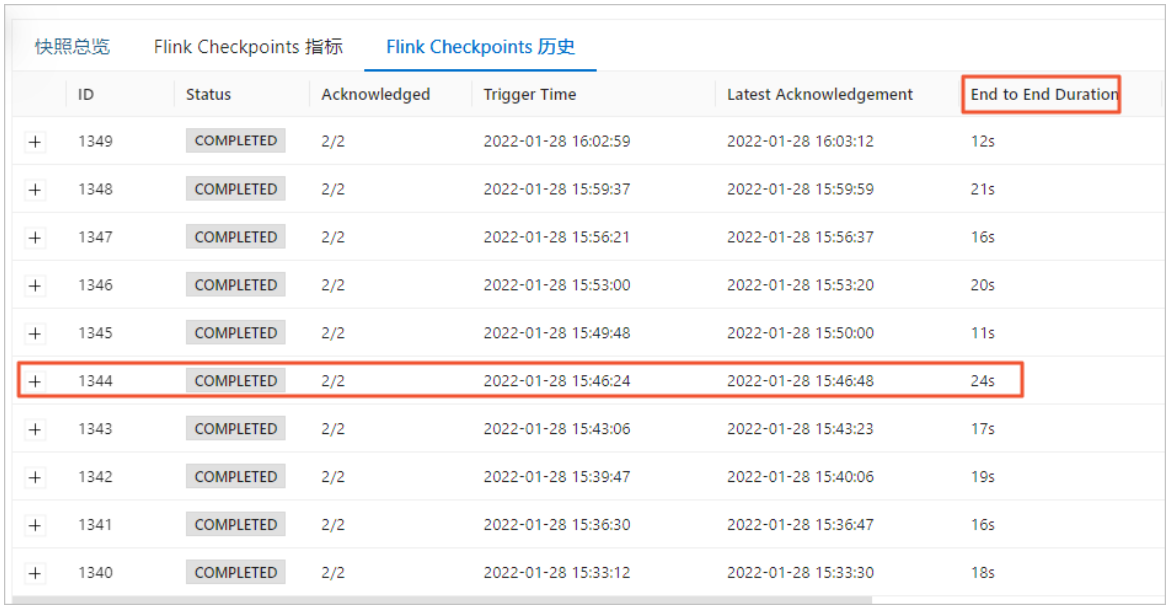


7. 单击**失效Task Managers**页签。

定位慢Checkpoint并查看对应Task Managers的日志

1. 进入作业运维详情页面。
   i. 登录实时计算控制台。
   ii. 在**Flink全托管**页签，单击目标工作空间**操作**列下的**开发控制台**。
   iii. 在左侧导航栏上，选择**应用 > 作业运维**。
   iv. 单击目标作业名称。
2. 在**作业快照**页签，单击**Flink Checkpoints历史**。

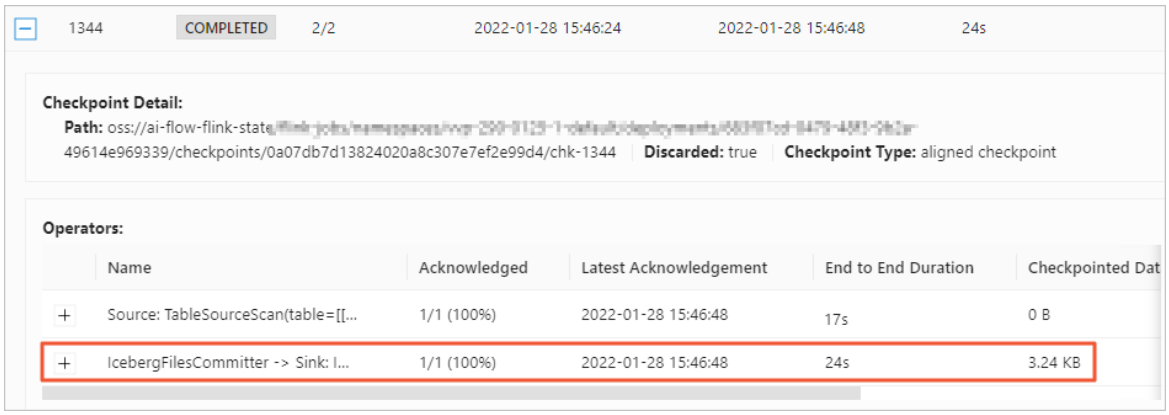
3. 查看**End to End Duration**列值，确认慢Checkpoint对应的ID。



快照总览 Flink Checkpoints 指标 Flink Checkpoints 历史

	ID	Status	Acknowledged	Trigger Time	Latest Acknowledgement	End to End Duration
+	1349	COMPLETED	2/2	2022-01-28 16:02:59	2022-01-28 16:03:12	12s
+	1348	COMPLETED	2/2	2022-01-28 15:59:37	2022-01-28 15:59:59	21s
+	1347	COMPLETED	2/2	2022-01-28 15:56:21	2022-01-28 15:56:37	16s
+	1346	COMPLETED	2/2	2022-01-28 15:53:00	2022-01-28 15:53:20	20s
+	1345	COMPLETED	2/2	2022-01-28 15:49:48	2022-01-28 15:50:00	11s
+	1344	COMPLETED	2/2	2022-01-28 15:46:24	2022-01-28 15:46:48	24s
+	1343	COMPLETED	2/2	2022-01-28 15:43:06	2022-01-28 15:43:23	17s
+	1342	COMPLETED	2/2	2022-01-28 15:39:47	2022-01-28 15:40:06	19s
+	1341	COMPLETED	2/2	2022-01-28 15:36:30	2022-01-28 15:36:47	16s
+	1340	COMPLETED	2/2	2022-01-28 15:33:12	2022-01-28 15:33:30	18s

4. 单击慢Checkpoint对应的ID左侧的 ＋ ，确认慢Checkpoint对应的Operators节点。



1344 COMPLETED 2/2 2022-01-28 15:46:24 2022-01-28 15:46:48 24s

Checkpoint Detail:
Path: oss://ai-flow-flink-state[illegible]
49614e969339/checkpoints/0a07db7d13824020a8c307e7ef2e99d4/chk-1344 **Discarded:** true **Checkpoint Type:** aligned checkpoint

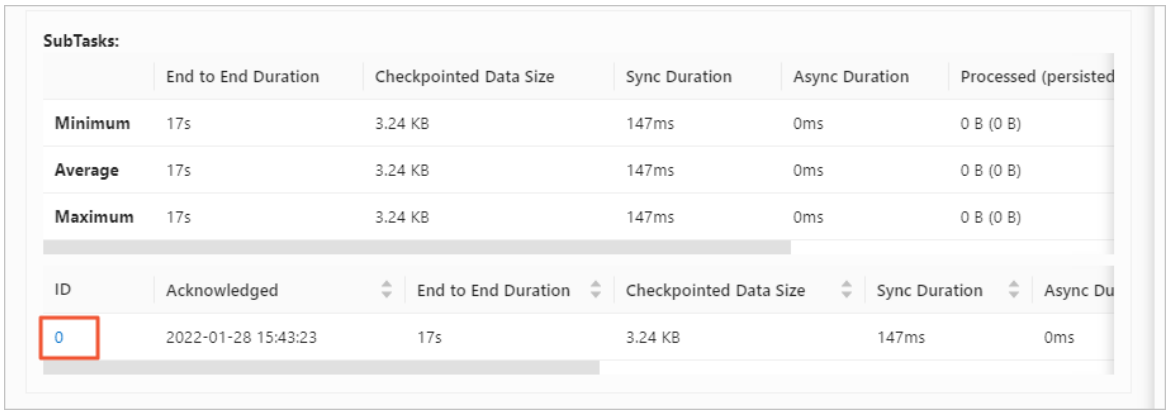
Operators:

	Name	Acknowledged	Latest Acknowledgement	End to End Duration	Checkpointed Dat
+	Source: TableSourceScan(table=[[...	1/1 (100%)	2022-01-28 15:46:48	17s	0 B
+	IcebergFilesCommitter -> Sink: I...	1/1 (100%)	2022-01-28 15:46:48	24s	3.24 KB

5. 单击Operators节点左侧的 ＋ ，展开查看每个Task的情况。

6. 单击慢Checkpoint对应的Task节点对应的**ID**。

   您可以单击**End to End Duration**列右侧的排序按钮，按照从大到小排序**End to End Duration**列的取值，即可快速定位到慢Checkpoint对应的Task节点。



SubTasks:

	End to End Duration	Checkpointed Data Size	Sync Duration	Async Duration	Processed (persisted
Minimum	17s	3.24 KB	147ms	0ms	0 B (0 B)
Average	17s	3.24 KB	147ms	0ms	0 B (0 B)
Maximum	17s	3.24 KB	147ms	0ms	0 B (0 B)

ID	Acknowledged	End to End Duration	Checkpointed Data Size	Sync Duration	Async Du
0	2022-01-28 15:43:23	17s	3.24 KB	147ms	0ms

7. 在跳转的页面，单击**运行Task Managers**页签，查看慢Checkpoint日志信息。

8.4.3. 查看运行事件

作业启动后，会产出运行事件。运行事件可以帮忙查看作业启动的过程中每个时间点的启动状态。本文为您介绍如何查看运行事件。

操作步骤

1. 登录实时计算控制台。
2. 在**Flink全托管**页签，单击目标工作空间**操作**列下的**开发控制台**。
3. 在左侧导航栏上，选择**应用 > 作业运维**。
4. 单击目标作业名称。
5. 单击**运行事件**。

6. 查看运行事件详情。

   运行事件页面上，默认展示事件发生的**时间**、**信息**和**操作**三部分信息，其中具体操作如下：

   - **复制信息**：复制信息内容。
   - **实例详情**：查看作业运行事件的详细信息，包括实例信息、SQL配置、资源配置、Flink配置、日志配置和Logging Level配置等。

     > **说明**
     > - 在**作业总览**页面右上角，单击**Flink UI**，可以跳转到运行作业的开源Flink UI界面。
     > - 单击**指标**，可以跳转到运行作业的VVP（Ververica Platform）的**指标**界面。**指标**界面默认展示的是从事件发生时间起30分钟内的**指标**信息。

   - **搜索该实例事件**：过滤出该实例的事件。

8.4.4. 查看启动和运行日志

在作业为运行状态时，您可以查看其启动日志和运行日志。本文为您介绍如何查看启动日志和运行日志。

背景信息

类型	详情
启动日志	Flink作业启动之后的日志，即从Flink环境被初始化一直到Job Manager被启动起来中间日志。关于如何查看启动日志的详情，请参见查看启动日志。 **说明** 启动日志中不包含调度日志，需要在运行事件中去查看相应的日志信息。例如资源不足、交换机IP不足都属于调度方面的。
运行日志	Flink的Job Manager和Task Managers的日志可以在作业探查页面查看，也可以在Flink UI界面查看。关于如何查看运行日志的详情，请参见查看运行日志。

前提条件

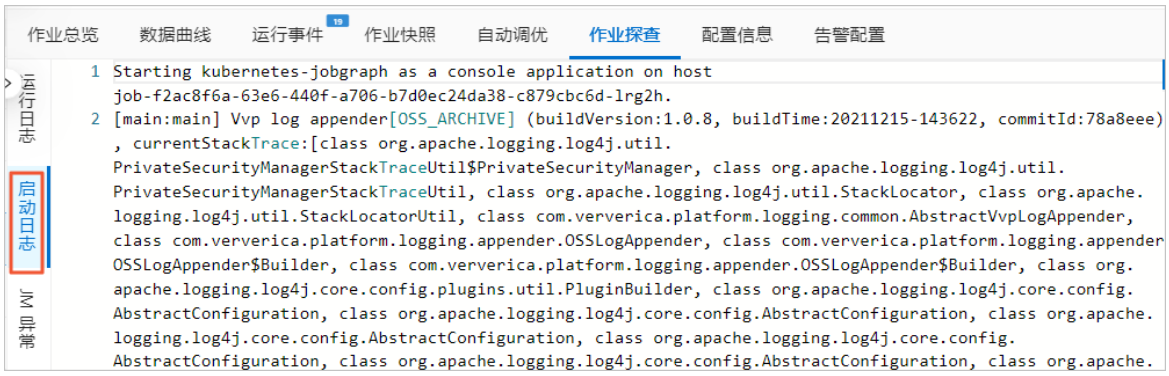
作业实例为运行状态。

注意事项

支持日志分页，每页显示1 MB，大概会显示8000~9000行日志，基本可以帮您排查错误。如果在第一页还未定位到问题，请注意切换日志查看页码。

查看启动日志

1. 登录实时计算控制台。
2. 在**Flink全托管**页签，单击目标工作空间**操作**列下的**开发控制台**。
3. 在左侧导航栏上，选择**应用 > 作业运维**。
4. 单击目标作业名称。
5. 单击**作业探查**。
6. 在左侧**启动日志**页签，查看启动日志。



查看运行日志

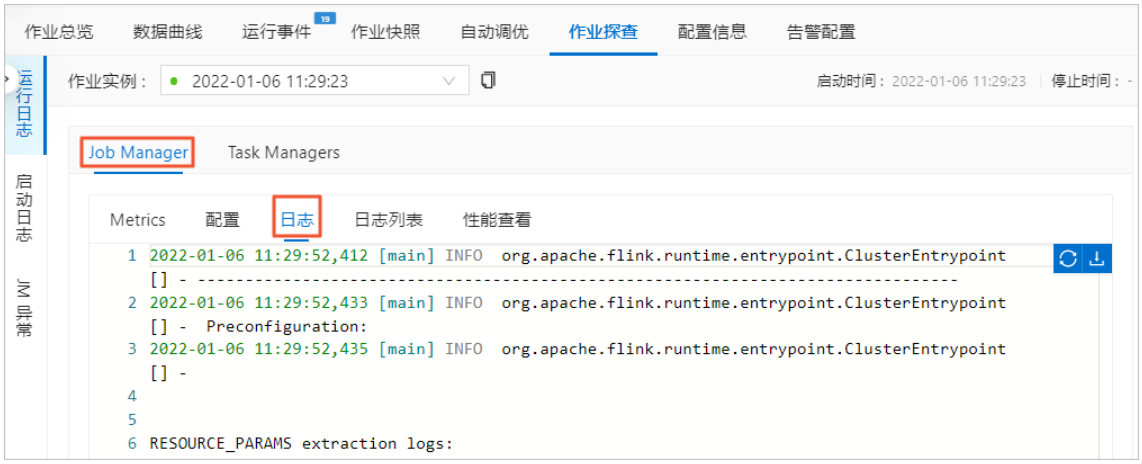
1. 登录实时计算控制台。
2. 在**Flink全托管**页签，单击目标工作空间**操作**列下的**开发控制台**。

3. 在左侧导航栏上，选择**应用 > 作业运维**。
4. 单击目标作业名称。
5. 单击**作业探查**。
6. 在左侧**运行日志**页签，选择正在运行的作业实例。

   左侧绿色小点表示此实例为运行的实例。

7. 查看Job Manager和Task Managers日志。
   - 查看Job Manager日志

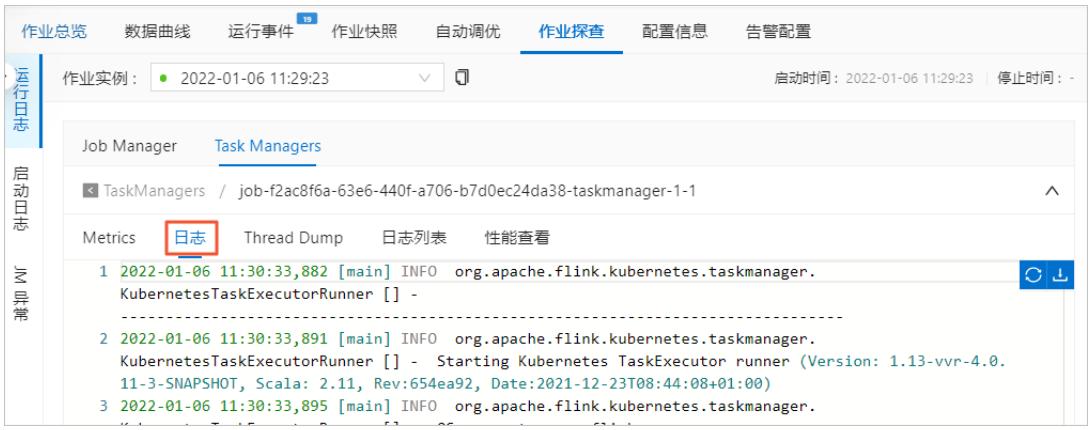
     在**Job Manager**页签，单击**日志**。



   - 查看Task Managers日志

     a. 在**Task Managers**页签，单击**Path, ID**。

b. 单击**日志**。



1. 登录实时计算控制台。
2. 在**Flink全托管**页签，单击目标工作空间**操作**列下的**开发控制台**。
3. 在左侧导航栏上，选择**应用 > 作业运维**。
4. 单击目标作业名称。
5. 在**作业概览**页面，单击**Flink UI**。
6. 在左侧导航栏，单击**Task Manager** 或**Job Manager**。
   - 单击**Task Manager** 后，单击目标**Path, ID**，查看作业Task Manager日志。
   - 单击**Job Manager**后，在**logs**页签，查看作业Job Manager日志。

在作业探查页面查看运行日志

在Flink UI页面查看运行日志

8.4.5. 配置历史作业实例日志输出

除过直接在作业探查页面查看历史作业实例日志外，您还可以将历史作业实例日志配置输出至外部存储（OSS或SLS）后进行查看。本文为您介绍如何配置历史作业实例日志输出至OSS或SLS。

背景信息

在作业实例在运行状态下，您可以在作业探查页面或Flink UI页面查看运行作业实例的日志，详情请参见查看启动和运行日志。

在配置历史作业实例日志输出至OSS或SLS后，您就可以在OSS或SLS上查看历史作业实例日志，详情请参见查看历史作业实例日志。

本文从以下场景为您介绍如何配置历史作业实例日志输出至OSS或SLS：

- 配置单个作业（新建作业）日志输出到OSS
- 配置单个作业（已有作业）日志输出到OSS
- 配置工作空间下所有作业日志输出到OSS
- 配置单个作业（新建作业）日志输出到SLS
- 配置单个作业（已有作业）日志输出到SLS
- 配置工作空间下所有作业日志输出到SLS

注意事项

配置日志输出至OSS或SLS后，需要重启作业。

配置单个作业（新建作业）日志输出到OSS

1. 进入作业开发界面，配置作业信息。
   i. 登录实时计算管理控制台。
   ii. 在**Flink全托管**页签，单击目标工作空间**操作**列下的**控制台**。
   iii. 在左侧导航栏，单击**作业开发**。
   iv. 单击**新建**。
   v. 填写作业信息后，在作业开发页面，填写SQL代码或者上传JAR包。
2. 配置作业日志。
   i. 单击右侧**高级配置**页签。
   ii. 在**日志配置**区域，**Logging Profile**选择为**Custom Template**。
   iii. 将以下文本粘贴到输入框中，并修改以下参数为您OSS Bucket信息。
      - Per-Job模式

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration xmlns="http://logging.apache.org/log4j/2.0/config"
strict="true" packages="com.ververica.platform.logging.appender" status="WARN">
  <Appenders>
    <Appender name="StdOut" type="Console">
      <Layout pattern="%d{yyyy-MM-dd HH:mm:ss,SSS}{GMT+8} %-5p %-60c %x - %m%n" t
ype="PatternLayout" charset="UTF-8"/>
    </Appender>
    <Appender name="RollingFile" type="RollingFile" fileName="${sys:log.file}" fi
lePattern="${sys:log.file}.%i">
      <Layout pattern="%d{yyyy-MM-dd HH:mm:ss,SSS}{GMT+8} %-5p %-60c %x - %m%n" t
ype="PatternLayout" charset="UTF-8"/>
      <Policies>
        <SizeBasedTriggeringPolicy size="20 MB"/>
      </Policies>
      <DefaultRolloverStrategy max="4"/>
    </Appender>
    <Appender name="OSS" type="OSS">
      <Layout pattern="%d{yyyy-MM-dd HH:mm:ss,SSS}{GMT+8} %-5p %-60c %x - %m%n" t
ype="PatternLayout" charset="UTF-8"/>
      <!-- The final effective log path is: ${baseUri}/logs/${namespace}/${deploy
mentId}/{jobId}/ -->
      <Property name="namespace">{{ namespace }}</Property> <!-- Do not modify th
is line -->
      <Property name="baseUri">oss://YOUR-BUCKET-NAME/</Property>
      <Property name="endpoint">https://YOUR-ENDPOINT</Property>
      <Property name="accessKeyId">YOUR-OSS-ACCESSKEYID</Property>
      <Property name="accessKeySecret">YOUR-OSS-ACCESSKEYSECRET</Property>
      <Property name="flushIntervalSeconds">10</Property>
      <Property name="flushIntervalEventCount">100</Property>
      <Property name="rollingBytes">10485760</Property>
    </Appender>
      <Appender name="StdOutErrConsoleAppender" type="Console">
          <Layout pattern="%m" type="PatternLayout" charset="UTF-8"/>
      </Appender>
```

```
      </Appender>
      <Appender name="StdOutFileAppender" type="RollingFile" fileName="${sys:stdo
ut.file}" filePattern="${sys:stdout.file}.%i">
          <Layout pattern="%m" type="PatternLayout" charset="UTF-8"/>
          <Policies>
          <SizeBasedTriggeringPolicy size="1 GB"/>
          </Policies>
          <DefaultRolloverStrategy max="2"/>
      </Appender>
      <Appender name="StdErrFileAppender" type="RollingFile" fileName="${sys:stde
rr.file}" filePattern="${sys:stderr.file}.%i">
          <Layout pattern="%m" type="PatternLayout" charset="UTF-8"/>
          <Policies>
          <SizeBasedTriggeringPolicy size="1 GB"/>
          </Policies>
          <DefaultRolloverStrategy max="2"/>
      </Appender>
  </Appenders>
  <Loggers>
    <Logger level="INFO" name="org.apache.hadoop"/>
    <Logger level="INFO" name="org.apache.kafka"/>
    <Logger level="INFO" name="org.apache.zookeeper"/>
    <Logger level="INFO" name="akka"/>
    <Logger level="ERROR" name="org.jboss.netty.channel.DefaultChannelPipeline"/>

    <Logger level="OFF" name="org.apache.flink.runtime.rest.handler.job.JobDetail
sHandler"/>
    <Logger level="ERROR" name="org.apache.flink.fs.osshadoop.shaded.com.aliyun.o
ss"/>
    <Logger level="INFO" name="StdOutErrRedirector.StdOut" additivity="false">
        <AppenderRef ref="StdOutFileAppender"/>
        <AppenderRef ref="StdOutErrConsoleAppender"/>
    </Logger>
    <Logger level="INFO" name="StdOutErrRedirector.StdErr" additivity="false">
        <AppenderRef ref="StdErrFileAppender"/>
        <AppenderRef ref="StdOutErrConsoleAppender"/>
    </Logger>
    {%- for name, level in userConfiguredLoggers -%}
      <Logger level="{{ level }}" name="{{ name }}"/>
    {%- endfor -%}
    <Root level="{{ rootLoggerLogLevel }}">
      <AppenderRef ref="StdOut"/>
      <AppenderRef ref="RollingFile"/>
      <AppenderRef ref="OSS"/>
    </Root>
  </Loggers>
</Configuration>
```

- Session模式

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration xmlns="http://logging.apache.org/log4j/2.0/config"
strict="true" packages="com.ververica.platform.logging.appender" status="WARN">
  <Appenders>
    <Appender name="StdOut" type="Console">
```

```
      <Layout pattern="%d{yyyy-MM-dd HH:mm:ss,SSS}{GMT+8} %-5p %-60c %x - %m%n" t
ype="PatternLayout" charset="UTF-8"/>
    </Appender>
    <Appender name="RollingFile" type="RollingFile" fileName="${sys:log.file}" fi
lePattern="${sys:log.file}.%i">
      <Layout pattern="%d{yyyy-MM-dd HH:mm:ss,SSS}{GMT+8} %-5p %-60c %x - %m%n" t
ype="PatternLayout" charset="UTF-8"/>
      <Policies>
        <SizeBasedTriggeringPolicy size="20 MB"/>
      </Policies>
      <DefaultRolloverStrategy max="4"/>
    </Appender>
    <Appender name="OSS" type="OSS">
      <Layout pattern="%d{yyyy-MM-dd HH:mm:ss,SSS}{GMT+8} %-5p %-60c %x - %m%n" t
ype="PatternLayout" charset="UTF-8"/>
      <!-- The final effective log path is: ${baseUri}/logs/${namespace}/${deploy
mentId}/{jobId}/ -->
      <Property name="namespace">{{ namespace }}</Property> <!-- Do not modify th
is line -->
      <Property name="baseUri">oss://YOUR-BUCKET-NAME/</Property>
      <Property name="endpoint">https://YOUR-ENDPOINT</Property>
      <Property name="accessKeyId">YOUR-OSS-ACCESSKEYID</Property>
      <Property name="accessKeySecret">YOUR-OSS-ACCESSKEYSECRET</Property>
      <Property name="flushIntervalSeconds">10</Property>
      <Property name="flushIntervalEventCount">100</Property>
      <Property name="rollingBytes">10485760</Property>
      <Property name="deploymentId">{{ sessionClusterName }}</Property>
      <Property name="jobId">{{ sessionClusterId }}</Property>
    </Appender>
      <Appender name="StdOutErrConsoleAppender" type="Console">
          <Layout pattern="%m" type="PatternLayout" charset="UTF-8"/>
      </Appender>
      <Appender name="StdOutFileAppender" type="RollingFile" fileName="${sys:stdo
ut.file}" filePattern="${sys:stdout.file}.%i">
          <Layout pattern="%m" type="PatternLayout" charset="UTF-8"/>
          <Policies>
          <SizeBasedTriggeringPolicy size="1 GB"/>
          </Policies>
          <DefaultRolloverStrategy max="2"/>
      </Appender>
      <Appender name="StdErrFileAppender" type="RollingFile" fileName="${sys:stde
rr.file}" filePattern="${sys:stderr.file}.%i">
          <Layout pattern="%m" type="PatternLayout" charset="UTF-8"/>
          <Policies>
          <SizeBasedTriggeringPolicy size="1 GB"/>
          </Policies>
          <DefaultRolloverStrategy max="2"/>
      </Appender>
  </Appenders>
  <Loggers>
    <Logger level="INFO" name="org.apache.hadoop"/>
    <Logger level="INFO" name="org.apache.kafka"/>
    <Logger level="INFO" name="org.apache.zookeeper"/>
    <Logger level="INFO" name="akka"/>
```

```
    <Logger level="ERROR" name="org.jboss.netty.channel.DefaultChannelPipeline"/>

    <Logger level="OFF" name="org.apache.flink.runtime.rest.handler.job.JobDetail
sHandler"/>
    <Logger level="ERROR" name="org.apache.flink.fs.osshadoop.shaded.com.aliyun.o
ss"/>
    <Logger level="INFO" name="StdOutErrRedirector.StdOut" additivity="false">
        <AppenderRef ref="StdOutFileAppender"/>
        <AppenderRef ref="StdOutErrConsoleAppender"/>
    </Logger>
    <Logger level="INFO" name="StdOutErrRedirector.StdErr" additivity="false">
        <AppenderRef ref="StdErrFileAppender"/>
        <AppenderRef ref="StdOutErrConsoleAppender"/>
    </Logger>
    {%- for name, level in userConfiguredLoggers -%}
      <Logger level="{{ level }}" name="{{ name }}"/>
    {%- endfor -%}
    <Root level="{{ rootLoggerLogLevel }}">
      <AppenderRef ref="StdOut"/>
      <AppenderRef ref="RollingFile"/>
      <AppenderRef ref="OSS"/>
    </Root>
  </Loggers>
</Configuration>
```

参数	说明
YOUR-BUCKET-NAME	替换成您OSS Bucket名称。
YOUR-ENDPOINT	替换成您OSS的Endpoint，详情请参见访问域名和数据中心。 Endpoint为**ECS的VPC网络访问（内网）** 所在行的**Endpoint（地域节点）**信息。
YOUR-OSS-ACCESSKEYID	替换成您配置OSS服务账号的**AccessKey ID**，获取方法请参见获取AccessKey。 **说明** 如果您配置与全托管集群同账号下的OSS，可以不填写该参数，不填写需要删除该参数。如果您配置与全托管集群不同账号下的OSS，该参数必填。
YOUR-OSS-ACCESSKEYSECRET	替换成您配置OSS服务账号的**AccessKey Secret**，获取方法请参见获取AccessKey。 **说明** 如果您配置与全托管集群同账号下的OSS，可以不填写该参数，不填写需要删除该参数。如果您配置与全托管集群不同账号下的OSS，该参数必填。

参数	说明
sessionClusterName	Session集群的名称。
sessionClusterId	Session集群的ID。

3. 启动作业。
   i. 在作业编辑页面右上角，单击**验证**。
      请编写完整的SQL逻辑后再进行验证，否则验证会失败。
   ii. 单击**上线**。
   iii. 在左侧导航栏上，单击**作业运维**。
   iv. 单击目标作业名称**操作**列的**启动**。
      您可以在OSS控制台查看作业日志，详情请参见在OSS上查看。

配置单个作业（已有作业）日志输出到OSS

1. 进入作业运维界面。
   i. 登录实时计算管理控制台。
   ii. 在**Flink全托管**页签，单击目标工作空间**操作**列下的**控制台**。
   iii. 在左侧导航栏上，单击**作业运维**。
2. 配置作业日志信息。
   i. 单击目标作业名称。
   ii. 在作业详情页面右上角，单击**编辑**。
   iii. 单击右侧**高级配置**页签。
   iv. 在**日志配置**区域，Logging Profile选择为Custom Template。
   v. 将以下文本粘贴到输入框中，并修改以下参数为您OSS Bucket信息。
      - Per-Job模式

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration xmlns="http://logging.apache.org/log4j/2.0/config"
strict="true" packages="com.ververica.platform.logging.appender" status="WARN">
  <Appenders>
    <Appender name="StdOut" type="Console">
      <Layout pattern="%d{yyyy-MM-dd HH:mm:ss,SSS}{GMT+8} %-5p %-60c %x - %m%n" t
ype="PatternLayout" charset="UTF-8"/>
    </Appender>
    <Appender name="RollingFile" type="RollingFile" fileName="${sys:log.file}" fi
lePattern="${sys:log.file}.%i">
      <Layout pattern="%d{yyyy-MM-dd HH:mm:ss,SSS}{GMT+8} %-5p %-60c %x - %m%n" t
ype="PatternLayout" charset="UTF-8"/>
      <Policies>
        <SizeBasedTriggeringPolicy size="20 MB"/>
      </Policies>
      <DefaultRolloverStrategy max="4"/>
    </Appender>
    <Appender name="OSS" type="OSS">
      <Layout pattern="%d{yyyy-MM-dd HH:mm:ss,SSS}{GMT+8} %-5p %-60c %x - %m%n" t
```

```
ype="PatternLayout" charset="UTF-8"/>
      <!-- The final effective log path is: ${baseUri}/logs/${namespace}/${deploy
mentId}/{jobId}/ -->
      <Property name="namespace">{{ namespace }}</Property> <!-- Do not modify th
is line -->
      <Property name="baseUri">oss://YOUR-BUCKET-NAME/</Property>
      <Property name="endpoint">https://YOUR-ENDPOINT</Property>
      <Property name="accessKeyId">YOUR-OSS-ACCESSKEYID</Property>
      <Property name="accessKeySecret">YOUR-OSS-ACCESSKEYSECRET</Property>
      <Property name="flushIntervalSeconds">10</Property>
      <Property name="flushIntervalEventCount">100</Property>
      <Property name="rollingBytes">10485760</Property>
    </Appender>
      <Appender name="StdOutErrConsoleAppender" type="Console">
          <Layout pattern="%m" type="PatternLayout" charset="UTF-8"/>
      </Appender>
      <Appender name="StdOutFileAppender" type="RollingFile" fileName="${sys:stdo
ut.file}" filePattern="${sys:stdout.file}.%i">
          <Layout pattern="%m" type="PatternLayout" charset="UTF-8"/>
          <Policies>
          <SizeBasedTriggeringPolicy size="1 GB"/>
          </Policies>
          <DefaultRolloverStrategy max="2"/>
      </Appender>
      <Appender name="StdErrFileAppender" type="RollingFile" fileName="${sys:stde
rr.file}" filePattern="${sys:stderr.file}.%i">
          <Layout pattern="%m" type="PatternLayout" charset="UTF-8"/>
          <Policies>
          <SizeBasedTriggeringPolicy size="1 GB"/>
          </Policies>
          <DefaultRolloverStrategy max="2"/>
      </Appender>
  </Appenders>
  <Loggers>
    <Logger level="INFO" name="org.apache.hadoop"/>
    <Logger level="INFO" name="org.apache.kafka"/>
    <Logger level="INFO" name="org.apache.zookeeper"/>
    <Logger level="INFO" name="akka"/>
    <Logger level="ERROR" name="org.jboss.netty.channel.DefaultChannelPipeline"/>

    <Logger level="OFF" name="org.apache.flink.runtime.rest.handler.job.JobDetail
sHandler"/>
    <Logger level="ERROR" name="org.apache.flink.fs.osshadoop.shaded.com.aliyun.o
ss"/>
    <Logger level="INFO" name="StdOutErrRedirector.StdOut" additivity="false">
        <AppenderRef ref="StdOutFileAppender"/>
        <AppenderRef ref="StdOutErrConsoleAppender"/>
    </Logger>
    <Logger level="INFO" name="StdOutErrRedirector.StdErr" additivity="false">
        <AppenderRef ref="StdErrFileAppender"/>
        <AppenderRef ref="StdOutErrConsoleAppender"/>
    </Logger>
    {%- for name, level in userConfiguredLoggers -%}
      <Logger level="{{ level }}" name="{{ name }}"/>
    {%- endfor -%}
```

```
    {%- endfor -%}
    <Root level="{{ rootLoggerLogLevel }}">
      <AppenderRef ref="StdOut"/>
      <AppenderRef ref="RollingFile"/>
      <AppenderRef ref="OSS"/>
    </Root>
  </Loggers>
</Configuration>
```

- Session模式

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration xmlns="http://logging.apache.org/log4j/2.0/config"
strict="true" packages="com.ververica.platform.logging.appender" status="WARN">
  <Appenders>
    <Appender name="StdOut" type="Console">
      <Layout pattern="%d{yyyy-MM-dd HH:mm:ss,SSS}{GMT+8} %-5p %-60c %x - %m%n" t
ype="PatternLayout" charset="UTF-8"/>
    </Appender>
    <Appender name="RollingFile" type="RollingFile" fileName="${sys:log.file}" fi
lePattern="${sys:log.file}.%i">
      <Layout pattern="%d{yyyy-MM-dd HH:mm:ss,SSS}{GMT+8} %-5p %-60c %x - %m%n" t
ype="PatternLayout" charset="UTF-8"/>
      <Policies>
        <SizeBasedTriggeringPolicy size="20 MB"/>
      </Policies>
      <DefaultRolloverStrategy max="4"/>
    </Appender>
    <Appender name="OSS" type="OSS">
      <Layout pattern="%d{yyyy-MM-dd HH:mm:ss,SSS}{GMT+8} %-5p %-60c %x - %m%n" t
ype="PatternLayout" charset="UTF-8"/>
      <!-- The final effective log path is: ${baseUri}/logs/${namespace}/${deploy
mentId}/{jobId}/ -->
      <Property name="namespace">{{ namespace }}</Property> <!-- Do not modify th
is line -->
      <Property name="baseUri">oss://YOUR-BUCKET-NAME/</Property>
      <Property name="endpoint">https://YOUR-ENDPOINT</Property>
      <Property name="accessKeyId">YOUR-OSS-ACCESSKEYID</Property>
      <Property name="accessKeySecret">YOUR-OSS-ACCESSKEYSECRET</Property>
      <Property name="flushIntervalSeconds">10</Property>
      <Property name="flushIntervalEventCount">100</Property>
      <Property name="rollingBytes">10485760</Property>
      <Property name="deploymentId">{{ sessionClusterName }}</Property>
      <Property name="jobId">{{ sessionClusterId }}</Property>
    </Appender>
      <Appender name="StdOutErrConsoleAppender" type="Console">
          <Layout pattern="%m" type="PatternLayout" charset="UTF-8"/>
      </Appender>
      <Appender name="StdOutFileAppender" type="RollingFile" fileName="${sys:stdo
ut.file}" filePattern="${sys:stdout.file}.%i">
          <Layout pattern="%m" type="PatternLayout" charset="UTF-8"/>
          <Policies>
          <SizeBasedTriggeringPolicy size="1 GB"/>
          </Policies>
          <DefaultRolloverStrategy max="2"/>
```

```
      </Appender>
      <Appender name="StdErrFileAppender" type="RollingFile" fileName="${sys:stde
rr.file}" filePattern="${sys:stderr.file}.%i">
          <Layout pattern="%m" type="PatternLayout" charset="UTF-8"/>
          <Policies>
          <SizeBasedTriggeringPolicy size="1 GB"/>
          </Policies>
          <DefaultRolloverStrategy max="2"/>
      </Appender>
  </Appenders>
  <Loggers>
    <Logger level="INFO" name="org.apache.hadoop"/>
    <Logger level="INFO" name="org.apache.kafka"/>
    <Logger level="INFO" name="org.apache.zookeeper"/>
    <Logger level="INFO" name="akka"/>
    <Logger level="ERROR" name="org.jboss.netty.channel.DefaultChannelPipeline"/>

    <Logger level="OFF" name="org.apache.flink.runtime.rest.handler.job.JobDetail
sHandler"/>
    <Logger level="ERROR" name="org.apache.flink.fs.osshadoop.shaded.com.aliyun.o
ss"/>
    <Logger level="INFO" name="StdOutErrRedirector.StdOut" additivity="false">
        <AppenderRef ref="StdOutFileAppender"/>
        <AppenderRef ref="StdOutErrConsoleAppender"/>
    </Logger>
    <Logger level="INFO" name="StdOutErrRedirector.StdErr" additivity="false">
        <AppenderRef ref="StdErrFileAppender"/>
        <AppenderRef ref="StdOutErrConsoleAppender"/>
    </Logger>
    {%- for name, level in userConfiguredLoggers -%}
      <Logger level="{{ level }}" name="{{ name }}"/>
    {%- endfor -%}
    <Root level="{{ rootLoggerLogLevel }}">
      <AppenderRef ref="StdOut"/>
      <AppenderRef ref="RollingFile"/>
      <AppenderRef ref="OSS"/>
    </Root>
  </Loggers>
</Configuration>
```

参数	说明
YOUR-BUCKET-NAME	替换成您OSS Bucket名称。
YOUR-ENDPOINT	替换成您OSS的Endpoint，详情请参见访问域名和数据中心。 Endpoint为**ECS的VPC网络访问（内网）** 所在行的**Endpoint（地域节点）**信息。

参数	说明
YOUR-OSS-ACCESSKEYID	替换成您配置OSS服务账号的**AccessKey ID**，获取方法请参见获取AccessKey。 **说明** 如果您配置与全托管集群同账号下的OSS，可以不填写该参数，不填写需要删除该参数。如果您配置与全托管集群不同账号下的OSS，该参数必填。
YOUR-OSS-ACCESSKEYSECRET	替换成您配置OSS服务账号的**AccessKey Secret**，获取方法请参见获取AccessKey。 **说明** 如果您配置与全托管集群同账号下的OSS，可以不填写该参数，不填写需要删除该参数。如果您配置与全托管集群不同账号下的OSS，该参数必填。
sessionClusterName	Session集群的名称。
sessionClusterId	Session集群的ID。

vi. 单击**保存**。

3. 在左侧导航栏上，单击**作业运维**。
4. 单击目标作业名称**操作**列的**启动**。

   您可以在OSS控制台查看作业日志，详情请参见在OSS上查看。

配置工作空间下所有作业日志输出到OSS

您可以通过配置**作业模板**的方式，配置工作空间下所有作业日志默认输出到OSS。

1. 登录实时计算管理控制台。
2. 在**Flink全托管**页签，单击目标工作空间**操作**列下的**控制台**。
3. 配置作业日志。
   i. 在左侧导航栏，选择**系统管理 > 作业模板**。
   ii. 在**日志配置**区域，**Logging Profile**选择为**Custom Template**。
   iii. 将以下文本粘贴到输入框中，并修改以下参数为您OSS Bucket信息。
   - Per-Job模式

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration xmlns="http://logging.apache.org/log4j/2.0/config"
strict="true" packages="com.ververica.platform.logging.appender" status="WARN">
  <Appenders>
    <Appender name="StdOut" type="Console">
      <Layout pattern="%d{yyyy-MM-dd HH:mm:ss,SSS}{GMT+8} %-5p %-60c %x - %m%n" t
ype="PatternLayout" charset="UTF-8"/>
    </Appender>
    <Appender name="RollingFile" type="RollingFile" fileName="${sys:log.file}" fi
```

```
lePattern="${sys:log.file}.%i">
      <Layout pattern="%d{yyyy-MM-dd HH:mm:ss,SSS}{GMT+8} %-5p %-60c %x - %m%n" t
ype="PatternLayout" charset="UTF-8"/>
      <Policies>
        <SizeBasedTriggeringPolicy size="20 MB"/>
      </Policies>
      <DefaultRolloverStrategy max="4"/>
    </Appender>
    <Appender name="OSS" type="OSS">
      <Layout pattern="%d{yyyy-MM-dd HH:mm:ss,SSS}{GMT+8} %-5p %-60c %x - %m%n" t
ype="PatternLayout" charset="UTF-8"/>
      <!-- The final effective log path is: ${baseUri}/logs/${namespace}/${deploy
mentId}/{jobId}/ -->
      <Property name="namespace">{{ namespace }}</Property> <!-- Do not modify th
is line -->
      <Property name="baseUri">oss://YOUR-BUCKET-NAME/</Property>
      <Property name="endpoint">https://YOUR-ENDPOINT</Property>
      <Property name="accessKeyId">YOUR-OSS-ACCESSKEYID</Property>
      <Property name="accessKeySecret">YOUR-OSS-ACCESSKEYSECRET</Property>
      <Property name="flushIntervalSeconds">10</Property>
      <Property name="flushIntervalEventCount">100</Property>
      <Property name="rollingBytes">10485760</Property>
    </Appender>
      <Appender name="StdOutErrConsoleAppender" type="Console">
          <Layout pattern="%m" type="PatternLayout" charset="UTF-8"/>
      </Appender>
      <Appender name="StdOutFileAppender" type="RollingFile" fileName="${sys:stdo
ut.file}" filePattern="${sys:stdout.file}.%i">
          <Layout pattern="%m" type="PatternLayout" charset="UTF-8"/>
          <Policies>
          <SizeBasedTriggeringPolicy size="1 GB"/>
          </Policies>
          <DefaultRolloverStrategy max="2"/>
      </Appender>
      <Appender name="StdErrFileAppender" type="RollingFile" fileName="${sys:stde
rr.file}" filePattern="${sys:stderr.file}.%i">
          <Layout pattern="%m" type="PatternLayout" charset="UTF-8"/>
          <Policies>
          <SizeBasedTriggeringPolicy size="1 GB"/>
          </Policies>
          <DefaultRolloverStrategy max="2"/>
      </Appender>
  </Appenders>
  <Loggers>
    <Logger level="INFO" name="org.apache.hadoop"/>
    <Logger level="INFO" name="org.apache.kafka"/>
    <Logger level="INFO" name="org.apache.zookeeper"/>
    <Logger level="INFO" name="akka"/>
    <Logger level="ERROR" name="org.jboss.netty.channel.DefaultChannelPipeline"/>

    <Logger level="OFF" name="org.apache.flink.runtime.rest.handler.job.JobDetail
sHandler"/>
    <Logger level="ERROR" name="org.apache.flink.fs.osshadoop.shaded.com.aliyun.o
ss"/>
    <Logger level="INFO" name="StdOutErrRedirector.StdOut" additivity="false">
```

```
    <Logger level="INFO" name="StdOutErrRedirector.StdOut" additivity="false">
        <AppenderRef ref="StdOutFileAppender"/>
        <AppenderRef ref="StdOutErrConsoleAppender"/>
    </Logger>
    <Logger level="INFO" name="StdOutErrRedirector.StdErr" additivity="false">
        <AppenderRef ref="StdErrFileAppender"/>
        <AppenderRef ref="StdOutErrConsoleAppender"/>
    </Logger>
    {%- for name, level in userConfiguredLoggers -%}
      <Logger level="{{ level }}" name="{{ name }}"/>
    {%- endfor -%}
    <Root level="{{ rootLoggerLogLevel }}">
      <AppenderRef ref="StdOut"/>
      <AppenderRef ref="RollingFile"/>
      <AppenderRef ref="OSS"/>
    </Root>
  </Loggers>
</Configuration>
```

- Session模式

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration xmlns="http://logging.apache.org/log4j/2.0/config"
strict="true" packages="com.ververica.platform.logging.appender" status="WARN">
  <Appenders>
    <Appender name="StdOut" type="Console">
      <Layout pattern="%d{yyyy-MM-dd HH:mm:ss,SSS}{GMT+8} %-5p %-60c %x - %m%n" t
ype="PatternLayout" charset="UTF-8"/>
    </Appender>
    <Appender name="RollingFile" type="RollingFile" fileName="${sys:log.file}" fi
lePattern="${sys:log.file}.%i">
      <Layout pattern="%d{yyyy-MM-dd HH:mm:ss,SSS}{GMT+8} %-5p %-60c %x - %m%n" t
ype="PatternLayout" charset="UTF-8"/>
      <Policies>
        <SizeBasedTriggeringPolicy size="20 MB"/>
      </Policies>
      <DefaultRolloverStrategy max="4"/>
    </Appender>
    <Appender name="OSS" type="OSS">
      <Layout pattern="%d{yyyy-MM-dd HH:mm:ss,SSS}{GMT+8} %-5p %-60c %x - %m%n" t
ype="PatternLayout" charset="UTF-8"/>
      <!-- The final effective log path is: ${baseUri}/logs/${namespace}/${deploy
mentId}/{jobId}/ -->
      <Property name="namespace">{{ namespace }}</Property> <!-- Do not modify th
is line -->
      <Property name="baseUri">oss://YOUR-BUCKET-NAME/</Property>
      <Property name="endpoint">https://YOUR-ENDPOINT</Property>
      <Property name="accessKeyId">YOUR-OSS-ACCESSKEYID</Property>
      <Property name="accessKeySecret">YOUR-OSS-ACCESSKEYSECRET</Property>
      <Property name="flushIntervalSeconds">10</Property>
      <Property name="flushIntervalEventCount">100</Property>
      <Property name="rollingBytes">10485760</Property>
      <Property name="deploymentId">{{ sessionClusterName }}</Property>
      <Property name="jobId">{{ sessionClusterId }}</Property>
    </Appender>
```

```
      <Appender name="StdOutErrConsoleAppender" type="Console">
          <Layout pattern="%m" type="PatternLayout" charset="UTF-8"/>
      </Appender>
      <Appender name="StdOutFileAppender" type="RollingFile" fileName="${sys:stdo
ut.file}" filePattern="${sys:stdout.file}.%i">
          <Layout pattern="%m" type="PatternLayout" charset="UTF-8"/>
          <Policies>
          <SizeBasedTriggeringPolicy size="1 GB"/>
          </Policies>
          <DefaultRolloverStrategy max="2"/>
      </Appender>
      <Appender name="StdErrFileAppender" type="RollingFile" fileName="${sys:stde
rr.file}" filePattern="${sys:stderr.file}.%i">
          <Layout pattern="%m" type="PatternLayout" charset="UTF-8"/>
          <Policies>
          <SizeBasedTriggeringPolicy size="1 GB"/>
          </Policies>
          <DefaultRolloverStrategy max="2"/>
      </Appender>
  </Appenders>
  <Loggers>
    <Logger level="INFO" name="org.apache.hadoop"/>
    <Logger level="INFO" name="org.apache.kafka"/>
    <Logger level="INFO" name="org.apache.zookeeper"/>
    <Logger level="INFO" name="akka"/>
    <Logger level="ERROR" name="org.jboss.netty.channel.DefaultChannelPipeline"/>

    <Logger level="OFF" name="org.apache.flink.runtime.rest.handler.job.JobDetail
sHandler"/>
    <Logger level="ERROR" name="org.apache.flink.fs.osshadoop.shaded.com.aliyun.o
ss"/>
    <Logger level="INFO" name="StdOutErrRedirector.StdOut" additivity="false">
        <AppenderRef ref="StdOutFileAppender"/>
        <AppenderRef ref="StdOutErrConsoleAppender"/>
    </Logger>
    <Logger level="INFO" name="StdOutErrRedirector.StdErr" additivity="false">
        <AppenderRef ref="StdErrFileAppender"/>
        <AppenderRef ref="StdOutErrConsoleAppender"/>
    </Logger>
    {%- for name, level in userConfiguredLoggers -%}
      <Logger level="{{ level }}" name="{{ name }}"/>
    {%- endfor -%}
    <Root level="{{ rootLoggerLogLevel }}">
      <AppenderRef ref="StdOut"/>
      <AppenderRef ref="RollingFile"/>
      <AppenderRef ref="OSS"/>
    </Root>
  </Loggers>
</Configuration>
```

参数	说明
YOUR-BUCKET-NAME	替换成您OSS Bucket名称。

参数	说明
YOUR-ENDPOINT	替换成您OSS的Endpoint，详情请参见访问域名和数据中心。 Endpoint为**ECS的VPC网络访问（内网）** 所在行的**Endpoint（地域节点）**信息。
YOUR-OSS-ACCESSKEYID	替换成您配置OSS服务账号的**AccessKey ID**，获取方法请参见获取AccessKey。 **说明** 如果您配置与全托管集群同账号下的OSS，可以不填写该参数，不填写需要删除该参数。如果您配置与全托管集群不同账号下的OSS，该参数必填。
YOUR-OSS-ACCESSKEYSECRET	替换成您配置OSS服务账号的**AccessKey Secret**，获取方法请参见获取AccessKey。 **说明** 如果您配置与全托管集群同账号下的OSS，可以不填写该参数，不填写需要删除该参数。如果您配置与全托管集群不同账号下的OSS，该参数必填。
sessionClusterName	Session集群的名称。
sessionClusterId	Session集群的ID。

iv. 单击**保存**。

> **说明** 配置**作业模板**后，后续该工作空间下创建的所有作业的日志都会被存储到OSS。您可以在OSS控制台查看作业日志，详情请参见在OSS上查看。

配置单个作业（新建作业）日志输出到SLS

1. 进入作业开发界面，配置作业信息。
   i. 登录实时计算管理控制台。
   ii. 在**Flink全托管**页签，单击目标工作空间**操作**列下的**控制台**。
   iii. 在左侧导航栏，单击**作业开发**。
   iv. 单击**新建**。
   v. 填写作业信息后，在作业开发页面，填写SQL代码或者上传JAR包。
2. 配置作业日志。
   i. 单击右侧**高级配置**页签。
   ii. 在**日志配置**区域，**Logging Profile**选择为**Custom Template**。
   iii. 将以下文本粘贴到**自定义模板**的输入框中，并修改以下参数为您SLS信息。
      - Per-Job模式

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration xmlns="http://logging.apache.org/log4j/2.0/config"
strict="true" packages="com.ververica.platform.logging.appender" status="WARN">
  <Appenders>
    <Appender name="StdOut" type="Console">
      <Layout pattern="%d{yyyy-MM-dd HH:mm:ss,SSS}{GMT+8} %-5p %-60c %x - %m%n" t
ype="PatternLayout" charset="UTF-8"/>
    </Appender>
    <Appender name="RollingFile" type="RollingFile" fileName="${sys:log.file}" fi
lePattern="${sys:log.file}.%i">
      <Layout pattern="%d{yyyy-MM-dd HH:mm:ss,SSS}{GMT+8} %-5p %-60c %x - %m%n" t
ype="PatternLayout" charset="UTF-8"/>
      <Policies>
        <SizeBasedTriggeringPolicy size="5 MB"/>
      </Policies>
      <DefaultRolloverStrategy max="1"/>
    </Appender>
    <Appender name="SLS" type="SLS">
      <Layout pattern="%d{yyyy-MM-dd HH:mm:ss,SSS}{GMT+8} %-5p %-60c %x - %m%n" t
ype="PatternLayout" charset="UTF-8"/>
      <!-- The final effective log path is: ${baseUri}/logs/${namespace}/${deploy
mentId}/{jobId}/ -->
      <Property name="namespace">{{ namespace }}</Property> <!-- Do not modify th
is line -->
      <Property name="project">YOUR-SLS-PROJECT</Property>
      <Property name="logStore">YOUR-SLS-LOGSTORE</Property>
      <Property name="endpoint">YOUR-SLS-ENDPOINT</Property>
      <Property name="accessKeyId">YOUR-SLS-ACCESSKEYID</Property>
      <Property name="accessKeySecret">YOUR-SLS-ACCESSKEYSECRET</Property>
      <Property name="topic">{{ namespace }}:{{ deploymentId }}:{{ jobId }}</Prop
erty>
      <Property name="deploymentName">{{ deploymentName }}</Property>
      <Property name="flushIntervalSeconds">10</Property>
      <Property name="flushIntervalEventCount">100</Property>
    </Appender>
      <Appender name="StdOutErrConsoleAppender" type="Console">
          <Layout pattern="%m" type="PatternLayout" charset="UTF-8"/>
      </Appender>
      <Appender name="StdOutFileAppender" type="RollingFile" fileName="${sys:stdo
ut.file}" filePattern="${sys:stdout.file}.%i">
          <Layout pattern="%m" type="PatternLayout" charset="UTF-8"/>
          <Policies>
          <SizeBasedTriggeringPolicy size="1 GB"/>
          </Policies>
          <DefaultRolloverStrategy max="2"/>
      </Appender>
      <Appender name="StdErrFileAppender" type="RollingFile" fileName="${sys:stde
rr.file}" filePattern="${sys:stderr.file}.%i">
          <Layout pattern="%m" type="PatternLayout" charset="UTF-8"/>
          <Policies>
          <SizeBasedTriggeringPolicy size="1 GB"/>
          </Policies>
          <DefaultRolloverStrategy max="2"/>
      </Appender>
```

```
  </Appenders>
  <Loggers>
    <Logger level="INFO" name="org.apache.hadoop"/>
    <Logger level="INFO" name="org.apache.kafka"/>
    <Logger level="INFO" name="org.apache.zookeeper"/>
    <Logger level="INFO" name="akka"/>
    <Logger level="ERROR" name="org.jboss.netty.channel.DefaultChannelPipeline"/>

    <Logger level="OFF" name="org.apache.flink.runtime.rest.handler.job.JobDetail
sHandler"/>
    <Logger level="ERROR" name="org.apache.flink.fs.osshadoop.shaded.com.aliyun.o
ss"/>
    <Logger level="INFO" name="StdOutErrRedirector.StdOut" additivity="false">
        <AppenderRef ref="StdOutFileAppender"/>
        <AppenderRef ref="StdOutErrConsoleAppender"/>
    </Logger>
    <Logger level="INFO" name="StdOutErrRedirector.StdErr" additivity="false">
        <AppenderRef ref="StdErrFileAppender"/>
        <AppenderRef ref="StdOutErrConsoleAppender"/>
    </Logger>
    {%- for name, level in userConfiguredLoggers -%}
      <Logger level="{{ level }}" name="{{ name }}"/>
    {%- endfor -%}
    <Root level="{{ rootLoggerLogLevel }}">
      <AppenderRef ref="StdOut"/>
      <AppenderRef ref="RollingFile"/>
      <AppenderRef ref="SLS"/>
    </Root>
  </Loggers>
</Configuration>
```

- Session模式

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration xmlns="http://logging.apache.org/log4j/2.0/config"
strict="true" packages="com.ververica.platform.logging.appender" status="WARN">
  <Appenders>
    <Appender name="StdOut" type="Console">
      <Layout pattern="%d{yyyy-MM-dd HH:mm:ss,SSS}{GMT+8} %-5p %-60c %x - %m%n" t
ype="PatternLayout" charset="UTF-8"/>
    </Appender>
    <Appender name="RollingFile" type="RollingFile" fileName="${sys:log.file}" fi
lePattern="${sys:log.file}.%i">
      <Layout pattern="%d{yyyy-MM-dd HH:mm:ss,SSS}{GMT+8} %-5p %-60c %x - %m%n" t
ype="PatternLayout" charset="UTF-8"/>
      <Policies>
        <SizeBasedTriggeringPolicy size="5 MB"/>
      </Policies>
      <DefaultRolloverStrategy max="1"/>
    </Appender>
    <Appender name="SLS" type="SLS">
      <Layout pattern="%d{yyyy-MM-dd HH:mm:ss,SSS}{GMT+8} %-5p %-60c %x - %m%n" t
ype="PatternLayout" charset="UTF-8"/>
      <!-- The final effective log path is: ${baseUri}/logs/${namespace}/${deploy
mentId}/${jobId}/ -->
```

```
mentId}/{jobId}/ -->
      <Property name="namespace">{{ namespace }}</Property> <!-- Do not modify th
is line -->
      <Property name="project">YOUR-SLS-PROJECT</Property>
      <Property name="logStore">YOUR-SLS-LOGSTORE</Property>
      <Property name="endpoint">YOUR-SLS-ENDPOINT</Property>
      <Property name="accessKeyId">YOUR-SLS-ACCESSKEYID</Property>
      <Property name="accessKeySecret">YOUR-SLS-ACCESSKEYSECRET</Property>
      <Property name="topic">{{ namespace }}:{{ sessionClusterName }}:{{ sessionC
lusterId }}</Property>
      <Property name="flushIntervalSeconds">10</Property>
      <Property name="flushIntervalEventCount">100</Property>
      <Property name="deploymentId">{{ sessionClusterName }}</Property>
      <Property name="jobId">{{ sessionClusterId }}</Property>
    </Appender>
      <Appender name="StdOutErrConsoleAppender" type="Console">
          <Layout pattern="%m" type="PatternLayout" charset="UTF-8"/>
      </Appender>
      <Appender name="StdOutFileAppender" type="RollingFile" fileName="${sys:stdo
ut.file}" filePattern="${sys:stdout.file}.%i">
          <Layout pattern="%m" type="PatternLayout" charset="UTF-8"/>
          <Policies>
          <SizeBasedTriggeringPolicy size="1 GB"/>
          </Policies>
          <DefaultRolloverStrategy max="2"/>
      </Appender>
      <Appender name="StdErrFileAppender" type="RollingFile" fileName="${sys:stde
rr.file}" filePattern="${sys:stderr.file}.%i">
          <Layout pattern="%m" type="PatternLayout" charset="UTF-8"/>
          <Policies>
          <SizeBasedTriggeringPolicy size="1 GB"/>
          </Policies>
          <DefaultRolloverStrategy max="2"/>
      </Appender>
  </Appenders>
  <Loggers>
    <Logger level="INFO" name="org.apache.hadoop"/>
    <Logger level="INFO" name="org.apache.kafka"/>
    <Logger level="INFO" name="org.apache.zookeeper"/>
    <Logger level="INFO" name="akka"/>
    <Logger level="ERROR" name="org.jboss.netty.channel.DefaultChannelPipeline"/>

    <Logger level="OFF" name="org.apache.flink.runtime.rest.handler.job.JobDetail
sHandler"/>
    <Logger level="ERROR" name="org.apache.flink.fs.osshadoop.shaded.com.aliyun.o
ss"/>
    <Logger level="INFO" name="StdOutErrRedirector.StdOut" additivity="false">
        <AppenderRef ref="StdOutFileAppender"/>
        <AppenderRef ref="StdOutErrConsoleAppender"/>
    </Logger>
    <Logger level="INFO" name="StdOutErrRedirector.StdErr" additivity="false">
        <AppenderRef ref="StdErrFileAppender"/>
        <AppenderRef ref="StdOutErrConsoleAppender"/>
    </Logger>
    {%- for name, level in userConfiguredLoggers -%}
```

```
    <Logger level="{{ level }}" name="{{ name }}"/>
  {%- endfor -%}
  <Root level="{{ rootLoggerLogLevel }}">
    <AppenderRef ref="StdOut"/>
    <AppenderRef ref="RollingFile"/>
    <AppenderRef ref="SLS"/>
  </Root>
 </Loggers>
</Configuration>
```

文件夹	说明
YOUR-SLS-PROJECT	替换成您SLS的Project名称。
YOUR-SLS-LOGSTORE	替换成您SLS的Logstore名称。
YOUR-SLS-ENDPOINT	替换成您SLS所在地域的Endpoint，详情请参见服务入口。
YOUR-SLS-ACCESSKEYID	替换成您配置SLS服务账号的**AccessKey ID**，获取方法请参见获取AccessKey。 **说明** 如果您配置与全托管集群同账号下的SLS，可以不填写该参数，不填写需要删除该参数。如果您配置与全托管集群不同账号下的SLS，该参数必填。
YOUR-SLS-ACCESSKEYSECRET	替换成您配置SLS服务账号的**AccessKey Secret**，获取方法请参见获取AccessKey。 **说明** 如果您配置与全托管集群同账号下的SLS，可以不填写该参数，不填写需要删除该参数。如果您配置与全托管集群不同账号下的SLS，该参数必填。
sessionClusterName	Session集群的名称。
sessionClusterId	Session集群的ID。

说明

- 请确保您所使用的AccessKey ID和AccessKey Secret是您开通**Flink全托管**所使用账号下管理的密钥对。
- 请确保您开通**Flink全托管**服务所使用的账号，有读写该SLS项目的权限。

3. 启动作业。
   i. 在作业编辑页面右上角，单击**验证**。
   ii. 单击**上线**。
   iii. 在左侧导航栏上，单击**作业运维**。

iv. 单击目标作业名称**操作**列的**启动**。

您可以在SLS控制台查看作业日志，详情请参见在SLS上查看。

配置单个作业（已有作业）日志输出到SLS

1. 进入作业运维界面。
   i. 登录实时计算管理控制台。
   ii. 在**Flink全托管**页签，单击目标工作空间**操作**列下的**控制台**。
   iii. 在左侧导航栏上，单击**作业运维**。
2. 配置作业日志。
   i. 单击目标作业名称。
   ii. 在作业详情页面右上角，单击**编辑**。
   iii. 单击**高级配置**。
   iv. 在**日志配置**区域，Logging Profile选择为Custom Template。
   v. 将以下文本粘贴到**自定义模板**的输入框中，并修改以下参数为您SLS信息。
      - Per-Job模式

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration xmlns="http://logging.apache.org/log4j/2.0/config"
strict="true" packages="com.ververica.platform.logging.appender" status="WARN">
  <Appenders>
    <Appender name="StdOut" type="Console">
      <Layout pattern="%d{yyyy-MM-dd HH:mm:ss,SSS}{GMT+8} %-5p %-60c %x - %m%n" t
ype="PatternLayout" charset="UTF-8"/>
    </Appender>
    <Appender name="RollingFile" type="RollingFile" fileName="${sys:log.file}" fi
lePattern="${sys:log.file}.%i">
      <Layout pattern="%d{yyyy-MM-dd HH:mm:ss,SSS}{GMT+8} %-5p %-60c %x - %m%n" t
ype="PatternLayout" charset="UTF-8"/>
      <Policies>
        <SizeBasedTriggeringPolicy size="5 MB"/>
      </Policies>
      <DefaultRolloverStrategy max="1"/>
    </Appender>
    <Appender name="SLS" type="SLS">
      <Layout pattern="%d{yyyy-MM-dd HH:mm:ss,SSS}{GMT+8} %-5p %-60c %x - %m%n" t
ype="PatternLayout" charset="UTF-8"/>
      <!-- The final effective log path is: ${baseUri}/logs/${namespace}/${deploy
mentId}/{jobId}/ -->
      <Property name="namespace">{{ namespace }}</Property> <!-- Do not modify th
is line -->
      <Property name="project">YOUR-SLS-PROJECT</Property>
      <Property name="logStore">YOUR-SLS-LOGSTORE</Property>
      <Property name="endpoint">YOUR-SLS-ENDPOINT</Property>
      <Property name="accessKeyId">YOUR-SLS-ACCESSKEYID</Property>
      <Property name="accessKeySecret">YOUR-SLS-ACCESSKEYSECRET</Property>
      <Property name="topic">{{ namespace }}:{{ deploymentId }}:{{ jobId }}</Prop
erty>
      <Property name="deploymentName">{{ deploymentName }}</Property>
      <Property name="flushIntervalSeconds">10</Property>
```

```
      <Property name="flushIntervalEventCount">100</Property>
    </Appender>
      <Appender name="StdOutErrConsoleAppender" type="Console">
          <Layout pattern="%m" type="PatternLayout" charset="UTF-8"/>
      </Appender>
      <Appender name="StdOutFileAppender" type="RollingFile" fileName="${sys:stdo
ut.file}" filePattern="${sys:stdout.file}.%i">
          <Layout pattern="%m" type="PatternLayout" charset="UTF-8"/>
          <Policies>
          <SizeBasedTriggeringPolicy size="1 GB"/>
          </Policies>
          <DefaultRolloverStrategy max="2"/>
      </Appender>
      <Appender name="StdErrFileAppender" type="RollingFile" fileName="${sys:stde
rr.file}" filePattern="${sys:stderr.file}.%i">
          <Layout pattern="%m" type="PatternLayout" charset="UTF-8"/>
          <Policies>
          <SizeBasedTriggeringPolicy size="1 GB"/>
          </Policies>
          <DefaultRolloverStrategy max="2"/>
      </Appender>
  </Appenders>
  <Loggers>
    <Logger level="INFO" name="org.apache.hadoop"/>
    <Logger level="INFO" name="org.apache.kafka"/>
    <Logger level="INFO" name="org.apache.zookeeper"/>
    <Logger level="INFO" name="akka"/>
    <Logger level="ERROR" name="org.jboss.netty.channel.DefaultChannelPipeline"/>

    <Logger level="OFF" name="org.apache.flink.runtime.rest.handler.job.JobDetail
sHandler"/>
    <Logger level="ERROR" name="org.apache.flink.fs.osshadoop.shaded.com.aliyun.o
ss"/>
    <Logger level="INFO" name="StdOutErrRedirector.StdOut" additivity="false">
        <AppenderRef ref="StdOutFileAppender"/>
        <AppenderRef ref="StdOutErrConsoleAppender"/>
    </Logger>
    <Logger level="INFO" name="StdOutErrRedirector.StdErr" additivity="false">
        <AppenderRef ref="StdErrFileAppender"/>
        <AppenderRef ref="StdOutErrConsoleAppender"/>
    </Logger>
    {%- for name, level in userConfiguredLoggers -%}
      <Logger level="{{ level }}" name="{{ name }}"/>
    {%- endfor -%}
    <Root level="{{ rootLoggerLogLevel }}">
      <AppenderRef ref="StdOut"/>
      <AppenderRef ref="RollingFile"/>
      <AppenderRef ref="SLS"/>
    </Root>
  </Loggers>
</Configuration>
```

- Session模式

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<Configuration xmlns="http://logging.apache.org/log4j/2.0/config"
strict="true" packages="com.ververica.platform.logging.appender" status="WARN">
  <Appenders>
    <Appender name="StdOut" type="Console">
      <Layout pattern="%d{yyyy-MM-dd HH:mm:ss,SSS}{GMT+8} %-5p %-60c %x - %m%n" t
ype="PatternLayout" charset="UTF-8"/>
    </Appender>
    <Appender name="RollingFile" type="RollingFile" fileName="${sys:log.file}" fi
lePattern="${sys:log.file}.%i">
      <Layout pattern="%d{yyyy-MM-dd HH:mm:ss,SSS}{GMT+8} %-5p %-60c %x - %m%n" t
ype="PatternLayout" charset="UTF-8"/>
      <Policies>
        <SizeBasedTriggeringPolicy size="5 MB"/>
      </Policies>
      <DefaultRolloverStrategy max="1"/>
    </Appender>
    <Appender name="SLS" type="SLS">
      <Layout pattern="%d{yyyy-MM-dd HH:mm:ss,SSS}{GMT+8} %-5p %-60c %x - %m%n" t
ype="PatternLayout" charset="UTF-8"/>
      <!-- The final effective log path is: ${baseUri}/logs/${namespace}/${deploy
mentId}/{jobId}/ -->
      <Property name="namespace">{{ namespace }}</Property> <!-- Do not modify th
is line -->
      <Property name="project">YOUR-SLS-PROJECT</Property>
      <Property name="logStore">YOUR-SLS-LOGSTORE</Property>
      <Property name="endpoint">YOUR-SLS-ENDPOINT</Property>
      <Property name="accessKeyId">YOUR-SLS-ACCESSKEYID</Property>
      <Property name="accessKeySecret">YOUR-SLS-ACCESSKEYSECRET</Property>
      <Property name="topic">{{ namespace }}:{{ sessionClusterName }}:{{ sessionC
lusterId }}</Property>
      <Property name="flushIntervalSeconds">10</Property>
      <Property name="flushIntervalEventCount">100</Property>
      <Property name="deploymentId">{{ sessionClusterName }}</Property>
      <Property name="jobId">{{ sessionClusterId }}</Property>
    </Appender>
      <Appender name="StdOutErrConsoleAppender" type="Console">
          <Layout pattern="%m" type="PatternLayout" charset="UTF-8"/>
      </Appender>
      <Appender name="StdOutFileAppender" type="RollingFile" fileName="${sys:stdo
ut.file}" filePattern="${sys:stdout.file}.%i">
          <Layout pattern="%m" type="PatternLayout" charset="UTF-8"/>
          <Policies>
          <SizeBasedTriggeringPolicy size="1 GB"/>
          </Policies>
          <DefaultRolloverStrategy max="2"/>
      </Appender>
      <Appender name="StdErrFileAppender" type="RollingFile" fileName="${sys:stde
rr.file}" filePattern="${sys:stderr.file}.%i">
          <Layout pattern="%m" type="PatternLayout" charset="UTF-8"/>
          <Policies>
          <SizeBasedTriggeringPolicy size="1 GB"/>
          </Policies>
          <DefaultRolloverStrategy max="2"/>
      </Appender>
```

```
  </Appenders>
  <Loggers>
    <Logger level="INFO" name="org.apache.hadoop"/>
    <Logger level="INFO" name="org.apache.kafka"/>
    <Logger level="INFO" name="org.apache.zookeeper"/>
    <Logger level="INFO" name="akka"/>
    <Logger level="ERROR" name="org.jboss.netty.channel.DefaultChannelPipeline"/>

    <Logger level="OFF" name="org.apache.flink.runtime.rest.handler.job.JobDetail
sHandler"/>
    <Logger level="ERROR" name="org.apache.flink.fs.osshadoop.shaded.com.aliyun.o
ss"/>
    <Logger level="INFO" name="StdOutErrRedirector.StdOut" additivity="false">
        <AppenderRef ref="StdOutFileAppender"/>
        <AppenderRef ref="StdOutErrConsoleAppender"/>
    </Logger>
    <Logger level="INFO" name="StdOutErrRedirector.StdErr" additivity="false">
        <AppenderRef ref="StdErrFileAppender"/>
        <AppenderRef ref="StdOutErrConsoleAppender"/>
    </Logger>
    {%- for name, level in userConfiguredLoggers -%}
      <Logger level="{{ level }}" name="{{ name }}"/>
    {%- endfor -%}
    <Root level="{{ rootLoggerLogLevel }}">
      <AppenderRef ref="StdOut"/>
      <AppenderRef ref="RollingFile"/>
      <AppenderRef ref="SLS"/>
    </Root>
  </Loggers>
</Configuration>
```

文件夹	说明
YOUR-SLS-PROJECT	替换成您SLS的Project名称。
YOUR-SLS-LOGSTORE	替换成您SLS的Logstore名称。
YOUR-SLS-ENDPOINT	替换成您SLS所在地域的Endpoint，详情请参见服务入口。
YOUR-SLS-ACCESSKEYID	替换成您配置SLS服务账号的**AccessKey ID**，获取方法请参见获取AccessKey。 **说明** 如果您配置与全托管集群同账号下的SLS，可以不填写该参数，不填写需要删除该参数。如果您配置与全托管集群不同账号下的SLS，该参数必填。

文件夹	说明
YOUR-SLS-ACCESSKEYSECRET	替换成您配置SLS服务账号的**AccessKey Secret**，获取方法请参见获取AccessKey。 **说明** 如果您配置与全托管集群同账号下的SLS，可以不填写该参数，不填写需要删除该参数。如果您配置与全托管集群不同账号下的SLS，该参数必填。
sessionClusterName	Session集群的名称。
sessionClusterId	Session集群的ID。

说明

- 请确保您所使用的AccessKey ID和AccessKey Secret是您开通**Flink全托管**所使用账号下管理的密钥对。
- 请确保您开通**Flink全托管**服务所使用的账号，有读写该SLS项目的权限。

vi. 单击**保存**。

3. 在左侧导航栏上，单击**作业运维**。
4. 单击目标作业名称**操作**列的**启动**。

   您可以在SLS控制台查看作业日志，详情请参见在SLS上查看。

配置工作空间下所有作业日志输出到SLS

您可以通过配置**作业模板**的方式，配置工作空间下所有作业日志默认输出到SLS。

1. 登录实时计算管理控制台。
2. 在**Flink全托管**页签，单击目标工作空间**操作**列下的**控制台**。
3. 在左侧选择导航栏，选择**系统管理 > 作业模板**。
4. 配置作业日志。

   i. 在**日志配置**区域，**Logging Profile**选择为**Custom Template**。

   ii. 将以下文本粘贴到**自定义模板**的输入框中，并修改以下参数为您SLS信息。

   - Per-Job模式

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration xmlns="http://logging.apache.org/log4j/2.0/config"
strict="true" packages="com.ververica.platform.logging.appender" status="WARN">
  <Appenders>
    <Appender name="StdOut" type="Console">
      <Layout pattern="%d{yyyy-MM-dd HH:mm:ss,SSS}{GMT+8} %-5p %-60c %x - %m%n" t
ype="PatternLayout" charset="UTF-8"/>
    </Appender>
    <Appender name="RollingFile" type="RollingFile" fileName="${sys:log.file}" fi
lePattern="${sys:log.file}.%i">
      <Layout pattern="%d{yyyy-MM-dd HH:mm:ss,SSS}{GMT+8} %-5p %-60c %x - %m%n" t
ype="PatternLayout" charset="UTF-8"/>
```

```
      <Policies>
        <SizeBasedTriggeringPolicy size="5 MB"/>
      </Policies>
      <DefaultRolloverStrategy max="1"/>
    </Appender>
    <Appender name="SLS" type="SLS">
      <Layout pattern="%d{yyyy-MM-dd HH:mm:ss,SSS}{GMT+8} %-5p %-60c %x - %m%n" t
ype="PatternLayout" charset="UTF-8"/>
      <!-- The final effective log path is: ${baseUri}/logs/${namespace}/${deploy
mentId}/{jobId}/ -->
      <Property name="namespace">{{ namespace }}</Property> <!-- Do not modify th
is line -->
      <Property name="project">YOUR-SLS-PROJECT</Property>
      <Property name="logStore">YOUR-SLS-LOGSTORE</Property>
      <Property name="endpoint">YOUR-SLS-ENDPOINT</Property>
      <Property name="accessKeyId">YOUR-SLS-ACCESSKEYID</Property>
      <Property name="accessKeySecret">YOUR-SLS-ACCESSKEYSECRET</Property>
      <Property name="topic">{{ namespace }}:{{ deploymentId }}:{{ jobId }}</Prop
erty>
      <Property name="deploymentName">{{ deploymentName }}</Property>
      <Property name="flushIntervalSeconds">10</Property>
      <Property name="flushIntervalEventCount">100</Property>
    </Appender>
      <Appender name="StdOutErrConsoleAppender" type="Console">
          <Layout pattern="%m" type="PatternLayout" charset="UTF-8"/>
      </Appender>
      <Appender name="StdOutFileAppender" type="RollingFile" fileName="${sys:stdo
ut.file}" filePattern="${sys:stdout.file}.%i">
          <Layout pattern="%m" type="PatternLayout" charset="UTF-8"/>
          <Policies>
          <SizeBasedTriggeringPolicy size="1 GB"/>
          </Policies>
          <DefaultRolloverStrategy max="2"/>
      </Appender>
      <Appender name="StdErrFileAppender" type="RollingFile" fileName="${sys:stde
rr.file}" filePattern="${sys:stderr.file}.%i">
          <Layout pattern="%m" type="PatternLayout" charset="UTF-8"/>
          <Policies>
          <SizeBasedTriggeringPolicy size="1 GB"/>
          </Policies>
          <DefaultRolloverStrategy max="2"/>
      </Appender>
  </Appenders>
  <Loggers>
    <Logger level="INFO" name="org.apache.hadoop"/>
    <Logger level="INFO" name="org.apache.kafka"/>
    <Logger level="INFO" name="org.apache.zookeeper"/>
    <Logger level="INFO" name="akka"/>
    <Logger level="ERROR" name="org.jboss.netty.channel.DefaultChannelPipeline"/>

    <Logger level="OFF" name="org.apache.flink.runtime.rest.handler.job.JobDetail
sHandler"/>
    <Logger level="ERROR" name="org.apache.flink.fs.osshadoop.shaded.com.aliyun.o
ss"/>
```

```
    <Logger level="INFO" name="StdOutErrRedirector.StdOut" additivity="false">
        <AppenderRef ref="StdOutFileAppender"/>
        <AppenderRef ref="StdOutErrConsoleAppender"/>
    </Logger>
    <Logger level="INFO" name="StdOutErrRedirector.StdErr" additivity="false">
        <AppenderRef ref="StdErrFileAppender"/>
        <AppenderRef ref="StdOutErrConsoleAppender"/>
    </Logger>
    {%- for name, level in userConfiguredLoggers -%}
      <Logger level="{{ level }}" name="{{ name }}"/>
    {%- endfor -%}
    <Root level="{{ rootLoggerLogLevel }}">
      <AppenderRef ref="StdOut"/>
      <AppenderRef ref="RollingFile"/>
      <AppenderRef ref="SLS"/>
    </Root>
  </Loggers>
</Configuration>
```

- Session模式

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration xmlns="http://logging.apache.org/log4j/2.0/config"
strict="true" packages="com.ververica.platform.logging.appender" status="WARN">
  <Appenders>
    <Appender name="StdOut" type="Console">
      <Layout pattern="%d{yyyy-MM-dd HH:mm:ss,SSS}{GMT+8} %-5p %-60c %x - %m%n" t
ype="PatternLayout" charset="UTF-8"/>
    </Appender>
    <Appender name="RollingFile" type="RollingFile" fileName="${sys:log.file}" fi
lePattern="${sys:log.file}.%i">
      <Layout pattern="%d{yyyy-MM-dd HH:mm:ss,SSS}{GMT+8} %-5p %-60c %x - %m%n" t
ype="PatternLayout" charset="UTF-8"/>
      <Policies>
        <SizeBasedTriggeringPolicy size="5 MB"/>
      </Policies>
      <DefaultRolloverStrategy max="1"/>
    </Appender>
    <Appender name="SLS" type="SLS">
      <Layout pattern="%d{yyyy-MM-dd HH:mm:ss,SSS}{GMT+8} %-5p %-60c %x - %m%n" t
ype="PatternLayout" charset="UTF-8"/>
      <!-- The final effective log path is: ${baseUri}/logs/${namespace}/${deploy
mentId}/{jobId}/ -->
      <Property name="namespace">{{ namespace }}</Property> <!-- Do not modify th
is line -->
      <Property name="project">YOUR-SLS-PROJECT</Property>
      <Property name="logStore">YOUR-SLS-LOGSTORE</Property>
      <Property name="endpoint">YOUR-SLS-ENDPOINT</Property>
      <Property name="accessKeyId">YOUR-SLS-ACCESSKEYID</Property>
      <Property name="accessKeySecret">YOUR-SLS-ACCESSKEYSECRET</Property>
      <Property name="topic">{{ namespace }}:{{ sessionClusterName }}:{{ sessionC
lusterId }}</Property>
      <Property name="flushIntervalSeconds">10</Property>
      <Property name="flushIntervalEventCount">100</Property>
      <Property name="deploymentId">{{ sessionClusterName }}</Property>
```

```
      <Property name="jobId">{{ sessionClusterId }}</Property>
    </Appender>
      <Appender name="StdOutErrConsoleAppender" type="Console">
          <Layout pattern="%m" type="PatternLayout" charset="UTF-8"/>
      </Appender>
      <Appender name="StdOutFileAppender" type="RollingFile" fileName="${sys:stdo
ut.file}" filePattern="${sys:stdout.file}.%i">
          <Layout pattern="%m" type="PatternLayout" charset="UTF-8"/>
          <Policies>
          <SizeBasedTriggeringPolicy size="1 GB"/>
          </Policies>
          <DefaultRolloverStrategy max="2"/>
      </Appender>
      <Appender name="StdErrFileAppender" type="RollingFile" fileName="${sys:stde
rr.file}" filePattern="${sys:stderr.file}.%i">
          <Layout pattern="%m" type="PatternLayout" charset="UTF-8"/>
          <Policies>
          <SizeBasedTriggeringPolicy size="1 GB"/>
          </Policies>
          <DefaultRolloverStrategy max="2"/>
      </Appender>
  </Appenders>
  <Loggers>
    <Logger level="INFO" name="org.apache.hadoop"/>
    <Logger level="INFO" name="org.apache.kafka"/>
    <Logger level="INFO" name="org.apache.zookeeper"/>
    <Logger level="INFO" name="akka"/>
    <Logger level="ERROR" name="org.jboss.netty.channel.DefaultChannelPipeline"/>

    <Logger level="OFF" name="org.apache.flink.runtime.rest.handler.job.JobDetail
sHandler"/>
    <Logger level="ERROR" name="org.apache.flink.fs.osshadoop.shaded.com.aliyun.o
ss"/>
    <Logger level="INFO" name="StdOutErrRedirector.StdOut" additivity="false">
        <AppenderRef ref="StdOutFileAppender"/>
        <AppenderRef ref="StdOutErrConsoleAppender"/>
    </Logger>
    <Logger level="INFO" name="StdOutErrRedirector.StdErr" additivity="false">
        <AppenderRef ref="StdErrFileAppender"/>
        <AppenderRef ref="StdOutErrConsoleAppender"/>
    </Logger>
    {%- for name, level in userConfiguredLoggers -%}
      <Logger level="{{ level }}" name="{{ name }}"/>
    {%- endfor -%}
    <Root level="{{ rootLoggerLogLevel }}">
      <AppenderRef ref="StdOut"/>
      <AppenderRef ref="RollingFile"/>
      <AppenderRef ref="SLS"/>
    </Root>
  </Loggers>
</Configuration>
```

文件夹	说明
YOUR-SLS-PROJECT	替换成您SLS的Project名称。
YOUR-SLS-LOGSTORE	替换成您SLS的Logstore名称。
YOUR-SLS-ENDPOINT	替换成您SLS所在地域的Endpoint，详情请参见服务入口。
YOUR-SLS-ACCESSKEYID	替换成您配置SLS服务账号的**AccessKey ID**，获取方法请参见获取AccessKey。 **说明** 如果您配置与全托管集群同账号下的SLS，可以不填写该参数，不填写需要删除该参数。如果您配置与全托管集群不同账号下的SLS，该参数必填。
YOUR-SLS-ACCESSKEYSECRET	替换成您配置SLS服务账号的**AccessKey Secret**，获取方法请参见获取AccessKey。 **说明** 如果您配置与全托管集群同账号下的SLS，可以不填写该参数，不填写需要删除该参数。如果您配置与全托管集群不同账号下的SLS，该参数必填。
sessionClusterName	Session集群的名称。
sessionClusterId	Session集群的ID。

说明

- 请确保您所使用的AccessKey ID和AccessKey Secret是您开通**Flink全托管**所使用账号下管理的密钥对。
- 请确保您开通**Flink全托管**服务所使用的账号，有读写该SLS项目的权限。

5. 单击**保存**。

说明 配置**作业模板**后，后续该工作空间下创建的所有作业的日志都会被存储到您配置的SLS项目下的Logstore。您可以在SLS控制台查看作业日志，详情请参见在SLS上查看。

8.4.6. 查看历史作业实例日志

Flink全托管支持在作业探查界面、对象存储OSS或者日志服务SLS上查看历史作业实例的日志。本文为您介绍如何在作业探查界面、OSS或SLS上查看历史作业实例日志。

背景信息

如果出现以下情况，则您需要查看历史作业实例的日志：

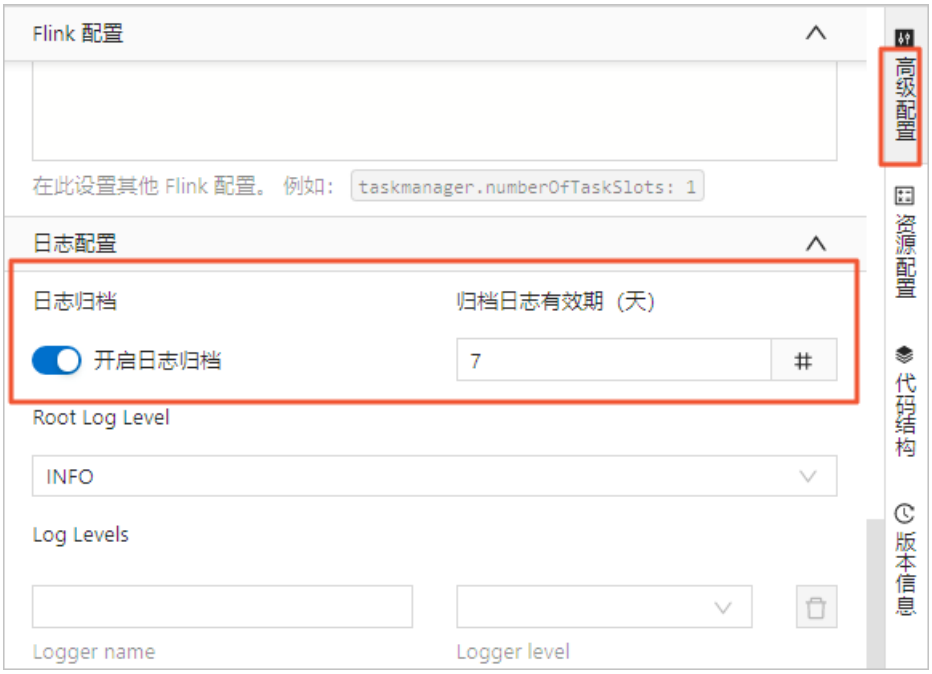
- 您需要查看、搜索和分析历史作业实例的运行日志（系统默认保留最近5 MB大小的运行日志）。
- Flink UI无法打开，需要查看Job Manager日志定位问题。

本文为您介绍查看历史作业实例日志的三种途径：

- 在作业探查界面查看
- 在OSS上查看
- 在SLS上查看

前提条件

- 在作业探查界面查看历史作业实例日志，需要您在作业开发页面的高级配置中已打开日志归档功能。默认已开启日志归档功能。



- 在OSS或者SLS上查看历史实例的日志，需要您在Flink全托管上已配置了历史作业实例的日志输出到OSS或者SLS，详情请参见配置历史作业实例日志输出。

使用限制

- 在VVR 3.x版本，仅VVR 3.0.7及以上版本支持查看历史作业实例的TM日志。
- 在VVR 4.x版本，仅VVR 4.0.11及以上版本支持查看历史作业实例的TM日志。

注意事项

- 如果Task Manager和Job Manager任意一个启动，日志都会写到SLS或OSS。但Task Manager和Job Manager都没有正常启动时，日志不会被写到SLS或OSS。
- 如果因为Job Manager和Task Manager都没有正常启动，导致您在OSS上没有找到对应的日志文件，请提交工单。

在作业探查界面查看

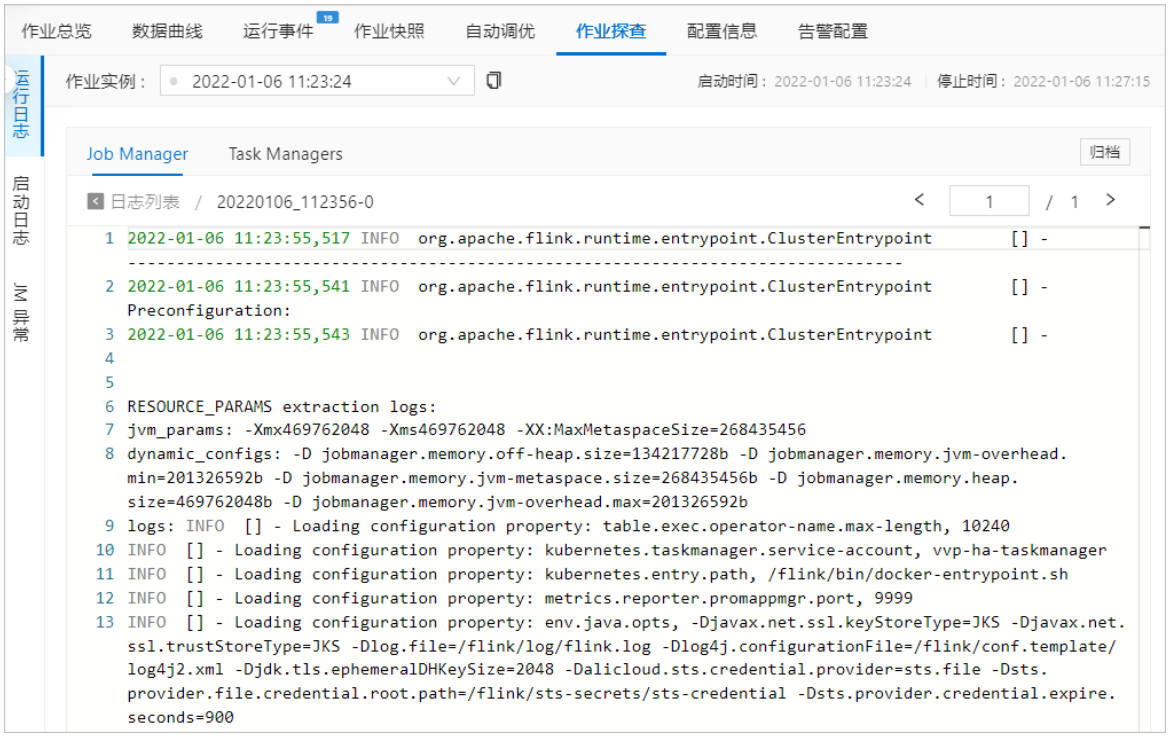
1. 登录实时计算控制台。
2. 在**Flink全托管**页签，单击目标工作空间**操作**列下的**开发控制台**。
3. 在左侧导航栏上，选择**应用 > 作业运维**。
4. 单击目标作业名称。
5. 单击**作业探查**。

6. 在左侧**运行日志**页签，选择正在运行的作业实例。

   左侧灰色小点表示此实例为历史作业实例。

7. 查看Job Manager和Task Managers日志。

   在**Job Manager**或**Task Managers**页签，分别单击对应的日志名称，进行查看。



在OSS上查看

1. 登录OSS管理控制台。
2. 在左侧导航栏，单击**Bucket列表**。
3. 单击您开通**Flink全托管**服务时，配置的Bucket名称。
4. 单击**文件管理**。
5. 在对应目录，查看日志信息。

文件夹	说明
artifacts	上传的JAR包会被存储到该目录。
flink-jobs	Flink作业的Checkpoint会被存储到该目录。
flink-savepoints	在**Flink全托管**开发控制台上，单击**Savepoint**，会触发**Savepoint**操作，最终的**Savepoint**文件会被存储到该目录。
logs	如果您作业的**日志模板**选择了**OSS**，则您的作业的日志会被存储到该目录。 **说明** *logs*目录下作业存储路径为：logs/${工作空间ID}/${作业ID}/${实例ID}。您可以通过该目录结构找到相应作业的JM（Job Manager）和TM（Task Manager）日志。

在SLS上查看

1. 登录日志服务控制台。
2. 在**全部的Project**区域，单击目标Project。
3. 在**日志库**页签，单击目标Logstore。
4. 输入查询分析语句，选择时间范围，单击**查询/分析**。

   详情请参见查询和分析日志。

9.开发参考

9.1. 支持的上下游存储

本文为您介绍Flink支持的上下游存储。

Flink支持的上下游存储包括SQL上下游存储和DataStream上下游存储，具体信息请参见如下内容：

- SQL上下游存储

作业类型	表类型	相关文档
	源表	○ 日志服务SLS源表 ○ 消息队列Kafka源表 ○ 数据总线DataHub源表 ○ 全量MaxCompute源表 ○ 增量MaxCompute源表 ○ 消息队列RocketMQ版源表 ○ 交互式分析Hologres源表 ○ 全量Elasticsearch源表 ○ Postgres的CDC源表（公测中） ○ MySQL的CDC源表 ○ Hudi源表 ○ Upsert Kafka源表 ○ Datagen源表 ○ 全量数据湖Iceberg源表 ○ 模拟数据生成源表

作业类型	表类型	相关文档
流作业	结果表	○ 日志服务SLS结果表 ○ 消息队列Kafka结果表 ○ 数据总线DataHub结果表 ○ 表格存储Tablestore结果表 ○ MaxCompute结果表 ○ 消息队列RocketMQ版结果表 ○ 云数据库HBase版结果表 ○ 云数据库Redis版结果表 ○ 交互式分析Hologres结果表 ○ 云数据库MongoDB版结果表 ○ Elasticsearch结果表 ○ Phoenix5结果表 ○ 云数据库RDS MySQL结果表 ○ 云原生数据仓库AnalyticDB MySQL版3.0结果表 ○ ClickHouse结果表 ○ print结果表 ○ Upsert Kafka结果表 ○ Hudi结果表 ○ Blackhole结果表 ○ InfluxDB结果表 ○ 数据湖Iceberg结果表 ○ MySQL结果表 ○ 云原生数据仓库AnalyticDB PostgreSQL版结果表
	维表	○ 云数据库HBase版维表 ○ 表格存储Tablestore维表 ○ MaxCompute维表 ○ 云数据库Redis版维表 ○ Elasticsearch维表 ○ 交互式分析Hologres维表 ○ 云数据库RDS MySQL维表 ○ 云原生数据仓库AnalyticDB MySQL版3.0维表 ○ FileSystem维表 ○ MySQL维表 ○ 模拟数据生成维表 ○ 云原生数据仓库AnalyticDB PostgreSQL版维表

作业类型	表类型	相关文档
批作业	源表	○ 消息队列RocketMQ版源表 ○ 全量MaxCompute源表 ○ 交互式分析Hologres源表 ○ 全量Elasticsearch源表
	结果表	○ 消息队列RocketMQ版结果表 ○ MaxCompute结果表 ○ 交互式分析Hologres结果表

- DataStream上下游存储
  - 开源Flink的上下游存储，请参见DataStream Connectors。
  - Flink全托管新增支持的上下游存储，请参见下表。

Connector版本	VVR/Flink版本	Connector类型
1.13-vvr-4.0.7	VVR 4.0.7（对应Flink 1.13）	■ ververica-connector-datahub ■ 文档请参见DataHub DataStream Connector。 ■ demo请参见ververica-connector-datahub-demo。 ■ ververica-connector-kafka ■ 文档请参见Kafka DataStream Connector。 ■ demo请参见ververica-connector-kafka-demo。 ■ ververica-connector-odps或者ververica-connector-continuous-odps ■ 文档请参见MaxCompute DataStream Connector。 ■ demo请参见ververica-connector-maxcompute-demo。 ■ ververica-connector-mysql-cdc 文档请参见MySQL CDC DataStream Connector。

9.2. Flink SQL参考

9.2.1. 概述

Flink SQL是为了简化计算模型、降低您使用Flink门槛而设计的一套符合标准SQL语义的开发语言。

本文通过以下方面，为您介绍Flink SQL的使用方法。

操作		文档
关键字		关键字
创建数据视图		创建数据视图
DDL数据定义语句	创建数据源表	• 日志服务SLS源表 • 消息队列Kafka源表 • 数据总线DataHub源表 • 全量MaxCompute源表 • 增量MaxCompute源表 • 消息队列RocketMQ版源表 • 交互式分析Hologres源表 • 全量Elasticsearch源表 • Postgres的CDC源表（公测中） • MySQL的CDC源表 • Upsert Kafka源表 • Datagen源表
	创建数据结果表	• 日志服务SLS结果表 • 消息队列Kafka结果表 • 数据总线DataHub结果表 • 表格存储Tablestore结果表 • MaxCompute结果表 • 消息队列RocketMQ版结果表 • 云数据库HBase版结果表 • 云数据库Redis版结果表 • 交互式分析Hologres结果表 • 云数据库MongoDB版结果表 • Elasticsearch结果表 • Phoenix5结果表 • 云数据库RDS MySQL结果表 • 云原生数据仓库AnalyticDB MySQL版3.0结果表 • ClickHouse结果表 • print结果表 • Upsert Kafka结果表 • Blackhole结果表 • InfluxDB结果表

操作		文档
	创建数据维表	• 云数据库HBase版维表 • 表格存储Tablestore维表 • MaxCompute维表 • 云数据库Redis版维表 • Elasticsearch维表 • 交互式分析Hologres维表 • 云数据库RDS MySQL维表 • 云原生数据仓库AnalyticDB MySQL版3.0维表 • FileSystem维表
DML数据操作语句（INSERT INTO）		INSERT INTO语句
DQL数据查询语句		DQL数据查询语句
窗口函数		• 概述 • 滚动窗口 • 滑动窗口 • 会话窗口 • OVER窗口
内置函数		• 概述 • 标量函数 • 表值函数 • 聚合函数 • 机器学习函数CLUSTER_SERVING
自定义函数	Java	• 概述 • 自定义标量函数（UDF） • 自定义聚合函数（UDAF） • 自定义表值函数（UDTF）
	Python	• 概述 • 自定义标量函数（UDF） • 自定义聚合函数（UDAF） • 自定义表值函数（UDTF）

9.2.2. 关键字

本文为您介绍Flink关键字。

Flink V1.11关键字详情，请参见Reserved Keywords。

> **说明** 如果您需要在DDL中定义Flink关键字，请在关键字两端添加反引号（`），例如 `` `value` ``。

9.2.3. 创建数据视图

您可以通过创建数据视图简化开发过程。

背景信息

通常，业务逻辑比较复杂，需要将多层嵌套写在DML语句中，但是这种方式定位问题比较困难。此时，您可以通过定义数据视图的方式，将多层嵌套写在数据视图中，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。

语法

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

- viewName：视图名称。
- columnName：字段名称。
- queryStatement：嵌套语句别名。

示例

```
CREATE TEMPORARY VIEW tmp_view AS
SELECT
    *
FROM
    datagen_source;
INSERT INTO
    rds_output
SELECT
    r,
    t,
    c,
    u
FROM
    tmp_view;
```

9.2.4. DDL数据定义语句

9.2.4.1. 创建数据源表

9.2.4.1.1. 日志服务SLS源表

本文为您介绍日志服务SLS源表的DDL定义、WITH参数、类型映射、属性字段和代码示例。

什么是日志服务

日志服务SLS是针对日志类数据的一站式服务。日志服务可以帮助您快捷地完成数据采集、消费、投递以及查询分析，提升运维和运营效率，建立海量日志处理能力。

前提条件

已创建日志服务Project和Logstore，详情请参见步骤二：创建Project和Logstore。

使用限制

仅Flink计算引擎VVR 2.0.0及以上版本支持日志服务SLS Connector。

DDL定义

```
create table sls_source(
  a int,
  b int,
  c varchar
) with (
  'connector' = 'sls',
  'endPoint' = '<yourEndPoint>',
  'accessId' = '<yourAccessId>',
  'accessKey' = '<yourAccessKey>',
  'startTime' = '<yourStartTime>',
  'project' = '<yourProjectName>',
  'logStore' = '<yourLogStoreName>',
  'consumerGroup' = '<yourConsumerGroupName>'
);
```

说明

- SLS暂不支持MAP类型的数据。
- SLS对于不存在字段会置为Null。
- DDL定义中字段顺序可以为无序，建议和物理表中字段顺序保持一致。

WITH参数

参数	说明	是否必选	备注
connector	源表类型。	是	固定值为 `sls` 。
endPoint	消费端点信息。	是	服务入口。
accessId	AccessKey ID。	是	无。
accessKey	AccessKey Secret。	是	无。
project	SLS项目名称。	是	无。

参数	说明	是否必选	备注
logStore	LogStore名称。	是	无。
startTime	消费日志的开始时间。	是	无。
consumerGroup	消费组名称。	否	您可以自定义消费组名（没有固定格式）。
batchGetSize	单次读取logGroup的条数。	否	默认值为100。 说明 • batchGetSize设置不能超过1000，否则会报错。 • batchGetSize设置单次读取logGroup的条数。如果单条logItem的大小和batchGetSize都很大，则可能会导致频繁的垃圾回收（Garbage Collection），您可以适当减小batchGetSize参数值。

类型映射

日志服务和Flink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	Flink字段类型
STRING	VARCHAR

属性字段

字段名	字段类型	说明
`__source__`	STRING METADATA VIRTUAL	消息源。
`__topic__`	STRING METADATA VIRTUAL	消息主题。
`__timestamp__`	BIGINT METADATA VIRTUAL	日志时间。
`__tag__`	MAP<VARCHAR, VARCHAR> METADATA VIRTUAL	消息TAG。对于属性"__tag__:__receive_time__":"1616742274"，'__receive_time__'和'1616742274'会被作为KV对，记录在Map中，在SQL中通过__tag__['__receive_time__']的方式访问。

> **说明** 仅在VVR 3.0.1版本及以后版本支持获取以上SLS属性字段。

代码示例

```
CREATE TEMPORARY TABLE sls_input(
  `time` BIGINT,
  url STRING,
  dt STRING,
  float_field FLOAT,
  double_field DOUBLE,
  boolean_field BOOLEAN,
  `__topic__` STRING METADATA VIRTUAL,
  `__source__` STRING METADATA VIRTUAL,
  `__timestamp__` BIGINT METADATA VIRTUAL,
   __tag__ MAP<VARCHAR, VARCHAR> METADATA VIRTUAL,
  proctime as PROCTIME()
) WITH (
  'connector' = 'sls',
  'endpoint' ='cn-hangzhou.log.aliyuncs.com',
  'accessid' ='xx',
  'accesskey' ='xxx',
  'starttime' = '2001-08-01 00:00:00',
  'project' ='sls-test',
  'logstore' ='sls-input'
);
CREATE TEMPORARY TABLE print_sink(
  `time` BIGINT,
  url STRING,
  dt STRING,
  float_field FLOAT,
  double_field DOUBLE,
  boolean_field BOOLEAN,
  `__topic__` STRING ,
  `__source__` STRING ,
  `__timestamp__` BIGINT ,
  receive_time BIGINT
) WITH (
  'connector' = 'print',
  'logger'='true'
);
INSERT INTO print_sink
SELECT
 `time`,
  url,
  dt,
  float_field,
  double_field,
  boolean_field,
  `__topic__` ,
  `__source__` ,
  `__timestamp__` ,
  cast(__tag__['__receive_time__'] as bigint) as receive_time
FROM sls_input;
```

9.2.4.1.2. 消息队列Kafka源表

本文为您介绍消息队列Kafka源表的DDL定义、WITH参数、元信息列和示例。

什么是Kafka源表

消息队列Kafka版是阿里云提供的分布式、高吞吐、可扩展的消息队列服务。消息队列Kafka版广泛用于日志收集、监控数据聚合、流式数据处理、在线和离线分析等大数据领域。

前提条件

- 已创建消息队列Kafka资源，详情请参见步骤三：创建资源。
- 已配置白名单，详情请参见配置白名单。

使用限制

- 仅Flink计算引擎VVR 2.0.0及以上版本支持消息队列Kafka Connector。
- 消息队列Kafka Connector仅支持读取kafka 0.10及以上版本的数据。
- 消息队列Kafka Connector仅支持Kafka 2.4版本的消费者配置项，详情请参见消费者配置项。
- 仅Flink计算引擎vvr-4.0.12-flink-1.13及以上版本支持Kafka作为CTAS的同步数据源。
- 仅JSON format支持类型推导和Schema变更，其它format暂不支持。
- 在Kafka作为CTAS语句的数据源时，仅支持将JSON变更同步到Hudi和Hologres结果表。
- 仅支持Kafka中value部分的类型推导和表结构变更。如果您需要同步Kafka key部分的列，则需要您手动在DDL中进行指定。详情请参见示例三。

DDL定义

以下为创建Kafka源表的DDL示例，消息格式为CSV，包含5个字段。

```
CREATE TABLE kafkaTable (
  `user_id` BIGINT,
  `item_id` BIGINT,
  `category_id` BIGINT,
  `behavior` STRING,
  `topic` STRING METADATA VIRTUAL,
  `partition` BIGINT METADATA VIRTUAL
) WITH (
  'connector' = 'kafka',
  'topic' = 'my_excellent_topic',
  'properties.bootstrap.servers' = 'mykafka:9092',
  'properties.group.id' = 'my_excellent_group'
  'format' = 'csv',
  'scan.startup.mode' = 'earliest-offset'
)
```

在实际使用中请根据实际情况配置字段名和WITH参数。

元信息列

您可以在源表中定义元信息列，以获取Kafka消息的元信息。例如，当WITH参数中定义了多个topic时，如果在Kafka源表中定义了元信息列，那么Flink读取到的数据就会被标识是从哪个topic中读取的数据。

Key	数据类型	说明
topic	STRING NOT NULL METADATA VIRTUAL	Kafka消息所在的Topic名称。
partition	INT NOT NULL METADATA VIRTUAL	Kafka消息所在的Partition ID。
headers	MAP<STRING, BYTES> NOT NULL METADATA VIRTUAL	Kafka消息的消息头（header）。
leader-epoch	INT NOT NULL METADATA VIRTUAL	Kafka消息的Leader epoch。
offset	BIGINT NOT NULL METADATA VIRTUAL	Kafka消息的偏移量（offset）。
timestamp	TIMESTAMP(3) WITH LOCAL TIME ZONE NOT NULL METADATA VIRTUAL	Kafka消息的时间戳。
timestamp-type	STRING NOT NULL METADATA VIRTUAL	Kafka消息的时间戳类型： • NoTimestampType：消息中没有定义时间戳。 • CreateTime：消息产生的时间。 • LogAppendTime：消息被添加到Kafka Broker的时间。

说明

- 仅VVR 3.0.0及以后版本支持定义元信息列。
- 如果您需要将消息队列Kafka作为结果表，且忽略只读元信息列时，则在源表中定义元信息列时，必须声明这些元信息列为VIRTUAL。

作为CTAS数据源

CTAS语句支持将消息队列Kafka，且format为JSON的表作为数据源。在数据同步过程中，如果某些字段并未出现在预定义的表结构中，Flink会尝试去自动推导该列的类型。如果自动推导的类型不能满足您的使用需求，您也可以通过辅助推导的方式对某些列的解析类型进行声明。关于类型推导和嵌套类型的处理方式，详情如下：

说明 关于JSON Format的详细描述，详情请参见JSON Format。

在类型推导过程中，Flink默认只会展开JSON文本中的第一层数据，根据其类型和数值，按照基本规则进行类型推导。当然您也可以在DDL中声明特定的列的解析类型以满足您的特殊需要，即辅助推导。基本规则和辅助推导详情如下：

- 基本规则

  类型映射基本规则如下表所示。

JSON类型	Flink SQL类型
BOOLEAN	BOOLEAN
STRING	DATE、TIMESTAMP、TIMESTAMP_LTZ、TIME或STRING

JSON类型	Flink SQL类型
INT或LONG	BIGINT
BIGINT	DECIMAL或STRING 说明 由于Flink中DECIMA的类型是有精度限制的。因此，如果整数的实际取值超过了DECIMAL类型最大精度，Flink会自动推导其类型为STRING，避免精度的损失。
FLOAT、DOUBLE或BIG DECIMAL	DOUBLE
ARRAY	STRING
OBJECT	STRING

示例

- JSON文本

```
{
  "id": 101,
  "name": "VVP",
  "properties": {
    "owner": "阿里云",
    "engine": "Flink"
  }
    "type": ["大数据"]
}
```

- Flink写入到下游存储的表信息

id	name	properties	type
101	"VVP"	{ "owner": "阿里云", "engine": "Flink" }	["大数据"]

- 辅助推导

  如果您觉得以上基本规则不符合您实际需要，则您可以在源表的DDL中声明特定列的解析类型。通过该方式，Flink会优先使用您声明的列类型去解析目标字段。针对以下示例，Flink会使用DECIMAL的方式去解析price字段，而不是使用默认的基本规则将其转换为DOUBLE类型。

```
CREATE TABLE evolvingKafkaSource (
  price DECIMAL(18, 2)
) WITH (
  'connector' = 'kafka',
  'properties.bootstrap.servers' = 'localhost:9092',
  'topic' = 'evolving_kafka_demo',
  'scan.startup.mode' = 'earliest-offset',
  'format' = 'json'
);
```

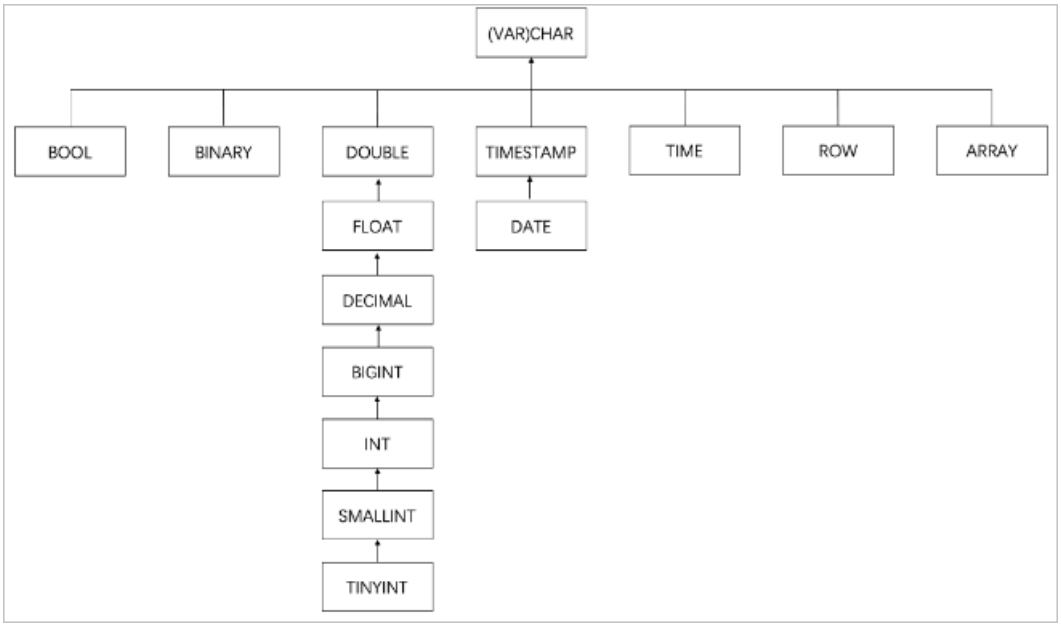
但是，如果您在DDL中指定的类型和实际数据中的类型不一致时，则可以按照以下方式进行处理：

- 在声明的类型比实际类型更宽泛时，以声明的类型自动去解析。

  例如，声明为DOUBLE，遇到的数据类型为BIGINT，则会以DOUBLE类型去解析。

- 在实际的类型比声明的类型更为宽泛或者两种类型不兼容时，由于当前CTAS不支持类型变更，因此会报错提示您相关信息，您需要重新启动作业并声明准确的类型去解析数据。

  对于类型的宽泛的程度以及兼容性，可以总结为下图。



上图表示越靠近根节点，其类型越宽泛。如果两个类型在不同的分支上，则表示这两个类型不兼容。

> **说明**
>
> - 不支持辅助推导复杂类型，包括ROW、ARRAY、MAP和MULTISET。
> - 对于复杂类型，Flink在默认情况下会处理为STRING。

通常，Kafka Topic中的JSON文本带有嵌套结构。如果您需要提取JSON文本中的嵌套列，则可以通过以下两种方式：

- 在源表DDL中声明'json.infer-schema.flatten-nested-columns.enable'='true'，来展开嵌套列中的所有元素至顶层。

  通过该方式，所有的嵌套列都会被依次展开。为了避免列名冲突，Flink采用索引到该列的路径作为展开后列名字。

> **说明** 目前不支持解决列名冲突。如果发生列名冲突，请在源表的DDL中声明json.ignore-parse-errors为true，来忽略存在冲突的数据。

示例

- JSON文本

```
{
  "nested": {
    "inner": {
      "col": true
    }
  }
}
```

- Flink写入到下游存储的表信息

neseted.inner.col
true

- 在DDL中CTAS语法中添加计算列 `rowkey` AS JSON_VALUE(`properties`, `$.rowkey`)，来指定要展开的列。

  详情请参见示例四。

WITH参数

参数	说明	是否必选	数据类型	备注
connector	源表类型。	是	String	固定值为 `kafka` 。
topic	topic名称。	是	String	以分号 (;) 分隔多个topic名称，例如 `topic-1;topic-2` 。 **注意** topic和topic-pattern两个选项只能指定其中一个。
topic-pattern	匹配读取topic名称的正则表达式。所有匹配该正则表达式的topic在作业运行时均会被订阅。	否	String	**注意** • 仅VVR 3.0.0及以后版本支持该参数。 • topic和topic-pattern两个选项只能指定其中一个。
properties.bootstrap.servers	Kafka Broker地址。	是	String	格式为 `host:port,host:port,host:port` ，以英文逗号（,）分割。

参数	说明	是否必选	数据类型	备注
properties.group.id	Kafka消费组ID。	是	String	无。
properties.*	Kafka配置。	否	String	后缀名必须匹配为Kafka官方文档中定义的配置。Flink会将properties.前缀移除，并将剩余的配置传递给Kafka客户端。例如可以通过 `'properties.allow.auto.create.topics' = 'false'` 来禁用自动创建topic。 不建议通过以上方式修改'key.deserializer'和'value.deserializer'参数，因为它们会被kafka配置覆盖。
format	Flink Kafka Connector在反序列化来自Kafka的消息体（value）时使用的格式。	是	String	取值如下： • csv • json • avro • debezium-json • canal-json • maxwell-json • avro-confluent • raw **说明** • 仅VVR 3.0.0及以后版本支持maxwell-json、avro-confluent和raw格式。 • format和value.format只能配置其中一个，如果同时配置两个，则会有冲突。

参数	说明	是否必选	数据类型	备注
value.format	Flink Kafka Connector在反序列化来自Kafka的消息体（value）时使用的格式。	是	String	取值如下： • csv • json • avro • debezium-json • canal-json • maxwell-json • avro-confluent • raw 说明 • 仅VVR 3.0.0及以后版本支持maxwell-json、avro-confluent和raw格式。 • format和value.format只能配置其中一个，如果同时配置两个，则会有冲突。
key.format	反序列化Kafka消息键（key）时使用的格式。	否	String	取值如下： • csv • json • avro • debezium-json • canal-json • maxwell-json • avro-confluent • raw 说明 • 仅VVR 3.0.0及以后版本支持maxwell-json、avro-confluent和raw格式。 • 如果指定了key.format参数，则也必须指定key.fields参数。

参数	说明	是否必选	数据类型	备注
key.fields	Kafka消息键（key）解析出来的数据存放的字段。	否	String	多个字段名以分号（;）分隔。例如 `field1;field2` 。默认不配置该参数，因此key不会被解析，key数据将被丢弃。 **说明** 仅VVR 3.0.0及以后版本支持该参数。
key.fields-prefix	为所有Kafka消息键（Key）指定自定义前缀，以避免与消息体（Value）格式字段重名。	否	String	默认情况下前缀为空。如果定义了前缀，表结构和配置项key.fields都需要使用带前缀的名称。 当构建消息键字段时，前缀会被移除，将使用无前缀的名称。 **注意** • 仅VVR 3.0.0及以后版本支持该参数。 • 该配置项要求必须将value.fields-include配置为EXCEPT_KEY。
value.fields-include	在解析消息体时，是否要包含消息键字段。	否	String	取值如下： • ALL（默认值）：所有定义的字段都存放消息体（Value）的解析出来的数据。 • EXCEPT_KEY：除去key.fields定义字段，剩余的定义字段可以用来存放消息体（Value）解析出来的数据。 **说明** 仅VVR 3.0.0及以后版本支持该参数。

参数	说明	是否必选	数据类型	备注
scan.startup.mode	Kafka读取数据的启动位点。	否	String	取值如下： • earliest-offset：从Kafka最早分区开始读取。 • latest-offset：从Kafka最新位点开始读取。 • group-offsets（默认值）：根据Group读取。 • timestamp：从Kafka指定时间点读取。 需要在WITH参数中指定scan.startup.timestamp-millis参数。 • specific-offsets：从Kafka指定分区指定偏移量读取。 需要在WITH参数中指定scan.startup.specific-offsets参数。
scan.startup.specific-offsets	在specific-offsets启动模式下，指定每个分区的启动偏移量。	否	String	例如：partition:0,offset:42;partition:1,offset:300
scan.startup.timestamp-millis	在timestamp启动模式下，指定启动位点时间戳。	否	Long	单位为毫秒。
value.fields-prefx	为所有Kafka消息键（Value）指定自定义前缀，以避免与消息体（Key）或Metadata字段重名。	否	String	如果定义了前缀，表结构需要使用带前缀的名称。当构建消息键字段时，前缀会被移除，将使用无前缀的名称。 **说明** 仅Flink计算引擎VVR 4.0.12及之后版本支持该参数。 默认情况下前缀为空。
json.infer-schema.flatten-nested-columns.enable	是否递归式地展开JSON中的嵌套列。	否	Boolean	参数取值如下： • true：递归式展开。 对于被展开的列，Flink使用索引该值的路径作为名字。例如，对于JSON{"nested": {"col": true}} 中的列col，它展开后的名字为nested.col。 • false（默认值）：将嵌套类型当作String处理。 **说明** 该参数仅在Kafka作为CTAS数据同步的数据源时生效。

参数	说明	是否必选	数据类型	备注
json.infer-schema.primitive-as-string	是否推导所有基本类型为String类型。	否	Boolean	参数取值如下： • true：推导所有基本类型为String。 • false（默认值）：按照基本规则进行推导。 说明 该参数仅在Kafka作为CTAS数据同步的数据源时生效。

Kafka Consumer配置参数详情请参见Kafka官网消费者配置项列表。如果您还需要直接配置Connector使用的Kafka Consumer，可以在Kafka Consumer配置参数前添加 `properties` 前缀，并将该Kafka Consumer配置信息追加至WITH参数。例如Kafka集群需要SASL（Simple Authentication and Security Layer）认证。

```
CREATE TABLE kafkaTable (
    ...
) WITH (
    ...
    'properties.security.protocol' = 'SASL_PLAINTEXT',
    'properties.sasl.mechanism' = 'PLAIN',
    'properties.sasl.jaas.config' = 'org.apache.flink.kafka.shaded.org.apache.kafka.common.
security.plain.PlainLoginModule required username="USERNAME" password="PASSWORD";'
);
```

说明 仅Per-Job集群支持Kafka认证。

代码示例

- 示例一：从Kafka中读取数据后插入Kafka。

  从名称为source的Topic中读取Kafka数据，再写入名称为sink的Topic，数据使用CSV格式。

```
CREATE TEMPORARY TABLE Kafka_source (
  id INT,
  name STRING,
  age INT
) WITH (
  'connector' = 'kafka',
  'topic' = 'source',
  'properties.bootstrap.servers' = '<yourKafkaBrokers>',
  'properties.group.id' = '<yourKafkaConsumerGroupId>',
  'format' = 'csv'
);
CREATE TEMPORARY TABLE Kafka_sink (
  id INT,
  name STRING,
  age INT
) WITH (
  'connector' = 'kafka',
  'topic' = 'sink',
  'properties.bootstrap.servers' = '<yourKafkaBrokers>',
  'properties.group.id' = '<yourKafkaConsumerGroupId>',
  'format' = 'csv'
);
INSERT INTO Kafka_sink SELECT id, name, age FROM Kafka_source;
```

- 示例二：同步表结构以及数据

  将Kafka Topic中的消息实时同步到Hologres中。在该情况下，您可以将Kafka消息的offset和partition id作为主键，从而保证在Failover时，Hologres中不会有重复消息。

```
CREATE TEMPORARY TABLE kafkaTable (
  `offset` INT NOT NULL METADATA,
  `part` BIGINT NOT NULL METADATA FROM 'partition'
) WITH (
  'connector' = 'kafka',
  'properties.bootstrap.servers' = '<yourKafkaBrokers>',
  'topic' = 'kafka_evolution_demo',
  'scan.startup.mode' = 'earliest-offset',
  'format' = 'json',
  'json.infer-schema.flatten-nested-columns.enable' = 'true'
    -- 可选，将嵌套列全部展开。
);
CREATE TABLE IF NOT EXISTS hologres.kafka.`sync_kafka`(
   PRIMARY KEY (`part`, `offset`) NOT ENFORCED
) WITH (
  'connector' = 'hologres'
) AS TABLE vvp.`default`.kafkaTable;
```

- 示例三：同步表结构以及Kafka消息的key和value数据。

  Kafka消息中的key部分已经存储了相关信息，您可以同时同步Kafka中的key和value。

```
CREATE TEMPORARY TABLE kafkaTable (
  `key_id` INT NOT NULL,
  `val_name` VARCHAR(200)
) WITH (
  'connector' = 'kafka',
  'properties.bootstrap.servers' = '<yourKafkaBrokers>',
  'topic' = 'kafka_evolution_demo',
  'scan.startup.mode' = 'earliest-offset',
  'key.format' = 'json',
  'value.format' = 'json',
  'key.fields' = 'key_id',
  'key.fields-prefix' = 'key_',
  'value.fields-prefix' = 'val_',
  'value.fields-include' = 'EXCEPT_KEY'
);
CREATE TABLE IF NOT EXISTS hologres.kafka.`sync_kafka`(
WITH (
  'connector' = 'hologres'
) AS TABLE vvp.`default`.kafkaTable;
```

说明 Kafka消息中的key部分不支持表结构变更和类型推导，需要您手动声明。

- 示例四：同步表结构和数据并进行计算。

  在同步Kafka数据到Hologres时，往往需要一些轻量级的计算，并将计算后的结果作为主键。

```
CREATE TEMPORARY TABLE kafkaTable (
  `distinct_id` INT NOT NULL,
  `properties` STRING,
  `timestamp` TIMESTAMP METADATA,
  `date` AS CAST(`timestamp` AS DATE)
) WITH (
  'connector' = 'kafka',
  'properties.bootstrap.servers' = '<yourKafkaBrokers>',
  'topic' = 'kafka_evolution_demo',
  'scan.startup.mode' = 'earliest-offset',
  'key.format' = 'json',
  'value.format' = 'json',
  'key.fields' = 'key_id',
  'key.fields-prefix' = 'key_'
);
CREATE TABLE IF NOT EXISTS hologres.kafka.`sync_kafka` WITH (
   'connector' = 'hologres'
) AS TABLE vvp.`default`.kafkaTable
ADD COLUMN
  `order_id` AS COALESCE(JSON_VALUE(`properties`, '$.order_id'), 'default');
-- 使用COALESCE处理空值情况。
```

常见问题

- Flink如何获取JSON数据？
- Flink和Kafka网络连通，但Flink无法消费或者写入数据？

- Flink中的Commit Offset有什么作用？
- 为什么Flink和Kafka之间的网络是连通的，但是依然会有timeout expired while fetching topic metadata的报错？
- 为什么Kafka源表数据基于Event Time的窗口后，不能输出数据？

9.2.4.1.3. 数据总线DataHub源表

本文为您介绍数据总线DataHub源表的DDL定义、WITH参数、类型映射、属性字段和常见问题。

什么是数据总线DataHub

阿里云流数据处理平台DataHub是流式数据（Streaming Data）的处理平台，提供对流式数据的发布（Publish）、订阅（Subscribe）和分发功能，让您可以轻松构建基于流式数据的分析和应用。

前提条件

已创建DataHub的Project和Topic，详情请参见创建Project和Topic。

使用限制

仅Flink计算引擎VVR 2.0.0及以上版本支持数据总线DataHub Connector。

DDL定义

```
create table datahub_source(
  name VARCHAR,
  age BIGINT,
  birthday BIGINT
) with (
  'connector' = 'datahub',
  'endPoint' = '<yourEndPoint>',
  'project' = '<yourProjectName>',
  'topic' = '<yourTopicName>',
  'subId' = '<yourSubId>',
  'accessId' = '<yourAccessId>',
  'accessKey' = '<yourAccessKey>'
);
```

WITH参数

参数	说明	是否必填	备注
connector	源表类型。	是	固定值为 `datahub` 。
endPoint	消费端点信息。	是	详情请参见DataHub域名列表。
accessId	AccessKey ID。	是	无。
accessKey	AccessKey Secret。	是	无。
project	项目名称。	是	无。

参数	说明	是否必填	备注
topic	Topic名称。	是	无。
subId	Topic的订阅ID。	是	多个任务不能同时使用同一个订阅。
startTime	启动位点的时间。	否	格式为 `yyyy-MM-dd hh:mm:ss` 。
retryTimeout	最大持续重试时间。	否	单位为毫秒，默认值为1800000毫秒（半小时）。
retryInterval	重试间隔。	否	单位为毫秒，默认值为1000。
maxFetchSize	单次读取条数。	否	默认值为50。
maxBufferSize	异步读取的最大缓存数据条数。	否	默认值为50。
lengthCheck	单行字段条数检查策略。	否	• NONE（默认值）： ○ 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，跳过该行数据。 • SKIP：解析出的字段数和定义字段数不同时跳过该行数据。 • 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，按从左到右的顺序，在行尾用Null填充缺少的字段。
columnErrorDebug	是否打开调试开关。	否	• false（默认值）：关闭调试功能。 • true：打开调试开关，打印解析异常的日志。

类型映射

DataHub和Flink字段类型对应关系如下，建议使用该对应关系进行DDL声明。

DataHub字段类型	Flink字段类型
STRING	VARCHAR
TIMESTAMP	BIGINT
TINYINT	TINYINT
SMALLINT	SMALLINT

DataHub字段类型	Flink字段类型
INTEGER	INTEGER
BIGINT	BIGINT
FLOAT	FLOAT
DOUBLE	DOUBLE
BOOLEAN	BOOLEAN
DECIMAL	DECIMAL

说明 如果您DataHub中存储的是二进制BLOB类型数据时，则Flink字段需要声明为VARBINARY类型，与METAQ类似。

属性字段

字段名	字段类型	说明
shard-id	BIGINT METADATA VIRTUAL	Shard的ID。
sequence	STRING METADATA VIRTUAL	数据顺序。
system-time	TIMESTAMP METADATA VIRTUAL	系统时间。

说明 仅在VVR 3.0.1及以上版本支持获取以上DataHub属性字段。

代码示例

```
CREATE TEMPORARY TABLE datahub_input (
  `time` BIGINT,
  `sequence`  STRING METADATA VIRTUAL,
  `shard-id` BIGINT METADATA VIRTUAL,
  `system-time` TIMESTAMP METADATA VIRTUAL
) WITH (
  'connector' = 'datahub',
  'subId' = '<yourSubId>',
  'endPoint' = '<yourEndPoint>',
  'project' = '<yourProjectName>',
  'topic' = '<yourTopicName>',
  'accessid' = '<yourAccessId>',
  'accesskey' = '<yourAccessKey>'
);
CREATE TEMPORARY TABLE test_out (
  `time` BIGINT,
  `sequence`  STRING,
  `shard-id` BIGINT,
  `system-time` TIMESTAMP
) WITH (
  'connector' = 'print',
  'logger' = 'true'
);
INSERT INTO test_out
SELECT
  `time`,
  `sequence` ,
  `shard-id`,
  `system-time`
FROM datahub_input;
```

常见问题

- 分裂或者缩容DataHub Topic后导致Flink作业失败，如何恢复？
- 可以删除正在消费的DataHub Topic吗？

9.2.4.1.4. 全量MaxCompute源表

本文为您介绍全量MaxCompute源表DDL定义、WITH参数、类型映射和常见问题。

> **说明** MaxCompute Connector可以作为Stream作业和Batch作业的源表使用。

什么是MaxCompute

大数据计算服务MaxCompute（原名ODPS）是一种快速、完全托管的EB级数据仓库解决方案，致力于批量结构化数据的存储和计算，提供海量数据仓库的解决方案及分析建模服务。MaxCompute详情请参见什么是MaxCompute。

前提条件

已创建MaxCompute表，详情请参见创建表。

使用限制

仅Flink计算引擎VVR 2.0.0及以上版本支持全量MaxCompute Connector。

DDL定义

```
create table odps_source(
  id INT,
  user_name VARCHAR,
  content VARCHAR
) with (
  'connector' = 'odps',
  'endpoint' = '<yourEndpoint>',
  'tunnelEndpoint' = '<yourTunnelEndpoint>',
  'project' = '<yourProjectName>',
  'tablename' = '<yourTableName>',
  'accessid' = '<yourAccessKeyId>',
  'accesskey' = '<yourAccessKeySecret>',
  'partition' = 'ds=2018****'
);
```

WITH参数

参数	说明	是否必填	备注
connector	源表类型。	是	固定值为 `odps` 。
endPoint	MaxCompute服务地址。	是	请参见Endpoint。
tunnelEndpoint	MaxCompute Tunnel服务的连接地址。	否	请参见Endpoint。 **说明** VPC环境下为必填。
project	MaxCompute项目名称。	是	无
tableName	MaxCompute表名。	是	无
accessId	AccessKey ID。	是	无
accessKey	AccessKey Secret。	是	无

参数	说明	是否必填	备注
partition	分区名。	否	• 只存在一级分区的MaxCompute表 例如，如果只存在1个分区列 `ds` ，则 `` `partition` = 'ds=20180905' `` 表示读 `ds=20180905` 分区的数据。 • 存在多级分区的MaxCompute表 例如，如果存在2个分区列 `ds` 和 `hh` ，则 `` `partition`='ds=20180905,hh=*' `` 表示读 `ds=20180905` 分区的数据。 **说明** 分区过滤时需要声明所有分区的值。例如，上述示例中，只声明 `` `partition` = 'ds=20180905' `` ，则不会读取任何分区。

类型映射

MaxCompute字段类型	Flink字段类型
TINYINT	TINYINT
SMALLINT	SMALLINT
INT	INT
BIGINT	BIGINT
FLOAT	FLOAT
DOUBLE	DOUBLE
BOOLEAN	BOOLEAN
DATETIME	TIMESTAMP
TIMESTAMP	TIMESTAMP
VARCHAR	VARCHAR
DECIMAL	DECIMAL
BINARY	VARBINARY
STRING	VARCHAR

代码示例

```
CREATE TEMPORARY TABLE odps_source (
  cid varchar,
  rt DOUBLE
) with (
  'connector' = 'odps',
  'endpoint' = '<yourEndpointName>',
  'tunnelEndpoint' = '<yourTunnelEndpoint>',
  'project' = '<yourProjectName>',
  'tablename' = '<yourTableName>',
  'accessid' = '<yourAccessId>',
  'accesskey' = '<yourAccessPassword>',
  'partition' = 'ds=20180905'
);
CREATE TEMPORARY TABLE blackhole_sink (
  cid varchar,
  invoke_count BIGINT
) with (
  'connector'='blackhole'
);
INSERT INTO blackhole_sink
SELECT
   cid,
   count(*) as invoke_count
FROM odps_source GROUP BY cid;
```

常见问题

- endPoint和tunnelEndpoint是指什么？如果配置错误会产生什么结果？
- 启动作业时出现Akka超时报错，但是全量MaxCompute源表和增量MaxCompute获取Metadata速率正常，应该如何处理？
- 全量MaxCompute和增量MaxCompute源表作业是否支持暂停作业后修改并发数，再恢复作业？
- 全量MaxCompute和增量MaxCompute是如何读取MaxCompute数据的？
- 引用MaxCompute作为数据源，在作业启动后，向已有的分区或者表里追加数据，这些新数据是否能被全量MaxCompute或增量MaxCompute源表读取？
- 作业启动位点设置了2019-10-11 00:00:00， 为什么启动位点前的分区也会被全量MaxCompute源表读取？
- 报错：ErrorMessage=Authorization Failed [4019], You have NO privilege'ODPS:***'

9.2.4.1.5. 增量MaxCompute源表

本文为您介绍增量MaxCompute源表DDL定义、WITH参数、类型映射和常见问题。

什么是MaxCompute

大数据计算服务MaxCompute（原名ODPS）是一种快速、完全托管的EB级数据仓库解决方案，致力于批量结构化数据的存储和计算，提供海量数据仓库的解决方案及分析建模服务。MaxCompute详情请参见什么是MaxCompute。

前提条件

已创建MaxCompute表，详情请参见创建表。

使用限制

- 仅Flink计算引擎VVR 2.1.2及以上版本支持增量MaxCompute Connector。
- 增量MaxCompute源表不支持作为维表使用。
- 增量MaxCompute源表只支持MaxCompute分区表，不支持非分区表。

DDL定义

```
create table odps_source(
  id INT,
  user_name VARCHAR,
  content VARCHAR
) with (
  'connector' = 'continuous-odps',
  'endpoint' = '<yourEndpoint>',
  'tunnelEndpoint' = '<yourTunnelEndpoint>',
  'project' = '<yourProjectName>',
  'tablename' = '<yourTableName>',
  'accessid' = '<yourAccessKeyId>',
  'accesskey' = '<yourAccessKeySecret>',
  'startpartition' = 'ds=2018****'
);
```

WITH参数

参数	说明	是否必填	备注
connector	源表类型。	是	固定值为 `continuous-odps` 。
endPoint	MaxCompute服务本身的连接地址。	是	请参见Endpoint。
tunnelEndpoint	MaxCompute Tunnel服务的连接地址。	是	请参见Endpoint。 **说明** VPC环境下为必填。
project	表所属的project名称。	是	无。
tableName	表名。	是	无。
accessId	AccessKey ID。	是	无。
accessKey	AccessKey Secret。	是	无。

参数	说明	是否必填	备注

startPartition	指定读取的起始分区。系统加载分区列表时，会把每个分区列表的所有分区和startPartition按照字母顺序进行比较，加载满足条件的分区的数据。
	此外，增量MaxCompute源表可以持续监听增量MaxCompute分区表。读完已有的分区后，任务不会退出，且持续监听并读入新分区。
	说明
	• 增量MaxCompute源表中必须存在一级分区，二级分区可选。
	• 如果指定二级分区，必须写在一级分区的后面。
	• 如果startPartition指定的分区不存在，系统会从下一个分区开始读取数据。
		是	例如，指定startPartition是ds=20191201，表示加载增量MaxCompute表里所有满足ds >= 20191201的分区数据。
			如果一个增量MaxCompute分区表，有一级分区ds和二级分区type两个分区列，假设表里有以下5个分区：
			• ds=20191201,type=a
			• ds=20191201,type=b
			• ds=20191202,type=a
			• ds=20191202,type=b
			• ds=20191202,type=c
			不同startPartition，满足分区的列表如下：
			• ds=20191202
			  ○ ds=20191202,type=a
			  ○ ds=20191202,type=b
			  ○ ds=20191202,type=c
			• ds=20191201,type=c
			  ○ ds=20191202,type=a
			  ○ ds=20191202,type=b
			  ○ ds=20191202,type=c

类型映射

MaxCompute字段类型	Flink字段类型
TINYINT	TINYINT
SMALLINT	SMALLINT
INT	INT
BIGINT	BIGINT
FLOAT	FLOAT
DOUBLE	DOUBLE
BOOLEAN	BOOLEAN
DATETIME	TIMESTAMP
TIMESTAMP	TIMESTAMP

MaxCompute字段类型	Flink字段类型
DECIMAL	DECIMAL
BINARY	VARBINARY
STRING	VARCHAR

注意

- 增量MaxCompute源表暂不支持CHAR、VARCHAR、ARRAY、MAP和STRUCT数据类型。
- 您可以临时使用STRING替换VARCHAR。

代码示例

增量MaxCompute源表每天产生一个分区，分区列是ds，从ds=20191201分区开始，加载后续的所有分区，并一直监听新分区的产生。

```
--读增量MaxCompute表，读取的分区范围是[ds=20191201，∞)。
CREATE TEMPORARY TABLE odps_source (
  cid VARCHAR,
  rt DOUBLE
) with (
  'connector' = 'continuous-odps',
  'endpoint' = '<yourEndpoint>',
  'tunnelEndpoint' = '<yourTunnelEndpoint>',
  'project' = '<yourProjectName>',
  'tablename' = '<yourTableName>',
  'accessid' = '<yourAccessKeyId>',
  'accesskey' = '<yourAccessKeySecret>',
  'startpartition' = 'ds=2018****'
);
CREATE TEMPORARY TABLE blackhole_sink (
  cid VARCHAR,
  rt DOUBLE
) WITH (
  'connector'='blackhole'
);
INSERT INTO blackhole_sink
SELECT
    cid, rt FROM odps_source;
```

常见问题

- endPoint和tunnelEndpoint是指什么？如果配置错误会产生什么结果？
- 启动作业时出现Akka超时报错，但是全量MaxCompute源表和增量MaxCompute获取Metadata速率正常，应该如何处理？
- 全量MaxCompute和增量MaxCompute源表作业是否支持暂停作业后修改并发数，再恢复作业？
- 全量MaxCompute和增量MaxCompute是如何读取MaxCompute数据的？
- 引用MaxCompute作为数据源，在作业启动后，向已有的分区或者表里追加数据，这些新数据是否能被全

量MaxCompute或增量MaxCompute源表读取？
- 作业启动位点设置了2019-10-11 00:00:00， 为什么启动位点前的分区也会被全量MaxCompute源表读取？
- 增量MaxCompute源表监听到新分区时，如果该分区还有数据没有写完，如何处理？
- 报错：ErrorMessage=Authorization Failed [4019], You have NO privilege'ODPS:***'

9.2.4.1.6. 消息队列RocketMQ版源表

本文为您介绍消息队列RocketMQ版源表DDL定义、WITH参数、类型映射、属性字段和代码示例。

> **说明** RocketMQ Connector可以作为Stream作业和Batch作业的源表使用。

什么是消息队列RocketMQ版

消息队列RocketMQ版是阿里云基于Apache RocketMQ构建的低延迟、高并发、高可用和高可靠的分布式消息中间件。消息队列RocketMQ版既可为分布式应用系统提供异步解耦和削峰填谷的能力，同时也具备互联网应用所需的海量消息堆积、高吞吐和可靠重试等特性。

前提条件

已创建了RocketMQ资源，详情请参见创建资源。

使用限制

仅Flink计算引擎VVR 2.0.0及以上版本支持消息队列RocketMQ Connector。

DDL定义

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

> **说明** RocketMQ是非结构化存储格式的消息中间件，对于数据的Schema不提供强制定义，完全由业务层指定。Flink仅支持CSV和二进制格式的RocketMQ消息。

WITH参数

参数	说明	是否必填	备注
connector	源表类型。	是	固定值为 `mq` 。
topic	topic名称。	是	无。

endPoint	EndPoint地址。	是	阿里云消息队列RocketMQ版接入地址支持以下两种类型：
			• VVR 3.0.1及以上版本的作业，需要使用TCP协议客户端接入点，详情请参见关于TCP内网接入点设置的公告。接入点获取方式如下：
			  ○ 内网服务MQ（阿里云经典网络/VPC）接入地址：在MQ控制台目标实例详情中，选择**接入点 > TCP协议客户端接入点 > 内网访问**，获取对应的EndPoint。
			  ○ 公网服务MQ接入地址：在MQ控制台目标实例详情中，选择**接入点 > TCP协议客户端接入点 > 公网访问**，获取对应的EndPoint。
			• VVR 3.0.1（不含）以下版本的作业，使用如下接入点：
			  ○ 内网服务MQ（阿里云经典网络/VPC）接入地址：
			  ■ 华东1（杭州）、华东2（上海）、华北1（青岛）、华北2（北京）、华南1（深圳）、中国（香港）：`onsaddr-internal.aliyun.com:8080`
			  ■ 亚太东南1（新加坡）：`ap-southeastaddr-internal.aliyun.com:8080` 。
			  ■ 中东东部1（迪拜）：`ons-me-east-1-internal.aliyuncs.com:8080` 。
			  ■ 亚太南部1（孟买）：`ons-ap-south-1-internal.aliyuncs.com:8080` 。
			  ■ 亚太东南3（吉隆坡）：`ons-ap-southeast-3-internal.aliyun.com:8080` 。
			  ○ 公网服务MQ接入地址：`onsaddr-internet.aliyun.com:80` 。

参数	说明	是否必填	备注
			注意 • 如果您已使用了VVR 3.0.1（不含）以下版本的RocketMQ Connector，则您需要将您的实时计算作业升级至VVR 3.0.1及以上版本，并将作业中EndPoint参数取值更改为新的RocketMQ接入点，旧的RocketMQ接入点存在稳定性风险或不可用的问题，详情请参见RocketMQ接入点变更导致实时计算作业适配升级公告。 • 内网服务无法跨域访问。例如，您所购买的实时计算服务的地域为华东1，但是购买的消息队列MQ服务的地域为华东2（上海），则无法访问。 • 由于阿里云网络安全策略动态变化，实时计算连接公网服务MQ时可能会出现网络连接问题，推荐您使用内网服务MQ，如果在使用公网服务MQ时出现异常，请您提交工单进行咨询。
accessId	AccessKey ID。	是	无。
accessKey	AccessKey Secret。	是	无。
consumerGroup	Consumer组名。	是	无。
pullIntervalMs	拉取数据的时间间隔。	是	单位为毫秒。
nameServerSubgroup	NameServer组。	否	• 内网服务（阿里云经典网络或VPC）：必须配置'nameServerSubgroup' = 'nsaddr4client-internal'。 • 公网服务：无需配置nameServerSubgroup。 说明 仅VVR 2.1.1 ~ VVR 3.0.0版本支持该参数，VVR 3.0.1及以后版本不支持该参数。
timeZone	时区。	否	例如，Asia/Shanghai。
startTimeMs	启动时间点。	否	无。
startMessageOffset	消息开始的偏移量。	否	如果填写该参数，则优先以startMessageoffset的位点开始加载数据。
tag	订阅的标签。	否	MQ作为数据源时，只能读取单个tag。
lineDelimiter	解析Block时，行分隔符。	否	默认值为 `\n` 。
fieldDelimiter	字段分隔符。	否	根据MQ终端的模式，分隔符分别为： • 在只读模式下（默认模式），分隔符为 `u&'\0001` 。该模式下，分隔符不可见。 • 在编辑模式下，分隔符为 `^A` 。
encoding	编码格式。	否	默认值为 `utf-8` 。

参数	说明	是否必填	备注

lengthCheck	单行字段条数检查策略。	否	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。	否	默认值为FALSE。如果设置为TRUE，则打印解析异常的Log。
instanceID	实例ID。	否	根据RocketMQ实例是否有独立命名空间，执行如下操作： • 是，必须配置instanceID参数。 • 否，不能配置instanceID参数。

类型映射

RocketMQ字段类型	Flink字段类型
STRING	VARCHAR

属性字段

字段名	字段类型	说明
topic	VARCHAR METADATA VIRTUAL	RocketMQ消息Topic。
queue-id	INT METADATA VIRTUAL	RocketMQ消息队列ID。
queue-offset	BIGINT METADATA VIRTUAL	RocketMQ消息队列的消费位点。
msg-id	VARCHAR METADATA VIRTUAL	RocketMQ消息ID。
store-timestamp	TIMESTAMP(3) METADATA VIRTUAL	RocketMQ消息存储时间。
born-timestamp	TIMESTAMP(3) METADATA VIRTUAL	RocketMQ消息生成时间。
keys	VARCHAR METADATA VIRTUAL	RocketMQ消息Keys。

字段名	字段类型	说明
tags	VARCHAR METADATA VIRTUAL	RocketMQ消息Tags。

说明 仅在VVR 3.0.1及以上版本支持获取以上RocketMQ属性字段。

代码示例

- CSV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RocketMQ消息可以包括0条到多条数据记录，记录之间使用 `\n` 分隔。

Flink作业中，声明RocketMQ数据源表的DDL如下。

```
create table mq_source(
   id varchar,
   name varchar,
   gender varchar,
   topic varchar metadata virtual
) with (
   'connector' = 'mq',
   'topic' = 'mq-test',
   'endpoint' = '<yourEndpoint>',
   'pullIntervalMs' = '1000',
   'accessId' = '<yourAccessId>',
   'accessKey' = '<yourAccessSecret>',
   'startMessageOffset' = '1000',
   'consumerGroup' = 'mq-group',
   'fieldDelimiter' = '|'
);
```

- 二进制格式

```
create temporary table source_table (
  mess varbinary
) with (
  'connector' = 'mq',
  'endpoint' = '<yourEndpoint>',
  'pullIntervalMs' = '500',
  'accessId' = '<yourAccessId>',
  'accessKey' = '<yourAccessSecret>',
  'topic' = 'mq-test',
  'consumerGroup' = 'mq-group'
);
create temporary table out_table (
  commodity varchar
) with (
  'connector' = 'print'
);
insert into out_table
select
  cast(mess as varchar)
from source_table;
```

9.2.4.1.7. 交互式分析Hologres源表

本文为您介绍交互式分析Hologres源表DDL定义、WITH参数、代码示例和类型映射。

> **说明** Hologres Connector可以作为Stream作业和Batch作业的源表使用。

什么是交互式分析Hologres

交互式分析Hologres兼容PostgreSQL协议，与大数据生态紧密连接，支持高并发、低延时实时分析处理PB级数据，让您轻松使用现有BI（Business Intelligence）工具对数据进行多维分析和业务探索。

前提条件

已创建Hologres表，详情请参见表。

使用限制

- 仅Flink计算引擎VVR 2.0.0及以上版本支持交互式分析Hologres Connector。
- Hologres Connector不支持访问Hologres外部表。Hologres外部表详情请参见外部表。
- Flink默认以批模式读取Hologres源表数据，即只扫描一次Hologres全表，扫描结束，消费结束，新到Hologres源表的数据不会被读取。从VVR 3.0.0版本开始，支持实时消费Hologres数据。如何实时消费Hologres数据，请参见Flink实时消费Hologres。

DDL定义

```
create table hologres_source(
  name varchar,
  age BIGINT,
  birthday BIGINT
) with (
  'connector'='hologres',
  'dbname'='<yourDbname>',
  'tablename'='<yourTablename>',
  'username'='<yourAccessID>',
  'password'='<yourAccessSecret>',
  'endpoint'='<yourEndpoint>',
  'field_delimiter'='|' --该参数可选。
);
```

说明 Flink不支持在源表中定义计算列。

WITH参数

参数	说明	是否必填	备注
connector	源表类型。	是	固定值为*hologres*。
dbname	数据库名称。	是	无。
tablename	表名称。 **说明** 如果Schema不为Public时，需要在表名称前面添加模式名称的前缀，即schema.tableName。	是	无。
username	用户名，请填写阿里云账号的AccessKey ID。	是	无。
password	密码，请填写阿里云账号的AccessKey Secret。	是	无。
endpoint	Hologres服务地址。	是	详情请参见访问域名。
field_delimiter	导出数据时，不同行之间使用的分隔符。 **说明** 只有在bulkread = true时有效。	否	默认值为*"\u0002"*。

参数	说明	是否必填	备注
binlog	是否消费Binlog数据。	否	参数取值如下： • true：消费Binlog数据。 • false（默认值）：不消费Binlog数据。 注意 仅VVP 2.4.0及以上版本支持该参数。
binlogMaxRetryTimes	读取Binlog数据出错后的重试次数。	否	默认值为60。 注意 仅VVP 2.4.0及以上版本支持该参数。
binlogRetryIntervalMs	读取Binlog数据出错后的重试时间间隔。	否	默认值为2000，单位为毫秒。 注意 仅VVP 2.4.0及以上版本支持该参数。
binlogBatchReadSize	批量读取Binlog的数据行数。	否	默认值为16。 注意 仅VVP 2.4.0及以上版本支持该参数。
cdcMode	是否采用CDC模式读取Binlog数据。	否	参数取值如下： • true：CDC模式读取Binlog数据。 • false（默认值）：非CDC模式读取Binlog数据。 注意 仅VVP 2.4.0及以上版本支持该参数。
startTime	启动位点的时间。如果没有设置该参数，且作业没有从state恢复，则从最早的Binlog开始消费Hologres数据。	否	格式为yyyy-MM-dd hh:mm:ss。

代码示例

```
CREATE TEMPORARY TABLE hologres_source (
  name varchar,
  age BIGINT,
  birthday BIGINT
) with (
  'connector'='hologres',
  'dbname'='<yourDbname>',
  'tablename'='<yourTablename>',
  'username'='<yourAccessID>',
  'password'='<yourAccessSecret>',
  'endpoint'='<yourEndpoint>',
  'field_delimiter'='|' --该参数可选。
);
CREATE TEMPORARY TABLE blackhole_sink(
  name varchar,
  age BIGINT,
  birthday BIGINT
) with (
  'connector'='blackhole'
);
INSERT INTO blackhole_sink
SELECT
   name, age, birthday
from hologres_source;
```

类型映射

交互式分析Hologres和Flink字段类型对应关系如下，建议使用该对应关系进行DDL声明。

Hologres字段类型	Flink字段类型
INT	INT
INT[]	ARRAY<INT>
BIGINT	BIGINT
BIGINT[]	ARRAY<BIGINT>
REAL	FLOAT
REAL[]	ARRAY<REAL>
DOUBLE PRECISION	DOUBLE
DOUBLE PRECISION[]	ARRAY<DOUBLE PRECISION>
BOOLEAN	BOOLEAN
BOOLEAN[]	ARRAY<BOOLEAN>
TEXT	VARCHAR

Hologres字段类型	Flink字段类型
TEXT[]	ARRAY<TEXT>
NUMERIC	DECIMAL
DATE	DATE
TIMESTAMP WITH TIMEZONE（别名TIMESTAMPTZ）	TIMESTAMP

Flink实时消费Hologres

VVP 2.4.0及以上版本，Hologres Connector支持实时消费Hologres。实时消费Hologres，即实时消费Hologres的Binlog数据。以下为您介绍Flink实时消费Hologres的详情：

- 使用限制
  - 已存在的表无法修改表属性开启Binlog，需要重新建表。
  - 不支持开启分区表父表的Binlog，请使用非分区表。
  - 暂不支持实时消费TIMESTAMP类型的数据，因此创建Hologres表时，请使用TIMESTAMPTZ类型。
  - 开启Binlog后，理论上列存表的开销要大于行存表的开销。因此，对于数据更新频繁的场景，建议使用存储格式为行存的表开启Binlog。
- 开启Binlog

  实时消费功能默认关闭，因此在Hologres控制台上创建表的DDL时，需要设置binlog.level和binlog.ttl参数，示例如下。

```
begin;
create table test_message_src(
 id int primary key,
 title text not null,
 body text
);
call set_table_property('test_message_src', 'orientation', 'row');
call set_table_property('test_message_src', 'clustering_key', 'id');
call set_table_property('test_message_src', 'binlog.level', 'replica');
call set_table_property('test_message_src', 'binlog.ttl', '86400');
commit;
```

  其中，binlog.level设置为replica即代表开启Binlog，binlog.ttl为Binlog的TTL，单位为秒。
- 注意事项
  - UPDATE操作会产生两条Binlog记录，分别为更新操作前和操作后的数据记录，因此您会消费到两条数据。但是，Hologres Binlog功能会保证这两条记录是连续的且更新前的Binlog记录在前，更新后的Binlog记录在后。
  - 建议Flink作业并发数和Hologres Table的Shard个数保持一致。

    您可以在Hologres控制台上，使用以下语句查看Table的Shard数，其中tablename为您的业务表名称。

```
select tg.property_value from hologres.hg_table_properties tb join hologres.hg_table_gr
oup_properties tg on tb.property_value = tg.tablegroup_name where tb.property_key = 'ta
ble_group' and tg.property_key = 'shard_count' and table_name = '<tablename>';
```

- 消费模式
  - 非CDC模式

    该模式下Source消费的Binlog数据是作为普通的Flink数据传递给下游节点的，即所有数据都是INSERT类型的数据，您可以根据业务情况选择如何处理特定hg_binlog_event_type类型的数据。源表DDL代码示例如下。

```
create table test_message_src_binlog_table(
  hg_binlog_lsn BIGINT,
  hg_binlog_event_type BIGINT,
  hg_binlog_timestamp_us BIGINT,
  id INTEGER,
  title VARCHAR,
  body VARCHAR
) with (
  'connector'='hologres',
  'dbname'='<yourDbname>',
  'tablename'='<yourTablename>',
  'username'='<yourAccessID>',
  'password'='<yourAccessSecret>',
  'endpoint'='<yourEndpoint>',
  'binlog' = 'true',
  'binlogMaxRetryTimes' = '10',
  'binlogRetryIntervalMs' = '500',
  'binlogBatchReadSize' = '100'
);
```

  - CDC模式

    该模式下Source消费的Binlog数据，将根据hg_binlog_event_type自动为每行数据设置准确的Flink RowKind类型，例如，INSERT、DELETE、UPDATE_BEFORE和UPDATE_AFTER类型，这样就能完成表的数据的镜像同步，类似MySQL或Postgres的CDC功能。源表DDL代码示例如下。

```
create table test_message_src_binlog_table(
  id INTEGER,
  title VARCHAR,
  body VARCHAR
) with (
  'connector'='hologres',
  'dbname'='<yourDbname>',
  'tablename'='<yourTablename>',
  'username'='<yourAccessID>',
  'password'='<yourAccessSecret>',
  'endpoint'='<yourEndpoint>',
  'binlog' = 'true',
  'cdcMode' = 'true',
  'binlogMaxRetryTimes' = '10',
  'binlogRetryIntervalMs' = '500',
  'binlogBatchReadSize' = '100'
);
```

- 实现原理

  一条Binlog的字段由Binlog系统字段和用户Table字段组成，字段定义如下：

字段名	字段类型	说明
hg_binlog_lsn	BIGINT	Binlog系统字段，表示Binlog序号，Shard内部单调递增不保证连续，不同Shard之间不保证唯一和有序。
hg_binlog_event_type	BIGINT	Binlog系统字段，表示当前记录所表示的修改类型，参数取值如下： ○ INSERT=5：表示当前Binlog为插入一条新的记录。 ○ DELETE=2：表示当前Binlog为删除一条已有的记录。 ○ BEFORE_UPDATE=3：表示当前Binlog为更新操作前的记录。 ○ AFTER_UPDATE=7：表示当前Binlog为更新操作后的记录。
hg_binlog_timestamp_us	BIGINT	Binlog系统字段，系统时间戳，单位为μs。
user_table_column_1	用户定义	用户的表字段。
...	...	用户的表字段。
user_table_column_n	用户定义	用户的表字段。

9.2.4.1.8. 全量Elasticsearch源表

本文为您介绍全量Elasticsearch（ES）源表DDL定义、WITH参数、类型映射和代码示例。

> **说明** Elasticsearch Connector可以作为Stream作业和Batch作业的源表使用。

什么是云数据库Elasticsearch

阿里云Elasticsearch兼容开源Elasticsearch的功能，以及Security、Machine Learning、Graph、APM等商业功能，致力于数据分析、数据搜索等场景服务。为您提供企业级权限管控、安全监控告警、自动报表生成等场景服务。

前提条件

- 已创建Elasticsearch索引，详情请参见步骤一：创建实例。
- 已配置Elasticsearch公网或私网访问白名单，详情请参见配置实例公网或私网访问白名单。

使用限制

- 仅支持Elasticsearch 5.5及以上版本。
- 仅Flink计算引擎VVR 2.0.0及以上版本支持全量Elasticsearch Connector。
- 仅支持全量Elasticsearch源表，不支持增量Elasticsearch源表。

DDL定义

```
 CREATE TABLE elasticsearch_source(
  name STRING,
  location STRING,
  `value` FLOAT
) WITH (
  'connector' ='elasticsearch',
  'endPoint' = '<yourEndPoint>',
  'accessId' = '<yourAccessId>',
  'accessKey' = '<yourAccessSecret>',
  'indexName' = '<yourIndexName>',
  'typeNames' = '<yourTypeName>'
);
```

说明 DDL中的字段均对应Elasticsearch文档中的字段，不支持将文档ID写入表中。

WITH参数

参数	说明	是否必填	备注
connector	源表类型。	是	固定值为 `elasticsearch` 。
endPoint	Server地址。	是	例如：*http://127.0.0.1:9200*。
accessId	Elasticsearch实例的用户名。	否	无。
accessKey	Elasticsearch实例的密码。	否	无。
indexName	索引名称。	是	无。
typeNames	Type名称。	否	默认值为 `_doc` 。 **说明** Elasticsearch 7.0以上版本不建议设置该参数。
batchSize	每个scroll请求从Elasticsearch集群获取的最大文档数。	否	默认值为2000。
keepScrollAliveSecs	scroll上下文保留的最长时间。	否	单位为秒，默认值为3600。

类型映射

Flink以JSON来解析Elasticsearch数据，详情请参见数据类型映射关系。

代码示例

```
CREATE TEMPORARY TABLE elasticsearch_source (
  name STRING,
  location STRING,
  `value` FLOAT
) WITH (
  'connector' ='elasticsearch',
  'endPoint' = '<yourEndPoint>',
  'accessId' = '<yourAccessId>',
  'accessKey' = '<yourAccessSecret>',
  'indexName' = '<yourIndexName>',
  'typeNames' = '<yourTypeName>'
);
CREATE TEMPORARY TABLE blackhole_sink (
  name STRING,
  location STRING,
  `value` FLOAT
) WITH (
  'connector' ='blackhole'
);
INSERT INTO blackhole_sink
SELECT name, location, `value`
FROM elasticsearch_source;
```

9.2.4.1.9. Postgres的CDC源表（公测中）

本文为您介绍Postgres的CDC（Change Data Capture）源表DDL定义、WITH参数、类型映射和代码示例。

> **注意** Postgres的CDC源表正处于公测中，如果您对作业稳定性要求较高时，建议不要使用Postgres的CDC源表。

什么是Postgres的CDC源表

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

> **说明** 如果您使用的是开源的PostgreSQL数据库，Postgres CDC connector支持读取的Postgres版本为9.6及以上版本。

前提条件

已创建Postgres数据库和表，创建RDS Postgres数据库和表的详情请参见创建数据库和账号。

使用限制

仅Flink计算引擎VVR 2.1.2及以上版本支持Postgres的CDC Connector。

DDL定义

```
CREATE TABLE postgrescdc_source (
  shipment_id INT,
  order_id INT,
  origin STRING,
  destination STRING,
  is_arrived BOOLEAN
) WITH (
  'connector' = 'postgres-cdc',
  'hostname' = '<yourHostname>',
  'port' = '5432',
  'username' = '<yourUserName>',
  'password' = '<yourPassWord>',
  'database-name' = '<yourDatabaseName>',
  'schema-name' = '<yourSchemaName>',
  'table-name' = '<yourTableName>'
);
```

WITH参数

参数	说明	是否必填	数据类型	备注
connector	源表类型	是	STRING	固定值为 `postgres-cdc` 。
hostname	Postgres数据库的IP地址或者Hostname。	是	STRING	无
username	Postgres数据库服务的用户名。	是	STRING	无
password	Postgres数据库服务的密码	是	STRING	无
database-name	数据库名称	是	STRING	数据库名称支持正则表达式以读取多个数据库的数据。
schema-name	Postgres Schema名称	是	STRING	Schema名称支持正则表达式以读取多个Schema的数据。
table-name	Postgres表名	是	STRING	表名支持正则表达式去读取多个表的数据。
port	Postgres数据库服务的端口号	否	INTEGER	默认值为5432。

参数	说明	是否必填	数据类型	备注
decoding.plugin.name	Postgres Logical Decoding插件名称	否	STRING	根据Postgres服务上安装的插件确定。支持的插件列表如下： • decoderbufs（默认值） • wal2json • wal2json_rds • wal2json_streaming • wal2json_rds_streaming • pgoutput 说明 如果您使用的是阿里云RDS PostgreSQL，你需要开启逻辑解码（wal2json）功能，详情请参见逻辑解码（wal2json）。
debezium.*	Debezium属性参数	否	STRING	更细粒度控制Debezium客户端的行为。例如 `'debezium.snapshot.mode' = 'never'`，详情请参见配置属性。 说明 建议每个表都设置 `debezium.slot.name` 参数，以避免出现 `PSQLException: ERROR: replication slot "debezium" is active for PID 974` 报错。

类型映射

Postgres CDC和Flink字段类型对应关系如下。

Postgres CDC字段类型	Flink字段类型
SMALLINT	SMALLINT
INT2	
SMALLSERIAL	
SERIAL2	
INTEGER	INT
SERIAL	
BIGINT	BIGINT
BIGSERIAL	
BIGINT	BIGINT

Postgres CDC字段类型	Flink字段类型
REAL	FLOAT
FLOAT4	
FLOAT8	DOUBLE
DOUBLE PRECISION	
NUMERIC(p, s)	DECIMAL(p, s)
DECIMAL(p, s)	
BOOLEAN	BOOLEAN
DATE	DATE
TIME [(p)] [WITHOUT TIMEZONE]	TIME [(p)] [WITHOUT TIMEZONE]
TIMESTAMP [(p)] [WITHOUT TIMEZONE]	TIMESTAMP [(p)] [WITHOUT TIMEZONE]
CHAR(n)	STRING
CHARACTER(n)	
VARCHAR(n)	
CHARACTER VARYING(n)	
TEXT	
BYTEA	BYTES

9.2.4.1.10. MySQL的CDC源表

本文为您介绍MySQL的CDC（Change Data Capture）源表DDL定义、WITH参数、类型映射和代码示例。

什么是MySQL的CDC源表

MySQL的CDC源表，即MySQL的流式源表，会先读取数据库的历史全量数据，并平滑切换到Binlog读取上，保证不多读一条也不少读一条数据。即使发生故障，也能保证通过Exactly Once语义处理数据。MySQL CDC Connector支持并发地读取全量数据，通过增量快照算法实现了全程无锁和断点续传。

支持以下核心特性：

- 流批一体，支持读取全量和增量数据，无需维护两套流程。
- 支持并发读取全量数据，性能水平扩展。
- 全量读取无缝切换增量读取，自动缩容，节省计算资源。
- 全量阶段读取支持断点续传，更稳定。
- 无锁读取全量数据，不影响在线业务。

实现原理

Source在启动时会扫描全表，将表按照主键分成多个chunk。并使用增量快照算法逐个读取每个chunk的数据。作业会周期性执行Checkpoint，记录下已经完成的chunk。当发生Failover时，只需要继续读取未完成的chunk。当chunk全部读取完后，会从之前获取的Binlog位点读取增量的变更记录。Flink作业会继续周期性执行Checkpoint，记录下Binlog位点，当作业发生Failover，便会从之前记录的Binlog位点继续处理，从而实现Exactly Once语义。更详细的增量快照算法，请参见MySQL CDC Connector。

前提条件

- MySQL和VVP的网络连通。
- MySQL服务器配置如下：
  - MySQL版本为5.7和8.0.X。
  - 已开启了Binlog。
  - Binlog格式已设置为ROW。
  - binlog_row_image已设置为FULL。
- 已在MySQL配置文件中配置了交互超时或等待超时参数。
- 已创建MySQL用户，并授予了SELECT、 SHOW DATABASES 、REPLICATION SLAVE和REPLICATION CLIENT权限。

> **说明** 以上配置需要在RDS MySQL、PolarDB MySQL或者自建MySQL上的操作，详情请参见配置MySQL。

使用限制

- 仅vvr-4.0.8-flink-1.13及以上引擎版本支持无锁读取和并发读取功能。
- MySQL CDC Connector支持读取的MySQL版本为5.7和8.0.X。

> **说明** VVR 4.0.11及以上版本支持读取MySQL 5.6。

- MySQL CDC 源表暂不支持定义Watermark。如果您需要进行窗口聚合，您可以采用非窗口聚合的方式，详情请参见不支持定义Watermark，那如何进行窗口聚合？。
- MySQL的CDC源表需要一个有特定权限（包括SELECT、SHOW DATABASES、REPLICATION SLAVE和REPLICATION CLIENT）的MySQL用户，才能读取全量和增量数据。

注意事项

- 每个作业需显式配置不同的Server ID。

  每个同步数据库数据的客户端，都会有一个唯一ID，即Server ID。MySQL SERVER会根据该ID来维护网络连接以及Binlog位点。因此如果有大量不同的Server ID的客户端一起连接MySQL SERVER，可能导致MySQL SERVER的CPU陡增，影响线上业务稳定性。

  此外，多个作业共享相同的Server ID，会导致Binlog位点错乱，多读或少读数据。因此建议每个CDC作业都配置不同的Server ID。建议通过动态Hints来配置Server ID，而不是在DDL参数中配置Server ID。配置不同的Server ID代码示例如下。

  ```
  SELECT * FROM source_table  /*+ OPTIONS('server-id'='123456') */ ;
  ```

  动态Hints详情请参见动态Hints。

> **说明** 如果开启增量快照读取数据，则server-id配置项需要配置成与作业并发匹配的Server ID范围。

- 仅VVR 4.0.8 及以上版本支持全量阶段的无锁读取、并发读取、断点续传等功能。

  如果您使用的是VVR 4.0.8以下版本，需要对MySQL用户授予RELOAD权限用来获取全局读锁，保证数据读取的一致性。全局读锁会阻塞写入操作，持锁时间可能达到秒级，因此可能对线上业务造成影响。

  此外，VVR 4.0.8以下版本在全量读取阶段无法执行Checkpoint，全量阶段的作业失败会导致作业重新读取全量数据，稳定性不佳。因此建议您将作业升级到VVR 4.0.8及以上版本。

DDL定义

```
CREATE TABLE mysqlcdc_source (
   order_id INT,
   order_date TIMESTAMP(0),
   customer_name STRING,
   price DECIMAL(10, 5),
   product_id INT,
   order_status BOOLEAN,
   PRIMARY KEY(order_id) NOT ENFORCED
) WITH (
  'connector' = 'mysql-cdc',
  'hostname' = '<yourHostname>',
  'port' = '3306',
  'username' = '<yourUsername>',
  'password' = '<yourPassword>',
  'database-name' = '<yourDatabaseName>',
  'table-name' = '<yourTableName>'
);
```

WITH参数

参数	说明	是否必填	数据类型	备注
connector	源表类型。	是	STRING	可以填写为 `mysql-cdc` 或者 `mysql` ，二者等价。
hostname	MySQL数据库的IP地址或者Hostname。	是	STRING	无。
username	MySQL数据库服务的用户名。	是	STRING	无。
password	MySQL数据库服务的密码。	是	STRING	无。
database-name	MySQL数据库名称。	是	STRING	数据库名称支持正则表达式以读取多个数据库的数据。
table-name	MySQL表名。	是	STRING	表名支持正则表达式以读取多个表的数据。

参数	说明	是否必填	数据类型	备注
port	MySQL数据库服务的端口号。	否	INTEGER	默认值为3306。
server-id	数据库客户端的一个数字 ID。	否	STRING	该ID必须是MySQL集群中全局唯一的。建议针对同一个数据库的每个作业都设置一个不同的ID。默认会随机生成一个5400~6400的值。 该参数也支持ID范围的格式，例如5400-5408。在开启增量读取模式时支持多并发读取，此时推荐设定为ID范围，使得每个并发使用不同的ID。
scan.incremental.snapshot.enabled	是否开启增量快照。	否	BOOLEAN	默认开启增量快照。增量快照是一种读取全量数据快照的新机制。与旧的快照读取相比，增量快照有很多优点，包括： • 读取全量数据时，Source可以是并行读取。 • 读取全量数据时，Source支持chunk粒度的检查点。 • 读取全量数据时，Source不需要获取全局读锁（FLUSH TABLES WITH read lock）。 如果您希望Source支持并发读取，每个并发的Reader需要有一个唯一的服务器ID，因此server-id必须是5400-6400这样的范围，并且范围必须大于等于并发数。
scan.incremental.snapshot.chunk.size	表的chunk的大小（行数）。	否	Integer	默认值为8096。当开启增量快照读取时，表会被切分成多个chunk读取。在读完chunk的数据之前，chunk的数据会先缓存在内存中，因此chunk太大，可能导致内存OOM。chunk越小，故障恢复的粒度也越小，但也会降低吞吐。
scan.snapshot.fetch.size	当读取表的全量数据时，每次最多拉取的记录数。	否	Integer	默认值为1024。
scan.startup.mode	消费数据时的启动模式。	否	STRING	参数取值如下： • initial（默认）：在第一次启动时，会先扫描历史全量数据，然后读取最新的Binlog数据。 • latest-offset：在第一次启动时，不会扫描历史全量数据，直接从Binlog的末尾（最新的Binlog处）开始读取，即只读取该Connector启动以后的最新变更。 。
server-time-zone	数据库在使用的会话时区。	否	STRING	例如Asia/Shanghai，该参数控制了MySQL中的TIMESTAMP类型如何转成STRING类型。更多信息请参见Debezium时间类型。

参数	说明	是否必填	数据类型	备注
debezium.min.row.count.to.stream.results	当表的条数大于该值时，会使用分批读取模式。	否	INTEGER	默认值为1000。Flink采用以下方式读取MySQL源表数据： • 全量读取：直接将整个表的数据读取到内存里。优点是速度快，缺点是会消耗对应大小的内存，如果源表数据量非常大，可能会有OOM风险。 • 分批读取：分多次读取，每次读取一定数量的行数，直到读取完所有数据。优点是读取数据量比较大的表没有OOM风险，缺点是读取速度相对较慢。
connect.timeout	在尝试连接MySQL数据库服务器之后，连接器在超时之前应该等待的最大时间。	否	Duration	默认值为30秒。

元数据

元数据在分库分表合并同步场景非常实用，因为分库分表合并后，一般业务还是希望区分每条数据的库名和表名来源，而元数据列可以访问源表的库名和表名信息。因此通过元数据列可以非常方便地将多张分表合并到一张目的表。

自vvr-4.0.11-flink-1.13版本开始，MySQL CDC Source支持元数据列语法，您可以通过元数据列访问以下元数据。

元数据key	元数据类型	描述
database_name	STRING NOT NULL	包含该行记录的库名。
table_name	STRING NOT NULL	包含该行记录的表名。
op_ts	TIMESTAMP_LTZ(3) NOT NULL	该行记录在数据库中的变更时间，如果该记录来自表的存量历史数据而不是Binlog中获取，则该值总是0。

将MySQL实例中多个分库下的多张orders表，合并同步到下游Hologres的holo_orders表中，代码示例如下所示。

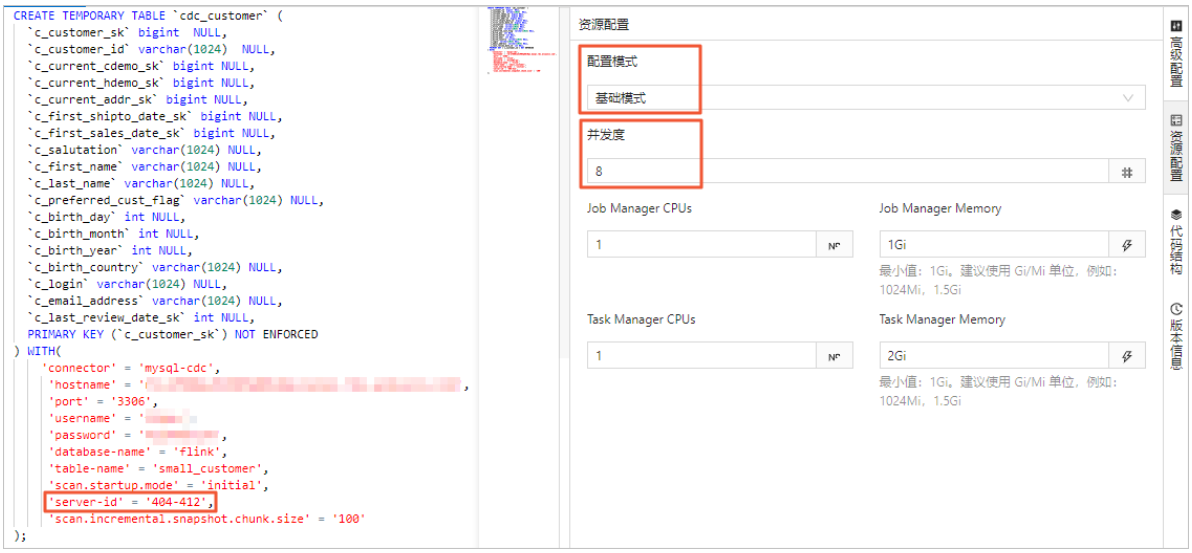
```
CREATE TABLE mysql_orders (
  db_name STRING METADATA FROM 'database_name' VIRTUAL,  -- 读取库名。
  table_name STRING METADATA  FROM 'table_name' VIRTUAL, -- 读取表名。
  operation_ts TIMESTAMP_LTZ(3) METADATA FROM 'op_ts' VIRTUAL, -- 读取变更时间。
  order_id INT,
  order_date TIMESTAMP(0),
  customer_name STRING,
  price DECIMAL(10, 5),
  product_id INT,
  order_status BOOLEAN,
  PRIMARY KEY(order_id) NOT ENFORCED
) WITH (
  'connector' = 'mysql-cdc',
  'hostname' = 'localhost',
  'port' = '3306',
  'username' = 'flinkuser',
  'password' = 'flinkpw',
  'database-name' = 'mydb_.*', -- 正则匹配多个分库。
  'table-name' = 'orders_.*'   -- 正则匹配多张分表。
);
INSERT INTO holo_orders SELECT * FROM mysql_orders;
```

并发控制

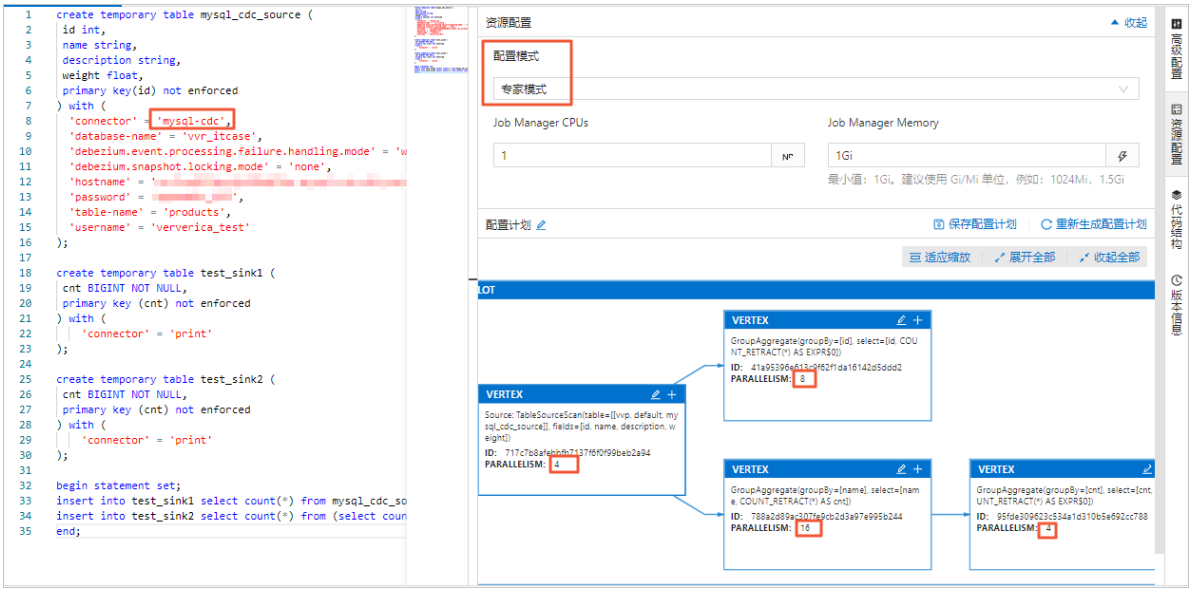
MySQL-CDC connector支持多并发读取全量数据，能够提高数据加载效率。同时配合Flink VVP平台的Autopilot自动调优功能，在多并发读取完成后增量阶段，能够自动缩容，节约计算资源。

在Flink全托管VVP平台，您可以在资源配置页面的基础模式或专家模式中设置作业的并发数。设置并发的区别如下：

- 基础模式设置的并发数为整个作业的全局并发数。



- 专家模式支持按需为某个VERTEX设置并发数。



资源配置详情请参见配置细粒度资源。

> **注意** 无论是基础模式还是专家模式，在设置的并发时，表中声明的server-id范围必须大于等于作业的并发数。例如server-id的范围为5404-5412，则共有8个唯一的server-id，因此作业最多可以设置8个并发，且不同的作业对于同一个MySQL实例的server-id范围不能有重叠，即每个作业需显式配置不同的server-id。

Autopilot自动缩容

全量阶段积累了大量历史数据，为了提高读取效率，通常采用并发的方式读取历史数据。而在Binlog增量阶段，因为Binlog数据量少且为了保证全局有序，通常只需要单并发读取。全量阶段和增量阶段对资源的不同需求，可以通过自动调优功能自动帮您实现性能和资源的平衡。

自动调优会监控MySQL CDC Source的每个task的流量。当进入Binlog阶段，如果只有一个task在负责Binlog读取，其他task均空闲时，自动调优便会自动缩小Source的CU数和并发。开启自动调优只需要在作业运维页面，将自动调优的模式设置为Active模式。

> **说明** 默认调低并发度的最小触发时间间隔为24小时。更多自动调优的参数和细节，请参见配置自动调优。

类型映射

MySQL的CDC和Flink字段类型对应关系如下。

MySQL CDC字段类型	Flink字段类型
TINYINT	TINYINT
SMALLINT	SMALLINT
TINYINT UNSIGNED	
INT	
MEDIUMINT	

MySQL CDC字段类型	Flink字段类型
SMALLINT UNSIGNED	
BIGINT	BIGINT
INT UNSIGNED	
BIGINT UNSIGNED	DECIMAL(20, 0)
BIGINT	BIGINT
FLOAT	FLOAT
DOUBLE	DOUBLE
DOUBLE PRECISION	
NUMERIC(p, s)	DECIMAL(p, s)
DECIMAL(p, s)	
BOOLEAN	BOOLEAN
TINYINT(1)	
DATE	DATE
TIME [(p)]	TIME [(p)] [WITHOUT TIMEZONE]
DATETIME [(p)]	TIMESTAMP [(p)] [WITHOUT TIMEZONE]
TIMESTAMP [(p)]	TIMESTAMP [(p)]
	TIMESTAMP [(p)] WITH LOCAL TIME ZONE
CHAR(n)	STRING
VARCHAR(n)	
TEXT	
BINARY	BYTES
VARBINARY	
BLOB	

常见问题

- 不支持定义Watermark，那如何进行窗口聚合？
- 如何跳过Snapshot阶段，只从变更数据开始读取？

- 如何读取一个分库分表的MySQL数据库？
- 全表读取阶段效率慢、存在反压，应该如何解决？
- 如何判断CDC作业是否已完成全量数据同步？
- 报错：com.github.shyiko.mysql.binlog.network.ServerException
- 报错：The primary key is necessary when enable 'Key: 'scan.incremental.snapshot.enabled'

9.2.4.1.11. Hudi源表

本文为您介绍Hudi的背景信息、使用限制、 DDL定义、WITH参数和代码示例。

背景信息

- 什么是Hudi

  Hudi的定义、特性及典型场景详情如下表所示。

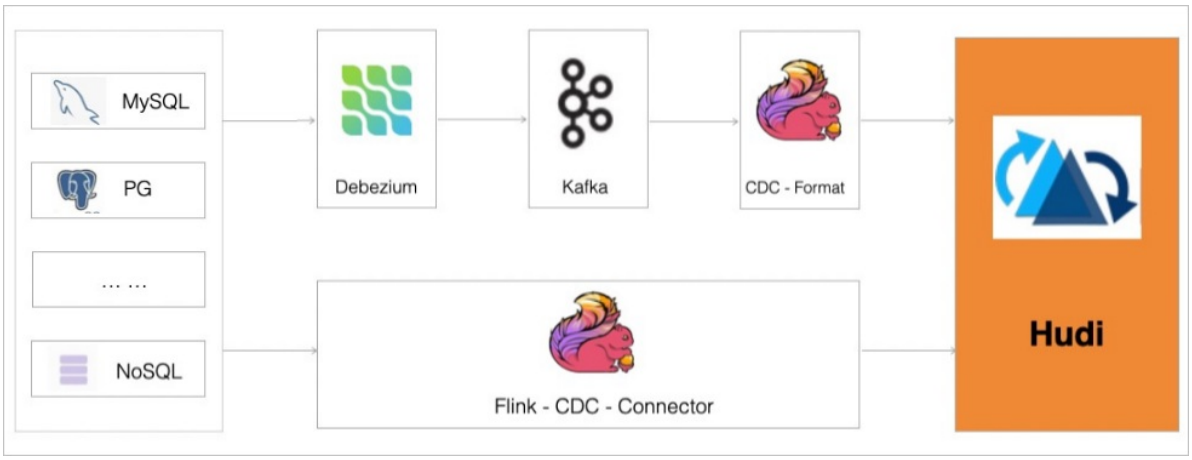
类别	详情
定义	Apache Hudi是一种开源的数据湖表格式框架。Hudi基于对象存储或者HDFS组织文件布局，保证ACID，支持行级别的高效更新和删除，从而降低数据ETL开发门槛。同时该框架还支持自动管理及合并小文件，保持指定的文件大小，从而在处理数据插入和更新时，不会创建过多的小文件，引发查询端性能降低，避免手动监控和重写小文件的运维负担。结合Flink、Presto、Spark等计算引擎进行数据入湖和计算分析，常用来支持DB入湖加速、增量数据实时消费和数仓回填等需求。详情请参见Apache Hudi。
特性	○ 支持ACID：支持ACID语义，提供事务的线性隔离级别。 ○ 支持UPSERT语义：UPSERT语义即就是INSERT和UPDATE两种语义的合并。在UPSERT语义时，如果记录不存在则插入；如果记录存在则更新。通过INSERT INTO语法可以大幅简化开发代码的复杂度，提升效率。 ○ 支持Data Version：通过时间旅行（Time Travel）特性，提供任意时间点的数据版本历史，便于数据运维，提升数据质量。 ○ 支持Schema Evolution：支持动态增加列，类型变更等Schema操作。
典型场景	○ DB入湖加速 相比昂贵且低效的传统批量加载和Merge，Hudi提供超大数据集的实时流式更新写入。通过实时的ETL，您可以直接将CDC（change data capture）数据写入数据湖，供下游业务使用。典型案例为采用Flink MySQL CDC Connector将RDBMS（MySQL）的Binlog写入Hudi表。 ○ 增量ETL 通过增量拉取的方式获取Hudi中的变更数据流，相对离线ETL调度，实时性更好且更轻量。典型场景是增量拉取在线服务数据到离线存储中，通过Flink引擎写入Hudi表，借助Presto或Spark引擎实现高效的OLAP分析。 ○ 消息队列 在小体量的数据场景下，Hudi也可以作为消息队列替代Kafka，简化应用开发架构。 ○ 数仓回填（backfill） 针对历史全量数据进行部分行、列的更新场景，通过数据湖极大减少计算资源消耗，提升了端到端的性能。典型案例是Hive场景下全量和增量的打宽。

- 全托管Flink集成Hudi功能优势

  相比开源社区Hudi，全托管Flink平台集成Hudi具有的功能优势详情如下表所示。

功能优势	详情
平台侧与Flink全托管集成，免运维	Flink全托管内置Hudi Connector，降低运维复杂度，提供SLA保障。
完善的数据连通性	对接多个阿里云大数据计算分析引擎，数据与计算引擎解耦，可以在Flink、Spark、Presto或Hive间无缝流转。
深度打磨DB入湖场景	与Flink CDC Connector联动，降低开发门槛。
提供企业级特性	包括集成DLF统一元数据视图、自动且轻量化的表结构变更。
内置阿里云OSS存储，低成本存储，弹性扩展	数据以开放的Parquet、Avro格式存储在阿里云OSS，存储计算分离，资源灵活弹性扩展。

- CDC数据同步



  CDC数据保存了完整的数据库变更，您可以通过以下任意一种方式将数据导入Hudi：

  - 对接CDC格式，消费Kafka数据的同时导入Hudi。

    支持debezium-json、canal-json和maxwell-json三种格式，该方式优点是可扩展性强，缺点是需要依赖Kafka和Debezium数据同步工具。

  - 通过Flink-CDC-Connector直接对接DB的Binlog，将数据导入Hudi。

    该方式优点是使用轻量化组件，对工具依赖少。

> **说明**
>
> - 如果无法保证上游数据顺序，则需要指定write.precombine.field字段。
> - 在CDC场景下，需要开启changelog模式，即changelog.enabled设为true。

使用限制

- 仅Flink计算引擎vvr-4.0.11-flink-1.13及以上版本支持Hudi Connector。
- 文件系统仅支持HDFS或阿里云OSS服务。
- 不支持以Session模式提交作业。

DDL定义

```
CREATE TEMPORARY TABLE hudi_sink (
  uuid BIGINT,
  data STRING,
  ts   TIMESTAMP(3)
) WITH (
  'connector' = 'hudi',
  'table.type' = 'MERGE_ON_READ',
  'oss.endpoint' = '<yourOSSEndpoint>',
  'path' = 'oss://<yourOSSBucket>/<自定义存储位置>',
  'accessKeyId' = '<yourAccessKeyId>',
  'accessKeySecret' = '<yourAccessKeySecret>',
  'read.streaming.enabled' = 'true'
);
```

WITH参数

- 基础参数

参数	说明	是否必选	备注
connector	源表类型。	是	固定值为 `hudi` 。
table.type	表类型。	是	参数取值如下： ○ COPY_ON_WRITE：使用Parquet列式存储，每次更新数据，创建一个新的base文件。 ○ MERGE_ON_READ：使用Parquet列式和Avro行式存储，更新操作将会被写入delta日志文件，异步合并delta日志文件和Parquet列式文件生成新版本文件。
path	表存储路径。	是	支持阿里云OSS和HDFS两种路径。例如*oss://<bucket name>/table*或*hdfs://<ip>:<port>/table*。
read.streaming.enabled	是否开启流读。	是	参数取值如下： ○ true：开启。 ○ false（默认值）：关闭。
oss.endpoint	阿里云对象存储服务OSS的Endpoint。	否	如果使用OSS作为存储，则必需填写。参数取值请详情参见访问域名和数据中心。
accessKeyId	阿里云账号的AccessKey ID。	否	如果使用OSS作为存储，则必需填写。获取方法请参见获取AccessKey。
accessKeySecret	阿里云账号的AccessKey Secret。	否	如果使用OSS作为存储，则必需填写。获取方法请参见获取AccessKey。

参数	说明	是否必选	备注
hive_sync.enable	是否开启同步元数据到Hive功能。	否	参数取值如下： ○ true：开启。 ○ false：不开启。
hive_sync.mode	Hive数据同步模式。	否	参数取值如下： ○ hms（推荐值）：采用Hive Metastore或者DLF Catalog时，需要设置为hms。 ○ jdbc（默认值）：采用jdbc Catalog时，需要设置为jdbc。
hive_sync.db	同步到Hive的数据库名称。	否	无。
hive_sync.table	同步到Hive的表名称。	否	无。
dlf.catalog.region	DLF服务的地域名。	否	详情请参见已开通的地域和访问域名。 说明 请和dlf.catalog.endpoint选择的地域保持一致。
dlf.catalog.endpoint	DLF服务的Endpoint。	否	详情请参见已开通的地域和访问域名。 说明 ○ 推荐您为dlf.catalog.endpoint参数配置DLF的VPC Endpoint。例如，如果您选择的地域为cn-hangzhou地域，则dlf.catalog.endpoint参数需要配置为dlf-vpc.cn-hangzhou.aliyuncs.com。 ○ 如果您需要跨VPC访问DLF，则请参见如何访问跨VPC里的存储资源？。
write.operation	写入操作模式。	否	参数取值如下： ○ insert 模式：数据追加写。 ○ upsert模式（默认值）：数据更新。
write.precombine.field	版本字段，基于此字段的大小来判断消息是否进行更新。	否	默认字段名称为ts，如果没有ts字段则为处理顺序。

参数	说明	是否必选	备注
changelog.enabled	是否开启Changelog模式。	否	参数取值如下： ○ true：开启Changelog模式。 Hudi支持保留消息的所有变更，对接Flink引擎的后，实现全链路近实时数仓生产。Hudi的MOR表以行存格式保留消息的所有变更，通过流读MOR表可以消费到所有的变更记录。此时，您需要开启Changelog模式，changelog.enabled设置为true。 开启changelog.enabled参数后，支持消费所有变更。异步的合并任务会将中间变更合并成1条。所以如果流读消费不够及时，被压缩后只能读到最后一条记录。当然，通过调整压缩的buffer时间可以预留一定的时间buffer给reader。例如调整合并的compaction.delta_commits和compaction.delta_seconds参数。 ○ false（默认值）：关闭Changelog模式。关闭Changelog模式时，即支持UPSERT语义，所有的消息仅保证最后一条合并消息，中间的变更可能会被合并。 **说明** 流读取会展示每次变更，批读只会展示合并后的变更结构。

- 高阶参数
  - 内存参数

> **说明**
> - 所有的内存参数单位都是MB。
> - 影响内存的三个因素包括TaskManager的数量和内存配置、write task的并发和每个write task能够分配到的内存。因此建议您确认每个write task能够分配到的内存，再考虑相关的内存参数设置。

参数	说明	默认值	备注

参数	说明	默认值	备注
write.task.max.size	一个write task的最大可用内存。	1024	预留给每个write task的内存buffer大小为write.task.max.size参数值减去compaction.max_memory参数值的差值。 当write task的内存buffer达到阈值后，会将内存的数据落盘。 您需要关注TM分配给每个write task的内存，保证每个write task能够分配到 write.task.max.size所配置的大小。例如TM的内存是4 GB，运行了2个StreamWriteFunction，那每个write function能分配到2 GB。此时你需要预留一些buffer，因为网络buffer，TM上其他类型任务，例如BucketAssignFunction也会消耗一些内存。
compaction.max_memory	合并文件时的最大可用内存。	100	如果是在线合并，资源充足时可以调大该参数，例如调为1 GB。 您需要关注compaction内存的变化。因为compaction.max_memory控制了每个compaction task读log时可以利用的内存大小。在内存资源充足时，有以下建议： ■ 如果是MOR表，可以将compaction.max_memory参数值调大些。 ■ 如果是COW表，可以将write.task.max.size和write.merge.max_memory参数值同时调大。
write.merge.max_memory	COW写操作，会有增量文件和全量文件的合并过程，增量的数据会缓存在内存中，该参数控制使用的堆内存大小。	100	通常您不需要设置该参数，保持默认值即可。

- 并发参数

参数	说明	默认值	备注
write.tasks	写任务（write）的并发写。每个write顺序写1~N个buckets。	4	增加并发不影响小文件个数。
write.bucket_assign.tasks	bucket assigner的并发。	1	增加该参数值会导致bucket数量增加，即增加小文件数。
write.index_bootstrap.tasks	Index bootstrap算子的并发。增加并发可以加快bootstrap阶段的效率，bootstrap阶段会阻塞checkpoint，因此需要设置多一些的checkpoint失败容忍次数。	不显式指定该参数时，默认使用Flink算子的并行度。	仅在index.bootstrap.enabled为true时生效。

- 在线合并参数

参数	说明	默认值	备注
compaction.tasks	在线合并算子的并发。	4	在线合并会消耗计算资源。
compaction.trigger.strategy	合并策略。	num_commits	支持以下四种策略： ■ num_commits：根据commits数量合并。 ■ time_elapsed：根据时间合并。 ■ num_and_time：根据数据量和时间合并。 ■ num_or_time：根据数量或时间合并。
compaction.delta_commits	delta commits文件的个数。	5	取值范围是整数，建议不超过20个。默认值的含义为5个delta文件触发一次合并。
compaction.delta_seconds	在线合并的时间间隔。	3600	单位为秒。
compaction.target_io	每个压缩任务的IO上限。	500（GB）	无。

示例

本示例为您介绍如何通过datagen connector随机生成流式数据写入Hudi表，然后从Hudi表流式读取数据写入blackhole。

1. 创建OSS Bucket。

   详情请参见创建存储空间。

2. 在**作业开发**页面，在目标作业文本编辑区域，编写SQL设计流作业。

```
CREATE TEMPORARY TABLE datagen(
  id INT NOT NULL PRIMARY KEY NOT ENFORCED,
  data  STRING,
  ts TIMESTAMP(3)
) WITH (
  'connector' = 'datagen' ,
  'rows-per-second'='100'
);
CREATE TEMPORARY TABLE datasink (
  id INT NOT NULL PRIMARY KEY NOT ENFORCED,
  data STRING,
  ts TIMESTAMP(3)
) WITH (
  'connector' = 'blackhole'
);
CREATE TEMPORARY TABLE hudi (
  id INT NOT NULL PRIMARY KEY NOT ENFORCED,
  data STRING,
  ts TIMESTAMP(3)
) WITH (
  'connector' = 'hudi',
  'oss.endpoint' = '<yourOSSEndpoint>',
  'accessKeyId' = '<yourAccessKeyId>',
  'accessKeySecret' = '<yourAccessKeySecret>',
  'path' = 'oss://<yourOSSBucket>/<自定义存储位置>',
  'table.type' = 'MERGE_ON_READ',
  'read.streaming.enabled' = 'true'
);
BEGIN STATEMENT SET;
INSERT INTO hudi SELECT * from datagen;
INSERT INTO datasink SELECT * FROM hudi;
END;
```

3. 在作业开发页面右侧**高级配置**面版中，引擎版本配置为**vvr-4.0.11-flink-1.13**。

4. 单击**验证**。
5. 单击**上线**。
6. 在**作业运维**页面，单击目标作业名称**操作**列的**启动**。

7. 在OSS控制台查看写入的测试数据。

   等第一次Checkpoint完成之后，您将能看到写入的测试数据了。

9.2.4.1.12. Upsert Kafka源表

本文为您介绍Upsert Kafka源表的DDL定义、WITH参数和示例。

什么是Upsert Kafka

Upsert Kafka基于开源Flink社区的Upsert-Kafka版本实现，详情请参见Upsert Kafka SQL 连接器。Upsert Kafka可以产生Changelog流，支持以UPSERT方式从Kafka Topic中读取数据或将数据写入Kafka Topic。其中每条数据记录代表一个更新或删除事件，原理如下：

- Kafka Topic中存在相应的Key，则以UPDATE操作将Key的值更新为数据记录中的Value。
- Kafka Topic中不存在相应的Key，则以INSERT操作将Key的值写入Kafka Topic。
- Key对应的Value为空，会被视作DELETE操作。

前提条件

已创建消息队列Kafka资源，详情请参见步骤三：创建资源。

使用限制

仅Flink计算引擎VVR 3.0.1及以上版本支持Upsert Kafka Connector。

DDL定义

以表名pageviews_per_region为例介绍DDL定义。

```
CREATE TABLE pageviews_per_region (
  user_region STRING,
  pv BIGINT,
  uv BIGINT,
  PRIMARY KEY (user_region) NOT ENFORCED
) WITH (
  'connector' = 'upsert-kafka',
  'topic' = '<yourTopicName>',
  'properties.bootstrap.servers' = '<yourServers>',
  'key.format' = 'avro',
  'value.format' = 'avro'
);
```

> **说明** Upsert Kafka必须声明主键。

WITH参数

参数	说明	是否必选	数据类型	备注
connector	源表类型	是	String	固定值为 `upsert-kafka` 。
topic	源表对应的Topic	是	String	无

参数	说明	是否必选	数据类型	备注
properties.bootstrap.servers	Kafka Broker地址	是	String	格式为 `host:port,host:port,host:port` ，以英文逗号（,）分割。
key.format	Upsert Kafka消息中Key部分反序列化的格式	是	String	支持的格式如下： • csv • json • avro
value.format	Upsert Kafka消息中Value部分反序列化的格式	是	String	支持的格式如下： • csv • json • avro
value.fields-include	指定出现在Value中的字段	是	String	取值如下： • ALL：默认值，Schema中所有字段，包括主键字段。 • EXCEPT_KEY：Schema中所有字段，不包括主键字段。
key.fields-prefix	为主键的所有字段定义一个自定义前缀，避免Key与Value字段的名称冲突。	否	String	默认情况下，前缀为空。如果定义了前缀，则表的Schema和key.fields都将使用前缀名称。 当构造Kafka Key对应格式的数据类型时，主键名中的前缀将被删除，并采用名字中非前缀部分作为最终的名字。 **说明** key.fields-prefix参数生效，要求value.fields-include必须设置为EXCEPT_KEY。
properties.*	指定Kafka参数	否	String	后缀名必须匹配定义在Apache Kafka文档中的参数名。Flink会自动移除properties.前缀，并将转换后的参数名及值传入Kafka客户端。例如，您可以通过设置 `'properties.allow.auto.create.topics' = 'false'` 来禁止自动创建Topic。 **说明** 参数 `key.deserializer` 和 `value.deserializer` 是不允许通过该方式指定参数的，因为Flink会重写这些参数的值。

示例

```
--创建upsert-kafka类型源表。
CREATE TABLE pageviews_per_region (
  user_region STRING,
  pv BIGINT,
  uv BIGINT,
  PRIMARY KEY (user_region) NOT ENFORCED
) WITH (
  'connector' = 'upsert-kafka',
  'topic' = '<yourTopicName>',
  'properties.bootstrap.servers' = '<yourServers>',
  'key.format' = 'avro',
  'value.format' = 'avro'
);
--创建Kafka类型结果表。
CREATE TABLE pageviews_per_region_db (
  user_region STRING,
  pv BIGINT,
  uv BIGINT,
  PRIMARY KEY (user_region) NOT ENFORCED
) WITH (
  'connector' = 'kafka',
  'topic' = '<yourTopicName>',
  'properties.bootstrap.servers' = '<yourKafkaBrokers>',
  'format' = 'csv'
);
--将upsert-kafka源表数据写入Kafka结果表。
INSERT INTO pageviews_per_region_db
SELECT user_region, pv, uv FROM pageviews_per_region;
```

9.2.4.1.13. Datagen源表

本文为您介绍Datagen源表的DDL定义、WITH参数和代码示例。

什么是Datagen源表

Datagen源表是系统内置的Connector，可以周期性地生成Datagen源表对应类型的随机数据。如果您在注册其他类型的Connector源表时报错，但您不确定是Flink系统问题还是源表WITH参数错误，您可以将WITH参数修改为'connector' = 'datagen'后，单击**运行**。如果不再报错，则证明Flink系统没有问题，您需要确认修改WITH参数。

使用限制

仅Flink计算引擎VVR 2.0.0及以上版本支持Datagen Connector。

DDL定义

```
CREATE TABLE datagen_source (
  name VARCHAR,
  score BIGINT
) WITH (
  'connector' = 'datagen'
);
```

WITH参数

参数	说明	是否必填	备注
connector	源表类型。	是	固定值为datagen。
rows-per-second	生成随机数据的速率。	否	默认值为10000条/秒。
fields.#.length	生成随机字符串数据的长度。	否	默认值为100。支持CHAR、VARCHAR和STRING。
fields.#.kind	• 在数据源是无界的，即fields.#.kind为随机发生器（默认值）。 • 在数据源是有界的，即fields.#.kind为序列发生器。	否	无。
fields.#.max	随机发生器的最大值。	否	仅支持数字类型。
fields.#.min	随机发生器的最小值。	否	仅支持数字类型。
fields.#.start	序列发生器的起始值。	否	无。
fields.#.end	序列发生器的结束值。	否	无。

代码示例

```
CREATE TEMPORARY table datahub_source(
  name VARCHAR
) WITH (
  'connector' = 'datagen'
);
CREATE TEMPORARY table datahub_sink(
  name  VARCHAR
) WITH (
  'connector'='datahub',
  'endpoint'='<yourEndpoint>',
  'project'='<yourProject>',
  'topic'='<yourTopic>',
  'accessId'='<yourAccessId>',
  'accessKey'='<yourAccessKey>'
);
INSERT INTO datahub_sink
SELECT
  LOWER(name)
from datahub_source;
```

9.2.4.1.14. 全量数据湖Iceberg源表

本文为您介绍全量数据湖Iceberg源表的DDL定义、WITH参数、类型映射和代码示例。

什么是数据湖Iceberg

Apache Iceberg是一种开放的数据湖表格格式。您可以借助Apache Iceberg快速地在HDFS或者云端OSS上构建自己的数据湖存储服务，并借助开源大数据生态的Flink、Spark、Hive、Presto等计算引擎来实现数据湖的分析。目前Apache Iceberg提供以下核心能力：

- 基于HDFS或者对象存储构建低成本的轻量级数据湖存储服务。
- 完善的ACID语义。
- 支持历史版本回溯。
- 支持高效的数据过滤。
- 支持Schema Evolution。
- 支持Partition Evolution。

您可以借助Flink高效的容错能力和流处理能力，把海量的日志行为数据实时导入到Apache Iceberg数据湖内，再借助Flink或者其他分析引擎来实现数据价值的提取。

> **说明** Iceberg Connector既可以作为Flink Stream作业的结果表，也可以作为Flink批作业的源表和结果表。本文主要介绍Iceberg表作为Flink批作业的源表。

使用限制

仅Flink计算引擎vvr-4.0.8-flink-1.13及以上版本支持Iceberg Connector。

DDL定义

```
CREATE TABLE dlf_iceberg (
  id    BIGINT,
  data  STRING
) WITH (
  'connector' = 'iceberg',
  'catalog-name' = '<yourCatalogName>',
  'catalog-type' = 'custom',
  'catalog-database' = '<yourDatabaseName>',
  'io-impl' = 'org.apache.iceberg.aliyun.oss.OSSFileIO',
  'oss.endpoint' = '<yourOSSEndpoint>',
  'access.key.id' = '<yourAccessKeyId>',
  'access.key.secret' = '<yourAccessKeySecret>',
  'catalog-impl' = 'org.apache.iceberg.aliyun.dlf.DlfCatalog',
  'warehouse' = '<yourOSSWarehousePath>',
  'dlf.catalog-id' = '<yourCatalogId>',
  'dlf.endpoint' = '<yourDLFEndpoint>',
  'dlf.region-id' = '<yourDLFRegionId>'
);
```

WITH参数

参数	说明	是否必选	备注
connector	源表类型。	是	固定值为 `iceberg` 。
catalog-name	Catalog名称。	是	请填写为自定义的英文名。
catalog-type	Catalog类型。	是	固定值为custom。
catalog-database	数据库名称。	是	对应用户在DLF上创建的数据库名称，例如dlf_db。
io-impl	分布式文件系统的实现类名。	是	固定值为 `org.apache.iceberg.aliyun.oss.OSSFileIO` 。
oss.endpoint	阿里云对象存储服务OSS的Endpoint。	否	请详情参见访问域名和数据中心。 **说明** • 推荐您为oss.endpoint参数配置OSS的VPC Endpoint。例如，如果您选择的地域为cn-hangzhou地域，则oss.endpoint需要配置为oss-cn-hangzhou-internal.aliyuncs.com。 • 如果您需要跨VPC访问OSS，则请参见如何访问跨VPC里的存储资源？
access.key.id	阿里云账号的Access Key。	是	获取方法请参见获取AccessKey。
access.key.secret	阿里云账号的Access Secret。	是	获取方法请参见获取AccessKey。
catalog-impl	Catalog的Class类名。	是	固定值为 `org.apache.iceberg.aliyun.dlf.DlfCatalog`
warehouse	表数据存放在OSS的路径。	是	无。
dlf.catalog-id	阿里云账号的账号ID。	是	登录账号信息，请通过用户信息页面获取。

参数	说明	是否必选	备注
dlf.endpoint	DLF服务的Endpoint。	是	详情请参见已开通的地域和访问域名。 说明 • 推荐您为dlf.endpoint参数配置DLF的VPC Endpoint。例如，如果您选择的地域为cn-hangzhou地域，则dlf.endpoint参数需要配置为dlf-vpc.cn-hangzhou.aliyuncs.com。 • 如果您需要跨VPC访问DLF，则请参见如何访问跨VPC里的存储资源？
dlf.region-id	DLF服务的地域名。	是	详情请参见已开通的地域和访问域名。 说明 请和dlf.endpoint选择的地域保持一致。

类型映射

Iceberg字段类型	Flink字段类型
BOOLEAN	BOOLEAN
INT	INT
LONG	BIGINT
FLOAT	FLOAT
DOUBLE	DOUBLE
DECIMAL(P,S)	DECIMAL(P,S)
DATE	DATE
TIME	TIME 说明 Iceberg时间戳精度为微秒，Flink时间戳精度为毫秒。在使用Flink读取Iceberg数据时，时间精度会对齐到毫秒。
TIMESTAMP	TIMESTAMP
TIMESTAMPTZ	TIMESTAMP_LTZ

Iceberg字段类型	Flink字段类型
STRING	STRING
FIXED(L)	BYTES
BINARY	VARBINARY
STRUCT<...>	ROW
LIST<E>	LIST
MAP<K, V>	MAP

代码示例

以下内容将为您介绍如何通过Flink作业将Iceberg源表的数据导入到另一个Iceberg结果表中，实现基本的ETL操作。

1. 确认已成功创建了DLF数据库。

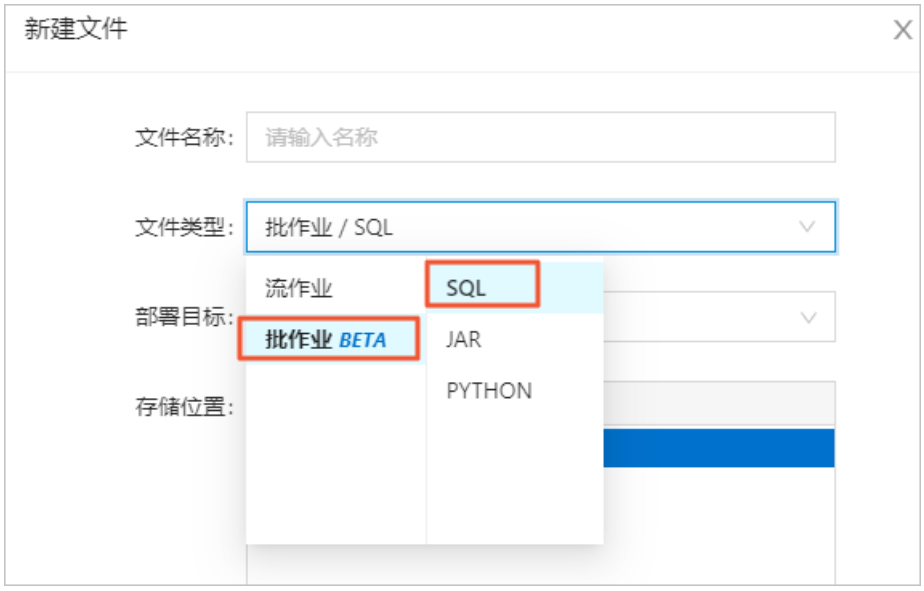
   如果您没有创建对应的DLF数据库，请创建DLF数据库，详情请参见创建元数据库。

   > **说明** 创建数据库选择OSS路径时，请将名称设置为*${warehouse}/${database_name}.db*。例如，如果您将warehouse地址设置为*oss://iceberg-test/warehouse*，数据库的名称设置为dlf_db，则dlf_db的OSS路径需要设置为*oss://iceberg-test/warehouse/dlf_db.db*。

2. 编写Flink Batch SQL作业生成测试数据，并导入到另一个Iceberg表中。

   i. 在左侧导航栏，单击**作业开发**。

   ii. 在页面左上角，单击**新建**，文件类型选择为批作业的SQL。



iii. 在文本编辑区域编写SQL，代码示例如下。

```
CREATE TEMPORARY TABLE src_iceberg (
  id    BIGINT,
  data  STRING
) WITH (
  'connector' = 'iceberg',
  'catalog-name' = '<yourCatalogName>',
  'catalog-type' = 'custom',
  'catalog-database' = '<yourDatabaseName>',
  'io-impl' = 'org.apache.iceberg.aliyun.oss.OSSFileIO',
  'oss.endpoint' = '<yourOSSEndpoint>',
  'access.key.id' = '<yourAccessKeyId>',
  'access.key.secret' = '<yourAccessKeySecret>',
  'catalog-impl' = 'org.apache.iceberg.aliyun.dlf.DlfCatalog',
  'warehouse' = '<yourOSSWarehousePath>',
  'dlf.catalog-id' = '<yourCatalogId>',
  'dlf.endpoint' = '<yourDLFEndpoint>',
  'dlf.region-id' = '<yourDLFRegionId>'
);
CREATE TEMPORARY TABLE dst_iceberg (
  id    BIGINT,
  data  STRING
) WITH (
  'connector' = 'iceberg',
  'catalog-name' = '<yourCatalogName>',
  'catalog-type' = 'custom',
  'catalog-database' = '<yourDatabaseName>',
  'io-impl' = 'org.apache.iceberg.aliyun.oss.OSSFileIO',
  'oss.endpoint' = '<yourOSSEndpoint>',
  'access.key.id' = '<yourAccessKeyId>',
  'access.key.secret' = '<yourAccessKeySecret>',
  'catalog-impl' = 'org.apache.iceberg.aliyun.dlf.DlfCatalog',
  'warehouse' = '<yourOSSWarehousePath>',
  'dlf.catalog-id' = '<yourCatalogId>',
  'dlf.endpoint' = '<yourDLFEndpoint>',
  'dlf.region-id' = '<yourDLFRegionId>'
);
BEGIN STATEMENT SET;
INSERT INTO src_iceberg VALUES (1, 'AAA'), (2, 'BBB'), (3, 'CCC'), (4, 'DDD'), (5,
'EEE');
INSERT INTO dst_iceberg SELECT * FROM src_iceberg;
END;
```

3. 在作业开发页面右侧**高级配置**面版中，引擎版本配置为**vvr-4.0.8-flink-1.13vvr-4.0.11-flink-1.13**。

4. 单击**验证**。
5. 单击**上线**。
6. 在OSS控制台查看写入的测试数据。

   等整个批作业完成后，您就可以在OSS路径下查看dst_iceberg表中写入的数据了。

9.2.4.1.15. 模拟数据生成源表

本文为您介绍模拟数据生成源表的使用限制、DDL定义、WITH参数和代码示例。

什么是模拟数据生成源表

模拟数据生成源表是系统内置的Connector，它根据表中每个字段提供的Java Faker表达式生成测试数据。当您在开发或测试时，需要使用一些测试数据来验证业务逻辑时，推荐您使用模拟数据生成Connector。

使用限制

- 仅Flink计算引擎VVR 4.0.12及以上版本支持模拟数据生成Connector。
- 支持部分数据类型，包括CHAR(n)、VARCHAR(n)、STRING、TINYINT、SMALLINT、INT、BIGINT、FLOAT、DOUBLE、DECIMAL、BOOLEAN、TIMESTAMP、ARRAY、MAP、MULTISET和ROW。

DDL定义

```
CREATE TABLE faker_source (
  `name` STRING,
  `age` INT
) WITH (
  'connector' = 'faker',
  'fields.name.expression' = '#{superhero.name}',
  'fields.age.expression' = '#{number.numberBetween ''0'',''1000''}'
);
```

WITH参数

参数	说明	数据类型	是否必填	备注
connector	源表类型。	STRING	是	固定值为faker。
number-of-rows	生成的数据行数。	INTEGER	否	如果设置了该参数，则源表是有界的，否则是无界的。
rows-per-second	生成数据的速率。	INTEGER	否	默认值为10000条/秒。
fields.<field>.expression	生成该字段值的Java Faker表达式。	STRING	是	详情请参见字段表达式。
fields.<field>.null-rate	该字段值为空的占比。	FLOAT	否	默认值为0.0。
fields.<field>.length	ARRAY、MAP或MULTISET集合类型的大小。	INTEGER	否	默认值为1。

字段表达式

- 操作方法

  使用模拟数据生成Connector时，DDL中定义的每个字段，都需要在WITH语句中提供具体的表达式。表达式的固定格式为'fields.<field>.expression' = '#{className.methodName ''parameter'', ...}'。相关的参数说明如下表所示。

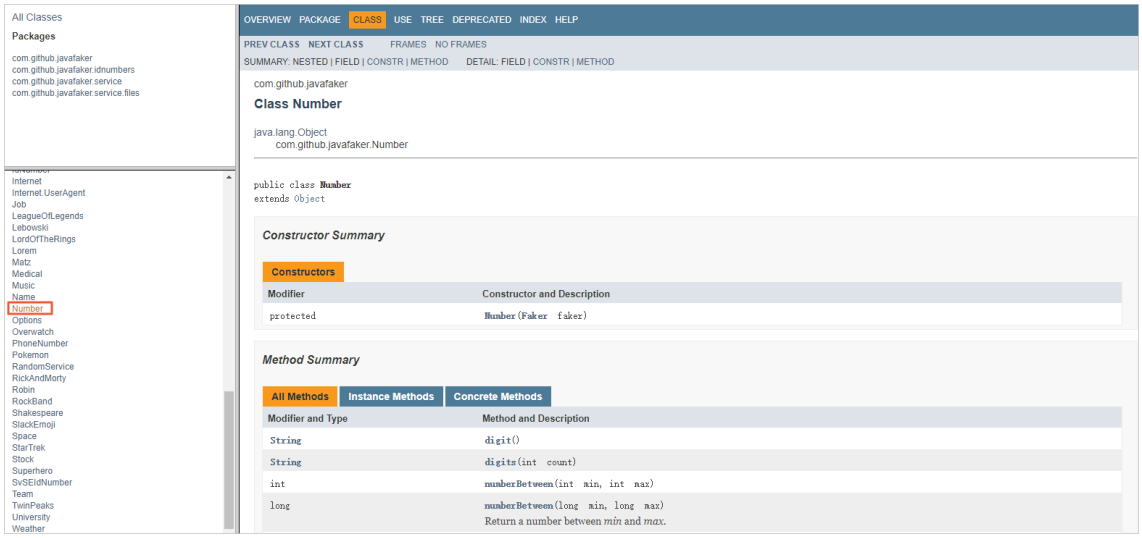
参数	说明
field	表示DDL中具体的字段名。
className	表示Faker类的类名。 Java Faker提供了大约80个Faker类用于生成您需要的字段表达式，您可以根据需要选择对应的类。 **说明** Faker类的类名不区分大小写。
methodName	表示方法名。 **说明** 方法名不区分大小写。

参数	说明
parameter	表示方法的入参。 说明 ○ 方法的入参需要使用两个半角单引号（' ）括起来。 ○ 多个参数使用半角逗号（,）进行分隔。

- 示例

  本文结合Java Faker API文档和DDL定义部分中的age字段表达式'fields.age.expression' = '#{number.numberBetween ''0'',''1000''}' 为例，为您介绍如何为DDL中的字段正确地生成SQL表达式。

  i. 在Java Faker API文档中，找到Number类。



  ii. 在Number类中找到numberBetween方法，并查看其方法描述。

Method Summary

All Methods | Instance Methods | Concrete Methods

Modifier and Type	Method and Description
String	digit()
String	digits(int count)
int	numberBetween(int min, int max)
long	numberBetween(long min, long max) Return a number between *min* and *max*.
int	randomDigit() Returns a random number from 0-9 (both inclusive)
int	randomDigitNotZero() Returns a random number from 1-9 (both inclusive)
double	randomDouble(int maxNumberOfDecimals, int min, int max)
double	randomDouble(int maxNumberOfDecimals, long min, long max) Returns a random double
long	randomNumber() Returns a random number
long	randomNumber(int numberOfDigits, boolean strict)

numberBetween方法表示返回指定数字范围的值。

iii. 根据类名Number和方法名numberBetween传入方法的参数0和1000，生成age字段的SQL表达式'fields.age.expression' = '#{number.numberBetween ''0'',''1000''}' 。

表示生成的age字段的值在0到1000范围内。

代码示例

```
CREATE TEMPORARY TABLE heros_source (
  `name` STRING,
  `power` STRING,
  `age` INT
) WITH (
  'connector' = 'faker',
  'fields.name.expression' = '#{superhero.name}',
  'fields.power.expression' = '#{superhero.power}',
  'fields.power.null-rate' = '0.05',
  'fields.age.expression' = '#{number.numberBetween ''0'',''1000''}'
);
CREATE TEMPORARY table blackhole_sink(
  `name` STRING,
  `power` STRING,
  `age` INT
) WITH (
  'connector' = 'blackhole'
);
INSERT INTO blackhole_sink SELECT * FROM heros_source;
```

9.2.4.2. 创建数据结果表

9.2.4.2.1. 日志服务SLS结果表

本文为您介绍日志服务SLS结果表的DDL定义、WITH参数、类型映射和相关文档。

什么是日志服务

日志服务SLS是针对日志类数据的一站式服务。日志服务可以帮助您快捷地完成数据采集、消费、投递以及查询分析，提升运维和运营效率，建立海量日志处理能力。日志服务本身是流数据存储，Flink支持将其作为流式数据的输出。

前提条件

已创建日志服务Project和Logstore，详情请参见步骤二：创建Project和Logstore。

使用限制

仅Flink计算引擎VVR 2.0.0及以上版本支持日志服务SLS Connector。

DDL定义

```
create table sls_sink(
 name varchar,
 age int,
 birthday int
)with(
 'connector'='sls',
 'endPoint'='<endPoint>',
 'accessId'='<yourAccessId>',
 'accessKey'='<yourAccessKey>',
 'project'='<yourProjectName>',
 'logstore'='<yourLogstoreName>'
);
```

WITH参数

参数	说明	是否必填	备注
connector	结果表类型	是	固定值为 `sls` 。
endPoint	EndPoint地址	是	服务入口。
project	项目名	是	无
logstore	表名	是	无
accessId	AccessKey ID	是	无
accessKey	AccessKey Secret	是	无

类型映射

日志服务和Flink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	Flink字段类型
STRING	VARCHAR

9.2.4.2.2. 消息队列Kafka结果表

本文为您介绍消息队列Kafka结果表的DDL定义、WITH参数和示例。

什么是消息队列Kafka

消息队列Kafka版是阿里云提供的分布式、高吞吐、可扩展的消息队列服务。消息队列Kafka版广泛用于日志收集、监控数据聚合、流式数据处理、在线和离线分析等大数据领域。

前提条件

- 已创建消息队列Kafka资源，详情请参见步骤三：创建资源。
- 已配置白名单，详情请参见配置白名单。

使用限制

- 仅Flink计算引擎VVR 2.0.0及以上版本支持消息队列Kafka Connector。
- 消息队列Kafka Connector仅支持将结果数据写入到kafka 0.10及以上版本。
- 消息队列Kafka Connector仅支持Kafka 2.4的生产者配置项，详情请参见生产者配置项。

DDL定义

```
create table kafka_sink(
  user_id BIGINT,
  item_id BIGINT,
  category_id BIGINT,
  behavior STRING,
  ts TIMESTAMP(3)
) with (
  'connector' = 'kafka',
  'topic' = '<yourTopicName>',
  'properties.bootstrap.servers' = '<yourKafkaBrokers>',
  'format' = 'csv'
);
```

WITH参数

参数	说明	是否必选	数据类型	备注
connector	结果表类型	是	STRING	固定值为 `kafka` 。
topic	结果表对应的Topic	是	STRING	无

参数	说明	是否必选	数据类型	备注
properties.bootstrap.servers	Kafka Broker地址	是	STRING	格式为 `host:port,host:port,host:port` ，以英文逗号（,）分割。
format	Flink Kafka Connector在序列化来自Kafka的消息时使用的格式。	是	STRING	格式取值如下： • csv • json • avro
sink.partitioner	从Flink分区到Kafka分区的映射模式。	否	STRING	映射模式取值如下： • fixed（默认值）：每个Flink分区对应至多一个Kafka分区。 • round-robin：Flink分区中的数据将被轮流分配至Kafka的各个分区。 • 自定义分区映射模式：如果fixed和round-robin不满足您的需求，您可以创建一个FlinkKafkaPartitioner的子类来自定义分区映射模式。例如 `org.mycompany.MyPartitioner` 。

Kafka Producer配置参数详情请参见Kafka官网生产者配置项列表。如果您还需要直接配置Connector使用的Kafka Producer，可以在Kafka Producer配置参数前添加 `properties` 前缀，并将该Kafka Producer配置信息追加至WITH参数。例如Kafka集群需要SASL（Simple Authentication and Security Layer）认证，代码示例如下。

```
CREATE TABLE kafkaTable (
    ...
) WITH (
    ...
    'properties.security.protocol' = 'SASL_PLAINTEXT',
    'properties.sasl.mechanism' = 'PLAIN',
    'properties.sasl.jaas.config' = 'org.apache.flink.kafka.shaded.org.apache.kafka.common.
security.plain.PlainLoginModule required username="USERNAME" password="PASSWORD";'
);
```

从Kafka中读取数据后插入Kafka示例

从名称为source的Topic中读取Kafka数据，再写入名称为sink的Topic，数据使用CSV格式。

```
CREATE TEMPORARY TABLE kafka_source (
    id INT,
    name STRING,
    age INT
) WITH (
    'connector' = 'kafka',
    'topic' = '<yourTopicName>',
    'properties.bootstrap.servers' = '<yourKafkaBrokers>',
    'properties.group.id' = '<yourPropertiesGroupid>',
    'format' = 'csv'
);
CREATE TEMPORARY TABLE kafka_sink (
    id INT,
    name STRING,
    age INT
) WITH (
    'connector' = 'kafka',
    'topic' = '<yourTopicName>',
    'properties.bootstrap.servers' = '<yourKafkaBrokers>',
    'format' = 'csv'
);
INSERT INTO kafka_sink SELECT id, name, age FROM kafka_source;
```

常见问题

Flink和Kafka网络连通，但Flink无法消费或者写入数据？

9.2.4.2.3. 数据总线DataHub结果表

本文为您介绍数据总线DataHub结果表的DDL定义、WITH参数和类型映射。

什么是数据总线DataHub

阿里云流数据处理平台DataHub是流式数据（Streaming Data）的处理平台，提供对流式数据的发布（Publish）、订阅（Subscribe）和分发功能，让您可以轻松构建基于流式数据的分析和应用。Flink支持将DataHub作为流式数据的输出。

前提条件

已创建DataHub的Project和Topic，详情请参见创建Project和Topic。

使用限制

仅Flink计算引擎VVR 2.0.0及以上版本支持数据总线DataHub Connector。

DDL定义

```
create table datahub_sink(
  name varchar,
  age BIGINT,
  birthday BIGINT
) WITH (
  'connector'='datahub',
  'endpoint'='<endPoint>',
  'project'='<yourProjectName>',
  'topic'='<yourTopicName>',
  'accessId'='<yourAccessId>',
  'accessKey'='<yourAccessKey>',
  'batchCount'='500',
  'batchSize'='512000',
  'flushInterval'='5000'
);
```

WITH参数

参数	说明	是否必填	备注
connector	结果表类型。	是	固定值为 `datahub` 。
endPoint	消费端点信息。	是	详情请参见DataHub域名列表。
accessId	AccessKey ID。	是	无。
accessKey	AccessKey Secret。	是	无。
project	项目名称。	是	无。
topic	Topic名称。	是	无。
retryTimeout	最大持续重试时间。	否	单位为毫秒，默认值为180000（半小时）。
retryInterval	重试间隔。	否	单位为毫秒，默认值为1000。
batchCount	每次批量写入数据的最大数据条数。	否	默认值为500。
batchSize	每次批量写入数据的大小。	否	默认值为512000 Byte。
flushInterval	缓存数据的最大超时时间。	否	单位毫秒，默认值为5000。
partitionBy	写入Sink节点前会根据partitionBy值进行hash，数据会流向对应的Sink节点。	否	默认值为空，随机发送。
hashFields	指定列名后，相同列的值会写入到同一个Shard。	否	默认值为Null，即随机写可以指明多个列值，用逗号（,）分隔。例如 `hashFields=a,b` 。

类型映射

DataHub和Flink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	Flink字段类型
STRING	VARCHAR
TIMESTAMP	BIGINT
TINYINT	TINYINT
SMALLINT	SMALLINT
INTEGER	INTEGER
BIGINT	BIGINT
FLOAT	FLOAT
DOUBLE	DOUBLE
BOOLEAN	BOOLEAN
DECIMAL	DECIMAL

代码示例

```
CREATE TEMPORARY table datahub_source(
  name VARCHAR
) with (
  'connector'='datahub',
  'endpoint'='<endPoint>',
  'project'='<yourProjectName>',
  'topic'='<yourTopicName>',
  'subId'='<yourSubId>',
  'accessId'='<yourAccessId>',
  'accessKey'='<yourAccessKey>',
  'startTime'='2018-06-01 00:00:00'
);
CREATE TEMPORARY table datahub_sink(
  name varchar
) with (
  'connector'='datahub',
  'endpoint'='<endPoint>',
  'project'='<yourProjectName>',
  'topic'='<yourTopicName>',
  'accessId'='<yourAccessId>',
  'accessKey'='<yourAccessKey>',
  'batchSize'='512000',
  'batchCount'='500'
);
INSERT INTO datahub_sink
SELECT
  LOWER(name)
from datahub_source;
```

9.2.4.2.4. 表格存储Tablestore结果表

本文为您介绍表格存储Tablestore结果表的DDL定义、WITH参数和映射关系。

什么是表格存储Tablestore

表格存储Tablestore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的无缝扩展，提供海量结构化数据的存储和实时访问服务。Flink支持将Tablestore作为流式数据的输出。

前提条件

已创建Tablestore数据表，详情请参见步骤三：创建数据表。

使用限制

仅Flink计算引擎VVR 2.0.0及以上版本支持表格存储Tablestore Connector。

DDL定义

```
CREATE TABLE ots_sink (
  name VARCHAR,
  age BIGINT,
  birthday BIGINT,
  primary key(name,age) not enforced
) WITH (
  'connector'='ots',
  'instanceName'='<yourInstanceName>',
  'tableName'='<yourTableName>',
  'accessId'='<yourAccessId>',
  'accessKey'='<yourAccessSecret>',
  'endPoint'='<yourEndpoint>',
  'valueColumns'='birthday'
);
```

说明 Tablestore结果表必须定义有 `Primary Key` ，输出数据以Update方式追加Tablestore表。

WITH参数

参数	说明	是否必填	备注
connector	结果表类型。	是	固定值为 `ots` 。
instanceName	实例名称。	是	无。
tableName	表名称。	是	无。
endPoint	实例访问地址。	是	请参见服务地址。
accessId	AccessKey ID。	是	无。
accessKey	AccessKey Secret。	是	无。
valueColumns	插入字段的列名。	是	多个字段以英文逗号（,）分割，例如 `ID,NAME` 。
bufferSize	流入多少条数据后开始输出。	否	默认值为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。	否	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。	否	默认值为100。
retryIntervalMs	重试间隔时间。	否	单位为毫秒，默认值为1000。
maxRetryTimes	最大重试次数。	否	默认值为10。

参数	说明	是否必填	备注
ignoreDelete	是否忽略DELETE操作。	否	默认值为False。
connectTimeout	Connector连接Tablestore的超时时间。	否	单位毫秒，默认值为30000（30秒）。
socketTimeout	Connector连接Tablestore的Socket超时时间。	否	单位毫秒，默认值为30000（30秒）。

类型映射

Tablestore字段类型	Flink字段类型
INTEGER	BIGINT
STRING	STRING
BOOLEAN	BOOLEAN
DOUBLE	DOUBLE

9.2.4.2.5. MaxCompute结果表

本文为您介绍MaxCompute结果表的DDL定义、WITH参数、类型映射和代码示例。

> **说明** MaxCompute Connector可以作为Stream作业和Batch作业的源表使用。

什么是MaxCompute

大数据计算服务MaxCompute（原名ODPS）是一种快速、完全托管的EB级数据仓库解决方案，致力于批量结构化数据的存储和计算，提供海量数据仓库的解决方案及分析建模服务。MaxCompute详情请参见什么是MaxCompute。

前提条件

已创建MaxCompute表，详情请参见创建表。

使用限制

仅Flink计算引擎VVR 2.0.0及以上版本支持MaxCompute Connector。

实现原理

MaxCompute Sink可以分为以下两个阶段：

1. 写入数据。调用MaxCompute SDK中的接口将数据写入缓冲区，在缓冲区大小超过64 MB或者每隔指定的时间间隔时，上传数据到MaxCompute的临时文件中。
2. 提交会话。在任务进行Checkpoint时， MaxCompute Sink会调用Tunnel的Commit方法，提交会话，移

动临时文件到MaxCompute表的数据目录，并修改元数据。

> **说明** Commit方法不能提供原子性。因此，MaxCompute Sink提供的是At least Once方式，而不是Exactly Once方式。

DDL定义

Flink支持将MaxCompute作为结果输出，示例代码如下。

```
create table odps_sink(
  id INT,
  user_name VARCHAR,
  content VARCHAR
) WITH (
  'connector' = 'odps',
  'endpoint' = '<yourEndpoint>',
  'tunnelEndpoint' = '<yourTunnelEndpoint>',
  'project' = '<yourProjectName>',
  'tablename' = '<yourTableName>',
  'accessid' = '<yourAccessKeyId>',
  'accesskey' = '<yourAccessKeySecret>',
  'partition' = 'ds=2018****'
);
```

> **说明** DDL定义中的字段名称、顺序和类型必须与MaxCompute物理表保持一致，否则可能导致MaxCompute物理表中查询的数据为 `/n` 。

WITH参数

参数	说明	是否必填	备注
connector	结果表类型。	是	固定值为 `odps` 。
endPoint	MaxCompute服务地址。	是	请参见Endpoint。
tunnelEndpoint	MaxCompute Tunnel服务的连接地址。	是	请参见Endpoint。 **说明** VPC环境下为必填。
project	MaxCompute项目名称。	是	无。
tableName	表名。	是	无。
accessId	AccessKey ID。	是	无。
accessKey	AccessKey Secret。	是	无。

参数	说明	是否必填	备注
partition	分区名。	否	如果存在分区表，则必填partition。填写partition需要注意以下两点： • 固定分区 例如 `` `partition` = 'ds=20180905' `` 表示将数据写入分区 `ds= 20180905` 。 • 动态分区 如果未明文显示分区值，则根据写入数据中分区列具体值，写入对应分区。例如 `` `partition`='ds' `` 表示根据 `ds` 字段值写入对应分区。 如果要创建多级动态分区，Partition中多个字段顺序必须和MaxCompute物理表保持一致，各个分区字段之间使用逗号（,）分割。 说明 ○ 动态分区列需要显式写在建表语句中。 ○ 在动态分区字段为空时，如果数据源中 `ds=null` 或者 `ds=''` ，则输出结果为ds=NULL的分区。
flushIntervalMs	Odps tunnel writer缓冲区Flush间隔。 MaxCompute Sink写入记录时，先将数据存储到MaxCompute的缓冲区中，等缓冲区溢出或者每隔一段时间（flushIntervalMs），再把缓冲区里的数据写到目标MaxCompute表。	否	单位为毫秒，默认值为30000（30秒）。
dynamicPartitionLimit	分区数目最大值。	否	默认值为100，系统会把已写入的分区和TunnelBufferedWriter的映射关系维护到一个Map里，如果该Map大小超过了dynamicPartitionLimit设定值，则会出现 `Too many dynamic partitions: 100, which exceeds the size limit: 100` 报错。

类型映射

MaxCompute字段类型	Flink字段类型
TINYINT	TINYINT
SMALLINT	SMALLINT
INT	INT
BIGINT	BIGINT
FLOAT	FLOAT
DOUBLE	DOUBLE
BOOLEAN	BOOLEAN
DATETIME	TIMESTAMP
TIMESTAMP	TIMESTAMP
VARCHAR	VARCHAR
STRING	VARCHAR
DECIMAL	DECIMAL

代码示例

- 写入固定分区

```
CREATE TEMPORARY TABLE datagen_source (
  id INT,
  len INT,
  content VARCHAR
) WITH (
  'connector' = 'datagen'
);
CREATE TEMPORARY TABLE odps_sink (
  id INT,
  len INT,
  content VARCHAR
) WITH (
  'connector' = 'odps',
  'endpoint' = '<yourEndpoint>',
  'tunnelEndpoint' = '<yourTunnelEndpoint>',
  'project' = '<yourProjectName>',
  'tablename' = '<yourTableName>',
  'accessid' = '<yourAccessKeyId>',
  'accesskey' = '<yourAccessKeySecret>',
  'partition' = 'ds=20180905'
);
INSERT INTO odps_sink
SELECT
  id, len, content
FROM datagen_source;
```

- 写入动态分区

```
CREATE TEMPORARY TABLE datagen_source (
  id INT,
  len INT,
  content VARCHAR,
  c TIMESTAMP
) WITH (
  'connector' = 'datagen'
);
CREATE TEMPORARY TABLE odps_sink (
  id  INT,
  len INT,
  content VARCHAR,
  ds VARCHAR --动态分区列需要显式声明。
) WITH (
  'connector' = 'odps',
  'endpoint' = '<yourEndpoint>',
  'tunnelEndpoint' = '<yourTunnelEndpoint>',
  'project' = '<yourProjectName>',
  'tablename' = '<yourTableName>',
  'accessid' = '<yourAccessKeyId>',
  'accesskey' = '<yourAccessKeySecret>',
  'partition' ='ds' --不写分区的值，表示根据ds字段的值写入不同分区。
);
INSERT INTO odps_sink
SELECT
   id,
   len,
   content,
   DATE_FORMAT(c, 'yyMMdd') as ds
FROM datagen_source;
```

常见问题

报错：ErrorMessage=Authorization Failed [4019], You have NO privilege'ODPS:***'

9.2.4.2.6. 消息队列RocketMQ版结果表

本文为您介绍消息队列RocketMQ版结果表DDL定义、WITH参数和示例代码等。

> **说明** RocketMQ Connector可以作为Stream作业和Batch作业的结果表使用。

什么是消息队列RocketMQ版

消息队列RocketMQ版是阿里云基于Apache RocketMQ构建的低延迟、高并发、高可用和高可靠的分布式消息中间件。消息队列RocketMQ版既可为分布式应用系统提供异步解耦和削峰填谷的能力，同时也具备互联网应用所需的海量消息堆积、高吞吐和可靠重试等特性。Flink支持将消息队列MQ作为流式数据的输出。

前提条件

已创建了RocketMQ资源，详情请参见创建资源。

使用限制

仅Flink计算引擎VVR 2.0.0及以上版本支持消息队列RocketMQ Connector。

DDL定义

```
create table mq_sink(
  x varchar,
  y varchar,
  z varchar
) with (
  'connector'='mq',
  'topic'='<yourTopicName>',
  'endpoint'='<yourEndpoint>',
  'accessId'='<yourAccessId>',
  'accessKey'='<yourAccessSecret>'
);
```

说明 MQ是非结构化存储格式的消息中间件，对于数据的Schema不提供强制定义，完全由业务层指定。Flink仅支持CSV和二进制格式的MQ消息。

WITH参数

参数	说明	是否必填	备注
connector	结果表类型。	是	固定值为 `mq` 。
topic	topic名称。	是	无。

			阿里云消息队列RocketMQ版接入地址支持以下两种类型：
			• VVR 3.0.1及以上版本的作业，需要使用TCP协议客户端接入点，详情请参见关于TCP内网接入点设置的公告。接入点获取方式如下：
			  ○ 内网服务MQ（阿里云经典网络/VPC）接入地址：在MQ控制台目标实例详情中，选择**接入点 > TCP协议客户端接入点 > 内网访问**，获取对应的endPoint。
			  ○ 公网服务MQ接入地址：在MQ控制台目标实例详情中，选择**接入点 > TCP协议客户端接入点 > 公网访问**，获取对应的endPoint。
			  说明 通过公网方式访问MQ，需要配置NAT，详情请参见创建和管理公网NAT网关实例。
			• VVR 3.0.1（不含）以下版本的作业，使用如下接入点：

参数	说明	是否必填	备注
endpoint	地址。	是	○ 内网服务MQ（阿里云经典网络/VPC）接入地址： ■ 华东1（杭州）、华东2（上海）、华北1（青岛）、华北2（北京）、华南1（深圳）、中国（香港）：`onsaddr-internal.aliyun.com:8080` ■ 亚太东南1（新加坡）：`ap-southeastaddr-internal.aliyun.com:8080`。 ■ 中东东部1（迪拜）：`ons-me-east-1-internal.aliyuncs.com:8080`。 ■ 亚太南部1（孟买）：`ons-ap-south-1-internal.aliyuncs.com:8080`。 ■ 亚太东南3（吉隆坡）：`ons-ap-southeast-3-internal.aliyun.com:8080`。 ○ 公网服务MQ接入地址：`onsaddr-internet.aliyun.com:80`。 **注意** • 如果您已使用了VVR 3.0.1（不含）以下版本的RocketMQ Connector，则您需要将您的实时计算作业升级至VVR 3.0.1及以上版本，并将作业中EndPoint参数取值更改为新的RocketMQ接入点，旧的RocketMQ接入点存在稳定性风险或不可用的问题，详情请参见RocketMQ接入点变更导致实时计算作业适配升级公告。 • 内网服务无法跨域访问。例如，您所购买的实时计算服务的地域为华东1，但是购买的消息队列MQ服务的地域为华东2，则无法访问。 • 由于阿里云网络安全策略动态变化，实时计算连接公网服务MQ时可能会出现网络连接问题，推荐您使用内网服务MQ，如果在使用公网服务MQ时出现异常，请您提交工单进行咨询。
accessId	AccessKey ID。	是	无。
accessKey	AccessKey Secret。	是	无。
producerGroup	写入的群组。	是	无。
tag	写入的标签。	否	支持设置多个tag，以逗号（,）进行分隔。

参数	说明	是否必填	备注
nameServerSubgroup	NameServer组。	否	• 内网服务（阿里云经典网络/VPC）：nsaddr4client-internal • 公网服务：nsaddr4client-internet 说明 仅VVR 2.1.1 ~ VVR 3.0.0版本支持该参数，VVR 3.0.1及以后版本不支持该参数。
encoding	编码类型。	否	默认值为 `utf-8` 。
retryTimes	写入的重试次数。	否	默认值为10。
sleepTimeMs	重试间隔时间。	否	默认值为1000（毫秒）。
instanceID	MQ实例ID。	否	• 如果MQ实例无独立命名空间，则不可以使用instanceID参数。 • 如果MQ实例有独立命名空间，则instanceID参数必选。

代码示例

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

说明 如果您的MQ消息为二进制格式，则DDL中只能定义1个字段，且字段类型必须为VARBINARY。

9.2.4.2.7. 云数据库HBase版结果表

本文为您介绍云数据库HBase版结果表DDL定义、WITH参数、转换关系、动态表和代码示例。

什么是云数据库HBase

云数据库HBase是低成本、高扩展、云智能的大数据NoSQL，兼容标准HBase访问协议，提供低成本存储、高扩展吞吐、智能数据处理等核心优势，是为淘宝推荐、花呗风控、广告投放、监控大屏、菜鸟物流轨迹、支付宝账单、手淘消息等众多阿里巴巴核心服务提供支撑的数据库，具备PB规模、千万级并发、秒级伸缩、毫秒响应、跨机房高可用、全托管、全球分布等企业能力。

前提条件

- 已购买HBase集群并创建表，详情请参见购买集群。
- 已设置白名单，详情请参见设置白名单。

使用限制

仅Flink计算引擎VVR 2.0.0及以上版本支持云数据库HBase Connector。

DDL定义

```
CREATE TABLE hbase_sink(
  rowkey INT,
  family1 ROW<q1 INT>,
  family2 ROW<q2 STRING, q3 BIGINT>,
  family3 ROW<q4 DOUBLE, q5 BOOLEAN, q6 STRING>
) with (
  'connector'='cloudhbase',
  'table-name'='<yourTableName>',
  'zookeeper.quorum'='<yourZookeeperQuorum>'
);
```

- HBase的列族（Column Family）必须声明为ROW类型，列族名即该ROW的字段名。例如，DDL定义中声明了family1、family2和family3三个列族。
- HBase列族中的列（Cloumn）与对应ROW中嵌套的每个字段对应，列名即字段名。例如，DDL定义中列族family2声明了q2和q3两列。
- 结果表中不需要将HBase表的所有列族和列都进行声明，只声明需要的即可。
- 除了类型为ROW的字段外，只能有一个原始类型（Atomic Type）的字段（例如STRING或BIGINT），该字段将被视作HBase的行键（Row Key），例如DDL定义中的rowkey。
- 必须将HBase的行键定义为结果表的主键（Primary Key），如果没有显示定义主键，默认使用行键作为主键。

WITH参数

参数	说明	是否必填	备注
connector	结果表类型。	是	固定值为 `cloudhbase` 。
table-name	HBase表名。	是	无。
zookeeper.quorum	HBase的zookeeper地址。	是	无。

参数	说明	是否必填	备注
zookeeper.znode.parent	HBase在zookeeper中的根目录。	否	默认值为 `/hbase` 。 说明 仅在HBase标准版中生效。
userName	用户名。	否	仅在HBase增强版中生效。
password	密码。	否	仅在HBase增强版中生效。
haclient.cluster.id	HBase高可用实例ID。	否	只有访问同城主备实例时才需要配置。 说明 仅在HBase增强版中生效。
retries.number	连接HBase客户端的重试次数。	否	默认值为31。
null-string-literal	HBase字段类型为字符串时，如果Flink字段数据为null，则将该字段赋值为null-string-literal，并写入HBase。	否	默认值为null。
sink.buffer-flush.max-size	写入HBase前，内存中缓存的数据量（字节）大小。调大该值有利于提高HBase写入性能，但会增加写入延迟和内存使用。	否	默认值为2 MB，支持字节单位B、KB、MB和GB，不区分大小写。设置为0表示不进行缓存。
sink.buffer-flush.max-rows	写入HBase前，内存中缓存的数据条数。调大该值有利于提高HBase写入性能，但会增加写入延迟和内存使用。	否	默认值为1000，设置为0表示不进行缓存。
sink.buffer-flush.interval	将缓存数据周期性写入到HBase的间隔，可以控制写入HBase的延迟。	否	默认值为1秒，支持时间单位ms、s、min、h和d。设置为0表示关闭定期写入。
dynamic.table	是否使用支持动态列的HBase表。	否	参数取值如下： • true：使用支持动态列的HBase表。 • false（默认值）：不使用支持动态列的HBase表。

转换关系

HBase数据通过org.apache.hadoop.hbase.util.Bytes转换成Flink的数据类型。解码过程有以下两种情况：

- 当数据为null时，对于非字符串类型的数据，会编码为空字节数组。
- 对于字符串类型的数据，取决于null-string-literal的值。

Flink与HBase的数据转换关系如下。

Flink类型	HBase转换函数
CHAR	byte[] toBytes(String s)
VARCHAR	
STRING	
BOOLEAN	byte[] toBytes(boolean b)
BINARY	byte[]
VARBINARY	
DECIMAL	byte[] toBytes(BigDecimal v)
TINYINT	new byte[] { val }
SMALLINT	byte[] toBytes(short val)
INT	byte[] toBytes(int val)
BIGINT	byte[] toBytes(long val)
FLOAT	byte[] toBytes(float val)
DOUBLE	byte[] toBytes(double val)
DATE	将日期转换成自1970.01.01以来的天数，用int表示，并通过byte[] toBytes(int val) 转换成字节数组。
TIME	将时间转换成自00:00:00以来的毫秒数，用int表示，并通过byte[] toBytes(int val) 转换成字节数组。
TIMESTAMP	将时间戳转换成自1970-01-01 00:00:00以来的毫秒数，用long表示，并通过byte[] toBytes(long val) 转换成字节数组。
ARRAY	不支持。
MAP	不支持。
MULTISET	
ROW	不支持。

动态表

Flink部分结果数据需要按某列的值，作为动态列输入HBase。例如，某商品每小时的成交额作为动态列的数据，示例如下。

```
CREATE TEMPORARY TABLE datagen_source (
  id INT,
  f1hour STRING,
  f1deal BIGINT,
  f2day STRING,
  f2deal BIGINT
) with (
  'connector'='datagen'
);
CREATE TEMPORARY TABLE hbase_sink (
  rowkey INT,
  f1 ROW<`hour` STRING, deal BIGINT>,
  f2 ROW<`day` STRING, deal BIGINT>
) with (
  'connector'='cloudhbase',
  'table-name'='<yourTableName>',
  'zookeeper.quorum'='<yourZookeeperQuorum>',
  'dynamic.table'='true'
);
INSERT INTO hbase_sink
SELECT id, ROW(f1hour, f1deal), ROW(f2day, f2deal) FROM datagen_source;
```

说明

- 当dynamic.table参数值为true时，表示使用支持动态列的HBase表。
- 每个列族对应的ROW中必须声明两个字段：第1个字段的值表示动态列，第2个字段的值表示动态列的值。
- 如果src表存在一条(1, "10", 100, "2020-7-26", 10000)数据，代表ID为1的商品，在10:00-11:00点之间的成交额是100，在2020年7月26日当天的成交额是10000，则HBase中将插入行键为1的行，其中f1:10为100，f2:2020-7-26为10000。

代码示例

```
CREATE TEMPORARY TABLE datagen_source (
  rowkey INT,
  f1q1 INT,
  f2q1 STRING,
  f2q2 BIGINT,
  f3q1 DOUBLE,
  f3q2 BOOLEAN,
  f3q3 STRING
) with (
  'connector'='datagen'
);
CREATE TEMPORARY TABLE hbase_sink (
  rowkey INT,
  family1 ROW<q1 INT>,
  family2 ROW<q1 STRING, q2 BIGINT>,
  family3 ROW<q1 DOUBLE, q2 BOOLEAN, q3 STRING>,
  PRIMARY KEY (rowkey) NOT ENFORCE
) with (
  'connector'='cloudhbase',
  'table-name'='<yourTableName>',
  'zookeeper.quorum'='<yourZookeeperQuorum>'
);
INSERT INTO hbase_sink
SELECT rowkey, ROW(f1q1), ROW(f2q1, f2q2), ROW(f3q1, f3q2, f3q3) FROM datagen_source;
```

9.2.4.2.8. 云数据库Redis版结果表

本文为您介绍云数据库Redis版结果表DDL定义、WITH参数、类型映射和代码示例。

> **说明** 支持自建Redis服务。

什么是云数据库Redis版

阿里云数据库Redis版是兼容开源Redis协议标准、提供内存加硬盘混合存储的数据库服务，基于高可靠双机热备架构及可平滑扩展的集群架构，充分满足高吞吐、低延迟及弹性变配的业务需求。Flink支持将其作为流式数据的输出。

前提条件

- 已创建云数据库Redis实例，详情请参见步骤1：创建实例。
- 已设置白名单，详情请参见步骤2：设置白名单。

使用限制

仅Flink计算引擎VVR 2.0.0及以上版本支持云数据库Redis Connector。

DDL定义

云数据库Redis版结果表支持5种Redis数据结构，其DDL定义如下：

- STRING类型

  DDL为两列：第1列为key，第2列为value。Redis插入数据的命令为 `set key value` 。

```
create table redis_sink (
  a STRING,
  b STRING,
  PRIMARY KEY (a) NOT ENFORCED -- 必填。
) with (
  'connector' = 'redis',
  'mode' = 'string',
  'host' = '<yourHost>',
  'port' = '<yourPort>',
  'password' = '<yourPassword>',
  'dbNum' = '<yourDbNum>',
  'ignoreDelete' = 'true' -- 收到Retraction时，是否删除已插入的数据，默认值为false。
);
```

- LIST类型

  DDL为两列：第1列为key，第2列为value。Redis插入数据的命令为 `lpush key value` 。

```
create table redis_sink (
  a STRING,
  b STRING,
  PRIMARY KEY (a) NOT ENFORCED -- 必填。
) with (
  'connector' = 'redis',
  'mode' = 'list',
  'host' = '<yourHost>',
  'port' = '<yourPort>',
  'password' = '<yourPassword>',
  'dbNum' = '<yourDbNum>',
  'ignoreDelete' = 'true' -- 收到Retraction时，是否删除已插入的数据，默认值为false。
);
```

- SET类型

  DDL为两列：第1列为key，第2列为value。Redis插入数据的命令为 `sadd key value` 。

```
create table redis_sink (
  a STRING,
  b STRING,
  PRIMARY KEY (a) NOT ENFORCED -- 必填。
) with (
  'connector' = 'redis',
  'mode' = 'set',
  'host' = '<yourHost>',
  'port' = '<yourPort>',
  'password' = '<yourPassword>',
  'dbNum' = '<yourDbNum>',
  'ignoreDelete' = 'true' -- 收到Retraction时，是否删除已插入的数据，默认值为false。
);
```

- HASHMAP类型

  DDL为三列：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 redis_sink (
  a STRING,
  b STRING,
  c STRING,
  PRIMARY KEY (a) NOT ENFORCED -- 必填。
) with (
  'connector' = 'redis',
  'mode' = 'hashmap',
  'host' = '<yourHost>',
  'port' = '<yourPort>',
  'password' = '<yourPassword>',
  'dbNum' = '<yourDbNum>',
  'ignoreDelete' = 'true' -- 收到Retraction时，是否删除已插入的数据，默认值为false。
);
```

- SORTEDSET类型

  DDL为三列：第1列为key，第2列为score，第3列为value。Redis插入数据的命令为 `zadd key score value` 。

```
create table redis_sink (
  a STRING,
  b DOUBLE, --必须为DOUBLE类型。
  c STRING,
  PRIMARY KEY (a) NOT ENFORCED -- 必填。
) with (
  'connector' = 'redis',
  'mode' = 'sortedset',
  'host' = '<yourHost>',
  'port' = '<yourPort>',
  'password' = '<yourPassword>',
  'dbNum' = '<yourDbNum>',
  'ignoreDelete' = 'true' -- 收到Retraction时，是否删除已插入的数据，默认值为false。
);
```

WITH参数

参数	说明	是否必填	取值
connector	结果表类型。	是	固定值为 `redis` 。
mode	对应Redis的数据结构。		取值如下： • string • list • set • hashmap • sortedset
host	Redis Server连接地址。		取值示例： `127.0.0.1` 。

参数	说明	是否必填	取值
port	Redis Server连接端口。	否	默认值为6379。
password	Redis密码。		默认值为空，不进行权限验证。
dbNum	选择操作的数据库。		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false，收到Retraction时，同时删除数据对应的key及已插入的数据。
clusterMode	Redis集群是否为cluster模式。		默认值为false。

类型映射

Redis字段类型	Flink字段类型
STRING	STRING
SCORE	DOUBLE

说明 因为Redis的SCORE类型应用于SORTEDSET（有序集合），所以需要手动为每个Value设置一个DOUBLE类型的SCORE，Value才能按照该SCORE从小到大进行排序。

代码示例

```
CREATE TEMPORARY TABLE datagen_source (
  v STRING,
  p STRING
) with (
  'connector' = 'datagen'
);
CREATE TEMPORARY TABLE redis_sink (
  a STRING,
  b STRING,
  PRIMARY KEY (a) NOT ENFORCED
) with (
  'connector' = 'redis',
  'mode' = 'string',
  'host' = '<yourHost>',
  'port' = '<yourPort>',
  'password' = '<yourPassword>'
);
INSERT INTO redis_sink
SELECT v, p
FROM datagen_source;
```

9.2.4.2.9. 交互式分析Hologres结果表

本文为您介绍交互式分析Hologres结果表DDL定义、WITH参数、流式语义和类型映射。

> **说明** Hologres Connector可以作为Stream作业和Batch作业的结果表使用。

背景信息

- 什么是交互式分析Hologres

  交互式分析Hologres兼容PostgreSQL协议，与大数据生态紧密连接，支持高并发、低延时实时分析处理PB级数据，让您轻松使用现有BI（Business Intelligence）工具对数据进行多维分析和业务探索。

- 功能特性

  Hologres Connector支持以下功能特性。

功能	详情
流式语义	支持写入changelog消息。
宽表Merge和局部更新功能	只更新修改部分的数据，而非整行更新。
作为CTAS和CDAS的目标端	支持实时同步整库、单表的数据以及相应的表结构变更到每张对应的Hologres表中。

前提条件

已创建Hologres表，详情请参见表。

使用限制

- 仅Flink计算引擎VVR 2.0.0及以上版本支持交互式分析Hologres Connector。
- Hologres Connector不支持将结果数据写入Hologres外部表。Hologres外部表详情请参见外部表。

DDL定义

```
create table hologres_sink(
  name varchar,
  age BIGINT,
  birthday BIGINT,
  PRIMARY KEY (name) NOT ENFORCE
) with (
  'connector'='hologres',
  'dbname'='<yourDbname>',
  'tablename'='<yourTablename>',
  'username'='<yourUsername>',
  'password'='<yourPassword>',
  'endpoint'='<yourEndpoint>',
  'field_delimiter'='|' --该参数可选。
);
```

WITH参数

参数	说明	是否必填	备注
connector	结果表类型。	是	固定值为*hologres*。
dbname	数据库名称。	是	无。
tablename	表名称。 **说明** 如果Schema不为Public时，则tablename需要填写为schema.tableName。	是	无。
username	用户名，请填写阿里云账号的AccessKey ID。	是	无。
password	密码，请填写阿里云账号的AccessKey Secret。	是	无。
endpoint	Hologres端点。	是	格式为*<ip>:<port>*。
field_delimiter	Hologres Sink支持将一个STRING字段按照field_delimiter切分成数组导入Hologres。	否	默认值为*"\u0002"*。
mutatetype	数据写入模式，详情请参见流式语义。	否	默认值为*insertorignore*。
partitionrouter	是否写入分区表。	否	默认值为*false*。
ignoredelete	是否忽略撤回消息。	否	默认值为*true*。 **说明** 仅在使用流式语义时生效。
createparttable	当写入分区表时，是否根据分区值自动创建不存在的分区表。 **说明** 如果分区值中存在短划线（-），暂不支持自动创建分区表。	否	• false（默认值）：不会自动创建。 • true：自动创建。 **说明** • VVR 2.1以上版本支持该参数。 • 请确保分区值不会出现脏数据，否则会创建错误的分区表导致Failover，建议慎用该参数。

参数	说明	是否必填	备注
useRpcMode	自VVP 2.4.0版本开始，默认通过JDBC实现Hologres Connector。VVP 2.4.0以前版本默认是通过RPC实现Hologres Connector。	否	通过JDBC实现会占用SQL连接，导致JDBC链接数增加。如果您需要降低SQL连接数，则可以设置该参数值为true。 **注意** 仅VVP 2.4.0及以上版本支持该参数。
connectionSize	单个Flink结果表任务所创建的JDBC连接池大小。 如果作业性能不足，建议您增加连接池大小。连接池大小和数据吞吐成正比。	否	默认值为3，单位为个。 **注意** 仅VVP 2.4.0及以上版本支持该参数。
jdbcWriteBatchSize	JDBC模式，Hologres Sink节点数据攒批的最大值。	否	默认值为256，单位为数据行数。 **注意** 仅VVP 2.4.0及以上版本支持该参数。
jdbcWriteFlushInterval	JDBC模式，Hologres Sink节点数据攒批写入Hologres的最长等待时间。	否	默认值为10000，即10秒。 **注意** 仅VVP 2.4.0及以上版本支持该参数。
ignoreNullWhenUpdate	当 `mutatetype='insertOrUpdate'` 时，是否忽略更新写入数据中的Null值。	否	取值说明如下： • false（默认值）：将Null值写到Hologres结果表里。 • true：忽略更新写入数据中的Null值。 **说明** 仅Flink计算引擎VVR 4.0及以上版本支持该参数。

流式语义

流处理，也称为流数据或流事件处理，即对一系列无界数据或事件连续处理。执行流数据或流事件处理的系统通常允许您指定一种可靠性模式或处理语义，保证整个系统处理数据的准确性，因为网络或设备故障等可能会导致数据丢失。

根据Hologres Sink的配置和Hologres表的属性，流式语义分为以下两种：

- Exactly-once（仅一次）：即使在发生各种故障的情况下，系统只处理一次数据或事件。
- At-least-once（至少一次）：如果在系统完全处理之前丢失了数据或事件，则从源头重新传输，因此可以多次处理数据或事件。如果第一次重试成功，则不必进行后续重试。

在Hologres结果表中使用流式语义，您需要注意以下几点：

- 如果Hologres物理表未设置主键，则Hologres Sink使用At-least-once语义。
- 如果Hologres物理表已设置主键，则Hologres Sink通过主键确保Exactly-once语义。当同主键数据出现多次时，您需要设置mutatetype参数确定更新结果表的方式，mutatetype取值如下：
  - insertorignore（默认值）：保留首次出现的数据，忽略后续所有数据。
  - insertorreplace：整行替换已有数据。
  - insertorupdate：替换部分已有数据。例如一张表有a、b、c和d四个字段，a是PK（Primary Key），写入Hologres时只写入a和b两个字段，在PK重复的情况下，系统只会更新b字段，c和d保持不变。

> **说明**
> - 当mutatetype设置为insertorupdate或insertorreplace时，系统根据主键更新数据。
> - Flink定义的结果表中的数据列数不一定要和Hologres物理表的列数一致，您需要保证缺失的列没有非空约束，即列值可以为Null，否则会报错。

- 默认情况下，Hologres Sink只能向一张表导入数据。如果导入数据至分区表的父表，即使导入成功，也会查询数据失败。您可以设置参数partitionRouter为true，开启自动将数据路由到对应分区表的功能。注意事项如下：
  - tablename参数需要填写为父表的表名。
  - 如果没有提前创建分区表，需要在WITH参数中启用createparttable=true，从而支持自动创建分区表，否则会导入失败。

宽表Merge和局部更新功能

在把多个流的数据写到一张Hologres宽表的场景中，会涉及到宽表Merge和数据的局部更新。下面通过一个示例来介绍如何设置。

假设有两个Flink数据流，一个数据流中包含A、B和C字段，另一个数据流中包含A、D和E字段，Hologres宽表WIDE_TABLE包含A、B、C、D和E字段，其中A字段为主键。具体操作如下：

1. 使用Flink SQL创建两张Hologres结果表，其中一张表只声明A、B和C字段，另一张表只声明A、D和E字段。这两张表都映射至宽表WIDE_TABLE。
2. 两张结果表的属性设置：
   - mutatetype设置为insertorupdate，可以根据主键更新数据。
   - ignoredelete设置为true，防止回撤消息产生Delete请求。
3. 将两个Flink数据流的数据分别INSERT至对应的结果表中。

> **说明** 在上述场景中，有如下限制：
> - 宽表必须有主键。
> - 每个数据流的数据都必须包含完整的主键字段。
> - 列存模式的宽表Merge场景在高RPS的情况下，CPU使用率会偏高，建议关闭表中字段的Dictionary encoding功能。

作为CTAS和CDAS的目标端

Hologres支持实时同步单表或整库级别的数据，在同步过程之中如果上游的表结构发生了变更也会实时同步到Hologres表中。

当上游需要同步的表发生了表结构变更，随着新数据流到Hologres表时，Flink会先触发Hologres修改相应的表结构，然后再将数据写入到相应的表中。以上过程，全部由Flink自动完成，您不需要关心实现细节。详情请参见CREATE TABLE AS(CTAS)语句、CREATE DATABASE AS(CDAS)语句和数据实时入仓入湖快速入门。

类型映射

Hologres字段类型	Flink字段类型
INT	INT
INT[]	ARRAY<INT>
BIGINT	BIGINT
BIGINT[]	ARRAY<BIGINT>
REAL	FLOAT
REAL[]	ARRAY<REAL>
DOUBLE PRECISION	DOUBLE
DOUBLE PRECISION[]	ARRAY<DOUBLE PRECISION>
BOOLEAN	BOOLEAN
BOOLEAN[]	ARRAY<BOOLEAN>
TEXT	VARCHAR
TEXT[]	ARRAY<TEXT>
NUMERIC	DECIMAL
DATE	DATE
TIMESTAMP WITH TIMEZONE	TIMESTAMP

9.2.4.2.10. 云数据库MongoDB版结果表

本文为您介绍云数据库MongoDB版结果表DDL定义、WITH参数和代码示例。

什么是云数据库MongoDB

云数据库MongoDB完全兼容MongoDB协议，基于飞天分布式系统和高可靠存储引擎，提供多节点高可用架构、弹性扩容、容灾、备份恢复、性能优化等功能。

前提条件

已创建云数据库MongoDB实例，详情请参见创建单节点实例。

使用限制

- 仅Flink计算引擎VVR 2.0.0及以上版本支持云数据库MongoDB Connector。

- 仅支持将云数据库MongoDB作为结果表使用。

DDL定义

```
CREATE TABLE mongodb_sink(
  id INT,
  number INT
) WITH (
  'connector' = 'mongodb',
  'database' = '<yourDatabase>',
  'collection' = '<yourCollection>',
  'uri' = '<yourUri>',
  'maxConnectionIdleTime' = '<yourMaxConnectionIdleTime>',
  'batchSize' = '1024'
);
```

WITH参数

参数	说明	是否必填	备注
connector	结果表类型。	是	固定值为 `mongodb` 。
database	数据库名称。	是	无。
collection	数据集合。	是	无。
uri	MongoDB连接串。	是	例如 `mongodb://123@dds-/admin?replicaSet=mgset-32966591` 。
maxConnectionIdleTime	连接超时时长。	否	默认值为60000，单位为毫秒。
batchSize	每次批量写入的条数。	否	默认值为1024。

代码示例

```
CREATE TEMPORARY TABLE datagen_source (
  v INT,
  p INT
) with (
  'connector' = 'datagen'
);
CREATE TEMPORARY TABLE mongodb_sink(
  id INT,
  number INT
) with (
  'connector'='mongodb',
  'database' = '<yourDatabase>',
  'collection' = '<yourCollection>',
  'uri'='<yourUri>'
);
INSERT INTO mongodb_sink
   SELECT v, p
FROM datagen_source;
```

9.2.4.2.11. Elasticsearch结果表

本文为您介绍Elasticsearch（ES）结果表DDL定义、WITH参数、类型映射和代码示例。

什么是云数据库Elasticsearch

阿里云Elasticsearch兼容开源Elasticsearch的功能，以及Security、Machine Learning、Graph、APM等商业功能，致力于数据分析、数据搜索等场景服务。为您提供企业级权限管控、安全监控告警、自动报表生成等场景服务。

前提条件

- 已创建Elasticsearch索引，详情请参见步骤一：创建实例。
- 已配置Elasticsearch公网或私网访问白名单，详情请参见配置实例公网或私网访问白名单。

使用限制

仅Flink计算引擎VVR 2.0.0及以上版本支持Elasticsearch Connector。

DDL定义

```
 CREATE TABLE es_sink(
   user_id   STRING,
   user_name   STRING,
   uv BIGINT,
   pv BIGINT,
   PRIMARY KEY (user_id) NOT ENFORCED  -- 主键可选，如果定义了主键，则作为文档ID，否则文档ID将为随机值。
)  WITH (
   'connector' = 'elasticsearch-6',
   'hosts' = '<yourHosts>',
   'index' = '<yourIndex>',
   'document-type' = '<yourEelasticsearch.types>',
   'username' ='<yourElasticsearch.accessId>',
   'password' ='<yourElasticsearch.accessKey>'
);
```

说明

- 仅支持Elasticsearch 6.x和7.x版本。
- DDL中的字段均对应Elasticsearch文档中的字段，不支持将文档ID写入表中。

WITH参数

参数	说明	是否必填	备注
connector	结果表类型。	是	固定值为 `elasticsearch-6` 或 `elasticsearch-7` 。
hosts	Server地址。	是	例如：*127.0.0.1:9200*。
index	索引名称。	是	默认为空，不进行权限验证。
document-type	文档类型。	• `elasticsearch-6` ：必填 • `elasticsearch-7` ：不支持	无。
username	用户名。	否	默认为空，不进行权限验证。
password	密码。	否	如果定义了username，则必须定义非空的password。
document-id.key-delimiter	文档ID的分隔符。	否	默认值为 `_` 。

参数	说明	是否必填	备注
failure-handler	Elasticsearch请求失败时的故障处理策略。	否	可选策略如下： • fail（默认值）：如果请求失败，则作业失败。 • ignore：忽略失败并删除请求。 • retry-rejected：重新添加由于队列容量满而失败的请求。 • custom class name：用于使用ActionRequestFailureHandler子类进行故障处理。
sink.flush-on-checkpoint	是否在checkpoint时执行flush。	否	默认值为true。禁用该功能后，在Elasticsearch进行Checkpoint时，connector将不等待确认所有pending请求已完成。因此，connector不会为请求提供at-least-once保证。
sink.bulk-flush.backoff.strategy	如果由于临时请求错误导致flush操作失败，则设置sink.bulk-flush.backoff.strategy指定重试策略。	否	• DISABLED（默认值）：不执行重试，即第一次请求错误后失败。 • CONSTANT：常量回退，即每次回退等待时间相同。 • EXPONENTIAL：指数回退，即每次回退等待时间指数递增。
sink.bulk-flush.backoff.max-retries	最大回退重试次数。	否	默认值为8。
sink.bulk-flush.backoff.delay	每次回退尝试之间的延迟。	否	• CONSTANT：每次重试之间的延迟。 • EXPONENTIAL：这是初始基准延迟。
sink.bulk-flush.max-actions	每个批量请求的最大缓冲操作数。	否	默认值为1000，0表示禁用该功能。
sink.bulk-flush.max-size	存放请求的缓冲区内存最大值。	否	单位为MB，默认值为2，0表示禁用该功能。
sink.bulk-flush.interval	flush的间隔。	否	默认值为1s，0表示禁用该功能。
connection.path-prefix	要添加到每个REST通信中的前缀字符串。	否	默认值为空。

文档ID

Elasticsearch Sink可以根据是否定义了主键确定是其在upsert模式还是在append模式下工作。如果定义了主键，Elasticsearch Sink将在upsert模式下工作，该模式可以消费包含UPDATE和DELETE的消息。如果未定义主键，Elasticsearch Sink将以append模式工作，该模式只能消费INSERT消息。

在Elasticsearch Sink中，主键用于计算Elasticsearch的文档ID。文档ID为最多512个字节不包含空格的字符串。Elasticsearch Sink通过使用document-id.key-delimiter指定的键分隔符按照DDL中定义的顺序连接所有主键字段，从而为每一行生成一个文档ID字符串。某些类型（例如BYTES、ROW、ARRAY和MAP等）由于没有对应的字符串表示形式，所以不允许其作为主键字段。如果未指定主键，Elasticsearch将自动生成随机的文档ID。

动态索引

Elasticsearch Sink同时支持静态索引和动态索引：

- 如果使用静态索引，则索引选项值应为纯字符串，例如myusers，所有记录都将被写入myusers索引。
- 如果使用动态索引，可以使用 `{field_name}` 引用记录中的字段值以动态生成目标索引。您还可以使用{field_name|date_format_string}将TIMESTAMP、DATE和TIME类型的字段值转换为date_format_string指定的格式。date_format_string与Java的DateTimeFormatter兼容。例如，如果设置为myusers-{log_ts|yyyy-MM-dd}，则log_ts字段值为2020-03-27 12:25:55的记录将被写入myusers-2020-03-27索引。

类型映射

Flink以JSON来解析Elasticsearch数据，详情请参见数据类型映射关系。

代码示例

```
CREATE TEMPORARY TABLE datagen_source (
  id STRING,
  name STRING,
  uv BIGINT
) with (
  'connector' = 'datagen'
);
CREATE TEMPORARY TABLE es_sink (
  user_id STRING,
  user_name STRING,
  uv BIGINT,
  PRIMARY KEY (user_id) NOT ENFORCED -- 主键可选，如果定义了主键，则作为文档ID，否则文档ID将为随机值。
) WITH (
  'connector' = 'elasticsearch-7',
  'hosts' = '<yourHosts>',
  'index' = '<yourIndex>',
  'document-type' = '<yourElasticsearch.types>',
  'username' ='<yourElasticsearch.accessId>',
  'password' ='<yourElasticsearch.accessKey>'
);
INSERT INTO es_sink
   SELECT id, name, uv
FROM datagen_source;
```

9.2.4.2.12. Phoenix5结果表

本文为您介绍Phoenix5结果表DDL定义、WITH参数和代码示例。

注意 Phoenix5结果表仅支持INSERT，不支持UPDATE和DELETE。

什么是Phoenix5

HBase SQL服务基于Phoenix 5.x为HBase 2.x提供SQL能力，通过轻客户端即可快速连接访问。SQL服务挂载了SLB负载均衡，通过round robin模式将请求均匀分发在每个query server节点。此外轻客户端还支持Python、Go等多语言访问。Phoenix5详情请参见HBase SQL(Phoenix) 5.x 使用说明。

前提条件

已开通HBase SQL服务，详情请参见HBase SQL(Phoenix) 5.x 使用说明。

使用限制

仅Flink计算引擎VVR 2.0.0及以上版本支持Phoenix5 Connector。

DDL定义

```
CREATE TABLE phoenix5_sink (
  `STATE` VARCHAR,
  CITY VARCHAR,
  POPULATION BIGINT,
  PRIMARY KEY (`STATE`, CITY) NOT ENFORCED
) WITH (
  'connector' = 'phoenix5',
  'serverUrl' = '<yourserverUrl>',
  'tableName' = '<yourTableName>',
  'batchsize' = '25'
);
```

WITH参数

参数	说明	是否必填	备注
connector	结果表类型。	是	默认值为 `phoenix5` 。
serverUrl	Phoenix5的Query Server地址： • 如果Phoenix5是在集群中创建的，则serverUrl是负载均衡服务的URL地址。 • 如果Phoenix5是在单机中创建的，则serverUrl是单机的URL地址。	是	您需要在云数据库HBase实例中开启Hbase SQL服务，详情请参见开启HBase。
tableName	Phoenix5表名。	是	无。
batchSize	一次批量写入的条数。	否	默认值为20。

代码示例

包含Phoenix5结果表的Flink作业代码示例如下。

```
CREATE TEMPORARY table datagen_source (
  `id` varchar,
  `name` varchar,
  `age` varchar,
  `birthday` varchar
) WITH (
  'connector' = 'datagen'
);
CREATE TEMPORARY table phoenix5_sink (
  `id` varchar,
  `name` varchar,
  `age` varchar,
  `birthday` varchar,
  primary key (id) NOT ENFORCED
) WITH (
  'connector' = 'phoenix5',
  'serverurl' = '<yourserverUrl>',
  'tablename' = '<yourTableName>',
  'batchsize' = '25'
);
INSERT INTO phoenix5_sink
  SELECT  `id` ,`name` , `age` ,`birthday`
FROM datagen_source;
```

9.2.4.2.13. 云数据库RDS MySQL结果表

本文为您介绍云数据库RDS MySQL结果表的DDL定义、WITH参数和类型映射。

什么是云数据库RDS MySQL

RDS MySQL基于阿里巴巴的MySQL源码分支，经过双十一高并发、大数据量的考验，拥有优良的性能。RDS MySQL支持实例管理、账号管理、数据库管理、备份恢复、白名单、透明数据加密以及数据迁移等基本功能。RDS MySQL详情请参见概述。

前提条件

- 已创建RDS MySQL数据库和表，详情请参见创建数据库和账号。
- 已设置IP白名单，详情请参见通过客户端、命令行连接RDS MySQL实例。

使用限制

仅Flink计算引擎VVR 2.0.0及以上版本支持云数据库RDS MySQL Connector。

DDL定义

```
CREATE TABLE rds_sink (
   id INT,
   num BIGINT,
   PRIMARY KEY (id) NOT ENFORCED
) WITH (
   'connector' = 'rds',
   'tableName' = '<yourTablename>',
   'userName' = '<yourUsername>',
   'password' = '<yourPassword>',
   'url' = '<yourUrl>'
);
```

说明

- Flink写入RDS或DRDS数据库结果表原理：针对Flink每行结果数据，拼接成一行SQL语句，输入至目标端数据库，然后执行。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，因此在DDL中不声明该自增字段。例如ID是自增字段，Flink DDL不声明该自增字段，则数据库在一行数据写入过程中会自动填补相关自增字段。
- 如果DRDS有分区表，拆分键必须在DDL里PRIMARY KEY（）中声明，否则拆分的表无法写入。
- DDL声明的字段必须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	数据类型	是否必填	备注
connector	结果表类型。	String	是	固定值为 `rds` 。
tableName	表名。	String	是	无。
userName	用户名。	String	是	无。
password	密码。	String	是	无。
url	URL地址。	String	是	云数据库RDS版专有网络VPC地址，即内网地址，详情请参见查看或修改内外网地址和端口。URL的格式为： `jdbc:mysql://<内网地址>/<databaseName>` ，其中databaseName为对应的数据库名称。 **说明** 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。

参数	说明	数据类型	是否必填	备注
maxRetryTimes	写入数据失败后，重试写入的最大次数。	Integer	否	参数默认值取值情况如下： • 在Flink计算引擎VVR 4.0.7及以上版本，该参数默认值为10。 • 在Flink计算引擎VVR 4.0.0~4.0.6版本，该参数默认值为3。 • 在Flink计算引擎VVR 3.x版本及以下版本，该参数默认值为3。 说明 Flink计算引擎版本推荐您使用4.0.10及以上或4.1.1及以上版本。
batchSize	一次批量写入的条数。	Integer	否	参数默认值取值情况如下： • 在Flink计算引擎VVR 4.0.7及以上版本，该参数默认值为4096。 • 在Flink计算引擎VVR 4.0.0~4.0.6版本，该参数默认值为5000。 • 在Flink计算引擎VVR 3.x版本及以下版本，该参数默认值为100。 说明 Flink计算引擎版本推荐您使用4.0.10及以上或4.1.1及以上版本。
bufferSize	内存中缓存的数据条数。batchSize或bufferSize任一到达阈值都会触发写入。	Integer	否	默认值为10000。 说明 • 仅Flink计算引擎VVR 4.0.7及以上版本支持该参数。 • 需指定主键后，该参数才生效。
flushIntervalMs	清空缓存的时间间隔。表示如果缓存中的数据在等待指定时间后，依然没有达到输出条件，系统统会自动输出缓存中的所有数据。	Integer	否	单位为毫秒，参数默认值取值情况如下： • 在Flink计算引擎VVR 4.0.7及以上版本，该参数默认值为2000。 • 在Flink计算引擎VVR 4.0.0~4.0.6版本，该参数默认值为0。 • 在Flink计算引擎VVR 3.x版本及以下版本，该参数默认值为1000。 说明 Flink计算引擎版本推荐您使用4.0.10及以上或4.1.1及以上版本。

参数	说明	数据类型	是否必填	备注
ignoreDelete	是否忽略Delete操作。	Boolean	否	参数取值如下： • true：忽略Delete操作。 • false（默认值）：接受Delete操作。 说明 仅Flink计算引擎VVR 4.0.7及以上版本支持该参数。
connectionMaxActive	数据库连接池大小。	Integer	否	默认值为40。 说明 • 仅Flink计算引擎VVR 4.0.7及以上版本支持该参数。 • 如果出现获取连接超时的问题，说明连接池不够用，可适当增大连接池的大小。

类型映射

RDS字段类型	Flink字段类型
BOOLEAN	BOOLEAN
TINYINT	TINYINT
SMALLINT	SMALLINT
TINYINT UNSIGNED	SMALLINT
INT	INT
SMALLINT UNSIGNED	INT
BIGINT	BIGINT
INT UNSIGNED	BIGINT
BIGINT UNSIGNED	DECIMAL(20,0)
FLOAT	FLOAT
DECIMAL	DECIMAL
DOUBLE	DOUBLE
DATE	DATE

RDS字段类型	Flink字段类型
TIME	TIME
TIMESTAMP	TIMESTAMP
VARCHAR	VARCHAR
VARBINARY	VARBINARY

代码示例

```
CREATE TEMPORARY TABLE datagen_source (
  `name` VARCHAR,
  `age` INT
) WITH (
  'connector' = 'datagen'
);
CREATE TEMPORARY TABLE rds_sink (
  `name` VARCHAR,
  `age` INT
) WITH (
  'connector' = 'rds',
  'password' = '<yourPassword>',
  'tableName' = '<yourTablename>',
  'url' = '<yourUrl>',
  'userName' = '<yourUsername>'
);
INSERT INTO rds_sink
SELECT  * FROM datagen_source;
```

常见问题

- Flink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？
- 使用RDS表中的唯一索引进行GROUP BY时需要注意什么？
- 为什么MySQL物理表（包含RDS MySQL和ADB）的INT UNSIGNED字段类型，在Flink SQL中要被声明为其他类型？
- 报错：Incorrect string value: '\xF0\x9F\x98\x80\xF0\x9F...' for column 'test' at row 1

9.2.4.2.14. 云原生数据仓库AnalyticDB MySQL版3.0结果表

本文为您介绍云原生数据仓库AnalyticDB MySQL版3.0结果表的DDL定义、WITH参数和类型映射。

什么是云原生数据仓库AnalyticDB MySQL版

云原生数据仓库AnalyticDB MySQL版是融合数据库、大数据技术于一体的云原生企业级数据仓库服务。AnalyticDB MySQL版支持高吞吐的数据实时增删改、低延时的实时分析和复杂ETL，兼容上下游生态工具，可用于构建企业级报表系统、数据仓库和数据服务引擎。

前提条件

- 已创建AnalyticDB MySQL集群并创建表，详情请参见创建集群和CREATE TABLE。

- 已设置白名单，详情请参见设置白名单。

使用限制

仅Flink计算引擎VVR 2.0.0及以上版本支持云原生数据仓库AnalyticDB MySQL版3.0 Connector。

DDL定义

```
CREATE TABLE adb_sink (
  id INT,
  num BIGINT,
  PRIMARY KEY (id) NOT ENFORCE
) WITH (
  'connector' = 'adb3.0',
  'tableName' = '<yourTablename>',
  'userName' = '<yourUsername>',
  'password' = '<yourPassword>',
  'url' = 'jdbc:mysql://<yourNetworkAddress>:<PortId>/<yourDatabaseName>'
);
```

WITH参数

参数	注释说明	是否必选	备注
connector	结果表类型。	是	固定值为 `adb3.0` 。
tableName	表名。	是	无。
username	用户名。	是	无。
password	密码。	是	无。
url	JDBC连接地址。	是	云原生数据仓库AnalyticDB MySQL版数据库地址。示例： `url='jdbc:mysql://databaseName****-cn-shenzhen-a.ads.aliyuncs.com:10014/databaseName'` 。 说明 • 云原生数据仓库AnalyticDB MySQL版数据库连接信息，请参见URL地址查询。 • databaseName为云原生数据仓库AnalyticDB MySQL版数据库名称。
maxRetryTimes	写入数据失败后，重试写入的最大次数。	否	参数默认值取值情况如下： • 在Flink计算引擎VVR 3.x版本及以下版本，该参数默认值为3。 • 在Flink计算引擎VVR 4.0.10及以上版本，该参数默认值为10。

参数	注释说明	是否必选	备注
batchSize	一次批量写入的条数。	否	参数默认值取值情况如下： • 在Flink计算引擎VVR 3.x版本及以下版本，该参数默认值为100。 • 在Flink计算引擎VVR 4.0.10及以上版本，该参数默认值为1000。
bufferSize	内存中缓存的数据条数。batchSize或bufferSize任一到达阈值都会触发写入。	否	默认值为1000。 说明 • 仅Flink计算引擎VVR 4.0.10及以上版本支持该参数。 • 需指定主键后，该参数才生效。
flushIntervalMs	清空缓存的时间间隔。表示如果缓存中的数据在等待指定时间后，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。	否	单位为毫秒，参数默认值取值情况如下： • 在Flink计算引擎VVR 3.x版本及以下版本，该参数默认值为1000。 • 在Flink计算引擎VVR 4.0.10及以上版本，该参数默认值为3000。
ignoreDelete	是否忽略Delete操作。	否	参数取值如下： • true：忽略Delete操作。 • false（默认值）：接受Delete操作。 说明 仅Flink计算引擎VVR 4.0.10及以上版本支持该参数。
replaceMode	是否采用replace into语法插入数据。	否	参数取值如下： • true（默认值）：采用replace into语法插入数据。 • false：采用insert into on duplicate key语法插入数据。 说明 • 仅Flink计算引擎VVR 4.0.10及以上版本支持该参数。 • 仅AnalyticDB MySQL 3.1.3.5及以上版本支持该参数。

参数	注释说明	是否必选	备注
excludeUpdateColumns	表示更新主键值相同的数据时，忽略指定字段的更新。	否	如果忽略指定的字段为多个时，则需要使用英文逗号（,）分割。例如 `excludeUpdateColumns=column1,column2` 。 **说明** 仅在 `replaceMode=false` 时，该参数才生效。在 `replaceMode=true` 时，对应字段会被更新为null。
connectionMaxActive	线程池大小	否	默认值为40。 **说明** 仅Flink计算引擎VVR 4.0.10及以上版本支持该参数。

类型映射

云原生数据仓库AnalyticDB MySQL版3.0字段类型	Flink字段类型
BOOLEAN	BOOLEAN
TINYINT	INT
SMALLINT	INT
INT	INT
BIGINT	BIGINT
DOUBLE	DOUBLE
VARCHAR	VARCHAR
DATETIME	TIMESTAMP
DATE	DATE

说明 因为MySQL的JDBC Driver在获取数据时，由于精度问题，Flink可能会采用不同的数据类型进行数据承接。

代码示例

```
CREATE TEMPORARY TABLE datagen_source (
  `name` VARCHAR,
  `age` INT
) WITH (
  'connector' = 'datagen'
);
CREATE TEMPORARY TABLE adb_sink (
  `name` VARCHAR,
  `age` INT
) WITH (
  'connector' = 'adb3.0',
  'password' = '<yourPassword>',
  'tableName' = '<yourTablename>',
  'url' = '<yourUrl>',
  'userName' = '<yourUsername>'
);
INSERT INTO adb_sink
SELECT  * FROM datagen_source;
```

常见问题

为什么MySQL物理表（包含RDS MySQL和ADB）的INT UNSIGNED字段类型，在Flink SQL中要被声明为其他类型？

9.2.4.2.15. ClickHouse结果表

本文为您介绍ClickHouse结果表的DDL定义，以及创建结果表时使用的WITH参数、类型映射和代码示例。

什么是ClickHouse

ClickHouse是一个用于联机分析（OLAP）的列式数据库管理系统，详情请参见什么是ClickHouse。

前提条件

- 已创建ClickHouse表，详情请参见创建表。
- 已配置白名单。
  - 如果您使用的是阿里云数据库ClickHouse，配置白名单详情请参见设置白名单。
  - 如果您使用的是阿里云E-MapReduce的ClickHouse，配置白名单详情请参见管理安全组。
  - 如果您使用的是阿里云ECS上自建的ClickHouse，配置白名单详情请参见安全组概述。
  - 如果为其他情况，请您自行配置ClickHouse所在机器的白名单让其可被Flink所在机器访问即可。

使用限制

- 仅Flink计算引擎VVR 3.0.2及以上版本支持ClickHouse Connector。
- 仅Flink计算引擎VVR 3.0.3，VVR 4.0.7及以上版本支持ignoreDelete选项。
- 仅Flink计算引擎VVR 4.0.10及以上版本支持ClickHouse的Nested类型。
- 仅Flink计算引擎VVR 4.0.11及以上版本支持直接将数据写入到ClickHouse分布式表对应的本地表。
- 仅Flink计算引擎VVR 4.0.11及以上版本提供写EMR的ClickHouse的Exactly Once语义。

DDL定义

```
CREATE TABLE clickhouse_sink (
  id INT,
  name VARCHAR,
  age BIGINT,
  rate FLOAT
) WITH (
  'connector' = 'clickhouse',
  'url' = '<yourUrl>',
  'userName' = '<yourUsername>',
  'password' = '<yourPassword>',
  'tableName' = '<yourTablename>',
  'maxRetryTimes' = '3',
  'batchSize' = '8000',
  'flushIntervalMs' = '1000'
  'ignoreDelete' = 'true',
  'shadWrite' = 'false',
  'writeMode' = 'partition',
  'shardingKey' = 'id'
);
```

WITH参数

参数	说明	是否必选	备注
connector	结果表类型。	是	固定值为*clickhouse*。
url	ClickHouse的JDBC连接地址。	是	URL格式为 `jdbc:clickhouse://<yourNetworAddress>:<PortId>/<yourDatabaseName>` 如果不写数据库名称，则使用默认的default数据库。 **说明** 如果您要将数据写入ClickHouse分布式表，则URL为该分布式表所在节点的JDBC URL。
userName	ClickHouse的用户名。	是	无。
password	ClickHouse的密码。	是	无。
tableName	ClickHouse的表名称。	是	无。
maxRetryTimes	向结果表插入数据失败后的最大尝试次数。	否	默认值为3。
batchSize	一次批量写入的数据条数。	否	默认值为100。 如果缓存中的数据条数达到了batchSize参数值，或者等待时间超过flushIntervalMs后，系统将会自动将缓存中的数据写入ClickHouse表中。

参数	说明	是否必选	备注
flushIntervalMs	清空缓存的时间间隔。	否	默认值为1000，单位为毫秒。
ignoreDelete	是否忽略Delete消息。	否	参数取值如下： • true（默认值）：忽略。 • false：不忽略。 如果为false，并且在DDL中声明了Primary Key，则会使用ClickHouse的ALTER语句来删除数据。
shardWrite	对于ClickHouse分布式表，是否直接写ClickHouse的本地表。	否	参数取值如下： • true：跳过分布式表，直接将数据写到该ClickHouse分布式表对应的本地表。 Flink将自动查询ClickHouse的集群信息，得到对应的本地表信息并进行写入。此时tableName应该为分布式表的名字。您也可以在URL中手动指定要将数据写到哪些节点的本地表中。此时tableName应该为本地表名字。示例如下。 `'url' = 'jdbc:clickhouse://192.168.1.1:8123/default;jdbc:clickhouse://192.168.1.2:8123/default;' 'tableName' = 'local_table'` 如果需要提高写ClickHouse分布式表的吞吐量，则建议将该值设置为true。 • false（默认值）：先写ClickHouse的分布式表，再由分布式表写入对应的本地表。此时tableName应为分布式表的名称。
writeMode	对于ClickHouse分布式表，采用何种策略写ClickHouse的本地表。	否	参数取值如下： • default（默认值）：表示总是写入到第一个节点的本地表。 • partition：表示将数据按key写到同一个节点的本地表。 • random：表示随机写到某个节点的本地表。
shardingKey	按何种key将数据写到同一个节点的本地表。	否	当writeMode取值为partition时，shardingKey值必填，可包含多个字段，多个字段以英文逗号（,）分隔。

参数	说明	是否必选	备注
exactlyOnce	是否开启exactlyOnce语义。 说明 • 目前仅支持写EMR的ClickHouse的Exactly Once语义。所以只有当您写EMR的ClickHouse时，才能将exactlyOnce设置为true。 • 不支持以partition策略写ClickHouse的本地表的Exactly Once语义。所以如果exactlyOnce设置为true，则writeMode不能设置为partition。	否	参数取值如下： • true：开启。 • false（默认值）：不开启。

类型映射

ClickHouse字段类型	Flink字段类型
UInt8	BOOLEN
Int8	TINYINT
Int16	SMALLINT
Int32	INTEGER
Int64	BIGINT
Float32	FLOAT
Float64	DOUBLE
FixedString	CHAR
String	VARCHAR
FixedString	BINARY

ClickHouse字段类型	Flink字段类型
String	VARBINARY
Date	DATE
DateTime	TIMESTAMP(0)
Datetime64(x)	TIMESTAMP(x)
Decimal	DECIMAL
ARRAY	ARRAY
Nested	

说明 ClickHouse暂不支持Flink的TIME、MAP、MULTISET和ROW类型。

对于ClickHouse的Nested类型，需要将其映射成Flink的ARRAY类型，例如：

```
// ClickHouse
CREATE TABLE visits (
  StartDate Date,
  Goals Nested
  (
    ID UInt32,
    OrderID String
  )
  ...
);
```

需要映射为：

```
// Flink
CREATE TABLE visits (
  StartDate DATE,
  `Goals.ID` ARRAY<INTEGER>,
  `Goals.OrderID` ARRAY<STRING>
);
```

注意 ClickHouse的DateTime类型可以精确到秒，Datetime64可以精确到纳秒。因为ClickHouse官方提供的JDBC写Datetime64数据类型会出现精度丢失，只能精确到秒的问题。所以目前通过Flink只能写入秒级别的TIMESTAMP，即TIMESTAMP(0)。

代码示例

- 示例1：写ClickHouse单节点表。

```
CREATE TEMPORARY TABLE clickhouse_source (
  id INT,
  name VARCHAR,
  age BIGINT,
  rate FLOAT
) WITH (
  'connector' = 'datagen',
  'rows-per-second' = '50'
);
CREATE TEMPORARY TABLE clickhouse_output (
  id INT,
  name VARCHAR,
  age BIGINT,
  rate FLOAT
) WITH (
  'connector' = 'clickhouse',
  'url' = '<yourUrl>',
  'userName' = '<yourUsername>',
  'password' = '<yourPassword>',
  'tableName' = '<yourTablename>'
);
INSERT INTO clickhouse_output
SELECT
  id,
  name,
  age,
  rate
FROM clickhouse_source;
```

- 示例二：写ClickHouse分布式表

  假设您已经有三个本地表，表名为local_table_test，分别在192.168.1.1、192.168.1.2和192.168.1.3节点上。然后基于这三个本地表，创建了一个分布式表distributed_table_test。此时，如果您希望Flink可以直接写本地表，并且可以按照某个key将相同key的数据写到同一个节点的本地表中，则DDL代码示例如下。

```
CREATE TEMPORARY TABLE clickhouse_source (
  id INT,
  name VARCHAR,
  age BIGINT,
  rate FLOAT
) WITH (
  'connector' = 'datagen',
  'rows-per-second' = '50'
);
CREATE TEMPORARY TABLE clickhouse_output (
  id INT,
  name VARCHAR,
  age BIGINT,
  rate FLOAT
) WITH (
  'connector' = 'clickhouse',
  'url' = '<yourUrl>',
  'userName' = '<yourUsername>',
  'password' = '<yourPassword>',
  'tableName' = 'local_table_test',
  'shardWrite' = 'true',
  'writeMode' = 'partition',
  'shardingKey' = 'name'
);
INSERT INTO clickhouse_output
SELECT
  id,
  name,
  age,
  rate
FROM clickhouse_source;
```

常见问题

ClickHouse结果表是否支持回撤更新数据？

9.2.4.2.16. print结果表

本文为您介绍print结果表的DDL定义，以及创建print结果表时使用的WITH参数。您可以通过print结果表测试流作业或者调试生产作业，验证数据结果是否符合预期。

使用限制

仅Flink计算引擎VVR 2.1.5及以上版本支持print Connector。

注意事项

- 因为print sink数据输出为info日志，所以如果您需要查看print结果表的结果数据，则需要将日志级别调至info。否则，会查不到结果数据。
- 应安全部门要求，系统对taskmanager.log日志展示数据条数进行了限制。如果您有排查脏数据或特定数据等需求，建议在Where条件中指定业务场景相关条件后，进行print操作，以避免因为数据条数限制无法排查。

DDL定义

```
CREATE TABLE print_table (
  a INT,
  b varchar
) WITH (
  'connector'='print',
  'logger'='true'
);
```

WITH参数

参数	说明	是否必填	备注
connector	结果表类型	是	固定值为print。
logger	控制台是否显示数据结果	否	取值如下： • false（默认值）：不显示。 • true：显示。
print-identifier	数据结果标识	否	在日志中通过数据结果标识检索信息。
sink.parallelism	结果表并行度	否	默认与上游并行度一致。

常见问题

如何在控制台查看print数据结果？

9.2.4.2.17. Upsert Kafka结果表

本文为您介绍Upsert Kafka结果表的DDL定义、WITH参数和示例。

什么是Upsert Kafka

Upsert Kafka基于开源Flink社区的Upsert Kafka版实现，详情请参见Upsert Kafka SQL连接器。Upsert Kafka可以消费ChangeLog流，支持Flink将INSERT和UPDATE_AFTER数据作为正常的Kafka消息写入Kafka Topic，并将DELETE数据以Value为空的Kafka消息写入Kafka Topic，即以Upsert方式将数据写入Kafka Topic。此外，Flink将根据主键列的值对数据进行分区，从而保证主键上的消息有序，因此同一主键上的更新或删除消息将落在同一分区中。

使用限制

- 仅Flink计算引擎VVR 2.1.4及以上版本支持Upsert Kafka Connector。
- Upsert Kafka Connector仅支持将结果数据写入到kafka 0.10及以上版本。
- Upsert Kafka Connector仅支持Kafka 2.4版本的生产者配置项，详情请参见生产者配置项。

DDL定义

以表名upsert_kafka_sink为例介绍DDL定义。

```
CREATE TABLE upsert_kafka_sink (
  user_region STRING,
  pv BIGINT,
  uv BIGINT,
  PRIMARY KEY (user_region) NOT ENFORCED
) WITH (
  'connector' = 'upsert-kafka',
  'topic' = '<yourTopicName>',
  'properties.bootstrap.servers' = '...',
  'key.format' = 'avro',
  'value.format' = 'avro'
);
```

说明 Upsert Kafka必须声明主键。

WITH参数

参数	说明	是否必选	数据类型	备注
connector	结果表类型。	是	STRING	固定值为 `upsert-kafka` 。
topic	结果表对应的Topic。	是	STRING	无。
properties.bootstrap.servers	Kafka Broker地址。	是	STRING	格式为 `host:port,host:port,host:port` ，以英文逗号（,）分割。
key.format	Upsert Kafka消息中Key部分序列化的格式。	是	STRING	支持的格式如下： • csv • json • avro
value.format	Upsert Kafka消息中Value部分序列化的格式。	是	STRING	支持的格式如下： • csv • json • avro
value.fields-include	指定出现在Value中的字段。	否	String	取值如下： • ALL（默认值）：Schema中所有字段，包括主键字段。 • EXCEPT_KEY：Schema中所有字段，不包括主键字段。

参数	说明	是否必选	数据类型	备注
properties.*	指定Kafka参数。	否	String	后缀名必须匹配定义在Apache Kafka文档中的参数名。Flink会自动移除properties. 前缀，并将转换后的参数名及值传入Kafka客户端。例如，您可以通过设置 `'properties.allow.auto.create.topics' = 'false'` 来禁止自动创建Topic。 **说明** 已在WITH参数定义的Kafka参数是不允许通过该方式指定参数，因为Flink会重写该参数的值。例如 `'key.deserializer'` 和 `'value.deserializer'` 。

Kafka Producer配置参数详情请参见Kafka官网生产者配置项列表。如果您还需要直接配置Connector使用的Kafka Producer，可以在Kafka Producer配置参数前添加 `properties` 前缀，并将该Kafka Producer配置信息追加至WITH参数。例如Kafka集群需要SASL（Simple Authentication and Security Layer）认证，代码示例如下。

```
CREATE TABLE kafkaTable (
    ...
) WITH (
    ...
    'properties.security.protocol' = 'SASL_PLAINTEXT',
    'properties.sasl.mechanism' = 'PLAIN',
    'properties.sasl.jaas.config' = 'org.apache.flink.kafka.shaded.org.apache.kafka.common.
security.plain.PlainLoginModule required username="USERNAME" password="PASSWORD";'
);
```

示例

```
--创建统计PV和UV的源表。
CREATE TABLE pageviews (
  user_id BIGINT,
  page_id BIGINT,
  viewtime TIMESTAMP,
  user_region STRING,
  WATERMARK FOR viewtime AS viewtime - INTERVAL '2' SECOND
) WITH (
  'connector' = 'kafka',
  'topic' = '<yourTopicName>',
  'properties.bootstrap.servers' = '...',
  'format' = 'json'
);
--创建Upsert Kafka结果表。
CREATE TABLE pageviews_per_region (
  user_region STRING,
  pv BIGINT,
  uv BIGINT,
  PRIMARY KEY (user_region) NOT ENFORCED
) WITH (
  'connector' = 'upsert-kafka',
  'topic' = '<yourTopicName>',
  'properties.bootstrap.servers' = '...',
  'key.format' = 'avro',
  'value.format' = 'avro'
);
--将统计的PV、UV数据写入结果表中。
INSERT INTO pageviews_per_region
SELECT
  user_region,
  COUNT(*),
  COUNT(DISTINCT user_id)
FROM pageviews
GROUP BY user_region;
```

9.2.4.2.18. Hudi结果表

本文为您介绍Hudi结果表的背景信息、使用限制、 DDL定义、WITH参数和示例。

背景信息

- 什么是Hudi

  Hudi的定义、特性及典型场景详情如下表所示。

类别	详情
定义	Apache Hudi是一种开源的数据湖表格式框架。Hudi基于对象存储或者HDFS组织文件布局，保证ACID，支持行级别的高效更新和删除，从而降低数据ETL开发门槛。同时该框架还支持自动管理及合并小文件，保持指定的文件大小，从而在处理数据插入和更新时，不会创建过多的小文件，引发查询端性能降低，避免手动监控和重写小文件的运维负担。结合Flink、Presto、Spark等计算引擎进行数据入湖和计算分析，常用来支持DB入湖加速、增量数据实时消费和数仓回填等需求。详情请参见Apache Hudi。

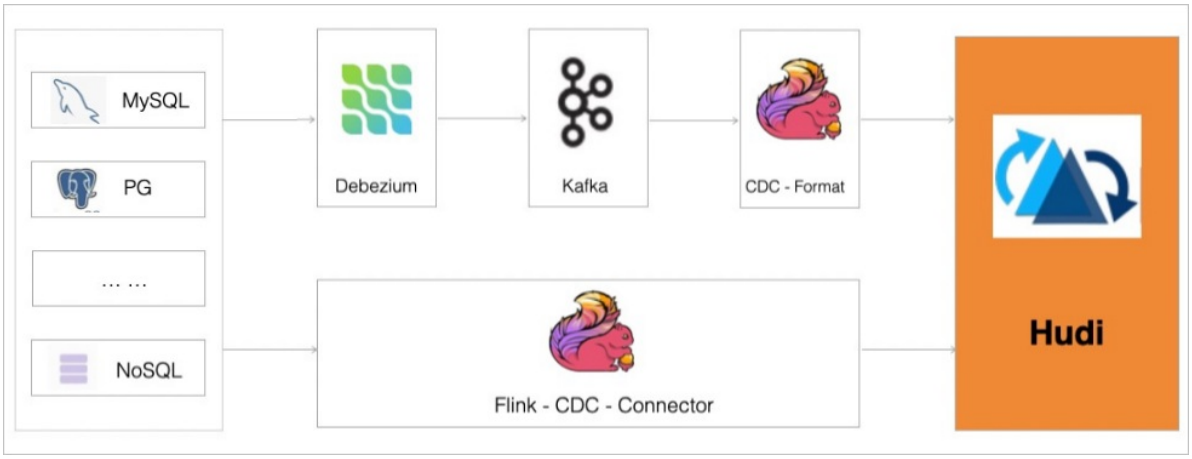
类别	详情
特性	○ 支持ACID：支持ACID语义，提供事务的线性隔离级别。 ○ 支持UPSERT语义：UPSERT语义即就是INSERT和UPDATE两种语义的合并。在UPSERT语义时，如果记录不存在则插入；如果记录存在则更新。通过INSERT INTO语法可以大幅简化开发代码的复杂度，提升效率。 ○ 支持Data Version：通过时间旅行（Time Travel）特性，提供任意时间点的数据版本历史，便于数据运维，提升数据质量。 ○ 支持Schema Evolution：支持动态增加列，类型变更等Schema操作。
典型场景	○ DB入湖加速 相比昂贵且低效的传统批量加载和Merge，Hudi提供超大数据集的实时流式更新写入。通过实时的ETL，您可以直接将CDC（change data capture）数据写入数据湖，供下游业务使用。典型案例为采用Flink MySQL CDC Connector将RDBMS（MySQL）的Binlog写入Hudi表。 ○ 增量ETL 通过增量拉取的方式获取Hudi中的变更数据流，相对离线ETL调度，实时性更好且更轻量。典型场景是增量拉取在线服务数据到离线存储中，通过Flink引擎写入Hudi表，借助Presto或Spark引擎实现高效的OLAP分析。 ○ 消息队列 在小体量的数据场景下，Hudi也可以作为消息队列替代Kafka，简化应用开发架构。 ○ 数仓回填（backfill） 针对历史全量数据进行部分行、列的更新场景，通过数据湖极大减少计算资源消耗，提升了端到端的性能。典型案例是Hive场景下全量和增量的打宽。

- 全托管Flink集成Hudi功能优势

  相比开源社区Hudi，全托管Flink平台集成Hudi具有的功能优势详情如下表所示。

功能优势	详情
平台侧与Flink全托管集成，免运维	Flink全托管内置Hudi Connector，降低运维复杂度，提供SLA保障。
完善的数据连通性	对接多个阿里云大数据计算分析引擎，数据与计算引擎解耦，可以在Flink、Spark、Presto或Hive间无缝流转。
深度打磨DB入湖场景	与Flink CDC Connector联动，降低开发门槛。
提供企业级特性	包括集成DLF统一元数据视图、自动且轻量化的表结构变更。
内置阿里云OSS存储，低成本存储，弹性扩展	数据以开放的Parquet、Avro格式存储在阿里云OSS，存储计算分离，资源灵活弹性扩展。

- CDC数据同步



CDC数据保存了完整的数据库变更，您可以通过以下任意一种方式将数据导入Hudi：

- 对接CDC Format，消费Kafka数据的同时导入Hudi。

  支持debezium-json、canal-json和maxwell-json三种格式，该方式优点是可扩展性强，缺点是需要依赖Kafka和Debezium数据同步工具。

- 通过Flink-CDC-Connector直接对接DB的Binlog，将数据导入Hudi。

  该方式优点是轻量化组件依赖少。

> **说明**
> - 如果无法保证上游数据顺序，则需要指定write.precombine.field字段。
> - 在CDC场景下，需要开启changelog模式，即changelog.enabled设为true。

使用限制

- 仅Flink计算引擎vvr-4.0.11-flink-1.13及以上版本支持Hudi Connector。
- 文件系统仅支持HDFS或阿里云OSS服务。
- 不支持以Session模式提交作业。

DDL定义

```
CREATE TEMPORARY TABLE hudi_sink (
  uuid BIGINT,
  data STRING,
  ts   TIMESTAMP(3)
) WITH (
  'connector' = 'hudi',
  'table.type' = 'COPY_ON_WRITE',
  'path' = 'oss://<yourOSSBucket>/<自定义存储位置>',
  'oss.endpoint' = '<yourOSSEndpoint>',
  'accessKeyId' = '<yourAccessKeyId>',
  'accessKeySecret' = '<yourAccessKeySecret>' ,
  'hive_sync.enable'='true',
  'hive_sync.db'='<db name>',
  'hive_sync.table' = '<table name>',
  'hive_sync.mode' = 'hms',
  'dlf.catalog.region' = 'cn-hangzhou',
  'dlf.catalog.endpoint' = 'dlf-vpc.cn-hangzhou.aliyuncs.com'
);
```

WITH参数

- 基础参数

参数	说明	是否必选	备注
connector	结果表类型。	是	固定值为 `hudi` 。
table.type	表类型。	是	参数取值如下： ○ COPY_ON_WRITE：使用Parquet列式存储，每次更新数据，创建一个新的base文件。 ○ MERGE_ON_READ：使用Parquet列式和Avro行式存储，更新操作将会被写入delta日志文件，异步合并Parquet列式文件生成新版本文件。
path	表存储路径。	是	支持阿里云OSS和HDFS两种路径。例如*oss://<bucket name>/table*或*hdfs://<ip>:<port>/table*。
oss.endpoint	阿里云对象存储服务OSS的Endpoint。	否	如果使用OSS作为存储，则必需填写。参数取值请详情参见访问域名和数据中心。
accessKeyId	阿里云账号的AccessKey ID。	否	如果使用OSS作为存储，则必需填写。获取方法请参见获取AccessKey。
accessKeySecret	阿里云账号的AccessKey Secret。	否	如果使用OSS作为存储，则必需填写。获取方法请参见获取AccessKey。
hive_sync.enable	是否开启同步元数据到Hive功能。	否	参数取值如下： ○ true：开启。 ○ false：不开启。

参数	说明	是否必选	备注
hive_sync.mode	Hive数据同步模式。	否	参数取值如下： ○ hms（推荐值）：采用Hive Metastore或者DLF Catalog时，需要设置为hms。 ○ jdbc（默认值）：采用jdbc Catalog时，需要设置为jdbc。
hive_sync.db	同步到Hive的数据库名称。	否	无。
hive_sync.table	同步到Hive的表名称。	否	无。
dlf.catalog.region	DLF服务的地域名。	否	详情请参见已开通的地域和访问域名。 说明 ○ 仅当hive_sync.mode设置为hms时，dlf.catalog.region参数设置才生效。 ○ 请和dlf.catalog.endpoint选择的地域保持一致。
dlf.catalog.endpoint	DLF服务的Endpoint。	否	详情请参见已开通的地域和访问域名。 说明 ○ 仅当hive_sync.mode设置为hms时，dlf.catalog.endpoint参数设置才生效。 ○ 推荐您为dlf.catalog.endpoint参数配置DLF的VPC Endpoint。例如，如果您选择的地域为cn-hangzhou地域，则dlf.catalog.endpoint参数需要配置为dlf-vpc.cn-hangzhou.aliyuncs.com。 ○ 如果您需要跨VPC访问DLF，则请参见如何访问跨VPC里的存储资源？。
write.operation	写入操作模式。	否	参数取值如下： ○ insert模式：数据追加写。 ○ upsert模式（默认值）：数据更新。 ○ bulk_insert模式：批量写入。

参数	说明	是否必选	备注
write.precombine.field	版本字段，基于此字段的大小来判断消息是否进行更新。	否	默认为ts字段，如果没有ts字段则为处理顺序。

- 高阶参数
  - 内存参数

> **说明**
> - 所有的内存参数单位都是MB。
> - 影响内存的三个因素包括TaskManager的数量和内存配置、write task的并发和每个write task能够分配到的内存。因此建议您确认每个write task能够分配到的内存，再考虑相关的内存参数设置。

参数	说明	默认值	备注
write.task.max.size	一个write task的最大可用内存。	1024	预留给每个write task的内存buffer大小为write.task.max.size参数值减去compaction.max_memory参数值的差值。 当write task的内存buffer达到阈值后，会将内存的数据落盘。 您需要关注TM分配给每个write task的内存，保证每个write task能够分配到 write.task.max.size所配置的大小。例如TM的内存是4 GB，运行了2个StreamWriteFunction，那每个write function能分配到2 GB。此时你需要预留一些buffer，因为网络buffer，TM上其他类型task，例如BucketAssignFunctio也会消耗一些内存。

参数	说明	默认值	备注
compaction.max_memory	合并文件时的最大可用内存。	100	如果是在线合并，资源充足时可以调大该参数，例如调为1 GB。 您需要关注compaction内存的变化。因为compaction.max_memory控制了每个compaction task读log时可以利用的内存大小。在内存资源充足时，有以下建议： ■ 如果是MOR表，可以将compaction.max_memory参数值调大些。 ■ 如果是COW表，可以将write.task.max.size和write.merge.max_memory参数值同时调大。
write.merge.max_memory	COW写操作，会有增量文件和全量文件数据合并的过程，增量的数据会缓存在内存中，该参数控制使用的堆内存大小。	100	通常您不需要设置该参数，保持默认值即可。

- 并发参数

参数	说明	默认值	备注
write.tasks	写任务（write）的并发读。每个write顺序写1~N个buckets。	4	增加并发不影响小文件个数。
write.bucket_assign.tasks	bucket assigner的并发。	1	增加该参数值会导致bucket数量增加，即增加小文件数。
write.index_bootstrap.tasks	Index bootstrap算子的并发。增加并发可以加快bootstrap阶段的效率，bootstrap阶段会阻塞checkpoint，因此需要设置多一些的checkpoint失败容忍次数。	不显式指定该参数时，默认使用Flink算子的并行度。	仅在index.bootstrap.enabled为true时生效。

- 在线合并参数

参数	说明	默认值	备注
compaction.tasks	在线合并算子的并发。	4	在线合并会消耗计算资源。
compaction.trigger.strategy	合并策略。	num_commits	支持以下四种策略： ■ num_commits：根据commits数量合并。 ■ time_elapsed：根据时间合并。 ■ num_and_time：根据数据量和时间合并。 ■ num_or_time：根据数量或时间合并。
compaction.delta_commits	delta commits文件的个数。	5	取值范围是整数，建议不超过20个。默认值的含义为5个delta文件触发一次合并。
compaction.delta_seconds	在线合并的时间间隔。	3600	单位为秒。
compaction.target_io	每个压缩任务的IO上限。	500（GB）	无。

- Changelog模式参数

  Hudi支持保留消息的所有变更，对接Flink引擎的后，实现全链路近实时数仓生产。Hudi的MOR表以行存格式保留消息的所有变更，通过流读MOR表可以消费到所有的变更记录。此时，您需要开启Changelog模式，changelog.enabled设置为true。

  开启changelog.enabled参数后，支持消费所有变更。异步的合并任务会将中间变更合并成1条。所以如果流读消费不够及时，被压缩后只能读到最后一条记录。当然，通过调整压缩的buffer时间可以预留一定的时间buffer给reader。例如调整合并的compaction.delta_commits和compaction.delta_seconds参数。

参数	说明	是否必填	备注

参数	说明	是否必填	备注
changelog.enabled	是否开启Changelog模式。	否	参数取值如下： ○ true：开启Changelog模式。 ○ false（默认值）：关闭Changelog模式。关闭Changelog模式时，即支持UPSERT语义，所有的消息仅保证最后一条合并消息，中间的变更可能会被合并。 **说明** 流读取会展示每次变更，批读只会展示合并后的变更结构。

- 批量导入参数

  如果存量数据来源于其他数据源，则可以使用批量导入功能，快速将存量数据转换为Hudi表格式。需要注意以下几点：

  - 批量导入省去avro的序列化以及数据的合并过程，后续不会再有去重操作，数据的唯一性需要您自己来保证。
  - write.operation参数在Batch Execuiton Mode下执行更高效，Batch模式默认会按照分区排序输入消息再写入Hudi，避免不同文件频繁切换处理导致性能下降。
  - 通过write.tasks参数指定bulk_insert write task的并发，并发的数量会影响到小文件的数量。

  其中涉及的WITH参数如下表所示。

参数	说明	是否必填	备注
write.tasks	写任务（write）的并发读。每个write顺序写1~N个buckets。	否	默认值为4。
write.bulk_insert.shuffle_by_partition	是否将数据按照partition字段打散，再通过write task写入。	否	默认值为true。 **说明** 开启该参数将减少小文件的数量，但可能存在数据倾斜的风险。

参数	说明	是否必填	备注
write.bulk_insert.sort_by_partition	是否将数据按照partition字段排序再写入。	否	默认值为true。 说明 当一个write task写多个 partition，开启该参数可以减少小文件数量。
write.sort.memory	sort算子的可用管理内存。	否	默认值为128，单位为MB。

- 全量接增量参数

  如果您已经有全量的离线Hudi表，需要导入离线数据后，再把新增数据写入表中，并且保证数据不重复，则可以把index.bootstrap.enabled参数设置为true，开启全量接增量功能。

  说明 如果发现耗时太长，则可以在写入全量数据时调大资源。全量数据写完后再写新数据时，可以将资源调小或者开启写入限流参数。

参数	说明	是否必填	备注
index.bootstrap.enabled	是否开启全量接增量功能。开启后存量表的索引数据一次性被加载到state中。	是	参数取值如下： ○ true：开启。 ○ false（默认值）：关闭。
index.partition.regex	设置正则表达式进行分区筛选。	否	默认加载全部分区。

- 写入限流参数

  写入量吞吐高，Partition随机引发的严重乱序，容易导致写入性能退化，出现吞吐毛刺的问题。此时您可以开启限流功能，保证写入流量平稳。例如，您可以先同步百亿数量级全量和增量数据到Kafka，再通过Flink流式消费的方式将库表数据直接导成Hudi格式的库表。

参数	说明	是否必填	备注
write.rate.limit	是否开启限流。	否	参数取值如下： ○ true：开启。 ○ false（默认值）：关闭。

- Append模式参数

参数	说明	是否必填	备注

参数	说明	是否必填	备注
write.insert.cluster	是否在写入时合并小文件。	否	默认值为false。 说明 ○ COW表的INSERT模式默认为Append，在这种模式下，文件不会有合并操作，但您可以启动小文件合并功能后进行合并。 ○ 开启该参数后，每次写入会优先合并之前的小文件，不会去重，吞吐会受影响。

示例

- 示例一：写Hudi结果表

  本示例为您介绍如何通过MySQL CDC流式读取数据然后写入Hudi表。

  i. 创建OSS Bucket。

  详情请参见创建存储空间。

  ii. 在**作业开发**页面，在目标作业文本编辑区域，编写SQL设计流作业。

```
CREATE TEMPORARY TABLE datagen(
  uuid    BIGINT,
  data  STRING,
  ts TIMESTAMP(3)
) WITH (
  'connector' = 'datagen'
);
CREATE TEMPORARY TABLE hudi_sink (
  uuid BIGINT,
  data STRING,
  ts TIMESTAMP(3)
) WITH (
  'connector' = 'hudi',
  'oss.endpoint' = '<yourOSSEndpoint>',
  'accessKeyId' = '<yourAccessKeyId>',
  'accessKeySecret' = '<yourAccessKeySecret>',
  'path' = 'oss://<yourOSSBucket>/<自定义存储位置>',
  'table.type' = 'COPY_ON_WRITE'
);
INSERT INTO hudi_sink SELECT * from datagen;
```

  iii. 在作业开发页面右侧**高级配置**面版中，引擎版本配置为vvr-4.0.11-flink-1.13。

iv. 单击**验证**。

v. 单击**上线**。

vi. 在**作业运维**页面，单击目标作业名称**操作**列的**启动**。

vii. 在OSS控制台查看写入的测试数据。

等第一次Checkpoint完成之后，您将能看到写入的测试数据了。

- 示例二：MySQL CDC数据入湖

本示例为您介绍如何通过MySQL的CDC源表读取数据然后写入Hudi表。

i. 创建OSS Bucket。

详情请参见创建存储空间。

ii. 在**作业开发**页面，在目标作业文本编辑区域，编写SQL设计流作业。

```
CREATE TEMPORARY TABLE mysql_src (
  id BIGINT,
  name STRING,
  PRIMARY KEY(id) NOT ENFORCED
) WITH (
  'connector' = 'mysql-cdc',
  'hostname' = '<yourRDSHostName>',
  'port' = '3306',
  'username' = '<yourRDSUserName>',
  'password' = '<yourRDSPassword>',
  'database-name' = 'user_db.*', -- 正则匹配多个分库。
  'table-name' = 'user.*'   -- 正则匹配多张分表。
);
CREATE TEMPORARY TABLE hudi_sink (
  id BIGINT PRIMARY KEY NOT ENFORCED,
  name STRING
) WITH (
  'connector' = 'hudi',
  'oss.endpoint' = '<yourOSSEndpoint>',
  'accessKeyId' = '<yourAccessKeyId>',
  'accessKeySecret' = '<yourAccessKeySecret>',
  'path' = 'oss://<yourOSSBucekt>/<Path to Table>/',
  'table.type' = 'MERGE_ON_READ'
);
INSERT INTO hudi_sink SELECT * FROM mysql_src;
```

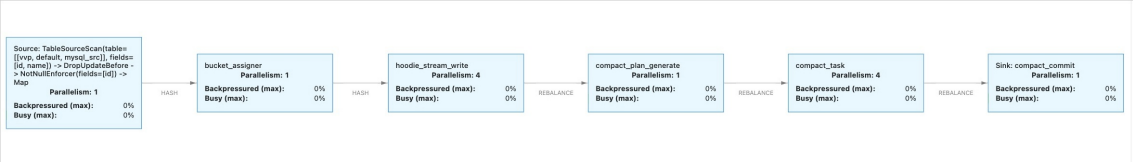
iii. 在作业开发页面右侧**高级配置**面版中，引擎版本配置为vvr-4.0.12-flink-1.13。

iv. 单击**验证**。

v. 单击**上线**。

vi. 在**作业运维**页面，单击目标作业名称**操作**列的**启动**。

作业上线后，您可以在**作业总览**页面，查看作业的Vertex图，了解作业运行过程。



vii. 在OSS控制台查看写入的测试数据。

等第一次Checkpoint完成之后，您将能看到写入的测试数据了。

9.2.4.2.19. Blackhole结果表

本文为您介绍Blackhole结果表的DDL定义、WITH参数和代码示例。

什么是Blackhole结果表

Blackhole结果表是系统内置的Connector。如果您在注册其他类型的Connector结果表时报错，但您不确定是系统问题还是结果表WITH参数错误，您可以将WITH参数修改为'connector' = 'blackhole'后，单击**运行**。如果不再报错，则证明系统没有问题，您需要确认修改WITH参数。

使用限制

仅Flink计算引擎VVR 2.0.0及以上版本支持Blackhole Connector。

DDL定义

```
create table blackhole_sink(
  name VARCHAR,
  score BIGINT
) with (
  'connector' = 'blackhole'
);
```

WITH参数

参数	说明	是否必填	备注
connector	结果表类型。	是	固定值为blackhole。

代码示例

```
CREATE TEMPORARY table datahub_source(
  name VARCHAR
) with (
  'connector'='datahub',
  'endpoint'='<yourEndpoint>',
  'project'='<yourProject>',
  'topic'='<yourTopic>',
  'subId'='<yourSubId>',
  'accessId'='<yourAccessId>',
  'accessKey'='<yourAccessKey>',
  'startTime'='2018-06-01 00:00:00'
);
CREATE TEMPORARY table blackhole_sink(
  name  VARCHAR
) with (
  'connector' = 'blackhole'
);
INSERT INTO blackhole_sink
SELECT
  LOWER(name)
from datahub_source;
```

9.2.4.2.20. InfluxDB结果表

本文为您介绍InfluxDB结果表的DDL定义、WITH参数、类型映射和代码示例。

什么是时序数据库InfluxDB®版

时序数据库InfluxDB®版是一款专门处理高写入和查询负载的时序数据库，用于存储大规模的时序数据并进行实时分析，包括来自DevOps监控、应用指标和IoT传感器上的数据。时序数据库InfluxDB®版详情请参见InfluxDB®☺介绍。

前提条件

已创建InfluxDB的数据库，详情请参见管理用户账号和数据库。

使用限制

仅Flink计算引擎VVR 2.1.5及以上版本支持InfluxDB Connector。

DDL定义

Flink支持使用InfluxDB作为结果表输出，示例代码如下。

```
CREATE TABLE stream_test_influxdb(
 `metric` VARCHAR,
 `timestamp` BIGINT,
 `tag_value1` VARCHAR,
 `field_fieldValue1` DOUBLE
) WITH (
  'connector' = 'influxdb',
  'url' = 'http://service.cn.influxdb.aliyuncs.com:****',
  'database' = '<yourDatabaseName>',
  'username' = '<yourDatabaseUserName>',
  'password' = '<yourDatabasePassword>',
  'batchSize' ='300',
  'retentionPolicy' = 'autogen',
  'ignoreErrorData' = 'false'
);
```

建表默认格式：

- 第0列：metric（VARCHAR），必填。
- 第1列：timestamp（BIGINT），必填，单位为毫秒。
- 第2列：tag_value1（VARCHAR），必填，最少填写一个。
- 第3列：field_fieldValue1（DOUBLE），必填，最少填写一个。

  写入多个field_fieldValue时，您需要按照如下格式填写。

  ```
  field_fieldValue1 类型,
  field_fieldValue2 类型,
  ...
  field_fieldValueN 类型
  ```

  示例如下。

  ```
  field_fieldValue1 DOUBLE,
  field_fieldValue2 INTEGER,
  ...
  field_fieldValueN INTEGER
  ```

> **说明** 结果表中只支持metric、timestamp、tag_*和field_*，不能出现其他的字段。

WITH参数

参数	说明	是否必填	备注
connector	结果表类型。	是	固定值为influxdb。
url	InfluxDB的服务地址。	是	在InfluxDB中，url为VPC网络地址，例如：*https://localhost:8086*或*http://localhost:3242*。 url支持HTTP和HTTPS。
database	InfluxDB的数据库名称。	是	例如db-flink。
username	数据库的用户名。	是	需要对目标数据库有写权限。用户名详情请参见管理用户账号和数据库。
password	数据库的密码。	是	密码详情请参见管理用户账号和数据库。
batchSize	批量提交的记录条数。	否	默认每次批量提交300条记录。
retentionPolicy	保留策略。	否	如果您不配置该参数时，该参数会被默认填写为每个数据库的默认保留策略autogen，保留策略详情请参见管理用户账号和数据库。
ignoreErrorData	是否忽略异常数据。	否	取值如下： • true：忽略异常数据。 • false（默认值）：不忽略异常数据。

类型映射

InfluxDB字段类型	Flink版字段类型
BOOLEAN	BOOLEAN
INT	INT
BIGINT	BIGINT
FLOAT	FLOAT
DECIMAL	DECIMAL
DOUBLE	DOUBLE
DATE	DATE
TIME	TIME
TIMESTAMP	TIMESTAMP
VARCHAR	VARCHAR

代码示例

```
CREATE TEMPORARY TABLE datahub_source(
 `metric` VARCHAR,
 `timestamp` BIGINT,
 `filedvalue` DOUBLE,
 `tagvalue` VARCHAR
) WITH (
  'connector' = 'datagen',
  'fields.metric.length' = '3',
  'fields.tagvalue.length' = '3',
  'fields.timestamp.min' = '1587539547000',
  'fields.timestamp.max' = '1619075547000',
  'fields.filedvalue.min' = '1',
  'fields.filedvalue.max' = '100000',
  'rows-per-second' = '50'
);
CREATE TEMPORARY TABLE influxdb_sink(
  `metric` VARCHAR,
  `timestamp` BIGINT,
  `field_fieldValue1` DOUBLE,
  `tag_value1` VARCHAR
) WITH (
  'connector' = 'influxdb',
  'url' = 'https://***********.influxdata.tsdb.aliyuncs.com:****',
  'database' = '<yourDatabaseName>',
  'username' = '<yourDatabaseUserName>',
  'password' = '<yourDatabasePassword>',
  'batchSize' ='100',
  'retentionPolicy' = 'autogen',
  'ignoreErrorData' = 'false'
);
INSERT INTO influxdb_sink
SELECT
   `metric`,
   `timestamp`,
   `filedvalue`,
   `tagvalue`
FROM datahub_source;
```

9.2.4.2.21. 数据湖Iceberg结果表

您可以通过Flink将数据写入Hive Metastore或者DLF Catalog下的Iceberg表。本文为您介绍如何将数据写入到数据湖Iceberg结果，包括DDL定义、WITH参数、类型映射和代码示例等。

背景信息

Apache Iceberg是一种开放的数据湖表格格式。您可以借助Apache Iceberg快速地在HDFS或者云端OSS上构建自己的数据湖存储服务，并借助开源大数据生态的Flink、Spark、Hive、Presto等计算引擎来实现数据湖的分析。目前Apache Iceberg提供以下核心能力：

- 基于HDFS或者对象存储构建低成本的轻量级数据湖存储服务。
- 完善的ACID语义。
- 支持历史版本回溯。

- 支持高效的数据过滤。
- 支持Schema Evolution。
- 支持Partition Evolution。

您可以借助Flink高效的容错能力和流处理能力，把海量的日志行为数据实时导入到Apache Iceberg数据湖内，再借助Flink或者其他分析引擎来实现数据价值的提取。您可以通过以下任意一种方式来使用Iceberg Connector：

- 将数据写入至Hive Metastore下的Iceberg表

  Apache Hive Metastore通常被作为开源大数据的统一元数据中心。如果您的元数据被维护在Apache Hive Metastore，希望通过Flink将数据写入Iceberg表内，然后将数据存储在阿里云OSS上，可以采用该方式来使用Iceberg Connector。

- 将数据写入至DLF Catalog下的Iceberg表

  阿里云数据湖构建（Data Lake Formation，DLF）是基于阿里云大数据生态产品设计和研发的统一元数据中心。借助该元数据中心您可以方便地采用开源大数据计算引擎（Spark、Hive、Presto或Flink）访问同一份数据湖内的数据。如果您希望元数据被存储在阿里云DLF上，可以采用该方式来使用Iceberg Connector。

> **说明** Iceberg Connector既可以作为Flink Stream作业的结果表，也可以作为Flink批作业的源表和结果表。

前提条件

将数据写入至DLF Catalog下的Iceberg表前，需要已开通阿里云DLF。如果您未开通阿里云DLF，请在数据湖构建控制台页面，单击**免费开通数据湖构建**。

使用限制

- 仅Flink计算引擎vvr-4.0.8-flink-1.13及以上版本支持Iceberg Connector。
- Iceberg Connector仅支持日志数据入湖，暂不支持CDC或者Binlog数据入湖。
- Iceberg Connector仅支持采用Hive Metastore和阿里云DLF作为Catalog，文件系统采用阿里云OSS服务。暂不支持采用开源HDFS作为Iceberg表的文件系统。

类型映射

Iceberg字段类型	Flink字段类型
BOOLEAN	BOOLEAN
INT	INT
LONG	BIGINT
FLOAT	FLOAT
DOUBLE	DOUBLE
DECIMAL(P,S)	DECIMAL(P,S)
DATE	DATE

Iceberg字段类型	Flink字段类型
TIME	TIME 说明 Iceberg时间戳精度为微秒，Flink时间戳精度为毫秒。在使用Flink读取Iceberg数据时，时间精度会对齐到毫秒。
TIMESTAMP	TIMESTAMP
TIMESTAMPTZ	TIMESTAMP_LTZ
STRING	STRING
FIXED(L)	BYTES
BINARY	VARBINARY
STRUCT<...>	ROW
LIST<E>	LIST
MAP<K, V>	MAP

将数据写入至Hive Metastore下的Iceberg表

下面为您介绍如何通过Flink将数据写入到Apache Hive Metastore上的Iceberg表。

1. 验证Hive Metastore依赖的Hadoop能否正确访问阿里云OSS数据。

   无论您使用的是阿里云EMR上的Hive Metastore，还是自建的Hive Metastore，都需要确保Hive Metastore所依赖的Hadoop能正确访问阿里云OSS上的数据。具体的确认方法如下：

   - 阿里云EMR上的Hive Metastore。

     通常，阿里云EMR上的Hive Metastore是可以直接访问该账号下的OSS数据的。如果您需要确认，则可以登录到Hive Metastore所在的主机，详情请参见登录集群。然后通过以下命令来验证Hadoop能否正确访问OSS数据。假如您需要访问*oss://table-format/*路径，则验证命令如下。

     ```
     hdfs dfs  -ls oss://table-format/
     ```

     说明 *oss://table-format/*为示例值。在实际验证时，需要替换为您账号下的Bucket名称和OSS路径。

   - 自建的Hive Metastore。

     如果您使用的是自建Hive Metastore，则需要手动配置Hadoop以便能正确地访问OSS数据。操作步骤详情如下。

a. 打开Hadoop的阿里云OSS插件开关，让Hadoop classpath可以自动加载阿里云OSS的相关依赖。阿里云OSS插件开关详情请参见Hadoop阿里云模块。不同Hadoop版本的操作方式如下表所示。

Hadoop版本	操作方式
hadoop 3.x.x	在*$HADOOP_HOME/etc/hadoop/hadoop-env.sh*中新增如下内容。 `export HADOOP_OPTIONAL_TOOLS="hadoop-aliyun"`
hadoop 2.9.x	在*$HADOOP_HOME/libexec/hadoop-config.sh*中新增如下内容。 `CLASSPATH=${CLASSPATH}:${TOOL_PATH}`
hadoop 2.8.x及以下版本	不支持访问阿里云OSS服务。

b. 在*$HADOOP_HOME/etc/hadoop/core-site.xml*中新增如下内容，并将fs.oss.endpoint、fs.oss.accessKeyId和fs.oss.accessKeySecret参数值替换为您的实际值。

```
<property>
    <name>fs.AbstractFileSystem.oss.impl</name>
    <value>org.apache.hadoop.fs.aliyun.oss.OSS</value>
</property>
<property>
    <name>fs.oss.impl</name>
    <value>org.apache.hadoop.fs.aliyun.oss.AliyunOSSFileSystem</value>
</property>
<property>
    <name>fs.oss.endpoint</name>
    <value>${YOUR_OSS_ENDPOINT}</value>
</property>
<property>
    <name>fs.oss.accessKeyId</name>
    <value>${YOUR_ACCESS_KEY_ID}</value>
</property>
<property>
    <name>fs.oss.accessKeySecret</name>
    <value>${YOUR_ACCESS_KEY_SECRET}</value>
</property>
```

其中涉及的参数解释如下表所示。

参数	说明
fs.oss.endpoint	阿里云对象存储OSS的域名，详情请参见访问域名和数据中心。
fs.oss.accessKeyId	阿里云账号的Access Key。获取方法请参见获取AccessKey。
fs.oss.accessKeySecret	阿里云账号的AccessKey Secret。获取方法请参见获取AccessKey。

c. 通过hdfs命令行来验证是否可以正确读取OSS数据。

```
hdfs dfs  -ls oss://table-format/
```

> **说明** *oss://table-format/*为示例值。在实际验证时，需要替换为您账号下的Bucket名称和OSS路径。

2. 配置网络以便Flink可以正确地访问Hive Metastore端口。

   配置和验证的方式，请参见管理Hive Metastore。

3. 编写SQL并执行入湖作业。

```
CREATE TEMPORARY TABLE flink_source(
  id BIGINT,
  data STRING
) WITH (
  'connector' = 'datagen'
);
CREATE TEMPORARY TABLE iceberg_sink (
  id BIGINT,
  data STRING
) WITH (
  'connector' = 'iceberg',
  'catalog-name' = 'hive_prod',
  'catalog-type' = 'hive',
  'engine.hive.enabled' = 'true',
  'uri' = 'thrift://<host>:<port>',
  'location' = 'oss://<oss-bucket>/<oss-object>',
  'io-impl' = 'org.apache.iceberg.aliyun.oss.OSSFileIO',
  'oss.endpoint' = '<aliyun-oss-endpoint>',
  'access.key.id' = '<aliyun-oss-access-key>',
  'access.key.secret' = '<aliyun-oss-access-key-secret>'
);
INSERT INTO iceberg_sink SELECT * FROM flink_source;
```

WITH参数解释如下表所示。

参数	说明	是否必选	备注
connector	结果表类型。	是	固定值为 `iceberg` 。
catalog-name	Catalog名称。	是	请填写为自定义的英文名。
catalog-type	Catalog类型。	是	固定值为hive。
catalog-database	数据库名称。	是	对应用户在Hive Metastore上创建的数据库名称，例如iceberg_db。

参数	说明	是否必选	备注
engine.hive.enabled	存放在Hive Metastore中的Iceberg表的元数据是否能被Hive计算引擎读取。	否	参数取值如下： ○ true（推荐值）：可以被Hive计算引擎直接读取。 ○ false：不可以被Hive计算引擎直接读取。
uri	Hive Metastore的thrift server的地址。	是	格式为*thrift://<host>:<port>*。其中： ○ host表示Hive Metastore的IP地址。 ○ port表示Hive Metstore的端口，默认值为9083。
location	结果表存放的OSS路径。	是	格式为*oss://<bucket>/<object>*。其中： ○ bucket表示您创建的OSS Bucket名称。 ○ object表示您存放数据的路径。
io-impl	分布式文件系统的实现类名称。	是	固定值为org.apache.iceberg.aliyun.oss.OSSFileIO。
oss.endpoint	阿里云对象存储服务OSS的Endpoint。	是	请详情参见访问域名和数据中心。 说明 ○ 推荐您为oss.endpoint参数配置OSS的VPC Endpoint。例如，如果您选择的地域为cn-hangzhou地域，则oss.endpoint需要配置为oss-cn-hangzhou-internal.aliyuncs.com。 ○ 如果您需要跨VPC访问OSS，则请参见如何访问跨VPC里的存储资源？
access.key.id	阿里云账号的AccessKey ID。	是	获取方法请参见获取AccessKey。
access.key.secret	阿里云账号的AccessKey Secret。	是	获取方法请参见获取AccessKey。

将数据写入至DLF Catalog下的Iceberg表

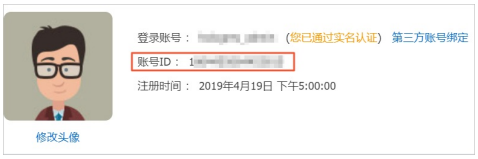
下面为您介绍如何通过Flink将数据写入到阿里云DLF上的Iceberg表。

- DDL定义

```
CREATE TABLE dlf_iceberg (
  id   BIGINT,
  data STRING
) WITH (
  'connector' = 'iceberg',
  'catalog-name' = '<yourCatalogName>',
  'catalog-type' = 'custom',
  'catalog-database' = '<yourDatabaseName>',
  'io-impl' = 'org.apache.iceberg.aliyun.oss.OSSFileIO',
  'oss.endpoint' = '<yourOSSEndpoint>',
  'access.key.id' = '<yourAccessKeyId>',
  'access.key.secret' = '<yourAccessKeySecret>',
  'catalog-impl' = 'org.apache.iceberg.aliyun.dlf.DlfCatalog',
  'warehouse' = '<yourOSSWarehousePath>',
  'dlf.catalog-id' = '<yourCatalogId>',
  'dlf.endpoint' = '<yourDLFEndpoint>',
  'dlf.region-id' = '<yourDLFRegionId>'
);
```

- WITH参数说明

参数	说明	是否必选	备注
connector	结果表类型。	是	固定值为 `iceberg` 。
catalog-name	Catalog名称。	是	请填写为自定义的英文名。
catalog-type	Catalog类型。	是	固定值为custom。
catalog-database	数据库名称。	是	对应用户在DLF上创建的数据库名称，例如dlf_db。
io-impl	分布式文件系统的实现类名。	是	固定值为 `org.apache.iceberg.aliyun.oss.OSSFileIO` 。
oss.endpoint	阿里云对象存储服务OSS的Endpoint。	是	请详情参见访问域名和数据中心。 说明 ○ 推荐您为oss.endpoint参数配置OSS的VPC Endpoint。例如，如果您选择的地域为cn-hangzhou地域，则oss.endpoint需要配置为oss-cn-hangzhou-internal.aliyuncs.com。 ○ 如果您需要跨VPC访问OSS，则请参见如何访问跨VPC里的存储资源？
access.key.id	阿里云账号的AccessKey ID。	是	获取方法请参见获取AccessKey。

参数	说明	是否必选	备注
access.key.secret	阿里云账号的Access Secret。	是	获取方法请参见获取AccessKey。
catalog-impl	Catalog的Class类名。	是	固定值为 `org.apache.iceberg.aliyun.dlf.DlfCatalog`
warehouse	表数据存放在OSS的路径。	是	无。
dlf.catalog-id	阿里云账号的账号ID。	是	登录账号信息，请通过用户信息页面获取。 
dlf.endpoint	DLF服务的Endpoint。	是	详情请参见已开通的地域和访问域名。 **说明** ○ 推荐您为dlf.endpoint参数配置DLF的VPC Endpoint。例如，如果您选择的地域为cn-hangzhou地域，则dlf.endpoint参数需要配置为dlf-vpc.cn-hangzhou.aliyuncs.com。 ○ 如果您需要跨VPC访问DLF，则请参见如何访问跨VPC里的存储资源？。
dlf.region-id	DLF服务的地域名。	是	详情请参见已开通的地域和访问域名。 **说明** 请和dlf.endpoint选择的地域保持一致。

- 代码示例
  i. 创建DLF数据库。

     操作步骤详情请参见创建元数据库。

     > **说明** 创建数据库选择OSS路径时，请将DLF数据库OSS路径选择为*${warehouse}/${database_name}.db*。例如，如果您将warehouse地址设置为*oss://iceberg-test/warehouse*，数据库的名称设置为dlf_db，则dlf_db的OSS路径需要选择为*oss://iceberg-test/warehouse/dlf_db.db*。

  ii. 在**作业开发**页面，在文本编辑区域，编写SQL设计流作业。

```
CREATE TEMPORARY TABLE datagen(
  id    BIGINT,
  data  STRING
) WITH (
  'connector' = 'datagen'
);
CREATE TEMPORARY TABLE dlf_iceberg (
  id    BIGINT,
  data  STRING
) WITH (
  'connector' = 'iceberg',
  'catalog-name' = '<yourCatalogName>',
  'catalog-type' = 'custom',
  'catalog-database' = '<yourDatabaseName>',
  'io-impl' = 'org.apache.iceberg.aliyun.oss.OSSFileIO',
  'oss.endpoint' = '<yourOSSEndpoint>',
  'access.key.id' = '<yourAccessKeyId>',
  'access.key.secret' = '<yourAccessKeySecret>',
  'catalog-impl' = 'org.apache.iceberg.aliyun.dlf.DlfCatalog',
  'warehouse' = '<yourOSSWarehousePath>',
  'dlf.catalog-id' = '<yourCatalogId>',
  'dlf.endpoint' = '<yourDLFEndpoint>',
  'dlf.region-id' = '<yourDLFRegionId>'
);
INSERT INTO dlf_iceberg SELECT * FROM datagen;
```

iii. 在作业开发页面右侧**高级配置**面版中，引擎版本配置为**vvr-4.0.8-flink-1.13**。

iv. 单击**验证**。

v. 单击**上线**。

vi. 在OSS控制台查看写入的测试数据。

等第一次Checkpoint完成之后，您将能看到写入的测试数据了。

9.2.4.2.22. MySQL结果表

本文为您介绍MySQL结果表的DDL定义、WITH参数和类型映射。

什么是MySQL结果表

MySQL结果表支持所有兼容MySQL协议的数据库，包括RDS MySQL、PolarDB for MySQL或者自建MySQL，而云数据库RDS MySQL结果表专为云数据库RDS MySQL定制，支持更丰富的配置参数。

前提条件

- 已创建MySQL数据库和表，详情请参见RDS MySQL创建数据库和账号、PolarDB MySQL创建数据库和账号或自建MySQL创建数据库和账号。
- 已设置IP白名单，详情请参见RDS MySQL白名单设置、PolarDB MySQL白名单设置或自建MySQL白名单设置。

使用限制

仅Flink计算引擎VVR 4.0.11及以上版本支持MySQL Connector。

DDL定义

```
CREATE TABLE mysql_sink (
  id INT,
  num BIGINT,
  PRIMARY KEY (id) NOT ENFORCED
) WITH (
  'connector' = 'mysql',
  'hostname' = '<yourHostname>',
  'port' = '3306',
  'username' = '<yourUsername>',
  'password' = '<yourPassword>',
  'database-name' = '<yourDatabaseName>',
  'table-name' = '<yourTableName>'
);
```

说明

- RDS MySQL数据库支持自增主键，因此在DDL中不声明该自增字段。例如ID是自增字段，Flink DDL不声明该自增字段，则数据库在一行数据写入过程中会自动填补相关自增字段。
- DDL声明的字段必须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	数据类型	是否必填	备注
connector	结果表类型。	String	是	固定值为 `mysql` 。

参数	说明	数据类型	是否必填	备注
hostname	MySQL数据库的IP地址或者hostname。	String	是	建议填写专有网络VPC地址。 说明 如果MySQL与Flink全托管不在同一VPC，则需填写公网地址。
username	MySQL数据库服务的用户名。	String	是	无。
password	MySQL数据库服务的密码。	String	是	无。
database-name	MySQL数据库名称。	String	是	无。
table-name	MySQL表名。	String	是	无。
port	MySQL数据库服务的端口号。	Integer	否	默认值为3306。
sink.max-retries	写入数据失败后，重试写入的最大次数。	Integer	否	默认值为3。
sink.buffer-flush.max-rows	内存中缓存的数据条数。	Integer	否	默认值为100。
sink.buffer-flush.interval	清空缓存的时间间隔。表示如果缓存中的数据在等待指定时间后，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。	Duration	否	默认值为1秒（1s）。

类型映射

MySQL字段类型	Flink字段类型
TINYINT	TINYINT
SMALLINT	

MySQL字段类型	Flink字段类型
TINYINT UNSIGNED	SMALLINT
INT	INT
MEDIUMINT	
SMALLINT UNSIGNED	
BIGINT	BIGINT
INT UNSIGNED	
BIGINT UNSIGNED	DECIMAL(20, 0)
BIGINT	BIGINT
FLOAT	FLOAT
DOUBLE	DOUBLE
DOUBLE PRECISION	
NUMERIC(p, s)	DECIMAL(p, s) 说明 其中p <= 38。
DECIMAL(p, s)	
BOOLEAN	BOOLEAN
TINYINT(1)	
DATE	DATE
TIME [(p)]	TIME [(p)] [WITHOUT TIMEZONE]
DATETIME [(p)]	TIMESTAMP [(p)] [WITHOUT TIMEZONE]
TIMESTAMP [(p)]	
CHAR(n)	CHAR(n)
VARCHAR(n)	VARCHAR(n)
BIT(n)	BINARY(⌈n/8⌉)
BINARY(n)	BINARY(n)
VARBINARY(N)	VARBINARY(N)
TINYTEXT	

MySQL字段类型	Flink字段类型
TEXT	STRING
MEDIUMTEXT	
LONGTEXT	
TINYBLOB	BYTES 说明 Flink仅支持小于等于2,147,483,647(2^31 - 1)的MySQL BLOB类型的记录。
BLOB	
MEDIUMBLOB	
LONGBLOB	

代码示例

```
CREATE TEMPORARY TABLE datagen_source (
  `name` VARCHAR,
  `age` INT
) WITH (
  'connector' = 'datagen'
);
CREATE TEMPORARY TABLE mysql_sink (
  `name` VARCHAR,
  `age` INT
) WITH (
  'connector' = 'mysql',
  'hostname' = '<yourHostname>',
  'port' = '3306',
  'username' = '<yourUsername>',
  'password' = '<yourPassword>',
  'database-name' = '<yourDatabaseName>',
  'table-name' = '<yourTableName>'
);
INSERT INTO mysql_sink
SELECT  * FROM datagen_source;
```

9.2.4.2.23. 云原生数据仓库AnalyticDB PostgreSQL版结果表

本文为您介绍云原生数据仓库AnalyticDB PostgreSQL版结果表的DDL定义、WITH参数和类型映射。

什么是云原生数据仓库AnalyticDB PostgreSQL版

云原生数据仓库AnalyticDB PostgreSQL版基于开源项目Greenplum构建，由阿里云深度扩展，兼容ANSI SQL 2003，兼容PostgreSQL和Oracle数据库生态，支持行存储和列存储模式。既提供高性能离线数据处理，也支持高并发在线分析查询，是各行业有竞争力的PB级实时数据仓库方案。它是一种大规模并行处理（MPP）数据仓库服务，可提供海量数据在线分析服务。

前提条件

- 已创建AnalyticDB PostgreSQL实例并创建表，详情请参见创建实例和数据表管理。
- 已设置白名单，详情请参见设置白名单。

使用限制

仅Flink计算引擎VVR 5.0.0及以上版本支持云原生数据仓库AnalyticDB PostgreSQL版Connector。

DDL定义

```
CREATE TABLE adbpg_sink (
  id INT,
  len INT,
  content VARCHAR,
  PRIMARY KEY(id)
) WITH (
  'connector'='adbpg',
  'url'='jdbc:postgresql://<Address>:<PortId>/<DatabaseName>',
  'tableName'='<yourDatabaseTableName>',
  'userName'='<yourDatabaseUserName>',
  'password'='<yourDatabasePassword>'
);
```

WITH参数

参数	说明	是否必选	备注
connector	结果表类型。	是	固定值为 `adbpg` 。
url	JDBC连接地址。	是	格式为： `jdbc:postgresql://<内网地址>:<端口>/<连接的数据库名称>` ，例如 `jdbc:postgresql://gp-xxxxxx.gpdb.cn-chengdu.rds.aliyuncs.com:5432/postgres`
tableName	表名。	是	无。
userName	用户名。	是	无。
password	密码。	是	无。
maxRetryTimes	写入数据失败后，重试写入的最大次数。	否	默认值为3。
retryWaitTime	重试的时间间隔。	否	默认值为100，单位为毫秒。
batchSize	一次批量写入的数据条数。	否	默认值为500。
flushIntervalMs	清空缓存的时间间隔。	否	如果缓存中的数据在等待指定时间后，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。 默认值为5000，单位为毫秒。

参数	说明	是否必选	备注
writeMode	第一次尝试写入时的写入方式。	否	参数取值如下： • insert（默认值）：直接插入，冲突时请参见conflictMode的冲突策略。 • upsert：冲突时自动更新，只能用于有主键的表。
conflictMode	当insert写入出现主键冲突或者唯一索引冲突时的处理策略。	否	参数取值如下： • strict（默认值）：冲突时报错。 • ignore：冲突时忽略。 • update：冲突时自动更新，可用于无主键表，执行效率较低。 • upsert：冲突时自动更新，只能用于有主键表。
targetSchema	Schema名称。	否	默认值为public。
caseSensitive	大小写是否敏感。	否	参数取值如下： • true：大小写敏感。 • false（默认值）：大小写不敏感。
connectionMaxActive	连接池的最大连接数。	否	默认值为5。 说明 • 系统会自动释放和数据库服务的空闲连接。 • 设置过大可能会导致服务端连接数出现异常。

类型映射

云原生数据仓库AnalyticDB PostgreSQL版字段类型	Flink字段类型
BOOLEAN	BOOLEAN
SMALLINT	INT
INT	INT
BIGINT	BIGINT
FLOAT	DOUBLE
VARCHAR	VARCHAR
TIMESTAMP	TIMESTAMP
DATE	DATE

代码示例

```
CREATE TEMPORARY TABLE datagen_source (
  `name` VARCHAR,
  `age` INT
)
COMMENT 'datagen source table'
WITH (
   'connector' = 'datagen'
);
CREATE TABLE adbpg_sink (
  name VARCHAR,
  age INT
) WITH (
  'connector'='adbpg',
  'url'='jdbc:postgresql://<Address>:<PortId>/<DatabaseName>',
  'tableName'='<yourDatabaseTableName>',
  'userName'='<yourDatabaseUserName>',
  'password'='<yourDatabasePassword>'
);
INSERT INTO adbpg_sink
SELECT * FROM datagen_source;
```

9.2.4.3. 创建数据维表

9.2.4.3.1. 云数据库HBase版维表

本文为您介绍云数据库HBase版维表的DDL定义、WITH参数、CACHE参数和转换关系。

什么是云数据库HBase

云数据库HBase是低成本、高扩展、云智能的大数据NoSQL，兼容标准HBase访问协议，提供低成本存储、高扩展吞吐、智能数据处理等核心优势，是为淘宝推荐、花呗风控、广告投放、监控大屏、菜鸟物流轨迹、支付宝账单、手淘消息等众多阿里巴巴核心服务提供支撑的数据库，具备PB规模、千万级并发、秒级伸缩、毫秒响应、跨机房高可用、全托管、全球分布等企业能力。

前提条件

- 已购买HBase集群并创建表，详情请参见购买集群。
- 已设置白名单，详情请参见设置白名单。

使用限制

仅Flink计算引擎VVR 2.0.0及以上版本支持云数据库HBase Connector。

DDL定义

```
CREATE TABLE hbase_dim(
  rowkey INT,
  family1 ROW<q1 INT>,
  family2 ROW<q2 STRING, q3 BIGINT>,
  family3 ROW<q4 DOUBLE, q5 BOOLEAN, q6 STRING>
) with (
  'connector'='cloudhbase',
  'table-name'='<yourTableName>',
  'zookeeper.quorum'='<yourZookeeperQuorum>'
);
```

- HBase的列族（Column Family）必须声明为ROW类型，列族名即该ROW的字段名。例如，DDL定义中声明了family1、family2和family3三个列族。
- HBase列族中的列（Cloumn）与对应ROW中嵌套的每个字段对应，列名即字段名。例如，DDL定义中列族family2声明了q2和q3两列。
- 除了类型为ROW的字段外，只能有一个原始类型（Atomic Type）的字段（例如STRING或BIGINT），该字段将被视作HBase的行键（Row Key），例如DDL定义中的rowkey。
- 必须将HBase的行键定义为结果表的主键（Primary Key），如果没有显示定义主键，默认使用行键作为主键。

WITH参数

参数	说明	是否必填	备注
connector	维表类型。	是	固定值为 `cloudhbase` 。
table-name	HBase表名。	是	无。
zookeeper.quorum	HBase的zookeeper地址。	是	无。
zookeeper.znode.parent	HBase在zookeeper中的根目录。	否	默认值为 `/hbase` 。 说明 仅在HBase标准版中生效。
userName	用户名。	否	仅在HBase增强版中生效。
password	密码。	否	仅在HBase增强版中生效。
haclient.cluster.id	HBase高可用实例ID。	否	只有访问同城主备实例时才需要配置。 说明 仅在HBase增强版中生效。
retries.number	HBase客户端的重试次数。	否	默认值为31。

参数	说明	是否必填	备注
null-string-literal	字段类型为字符串时，如果HBase中的数据为该值的字节数组，则该字段的值为null。	否	默认值为Null。

CACHE参数

参数	描述	是否必选	示例值
cache	缓存策略。	否	目前云数据库HBase版维表支持以下三种缓存策略： • *None*：无缓存。 • *LRU*：缓存维表里的部分数据。源表的每条数据都会触发系统先在Cache中查找数据，如果没有找到，则去物理维表中查找。 需要配置相关参数：缓存大小（cacheSize）和缓存更新时间间隔（cacheTTLMs）。 • *ALL（默认值）*：缓存维表里的所有数据。在Job运行前，系统会将维表中所有数据加载到Cache中，之后所有的维表查找数据都会通过Cache进行。如果在Cache中无法找到数据，则KEY不存在，并在Cache过期后重新加载一遍全量Cache。 适用于远程表数据量小且MISS KEY（源表数据和维表JOIN时，ON条件无法关联）特别多的场景。 需要配置相关参数：缓存更新时间间隔（cacheTTLMs），更新时间黑名单（cacheReloadTimeBlackList）。 **说明** 因为系统会异步加载维表数据，所以在使用CACHE ALL时，需要增加维表JOIN节点的内存，增加的内存大小为远程表数据量的两倍。
cacheSize	缓存大小。	否	当缓存策略选择*LRU*时，可以设置缓存大小，默认为*10000*行。
cacheTTLMs	缓存失效时间，单位为毫秒。	否	cacheTTLMs配置和cache有关： • 如果cache配置为*None*，则cacheTTLMs可以不配置，表示缓存不超时。 • 如果cache配置为*LRU*，则cacheTTLMs为缓存超时时间。默认不过期。 • 如果cache配置为*ALL*，则cacheTTLMs为缓存加载时间。默认不重新加载。
cacheEmpty	是否缓存空结果	否	默认值为true。

参数	描述	是否必选	示例值
cacheReloadTimeBlackList	更新时间黑名单。在缓存策略选择为ALL时，启用更新时间黑名单，防止在此时间内做Cache更新（例如双11场景）。	否	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` 。分隔符的使用情况如下所示： • 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。读取全量HBase数据，RPC（Remote Procedure Call Protocol）服务端一次返回给客户端的行数。	否	默认值为100条。

转换关系

HBase数据通过org.apache.hadoop.hbase.util.Bytes转换成Flink的数据类型。解码过程有以下两种情况：

- 对于Flink的非字符串类型，如果HBase中的值为空字节数组，则解码为null。
- 对于Flink的字符串类型，如果HBase中的值为null-string-literal字节数组，则解码为null。

Flink与HBase的数据转换关系如下。

Flink类型	HBase转换函数
CHAR	String toString(byte[] b)
VARCHAR	
STRING	
BOOLEAN	boolean toBoolean(byte[] b)
BINARY	byte[]
VARBINARY	
DECIMAL	BigDecimal toBigDecimal(byte[] b)
TINYINT	bytes[0]
SMALLINT	short toShort(byte[] bytes)
INT	int toInt(byte[] bytes)
BIGINT	long toLong(byte[] bytes)
FLOAT	float toFloat(byte[] bytes)
DOUBLE	double toDouble(byte[] bytes)

Flink类型	HBase转换函数
DATE	HBase字节数组通过int toInt(byte[] bytes) 转换成int，表示自1970.01.01以来的天数。
TIME	HBase字节数组通过int toInt(byte[] bytes) 转换成int，表示自00:00:00以来的毫秒数。
TIMESTAMP	HBase字节数组通过long toLong(byte[] bytes) 转换成long，表示自1970-01-01 00:00:00以来的毫秒数。
ARRAY	不支持
MAP / MULTISET	不支持
ROW	不支持

代码示例

包含HBase维表的实时计算作业代码示例如下。

```
CREATE TABLE datagen_source (
  a INT,
  b BIGINT,
  c STRING,
  `proc_time` AS PROCTIME()
) with (
  'connector'='datagen'
);
CREATE TABLE hbase_dim (
  rowkey INT,
  family1 ROW<col1 INT>,
  family2 ROW<col1 STRING, col2 BIGINT>,
  family3 ROW<col1 DOUBLE, col2 BOOLEAN, col3 STRING>
) WITH (
  'connector' = 'cloudhbase',
  'table-name' = '<yourTableName>',
  'zookeeper.quorum' = '<yourZookeeperQuorum>'
);
CREATE TABLE blackhole_sink(
  a INT,
  f1c1 INT,
  f3c3 STRING
) with (
  'connector' = 'blackhole'
);
INSERT INTO blackhole_sink
     SELECT a, family1.col1 as f1c1,  family3.col3 as f3c3 FROM datagen_source
JOIN hbase_dim FOR SYSTEM_TIME AS OF src.`proc_time` as h ON src.a = h.rowkey;
```

9.2.4.3.2. 表格存储Tablestore维表

本文为您介绍表格存储Tablestore维表DDL定义、WITH参数、CACHE参数和常见问题。

什么是表格存储Tablestore

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的无缝扩展，提供海量结构化数据的存储和实时访问服务。

前提条件

已创建Tablestore数据表，详情请参见步骤三：创建数据表。

使用限制

仅Flink计算引擎VVR 2.0.0及以上版本支持表格存储Tablestore Connector。

DDL定义

```
CREATE TABLE ots_dim (
  id int,
  len int,
  content STRING
) WITH (
  'connector'='ots',
  'endPoint'='<yourEndpoint>',
  'instanceName'='<yourInstanceName>',
  'tableName'='<yourTableName>',
  'accessId'='<yourAccessId>',
  'accessKey'='<yourAccessKey>'
);
```

WITH参数

参数	说明	是否必填	备注
connector	维表类型。	是	固定值为 `ots` 。
instanceName	实例名称。	是	无。
tableName	表名称。	是	无。
endPoint	实例访问地址。	是	无。
accessId	AccessKey ID。	是	无。
accessKey	AccessKey Secret。	是	无。
retryIntervalMs	重试间隔时间。	否	单位毫秒，默认值为1000（1秒）。
maxRetryTimes	最大重试次数。	否	默认值为100。
connectTimeout	Connector连接Tablestore的超时时间。	否	单位毫秒，默认值为30000（30秒）。

参数	说明	是否必填	备注
socketTimeout	Connector连接Tablestore的Socket超时时间。	否	单位毫秒，默认值为30000（30秒）。

CACHE参数

参数	说明	备注
cache	缓存策略	表格存储维表支持以下两种缓存策略： • *None（默认值）*：无缓存。 • *LRU*：缓存维表里的部分数据。源表的每条数据都会触发系统先在Cache中查找数据，如果没有找到，则去物理维表中查找。 需要配置相关参数：缓存大小（cacheSize）和缓存更新时间间隔（cacheTTLMs）。
cacheSize	缓存大小	当选择LRU缓存策略后，可以设置缓存大小，默认值为10000行。
cacheTTLMs	缓存超时时间，单位为毫秒。	cacheTTLMs配置和cache有关： • 如果cache配置为*None*，则cacheTTLMs可以不配置，表示缓存不超时。 • 如果cache配置为*LRU*，则cacheTTLMs为缓存超时时间。默认不过期。

常见问题

维表进行JOIN时，如果查询不到数据，应该如何处理？

9.2.4.3.3. MaxCompute维表

本文为您介绍MaxCompute维表的DDL定义、WITH参数、CACHE参数、类型映射和常见问题。

什么是MaxCompute

大数据计算服务MaxCompute（原名ODPS）是一种快速、完全托管的EB级数据仓库解决方案，致力于批量结构化数据的存储和计算，提供海量数据仓库的解决方案及分析建模服务。MaxCompute详情请参见什么是MaxCompute。

前提条件

已创建MaxCompute表，详情请参见创建表。

使用限制

仅Flink计算引擎VVR 2.0.0及以上版本支持MaxCompute Connector。

DDL定义

```
create table odps_dim(
  id VARCHAR,
  name VARCHAR,
  age int,
  PRIMARY KEY (id, name) not enforced
) with (
  'connector' = 'odps',
  'endpoint' = '<yourEndpoint>',
  'tunnelEndpoint' = '<yourTunnelEndpoint>',
  'project' = '<yourProjectName>',
  'tablename' = '<yourTableName>',
  'accessid' = '<yourAccessKeyId>',
  'accesskey' = '<yourAccessKeySecret>',
  'partition' = 'ds=2018****',
  'cache' = 'ALL'
);
```

说明

- 在维表进行JOIN时，ON条件必须包含所有主键的等值条件。
- 不支持将MaxCompute维表分区列写入DDL定义中。
- 主键字段必须按照PRIMARY KEY括号中字段的先后顺序，放在非主键字段之前。
- 声明维表时，定义主键，则实现一对一需求。如果您需要实现一对多需求，则不定义主键即可。

WITH参数

参数	说明	是否必填	备注
connector	维表类型。	是	固定值为 `odps` 。
endPoint	MaxCompute服务地址。	是	请参见Endpoint。
tunnelEndpoint	MaxCompute Tunnel服务的连接地址。	是	请参见Endpoint。 说明 VPC环境下为必填。
project	MaxCompute项目名称。	是	无。
tableName	表名。	是	无。
accessId	AccessKey ID。	是	无。
accessKey	AccessKey Secret。	是	无。

参数	说明	是否必填	备注
partition	分区名。	否	• 固定分区 ○ 只存在一级分区的MaxCompute表 例如，如果只存在1个分区列 `ds` ，则 `` `partition` = 'ds=20180905' `` 表示读 `ds=20180905` 分区的数据。 ○ 存在多级分区的MaxCompute表 例如，如果存在2个分区列 `ds` 和 `hh` ，则 `` `partition`='ds=20180905,hh=*' `` 表示读 `ds=20180905` 分区的数据。 说明 分区过滤时需要声明所有分区的值。例如，上述示例中，只声明 `` `partition` = 'ds=20180905' ` ，则不会读取任何分区。 • 非固定分区 ○ 支持 `` `partition` = 'max_pt()' ` ，即每次加载所有分区列表中字典序最大的分区。 ○ 支持 `` `partition` = 'max_pt_with_done()' ` ，即每次加载所有分区列表中字典序最大且伴随有 `.done` 的分区。

CACHE参数

参数	参数说明	备注

参数	参数说明	备注
cache	缓存策略。	目前MaxCompute维表仅支持 `ALL` 策略，必须显式声明。 *ALL*策略：缓存维表里的所有数据。在Job运行前，系统会将维表中所有数据加载到Cache中，之后所有的维表查询都会通过Cache进行。如果在Cache中无法找到数据，则KEY不存在，并在Cache过期后重新加载一遍全量Cache。 适用于远程表数据量小且MISS KEY（源表数据和维表JOIN时，ON条件无法关联）特别多的场景。需要配置缓存更新时间间隔（cacheTTLMs）和更新时间黑名单（cacheReloadTimeBlackList）参数。 **说明** • 因为系统会异步加载维表数据，所以在使用CACHE ALL时，需要增加维表JOIN节点的内存，增加的内存大小为远程表数据量的至少4倍，具体值与MaxCompute存储压缩算法有关。 • 在使用超大MaxCompute维表时，如果频繁GC（Allocation Failure）导致作业异常，且在增加维表JOIN节点的内存仍无改善的情况下，建议改为支持LRU cache策略的KV型维表，例如云数据库Hbase版维表。
cacheSize	缓存大小。	可以设置缓存大小，MaxCompute默认缓存值为100000行。 如果您的数据超过100000行，建议设置cacheSize参数值大于实际值，否则会报错 `Partition null table count ODPS tables row count exceeds maxRowCount limit {2}` 。
cacheTTLMs	缓存超时时间。	单位为毫秒，如果cache选择为 `ALL` 策略，则为缓存加载的间隔时间，默认为不重新加载。
cacheReloadTimeBlackList	更新时间黑名单。在缓存策略选择为ALL时，启用更新时间黑名单，防止在此时间内做Cache更新（例如双11场景）。	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` 。分隔符的使用情况如下所示： • 用英文逗号（,）来分隔多个黑名单。 • 用箭头（->）来分割黑名单的起始结束时间。

类型映射

MaxCompute字段类型	Flink字段类型
TINYINT	TINYINT
SMALLINT	SMALLINT
INT	INT
BIGINT	BIGINT
FLOAT	FLOAT
DOUBLE	DOUBLE
BOOLEAN	BOOLEAN
DATETIME	TIMESTAMP
TIMESTAMP	TIMESTAMP
VARCHAR	VARCHAR
STRING	VARCHAR
DECIMAL	DECIMAL
BINARY	VARBINARY

常见问题

- max_pt()和max_pt_with_done()的区别是什么？
- endPoint和tunnelEndpoint是指什么？如果配置错误会产生什么结果？
- 报错：ErrorMessage=Authorization Failed [4019], You have NO privilege'ODPS:***'

9.2.4.3.4. 云数据库Redis版维表

本文为您介绍云数据库Redis维表DDL定义、WITH参数、CACHE参数、类型映射和代码示例。

> **说明** Flink支持读取自建Redis服务的数据。

什么是云数据库Redis版

阿里云数据库Redis版是兼容开源Redis协议标准、提供内存加硬盘混合存储的数据库服务，基于高可靠双机热备架构及可平滑扩展的集群架构，充分满足高吞吐、低延迟及弹性变配的业务需求。

前提条件

- 已创建云数据库Redis实例，详情请参见步骤1：创建实例。
- 已设置白名单，详情请参见步骤2：设置白名单。

使用限制

- 仅Flink计算引擎VVR 2.0.0及以上版本支持云数据库Redis Connector。
- Redis维表仅支持读取Redis数据存储中STRING类型的数据。

DDL定义

```
CREATE TABLE redis_dim (
  id STRING,
  name STRING,
  PRIMARY KEY (id) NOT ENFORCED
) WITH (
  'connector' = 'redis',
  'host' = '<yourHost>',
  'port' = '<yourPort>',
  'password' = '<yourPassword>',
  'dbNum' = '<yourDbNum>'
);
```

说明

- Redis维表必须声明且只能声明一个主键。
- Redis维表仅支持声明两个字段，且字段类型必须为STRING。
- 维表JOIN时，ON条件必须包含所有主键的等值条件。
- Redis仅支持None和LRU两种缓存策略。

WITH参数

参数	说明	是否必填	备注
connector	维表类型。	是	固定值为 `redis` 。
host	Redis连接地址。	是	无。
port	Redis连接端口。	否	默认值为6379。
dbNum	选择操作的数据库。	否	默认值为0。
password	Redis密码。	否	默认值为空，不进行权限验证。
clusterMode	Redis集群是否为cluster模式。	否	取值如下： • true：为cluster模式。 • false（默认值）：不为cluster模式。

参数	说明	是否必填	备注
hashName	Hash模式下的Hash Key名称。	否	默认值为空。通常，Redis维表中的数据类型为STRING类型，即 `key-value` 对。如果设置hashName参数，则Redis维表中的数据类型为HASHMAP类型，即 `key-{field-value}` 对，其中： • key为hashName参数值。 • field为您在CREATE TABLE中指明的key参数值。 • value为key对应的赋值，和STRING类型 `key-value` 中value语义相同。

CACHE参数

参数	说明	是否必填	备注
cache	缓存策略。	否	云数据库Redis维表支持以下两种缓存策略： • *None（默认值）*：无缓存。 • *LRU*：缓存维表里的部分数据。源表的每条数据都会触发系统先在Cache中查找数据，如果没有找到，则去物理维表中查找。 需要配置相关参数：缓存大小（cacheSize）和缓存更新时间间隔（cacheTTLMs）。
cacheSize	缓存大小。	否	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，单位为毫秒。	否	cacheTTLMs配置和cache有关： • 如果cache配置为*None*，则cacheTTLMs可以不配置，表示缓存不超时。 • 如果cache配置为*LRU*，则cacheTTLMs为缓存超时时间。默认不过期。
cacheEmpty	是否缓存空结果。	否	默认值为true。

类型映射

Redis字段类型	Flink字段类型
STRING	VARCHAR

代码示例

```
CREATE TEMPORARY TABLE datagen_source (
  id STRING,
  data STRING,
  proctime as PROCTIME()
) WITH (
  'connector' = 'datagen'
);
CREATE TEMPORARY TABLE redis_dim (
  id STRING,
  name STRING,
  PRIMARY KEY (id) NOT ENFORCED --Redis中的Row Key字段。
) WITH (
  'connector' = 'redis',
  'host' = '<yourHost>',
  'port' = '<yourPort>',
  'password' = '<yourPassword>'
);
CREATE TEMPORARY TABLE blackhole_sink (
  id STRING,
  data STRING,
  name STRING
) WITH (
  'connector' = 'blackhole'
);
INSERT INTO blackhole_sink
SELECT e.*, w.*
FROM datagen_source AS e
JOIN redis_dim FOR SYSTEM_TIME AS OF e.proctime AS w
ON e.id = w.id;
```

9.2.4.3.5. Elasticsearch维表

本文为您介绍Elasticsearch（ES）维表DDL定义、WITH参数、CACHE参数和代码示例。

> **说明** 该Connector支持的功能是和阿里云Elasticsearch产品对齐的，暂不支持HTTPS加密访问形式。

什么是阿里云Elasticsearch

阿里云Elasticsearch兼容开源Elasticsearch的功能，以及Security、Machine Learning、Graph、APM等商业功能，致力于数据分析、数据搜索等场景服务。为您提供企业级权限管控、安全监控告警、自动报表生成等场景服务。

前提条件

- 已创建Elasticsearch索引，详情请参见步骤一：创建实例。
- 已配置Elasticsearch公网或私网访问白名单，详情请参见配置实例公网或私网访问白名单。

使用限制

仅Flink计算引擎VVR 2.0.0及以上版本支持Elasticsearch Connector。

DDL定义

```
CREATE TABLE es_dim(
  field1 LONG, --作为JOIN时的key，必须为STRING类型。
  field2 VARBINARY,
  field3 VARCHAR
) WITH (
  'connector' ='elasticsearch',
  'endPoint' = '<yourEndPoint>',
  'accessId' = '<yourAccessId>',
  'accessKey' = '<yourAccessSecret>',
  'indexName' = '<yourIndexName>',
  'typeNames' = '<yourTypeName>'
);
```

说明

- 维表JOIN时的key有且只能有1个，且为Elasticsearch对应索引的文档ID。
- 支持Elasticsearch 5.5及以上版本。

WITH参数

参数	说明	是否必填	备注
connector	维表类型	是	固定值为 `elasticsearch` 。
endPoint	服务地址	是	例如：*http://127.0.0.1:9200*。
accessId	Elasticsearch实例的用户名	否	无。
accessKey	Elasticsearch实例的密码	否	无。
indexName	索引名称	是	无。
typeNames	Type名称	否	默认值为 `_doc` 。 **说明** Elasticsearch 7.0以上版本不建议设置该参数。

CACHE参数

参数	说明	是否必填	备注

参数	说明	是否必填	备注
cache	缓存策略。	否	Elasticsearch仅支持以下两种缓存策略： • *None（默认值）*：无缓存。 • *LRU*：缓存维表里的部分数据。源表的每条数据都会触发系统先在Cache中查找数据，如果没有找到，则去物理维表中查找。 需要配置相关参数：缓存大小（cacheSize）和缓存更新时间间隔（cacheTTLMs）。
cacheSize	缓存大小。	否	选择 `LRU` 缓存策略后，可以设置缓存大小，默认值为10000行。
cacheTTLMs	缓存超时时间，单位为毫秒。	否	cacheTTLMs配置和cache有关： • 如果cache配置为*None*，则cacheTTLMs可以不配置，表示缓存不超时。 • 如果cache配置为*LRU*，则cacheTTLMs为缓存超时时间。默认值为10000毫秒。
cacheEmpty	是否缓存空结果。	否	默认值为true。

类型映射

Flink以JSON来解析Elasticsearch数据，详情请参见数据类型映射关系。

代码示例

```
CREATE TEMPORARY TABLE datagen_source (
  id STRING,
  data STRING,
  proctime as PROCTIME()
) WITH (
  'connector' = 'datagen'
);
CREATE TEMPORARY TABLE es_dim (
  id STRING,
  `value` FLOAT
) WITH (
  'connector' ='elasticsearch',
  'endPoint' = '<yourEndPoint>',
  'accessId' = '<yourAccessId>',
  'accessKey' = '<yourAccessSecret>',
  'indexName' = '<yourIndexName>',
  'typeNames' = '<yourTypeName>'
);
CREATE TEMPORARY TABLE blackhole_sink (
  id STRING,
  data STRING,
  `value` FLOAT
) WITH (
  'connector' = 'blackhole'
);
INSERT INTO blackhole_sink
SELECT e.*, w.*
  FROM datagen_source AS e
JOIN es_dim FOR SYSTEM_TIME AS OF e.proctime AS w
ON e.id = w.id;
```

9.2.4.3.6. 交互式分析Hologres维表

本文为您介绍交互式分析Hologres维表DDL定义、WITH参数、CACHE参数和代码示例。

> **说明** Hologres Connector不支持访问Hologres外部表。Hologres外部表详情请参见外部表。

什么是交互式分析Hologres

交互式分析Hologres兼容PostgreSQL协议，与大数据生态紧密连接，支持高并发、低延时实时分析处理PB级数据，让您轻松使用现有BI（Business Intelligence）工具对数据进行多维分析和业务探索。

前提条件

已创建Hologres表，详情请参见表。

使用限制

- 仅Flink计算引擎VVR 2.0.0及以上版本支持交互式分析Hologres Connector。
- 创建Hologres维表时建议选择行存模式，列存模式对于点查场景性能开销较大。

  选择行存模式创建维表时必须设置主键，并且将主键设置为*clustering key*才可以工作。示例语句如下：

```
begin;
create table test(a int primary key, b text, c text, d float8, e int8);
call set_table_property('test', 'orientation', 'row');
call set_table_property('test', 'clustering_key', 'a');
commit;
```

- Hologres维表的主键必须是Flink Join On的字段，Flink Join On的字段也必须是维表完整的主键字段，两者必须完全匹配。
- Flink计算引擎为VVR 4.0及以上版本的Hologres Flink Connector支持非主键的维表JOIN，而VVR 4.0以下版本的Hologres Flink Connector仅支持主键的维表JOIN。

DDL定义

```
CREATE TABLE hologres_dim(
  id INT,
  len INT,
  content VARCHAR
) with (
  'connector'='hologres',
  'dbname'='<yourDbname>',
  'tablename'='<yourTablename>',
  'username'='<yourUsername>',
  'password'='<yourPassword>',
  'endpoint'='<yourEndpoint>'
);
```

WITH参数

参数	说明	是否必填	备注
connector	维表类型。	是	固定值为 `hologres` （小写）。
endPoint	Hologres服务地址。	是	详情请参见访问域名。
tablename	表名称。 **说明** 如果Schema不为Public时，则tableName需要填写为schema.tableName。	是	无。
dbname	数据库名称。	是	无。
username	用户名，请填写阿里云账号的AccessKey ID。	是	无。
password	密码，请填写阿里云账号的AccessKey Secret。	否	无。

参数	说明	是否必填	备注
useRpcMode	自VVP 2.4.0版本开始，默认通过JDBC实现Hologres Connector。VVP 2.4.0以前版本默认是通过RPC实现Hologres Connector。	否	通过JDBC实现会占用SQL连接，导致JDBC链接数增加。如果您需要降低SQL连接数，则可以设置该参数值为true。 **注意** 仅VVP 2.4.0及以上版本支持该参数。
connectionSize	单个Flink维表任务所创建的JDBC连接池大小。 如果作业性能不足，建议您增加连接池大小。连接池大小和数据吞吐成正比。	否	默认值为3，单位为个。 **注意** 仅VVP 2.4.0及以上版本支持该参数。

CACHE参数

参数	说明	是否必填	备注
cache	缓存策略。	否	Hologres仅支持以下两种缓存策略： • *LRU*：缓存维表里的部分数据。源表的每条数据都会触发系统先在Cache中查找数据，如果没有找到，则去物理维表中查找。 需要配置相关参数：缓存大小（cacheSize）和缓存更新时间间隔（cacheTTLMs）。 • *None*：无缓存。 **说明** cache参数默认值情况如下： • VVR 4.x版本及以上版本：None。 • VVR 4.x版本以下版本：LRU。
cacheSize	缓存大小。	否	选择 `LRU` 缓存策略后，可以设置缓存大小，默认值为10000行。
cacheTTLMs	缓存更新时间间隔。	否	cacheTTLMs配置和cache有关： • 如果cache配置为*LRU*，则cacheTTLMs为缓存超时时间。默认不过期。 • 如果cache配置为*None*，则cacheTTLMs可以不配置，表示缓存不超时。

参数	说明	是否必填	备注
async	是否异步同步数据。	否	取值如下： • true：表示异步同步数据。 • false（默认值）：表示不进行异步同步数据。

类型映射

Hologres字段类型	Flink字段类型
INT	INT
INT[]	ARRAY<INT>
BIGINT	BIGINT
BIGINT[]	ARRAY<BIGINT>
REAL	FLOAT
REAL[]	ARRAY<REAL>
DOUBLE PRECISION	DOUBLE
DOUBLE PRECISION[]	ARRAY<DOUBLE PRECISION>
BOOLEAN	BOOLEAN
BOOLEAN[]	ARRAY<BOOLEAN>
TEXT	VARCHAR
TEXT[]	ARRAY<TEXT>
NUMERIC	DECIMAL
DATE	DATE
TIMESTAMP WITH TIMEZONE	TIMESTAMP

说明 上表中未列出的Hologres类型，Connector还未支持转换。

代码示例

```
CREATE TEMPORARY TABLE datagen_source (
  a INT,
  b BIGINT,
  c STRING,
  proctime AS PROCTIME()
) WITH (
  'connector' = 'datagen'
);
CREATE TEMPORARY TABLE hologres_dim (
  a INT,
  b VARCHAR,
  c VARCHAR
) WITH (
  'connector' = 'hologres',
   ...
);
CREATE TEMPORARY TABLE blackhole_sink (
  a INT,
  b STRING
) WITH (
  'connector' = 'blackhole'
);
insert into blackhole_sink select T.a,H.b
FROM datagen_source AS T JOIN hologres_dim FOR SYSTEM_TIME AS OF T.proctime AS H ON T.a = H
.a;
```

9.2.4.3.7. 云数据库RDS MySQL维表

本文为您介绍云数据库RDS MySQL维表DDL定义、WITH参数、CACHE参数、类型映射和代码示例。

什么是云数据库RDS MySQL

RDS MySQL基于阿里巴巴的MySQL源码分支，经过双十一高并发、大数据量的考验，拥有优良的性能。RDS MySQL支持实例管理、账号管理、数据库管理、备份恢复、白名单、透明数据加密以及数据迁移等基本功能。RDS MySQL详情请参见概述。

前提条件

- 已创建RDS MySQL数据库和表，详情请参见创建数据库和账号。
- 已设置IP白名单，详情请参见通过客户端、命令行连接RDS MySQL实例。

使用限制

仅Flink计算引擎VVR 2.0.0及以上版本支持云数据库RDS MySQL Connector。

语法示例

```
CREATE TABLE rds_dim (
  id1 INT,
  id2 VARCHAR
) WITH (
  'connector' = 'rds',
  'password' = '<yourPassword>',
  'tableName' = '<yourTablename>',
  'url' = '<yourUrl>',
  'userName' = '<yourUsername>',
  'cache' = 'ALL'
);
```

WITH参数

参数	说明	是否必填	备注
connector	维表类型	是	固定值为 `rds` 。
password	密码	是	无
tableName	表名	是	无
url	URL地址	是	云数据库RDS MySQL专有网络VPC地址，即内网地址，详情请参见查看或修改内外网地址和端口。URL的格式为： `jdbc:mysql://<内网地址>/<databaseName>` ，其中databaseName为对应的数据库名称。
userName	用户名	是	无
maxRetryTimes	写入数据失败后，重试写入的次数	否	默认值为3。
CACHE	缓存策略、缓存大小和缓存超时时间	否	详情请参见CACHE参数。

CACHE参数

参数	说明	是否必填	备注

参数	说明	是否必填	备注
cache	缓存策略	否	云数据库RDS（DRDS）版维表支持以下三种缓存策略： • *None（默认值）*：无缓存。 说明 VVR 4.0.6及以后版本的缓存策略的默认值为ALL。 • *LRU*：缓存维表里的部分数据。源表的每条数据都会触发系统先在Cache中查找数据，如果没有找到，则去物理维表中查找。 • *ALL*：缓存维表里的所有数据。在Job运行前，系统会将维表中所有数据加载到Cache中，之后所有的维表查找数据都会通过Cache进行。如果在Cache中无法找到数据，则KEY不存在，并在Cache过期后重新加载一遍全量Cache。 适用于远程表数据量小且MISS KEY（源表数据和维表JOIN时，ON条件无法关联）特别多的场景。 说明 • 使用ALL或LRU缓存策略时，必须配置cacheSize参数。 • 如果使用CACHE ALL时，请注意节点内存大小，防止出现OOM。 • 因为系统会异步加载维表数据，所以在使用CACHE ALL时，需要增加维表JOIN节点的内存，增加的内存大小为远程表数据量的两倍。
cacheSize	缓存大小	否	• 当选择LRU缓存策略后，必须设置缓存大小。 • 当选择ALL缓存策略后，可以不设置缓存大小。
cacheTTLMs	缓存超时时间，单位为毫秒	否	cacheTTLMs配置和cache有关： • 如果cache配置为*None*，则cacheTTLMs可以不配置，表示缓存不超时。 • 如果cache配置为*LRU*，则cacheTTLMs为缓存超时时间。默认不过期。 • 如果cache配置为*ALL*，则cacheTTLMs为缓存加载时间。默认不重新加载。
maxJoinRows	主表中每一条数据查询维表时，匹配后最多返回的结果数。	否	默认值为1024。如果您可以预估一条数据对应的维表数据最多为n条，则可以设置maxJoinRows='n'，以确保实时计算匹配处理效率。 说明 进行Join时，主表输入一条数据，对应维表匹配后返回的结果总数受该参数限制。

类型映射

RDS字段类型	Flink字段类型
BOOLEAN	BOOLEAN
TINYINT	TINYINT
SMALLINT	SMALLINT
TINYINT UNSIGNED	SMALLINT
INT	INT
SMALLINT UNSIGNED	INT
BIGINT	BIGINT
INT UNSIGNED	BIGINT
BIGINT UNSIGNED	DECIMAL(20,0)
FLOAT	FLOAT
DECIMAL	DECIMAL
DOUBLE	DOUBLE
DATE	DATE
TIME	TIME
TIMESTAMP	TIMESTAMP
VARCHAR	VARCHAR
VARBINARY	VARBINARY

代码示例

```
CREATE TEMPORARY TABLE datagen_source(
  a INT,
  b BIGINT,
  c STRING,
  `proctime` AS PROCTIME()
) with (
  'connector' = 'datagen'
);
CREATE TEMPORARY TABLE rds_dim (
  a INT,
  b VARCHAR,
  c VARCHAR
) with (
  'connector' = 'rds',
  'password' = '<yourPassword>',
  'tableName' = '<yourTablename>',
  'url' = 'jdbc:mysql://xxx',
  'userName' = '<yourUsername>'
);
CREATE TEMPORARY TABLE blackhole_sink(
  a INT,
  b STRING
) with (
  'connector' = 'blackhole'
);
insert into blackhole_sink select T.a,H.b
FROM datagen_source AS T JOIN rds_dim FOR SYSTEM_TIME AS OF T.`proctime` AS H ON T.a = H.a;
```

常见问题

为什么MySQL物理表（包含RDS MySQL和ADB）的INT UNSIGNED字段类型，在Flink SQL中要被声明为其他类型？

9.2.4.3.8. 云原生数据仓库AnalyticDB MySQL版3.0维表

本文为您介绍云原生数据仓库AnalyticDB MySQL版3.0维表的DDL定义、WITH参数、CACHE参数和类型映射。

什么是云原生数据仓库AnalyticDB MySQL版

云原生数据仓库AnalyticDB MySQL版是融合数据库、大数据技术于一体的云原生企业级数据仓库服务。AnalyticDB MySQL版支持高吞吐的数据实时增删改、低延时的实时分析和复杂ETL，兼容上下游生态工具，可用于构建企业级报表系统、数据仓库和数据服务引擎。

前提条件

- 已创建AnalyticDB MySQL集群并创建表，详情请参见创建集群和CREATE TABLE。
- 已设置白名单，详情请参见设置白名单。

使用限制

仅Flink计算引擎VVR 2.0.0及以上版本支持云原生数据仓库AnalyticDB MySQL版3.0 Connector。

DDL定义

```
CREATE TABLE adb30_dim (
  id1 INT,
  id2 VARCHAR
) WITH (
  'connector' = 'adb3.0',
  'password' = '<yourPassword>',
  'tableName' = '<yourTablename>',
  'url' = '<yourUrl>',
  'userName' = '<yourUsername>',
  'cache' = 'ALL',
  'cacheSize' = '500'
);
```

WITH参数

参数	说明	是否必选	备注
connector	维表类型。	是	固定值为 `adb3.0` 。
password	密码。	是	无。
tableName	表名。	是	无。
url	URL地址。	是	云原生数据仓库AnalyticDB MySQL版3.0的专有网络VPC地址。
username	用户名。	是	无。
maxRetryTimes	写入数据失败后，重试写入的次数。	否	默认值为3。
CACHE	缓存策略、缓存大小和缓存超时时间。	否	详情请参见CACHE参数。

CACHE参数

参数	说明	是否必填	备注

参数	说明	是否必填	备注
cache	缓存策略。	否	云原生数据仓库AnalyticDB MySQL版3.0维表支持以下三种缓存策略： • *None（默认值）*：无缓存。 • *LRU*：缓存维表里的部分数据。源表的每条数据都会触发系统先在Cache中查找数据，如果没有找到，则去物理维表中查找。 • *ALL*：缓存维表里的所有数据。在Job运行前，系统会将维表中所有数据加载到Cache中，之后所有的维表查找数据都会通过Cache进行。如果在Cache中无法找到数据，则KEY不存在，并在Cache过期后重新加载一遍全量Cache。 适用于远程表数据量小且MISS KEY（源表数据和维表JOIN时，ON条件无法关联）特别多的场景。 **说明** • 使用ALL或LRU缓存策略时，必须配置cacheSize参数。 • 如果使用CACHE ALL时，请注意节点内存大小，防止出现OOM。 • 因为系统会异步加载维表数据，所以在使用CACHE ALL时，需要增加维表JOIN节点的内存，增加的内存大小为远程表数据量的两倍。
cacheSize	缓存大小，即缓存多少行数据。	否	cacheSize配置和cache有关： • 如果cache配置为*None*，则不用配置cacheSize参数（默认为空）。 • 如果cache配置为*LRU*，则必须配置cacheSize参数。 • 如果cache配置为*ALL*，则必须配置cacheSize参数。
cacheTTLMs	缓存超时时间，单位为毫秒。	否	cacheTTLMs配置和cache有关： • 如果cache配置为*None*，则cacheTTLMs可以不配置，表示缓存不超时。 • 如果cache配置为*LRU*，则cacheTTLMs为缓存超时时间，默认值为10000ms。 • 如果cache配置为*ALL*，则cacheTTLMs为缓存加载时间，默认值为10000ms。
maxJoinRows	主表中每一条数据查询维表时，匹配后最多返回的结果数。	否	默认值为1024。如果您可以预估一条数据对应的维表数据最多为n条，则可以设置maxJoinRows='n'，以确保实时计算匹配处理效率。 **说明** 进行Join时，主表输入一条数据，对应维表匹配后返回的结果总数受该参数限制。

类型映射

云原生数据仓库AnalyticDB MySQL版3.0字段类型	Flink字段类型
BOOLEAN	BOOLEAN
TINYINT	INT
SMALLINT	INT
INT	INT
BIGINT	BIGINT
DOUBLE	DOUBLE
VARCHAR	VARCHAR
DATETIME	TIMESTAMP
DATE	DATE

代码示例

```
CREATE TEMPORARY TABLE datagen_source(
  a INT,
  b VARCHAR,
  c STRING,
  `proctime` AS PROCTIME()
) with (
  'connector' = 'datagen'
);
CREATE TEMPORARY TABLE adb_dim (
  a INT,
  b VARCHAR,
  c VARCHAR
) with (
  'connector' = 'adb3.0',
  'password' = '<yourPassword>',
  'tableName' = '<yourTablename>',
  'url' = '<yourUrl>',
  'userName' = '<yourUsername>'
);
CREATE TEMPORARY TABLE blackhole_sink(
  a INT,
  b VARCHAR
) with (
  'connector' = 'blackhole'
);
insert into blackhole_sink select T.a,H.b
FROM datagen_source AS T JOIN adb_dim FOR SYSTEM_TIME AS OF T.proctime AS H ON T.a = H.a;
```

常见问题

为什么MySQL物理表（包含RDS MySQL和ADB）的INT UNSIGNED字段类型，在Flink SQL中要被声明为其他类型？

9.2.4.3.9. FileSystem维表

本文为您介绍FileSystem维表的DDL定义，以及创建维表时使用的WITH参数和代码示例。

前提条件

已创建OSS存储空间，详细请参见创建存储空间。

使用限制

仅Flink计算引擎VVR 3.0.0及以上版本支持FileSystem Connector。

DDL定义

```
CREATE TABLE filesystem_dim (
  id STRING,
  name STRING
) WITH (
  'connector' = 'filesystem',
  'path' = 'oss://my_path/my_file',
  'format' = 'csv'
)
```

WITH 参数

参数	说明	是否必选	备注
connector	维表类型	是	固定值为filesystem。
path	文件路径	是	URI格式，例如：*oss://my_path/my_file*。
format	文件格式	是	参数取值如下： • csv • json • avro • parquet • orc • raw format参数详情请参见formats。
lookup.join.cache.ttl	重新读取数据的TTL时间	否	默认值为60分钟，即每隔60分钟重新读取数据。

代码示例

```
--创建event源表。
CREATE TABLE event (
  id STRING,
  data STRING
) WITH (
  'connector' = 'datahub'
   ...
);
--创建white_list维表。
CREATE TABLE white_list (
  id STRING,
  name STRING,
) WITH (
  'connector' = 'filesystem',
  'path' = '${remote_path_uri}',
  'format' = '${format}'
);
--关联event源表和white_list维表。
SELECT e.*, w.*
FROM event AS e
JOIN white_list FOR SYSTEM_TIME AS OF proctime() AS w
ON e.id = w.id;
```

9.2.4.3.10. MySQL维表

本文为您介绍MySQL维表的DDL定义、WITH参数和类型映射。

什么是MySQL维表

MySQL维表支持所有兼容MySQL协议的数据库，包括RDS MySQL、PolarDB for MySQL或者自建MySQL，而云数据库RDS MySQL维表专为云数据库RDS MySQL定制，支持更丰富的配置参数。

前提条件

- 已创建MySQL数据库和表，详情请参见RDS MySQL创建数据库和账号、PolarDB MySQL创建数据库和账号或自建MySQL创建数据库和账号。
- 已设置IP白名单，详情请参见RDS MySQL白名单设置、PolarDB MySQL白名单设置或自建MySQL白名单设置。

使用限制

仅Flink计算引擎VVR 4.0.11及以上版本支持MySQL Connector。

DDL定义

```
CREATE TABLE mysql_dim (
   id INT,
   name STRING,
   PRIMARY KEY (id) NOT ENFORCED
) WITH (
   'connector' = 'mysql',
   'hostname' = '<yourHostname>',
   'port' = '3306',
   'username' = '<yourUsername>',
   'password' = '<yourPassword>',
   'database-name' = '<yourDatabaseName>',
   'table-name' = '<yourTableName>'
);
```

WITH参数

参数	说明	数据类型	是否必填	备注
connector	维表类型。	String	是	固定值为 `mysql` 。
hostname	MySQL数据库的IP地址或者hostname。	String	是	建议填写专有网络VPC地址。 **说明** 如果MySQL与Flink全托管不在同一VPC，则需填写公网地址。
username	MySQL数据库服务的用户名。	String	是	无。
password	MySQL数据库服务的密码。	String	是	无。
database-name	MySQL数据库名称。	String	是	无。
table-name	MySQL表名。	String	是	无。
port	MySQL数据库服务的端口号。	Integer	否	默认值为3306。

参数	说明	数据类型	是否必填	备注
lookup.cache.strategy	缓存策略。	String	否	支持以下三种缓存策略： • *None（默认值）*：无缓存。 • *LRU*：缓存维表里的部分数据。源表的每条数据都会触发系统先在Cache中查找数据。如果没有找到，则去物理维表中查找。 • *ALL*：缓存维表里的所有数据。在Job运行前，系统会将维表中所有数据加载到Cache中，之后所有的维表查找数据都会通过Cache进行。如果在Cache中无法找到数据，则KEY不存在，并在Cache过期后重新加载一遍全量Cache。 适用于远程表数据量小且MISS KEY（源表数据和维表JOIN时，ON条件无法关联）特别多的场景。 **说明** • 使用LRU缓存策略时，必须配置lookup.cache.max-rows参数。 • 如果使用CACHE ALL时，请注意节点内存大小，防止出现OOM。 • 因为系统会异步加载维表数据，所以在使用CACHE ALL时，需要增加维表JOIN节点的内存，增加的内存大小为远程表数据量的两倍。
lookup.cache.max-rows	最大缓存条数。	Integer	否	• 当选择LRU缓存策略后，必须设置缓存大小。 • 当选择ALL缓存策略后，可以不设置缓存大小。
lookup.cache.ttl	缓存超时时间。	Duration	否	lookup.cache.ttl的配置和lookup.cache.strategy有关，详情如下： • 如果lookup.cache.strategy配置为*None*，则lookup.cache.ttl可以不配置，表示缓存不超时。 • 如果lookup.cache.strategy配置为*LRU*，则lookup.cache.ttl为缓存超时时间。默认不过期。 • 如果lookup.cache.strategy配置为*ALL*，则lookup.cache.ttl为缓存加载时间。默认不重新加载。

类型映射

MySQL字段类型	Flink字段类型
TINYINT	TINYINT

MySQL字段类型	Flink字段类型
SMALLINT	SMALLINT
TINYINT UNSIGNED	
INT	INT
MEDIUMINT	
SMALLINT UNSIGNED	
BIGINT	BIGINT
INT UNSIGNED	
BIGINT UNSIGNED	DECIMAL(20, 0)
BIGINT	BIGINT
FLOAT	FLOAT
DOUBLE	DOUBLE
DOUBLE PRECISION	
NUMERIC(p, s)	DECIMAL(p, s) **说明** 其中p <= 38。
DECIMAL(p, s)	
BOOLEAN	BOOLEAN
TINYINT(1)	
DATE	DATE
TIME [(p)]	TIME [(p)] [WITHOUT TIMEZONE]
DATETIME [(p)]	TIMESTAMP [(p)] [WITHOUT TIMEZONE]
TIMESTAMP [(p)]	
CHAR(n)	CHAR(n)
VARCHAR(n)	VARCHAR(n)
BIT(n)	BINARY(⌈n/8⌉)
BINARY(n)	BINARY(n)
VARBINARY(N)	VARBINARY(N)

MySQL字段类型	Flink字段类型
TINYTEXT	STRING
TEXT	
MEDIUMTEXT	
LONGTEXT	
TINYBLOB	BYTES 说明 Flink仅支持小于等于2,147,483,647(2^31 - 1)的MySQL BLOB类型的记录。
BLOB	
MEDIUMBLOB	
LONGBLOB	

代码示例

```
CREATE TEMPORARY TABLE datagen_source(
  a INT,
  b BIGINT,
  c STRING,
  `proctime` AS PROCTIME()
) WITH (
  'connector' = 'datagen'
);
CREATE TEMPORARY TABLE mysql_dim (
  a INT,
  b VARCHAR,
  c VARCHAR
) WITH (
  'connector' = 'mysql',
  'hostname' = '<yourHostname>',
  'port' = '3306',
  'username' = '<yourUsername>',
  'password' = '<yourPassword>',
  'database-name' = '<yourDatabaseName>',
  'table-name' = '<yourTableName>'
);
CREATE TEMPORARY TABLE blackhole_sink(
  a INT,
  b STRING
) WITH (
  'connector' = 'blackhole'
);
INSERT INTO blackhole_sink
SELECT T.a, H.b
FROM datagen_source AS T JOIN mysql_dim FOR SYSTEM_TIME AS OF T.`proctime` AS H ON T.a = H.
a;
```

9.2.4.3.11. 模拟数据生成维表

本文为您介绍模拟数据生成维表的使用限制、DDL定义、WITH参数和代码示例。

什么是模拟数据生成维表

模拟数据生成维表是系统内置的Connector，它根据表中每个字段提供的Java Faker表达式生成测试数据。当您在开发或测试时，需要使用一些测试数据来验证业务逻辑时，推荐您使用模拟数据生成Connector。

使用限制

- 仅Flink计算引擎VVR 4.0.12及以上版本支持模拟数据生成Connector。
- 支持部分数据类型，包括CHAR(n)、VARCHAR(n)、STRING、TINYINT、SMALLINT、INT、BIGINT、FLOAT、DOUBLE、DECIMAL、BOOLEAN、TIMESTAMP、ARRAY、MAP、MULTISET和ROW。

DDL定义

```
CREATE TABLE faker_dim (
  `name` STRING,
  `age` INT
) WITH (
  'connector' = 'faker',
  'fields.name.expression' = '#{superhero.name}',
  'fields.age.expression' = '#{number.numberBetween ''0'',''1000''}'
);
```

WITH参数

参数	说明	数据类型	是否必填	备注
connector	维表类型。	STRING	是	固定值为faker。
fields.<field>.expression	生成该字段值的Java Faker表达式。	STRING	是	详情请参见下文字段表达式。
fields.<field>.null-rate	该字段值为空的占比。	FLOAT	否	默认值为0.0。
fields.<field>.length	ARRAY、MAP或MULTISET集合类型的大小。	INTEGER	否	默认值为1。

字段表达式

- 操作方法

  使用模拟数据生成Connector时，DDL中定义的每个字段，都需要在WITH语句中提供具体的表达式。表达式的固定格式为'fields.<field>.expression' = '#{className.methodName "parameter", ...}'。相关的参数说明如下表所示。

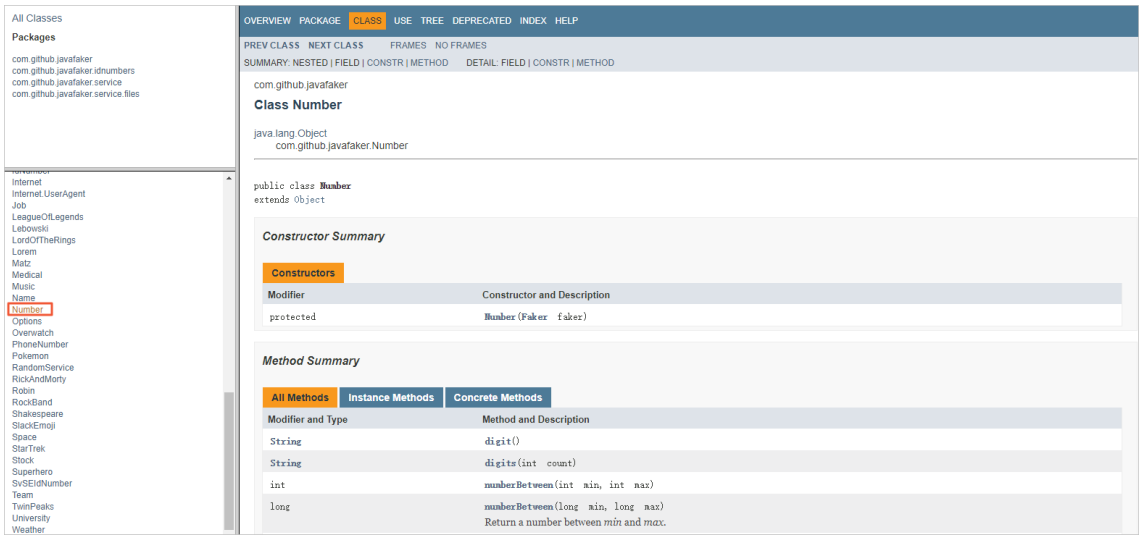
参数	说明

参数	说明
field	表示DDL中具体的字段名。
className	表示Faker类的类名。 Java Faker提供了大约80个Faker类用于生成您需要的字段表达式，您可以根据需要选择对应的类。 说明 Faker类的类名不区分大小写。
methodName	表示方法名。 说明 方法名不区分大小写。
parameter	表示方法的入参。 说明 ○ 方法的入参需要使用两个半角单引号（' ）括起来。 ○ 多个参数使用半角逗号（,）进行分隔。

- 示例

  本文结合Java Faker API文档和DDL定义部分中的age字段表达式'fields.age.expression' = '#{number.numberBetween ''0'',''1000''}' 为例，为您介绍如何为DDL中的字段正确地生成SQL表达式。

  i. 在Java Faker API文档中，找到Number类。



  ii. 在Number类中找到numberBetween方法，并查看其方法描述。

Method Summary

All Methods | Instance Methods | Concrete Methods

Modifier and Type	Method and Description
String	digit()
String	digits(int count)
int	numberBetween(int min, int max)
long	numberBetween(long min, long max) Return a number between *min* and *max*.
int	randomDigit() Returns a random number from 0-9 (both inclusive)
int	randomDigitNotZero() Returns a random number from 1-9 (both inclusive)
double	randomDouble(int maxNumberOfDecimals, int min, int max)
double	randomDouble(int maxNumberOfDecimals, long min, long max) Returns a random double
long	randomNumber() Returns a random number
long	randomNumber(int numberOfDigits, boolean strict)

numberBetween方法表示返回指定数字范围的值。

iii. 根据类名Number和方法名numberBetween传入方法的参数0和1000，生成age字段的SQL表达式'fields.age.expression' = '#{number.numberBetween ''0'',''1000''}' 。

表示生成的age字段的值在0到1000范围内。

代码示例

```
CREATE TEMPORARY TABLE datagen_source (
  `character_id` INT,
  `location` STRING,
  `proctime` AS PROCTIME()
) WITH (
  'connector' = 'datagen'
);
CREATE TEMPORARY TABLE faker_dim (
  `character_id` INT,
  `name` STRING
) WITH (
  'connector' = 'faker',
  'fields.character_id.expression' = '#{number.numberBetween ''0'',''100''}',
  'fields.name.expression' = '#{harry_potter.characters}'
);
SELECT
  c.character_id,
  l.location,
  c.name
FROM datagen_source AS l
JOIN faker_dim FOR SYSTEM_TIME AS OF proctime AS c
ON l.character_id = c.character_id;
```

9.2.4.3.12. 云原生数据仓库AnalyticDB PostgreSQL版维表

本文为您介绍云原生数据仓库AnalyticDB PostgreSQL版维表的DDL定义、WITH参数和类型映射。

什么是云原生数据仓库AnalyticDB PostgreSQL版

云原生数据仓库AnalyticDB PostgreSQL版基于开源项目Greenplum构建，由阿里云深度扩展，兼容ANSI SQL 2003，兼容PostgreSQL和Oracle数据库生态，支持行存储和列存储模式。既提供高性能离线数据处理，也支持高并发在线分析查询，是各行业有竞争力的PB级实时数据仓库方案。它是一种大规模并行处理（MPP）数据仓库服务，可提供海量数据在线分析服务。

前提条件

- 已创建AnalyticDB PostgreSQL实例并创建表，详情请参见创建实例和数据表管理。
- 已设置白名单，详情请参见设置白名单。

使用限制

仅Flink计算引擎VVR 5.0.0及以上版本支持云原生数据仓库AnalyticDB PostgreSQL版Connector。

DDL定义

```
CREATE TABLE adbpg_dim (
  id INT,
  len INT,
  content VARCHAR,
  PRIMARY KEY(id)
) WITH (
  'connector'='adbpg',
  'url'='jdbc:postgresql://<Address>:<PortId>/<DatabaseName>',
  'tableName'='<yourDatabaseTableName>',
  'userName'='<yourDatabaseUserName>',
  'password'='<yourDatabasePassword>'
);
```

WITH参数

参数	注释说明	是否必选	备注
connector	维表类型。	是	固定值为 `adbpg` 。
url	JDBC连接地址。	是	格式为：jdbc:postgresql://<Address>:<PortId>/<DatabaseName>
tableName	表名。	是	无。
username	用户名。	是	无。
password	密码。	是	无。

参数	注释说明	是否必选	备注
maxRetryTimes	写入数据失败后，重试写入的最大次数。	否	默认值为3。
targetSchema	Schema名称。	否	默认值为public。
caseSensitive	大小写是否敏感。	否	参数取值如下： • true：大小写敏感。 • false（默认值）：大小写不敏感。
connectionMaxActive	连接池的最大连接数。	否	默认值为5。 说明 • 系统会自动释放和数据库服务的空闲连接。 • 设置过大可能会导致服务端连接数出现异常。
maxJoinRows	单行数据Join的最多行数。	否	默认值为1024。
cache	缓存策略。	否	支持以下三种缓存策略： • *ALL*（默认值）：缓存维表里的所有数据。在Job运行前，系统会将维表中所有数据加载到Cache中，之后所有的维表查找数据都会通过Cache进行。如果在Cache中无法找到数据，则KEY不存在，并在Cache过期后重新加载一遍全量Cache。 • *LRU*：缓存维表里的部分数据。源表的每条数据都会触发系统先在Cache中查找数据。如果没有找到，则去物理维表中查找。 • *None（默认值）*：无缓存。
cacheSize	缓存大小，即缓存多少行数据。	否	当选择LRU缓存策略时，cacheSize参数生效。默认值为100000行。
cacheTTLMs	缓存失效的超时时间。	否	cacheTTLMs配置和cache有关： • 如果cache配置为*LRU*，则cacheTTLMs为缓存失效的超时时间。默认不过期。 • 如果cache配置为*ALL*，则cacheTTLMs为设置缓存重新加载的间隔时间，默认不重新加载。 单位为毫秒。

类型映射

云原生数据仓库AnalyticDB PostgreSQL版字段类型	Flink字段类型
BOOLEAN	BOOLEAN
SMALLINT	INT
INT	INT
BIGINT	BIGINT
FLOAT	DOUBLE
VARCHAR	VARCHAR
TIMESTAMP	TIMESTAMP
DATE	DATE

代码示例

```
CREATE TEMPORARY TABLE datagen_source(
  a INT,
  b BIGINT,
  c STRING,
  `proctime` AS PROCTIME()
)
COMMENT 'datagen source table'
WITH (
   'connector' = 'datagen'
};
CREATE TABLE adbpg_dim (
  a INT,
  b VARCHAR,
  c VARCHAR
) WITH (
  'connector'='adbpg',
  'url'='jdbc:postgresql://<Address>:<PortId>/<DatabaseName>',
  'tableName'='<yourDatabaseTableName>',
  'userName'='<yourDatabaseUserName>',
  'password'='<yourDatabasePassword>'
);
CREATE TEMPORARY TABLE blackhole_sink(
  a INT,
  b STRING
)
  COMMENT 'blackhole sink table'
  WITH (
   'connector' = 'blackhole'
);
INSERT INTO blackhole_sink SELECT T.a,H.b
FROM datagen_source AS T JOIN adb_dim FOR SYSTEM_TIME AS OF T.proctime AS H ON T.a = H.a;
```

9.2.5. DML数据操作语句

9.2.5.1. INSERT INTO语句

本文为您介绍如何在一个作业中写入一个Sink或多个Sink。

写入一个Sink示例

- 源表

```
CREATE TABLE datagen_source (
  name VARCHAR,
  score BIGINT
) WITH (
  'connector' = 'datagen'
);
```

- 结果表

```
create table blackhole_sink(
  name VARCHAR,
  score BIGINT
) WITH (
  'connector' = 'blackhole'
);
```

- DML

```
INSERT INTO blackhole_sink SELECT UPPER(name), score FROM datagen_source;
```

写入多个Sink示例

- 源表

```
CREATE TABLE datagen_source (
  name VARCHAR,
  score BIGINT
) WITH (
  'connector' = 'datagen'
);
```

- 结果表A

```
CREATE TABLE blackhole_sinkA(
  name VARCHAR,
  score BIGINT
) WITH (
  'connector' = 'blackhole'
);
```

- 结果表B

```
CREATE TABLE blackhole_sinkB(
  name VARCHAR,
  score BIGINT
) WITH (
  'connector' = 'blackhole'
);
```

- DML

  本文以写入两个Sink为例进行介绍。

```
BEGIN STATEMENT SET;      --写入多个Sink时，必填。
INSERT INTO blackhole_sinkA
  SELECT UPPER(name), sum(score)
  FROM datagen_source
  GROUP BY UPPER(name);
INSERT INTO blackhole_sinkB
  SELECT LOWER(name), max(score)
  FROM datagen_source
  GROUP BY LOWER(name);
END;      --写入多个Sink时，必填。
```

> **说明** 写入多个Sink语句时，需要以BEGIN STATEMENT SET;开头，以END;结尾。

9.2.5.2. CREATE TABLE AS(CTAS)语句

通过CTAS语句，在实时同步数据的同时，还能实时将上游表结构（Schema）的变更同步到下游表，提高您在目标存储中创建表和维护源表结构变更的效率。本文为您介绍CREATE TABLE AS(CTAS)的背景信息、前提条件、使用限制、基本语法和代码示例。

背景信息

- 功能特性

功能	详情
单表同步	支持实时同步源表的全量和增量数据到目标表中。
表结构变更同步	在实时同步数据的同时，还支持将源表的表结构变更（增加列信息等）实时同步到目标表中。
分库分表合并同步	支持使用正则表达式定义库名和表名，匹配数据源的多张分库分表，合并后同步到下游的一张表中。
自定义计算列同步	支持在源表上新增计算列，以支持您对源表的某些列进行转换计算。允许指定新增列的位置，并将其作为目标表的物理列，实时地将计算列的结果同步到目标表中。同时CTAS语法也支持修改目标表的主键，可以把新增列作为目标表的主键字段。
多CTAS语句	支持使用STATEMENT SET语法将多个CTAS语句作为一个作业一起提交，并支持对Source节点的合并复用，降低对数据源的压力。

- 启动流程

当执行CTAS语句时，阿里云Flink将会按照以下流程执行：

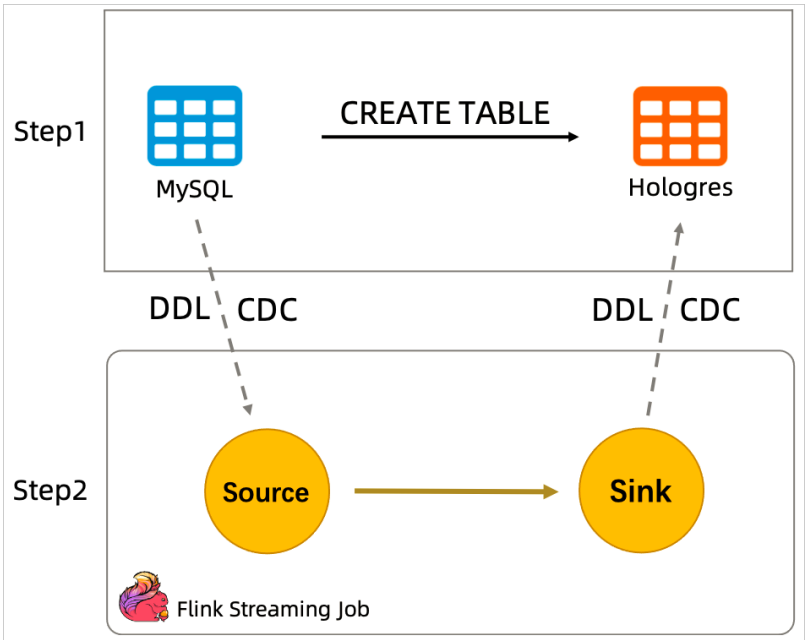
i. 检查目标存储中是否存在该目标表。

   如果不存在，则通过目标端Catalog去目标存储中创建相应的目标表，该目标表具有和数据源相同的Schema；如果存在，则跳过建表；如果已存在的目标表与源表Schema不一致，则会报错提示。

ii. 提交和启动相应的数据同步作业。

   将数据源的数据以及Schema的变更同步到目标表中。

从MySQL到Hologres同步CTAS数据同步流程如下图所示。



- 表结构变更同步策略

  通过CTAS语句，在实时同步数据的同时，还能将源表Schema的变更同步到目标表中。Schema变更包括初始的表创建以及未来的表变更。当前支持的Schema变更策略详情如下：

  - 添加可空列：会自动在目标表Schema末尾添加对应的列，并自动同步新增列的数据。
  - 删除可空列：不会直接在目标表中删除该列，而是将该列的数据自动填充为NULL值。
  - 重命名列：被看作为添加列和删除列。直接在目标表中末尾添加重名名后的列，并将重命名前的列数据自动填充为NULL值。例如如果col_a重命名为col_b，则会在目标表末尾添加col_b，并自动将col_a的数据填充为NULL值。

  暂不支持同步以下Schema的变更：

  - 数据类型的变更。

    例如由VARCHAR变为BIGINT、由NOT NULL变为NULLABLE属性。
  - 主键或索引等约束的变更。
  - 非空列的增加或删除的变更。

说明

- 如果在遇到以上不支持的Schema变更，则需要您手动删除下游目标表，重新启动CTAS作业，即重新创建目标表并重新同步历史数据。
- CTAS不会去识别具体的DDL类型，而是对比前后两条数据的Schema差异。因此，如果您先删除了某列后，又加回了该列，且这两个DDL之间无数据变化，那么CTAS会认为没有发生结构变更。同理，如果您添加了一列，直到该表有数据变化，CTAS才会感知到结构变更，才会同步结构变更到目标表。

前提条件

执行CTAS语法前，需要在您的工作空间中已注册目标端的Catalog。详情请参见管理Hive Metastore、管理Hologres Catalog或管理MySQL Catalog。

使用限制

- 仅Flink计算引擎vvr-4.0.11-flink-1.13及以上版本支持CTAS语法。
- 仅Flink计算引擎vvr-4.0.12-flink-1.13及以上版本支持同步自定义计算列。
- 目标端的Catalog仅支持Hologres Catalog。
- CTAS不支持进行作业调试，详情请参见作业调试。
- 目前支持同步表结构变更的数据源有MySQL的CDC源表和消息队列Kafka源表。

基本语法

```
CREATE TABLE IF NOT EXISTS <sink_table>
[
  ([CONSTRAINT constraint_name] PRIMARY KEY (column_name, ...) NOT ENFORCED)
]
[COMMENT table_comment]
[PARTITIONED BY (partition_column_name1, partition_column_name2, ...)]
WITH (key1=val1, key2=val2, ...)
AS TABLE <source_table> [/*+ OPTIONS(key1=val1, key2=val2, ... ) */]
[ADD COLUMN { <column_component> | (<column_component> [, ...])}];
<sink_table>:
  [catalog_name.][db_name.]table_name
<source_table>:
  [catalog_name.][db_name.]table_name
<column_component>:
  computed_column_definition [FIRST | AFTER column_name]
<computed_column_definition>:
  column_name AS computed_column_expression [COMMENT column_comment]
```

CTAS语法复用了CREATE TABLE语法的基本结构，其中的参数解释如下表所示。

参数	说明
sink_table	数据同步的目标表名，可以指定具体的Catalog名称和数据库名称。
PRIMARY KEY	目标表的主键。
COMMENT	目标表的描述，默认使用source_table的描述。

参数	说明
PARTITIONED BY	目标表的分区键。
WITH	目标表的参数，详情请参见每个结果表Connector文档的WITH参数。Connector文档请参见数据结果表。 说明 key和value都需要为字符串类型，例如'jdbcWriteBatchSize' = '1024'。
source_table	数据同步的源表表名，可指定具体的Catalog名称和Database名称。
OPTIONS	源表的参数，详情请参见每个源表Connector文档的WITH参数。Connector文档请参见数据源表。 说明 key和value都需要为字符串类型，例如'server-id' = '65500'。
ADD COLUMN	在源表上新增列，仅支持计算列。
column_component	新增列的描述。
computed_column_definition	计算列表达式的描述。
FIRST	新增列作为源表的第一个字段。如果不添加该参数，则新增列会默认作为源表的最后一个字段。
AFTER	新增列放在源表指定字段后面。

说明 因为IF NOT EXISTS关键字为必填，所以如果目标表在目标存储中并不存在，则会先创建该目标表，否则跳过创建步骤。创建的目标表Schema会使用源表的Schema，包括主键以及物理字段的字段名和字段类型，不包括计算列、meta字段、Watermark。其中源表到目标表的字段类型会经过类型映射，详见各个Connector页面的类型映射表。

代码示例

- 示例一：单表同步

  通常，CTAS都会配合数据源的Catalog和目标的Catalog一起使用，例如MySQL Catalog和Hologres Catalog结合CTAS语法，来完成MySQL到Hologres的全量和增量数据同步。使用MySQL Catalog可以自动解析源表的Schema及相应的参数，而不用手动编写DDL 。

  假设我们已经在工作空间中注册了名为holo的Hologres Catalog和名为mysql的MySQL Catalog。MySQL中的web_sales表同步到Hologres中，代码示例如下。

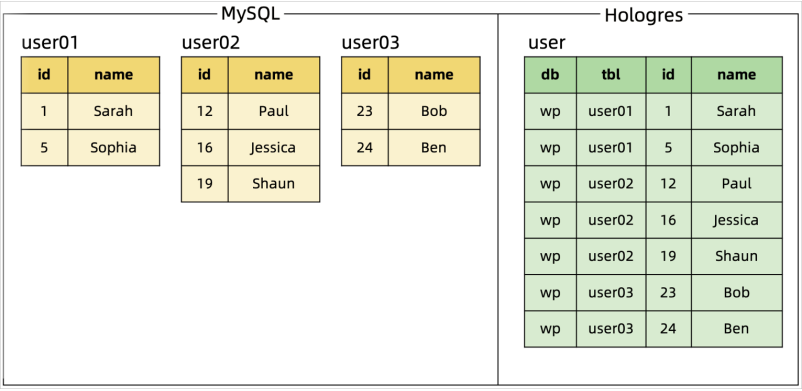
```
USE CATALOG holo;
CREATE TABLE IF NOT EXISTS web_sales
WITH ('jdbcWriteBatchSize' = '1024')    -- 可选，指定目标表的参数。
AS TABLE mysql.tpcds.web_sales
/*+ OPTIONS('server-id'='8001-8004') */;  -- 指定mysql-cdc源表的额外参数。
```

- 示例二：分库分表合并同步

  对于分库分表合并同步的场景，您可以结合MySQL Catalog，利用正则表达式的表名和库名来匹配所要同步的多张表。使用CTAS可以将这多张分库分表合并到一张Hologres表中，库名和表名会作为额外的两个字段写入到该表中，为保证主键唯一性，库名、表名和原主键一起作为该Hologres表的新联合主键。

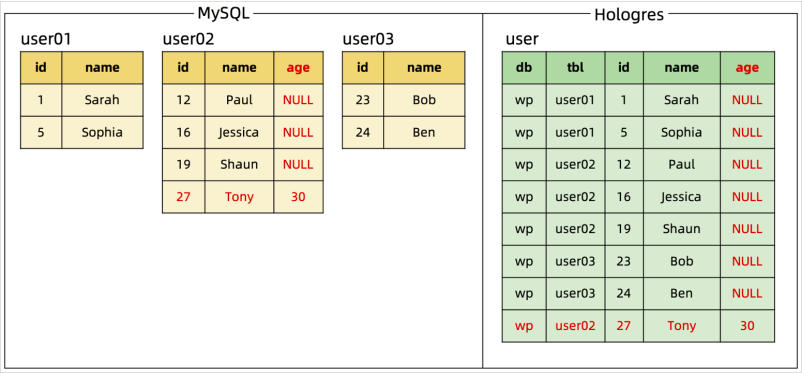
```
USE CATALOG holo;
CREATE TABLE IF NOT EXISTS user
WITH ('jdbcWriteBatchSize' = '1024')
AS TABLE mysql.`wp.*`.`user[0-9]+`
/*+ OPTIONS('server-id'='8001-8004') */;
```

  其合并的效果如下图所示。



  如果在user02新增一列age，并插入一条数据。此时虽然多张分表的Schema并不一致，但是user02上后续的数据和Schema变更都能实时地自动同步到下游表中。

```
ALTER TABLE `user02` ADD COLUMN `age` INT;
INSERT INTO `user02` (id, name, age) VALUES (27, 'Tony', 30);
```

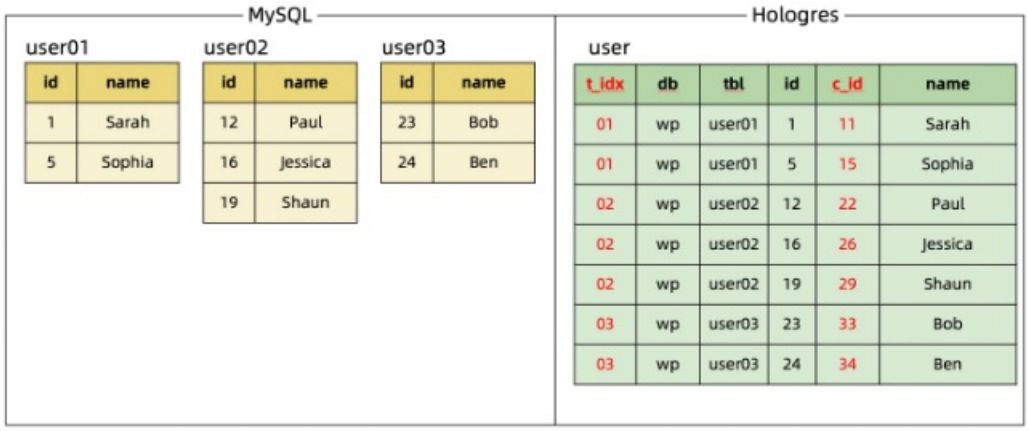


- 示例三：自定义计算列同步

  本示例以user分库分表合并同步作为基础，为您介绍在分库分表合并的过程中，如何进行一些转换计算。

```
USE CATALOG holo;
CREATE TABLE IF NOT EXISTS user (
  PRIMARY KEY (`t_idx`, `db`, `tbl`, `id`) NOT ENFORCED
) WITH ('jdbcWriteBatchSize' = '1024')
AS TABLE mysql.`wp.*`.`user[0-9]+`
/*+ OPTIONS('server-id'='8001-8004') */
ADD COLUMN (
  `t_idx` AS COALESCE(SPLIT_INDEX(`tbl`, 'r', 1), 'default') FIRST,
  `c_id` AS `id` + 10 AFTER `id`
);
```

新增计算列同步的效果如下图所示。



- 示例四：多CTAS语句

  阿里云Flink也支持使用STATEMENT SET语法将多个CTAS语句作为一个作业一起提交，并且 Flink还能对Source进行优化，复用一个Source节点读取多业务表的数据。这对于MySQL CDC数据源场景尤为适用，因为这可以减少server-id的使用，减少对数据库的连接数和读取压力。

  例如示例一同步了web_sales表，示例二同步了user分库分表，您可以使用STATEMENT SET语法将它们作为一个作业提交。一个作业便能完成多个业务表的同步，一个Source便能读取多个业务表数据，代码示例如下。

```
USE CATALOG holo;
BEGIN STATEMENT SET;
-- 同步web_sales表。
CREATE TABLE IF NOT EXISTS web_sales
AS TABLE mysql.tpcds.web_sales
/*+ OPTIONS('server-id'='8001-8004') */;
-- 同步user分库分表。
CREATE TABLE IF NOT EXISTS user
AS TABLE mysql.`wp.*`.`user[0-9]+`
/*+ OPTIONS('server-id'='8001-8004') */;
END;
```

9.2.5.3. CREATE DATABASE AS(CDAS)语句

CDAS支持整库级别的表结构和数据的实时同步，此外还支持表结构变更的同步。本文为您介绍CREATE DATABASE AS(CDAS)的背景信息、前提条件、使用限制、基本语法和示例。

背景信息

CDAS是CTAS语法的一个语法糖，以实现整库同步、多表同步的功能。阿里云Flink引擎会将CDAS语句为每个需要同步的表翻译成一个对应的CTAS语句。因此CDAS还拥有CTAS的数据同步，以及表结构变更同步的能力，常用于全自动化的数据集成场景，并且阿里云Flink还能对Source进行优化，复用一个Source节点读取多业务表的数据，这对于MySQL CDC数据源场景尤为适用。因为这不仅可以减少数据库的连接数，还能避免重复拉取Binlog数据，以降低数据库的读取压力。关于CDAS相关信息如下所示：

- 功能特性

功能	详情
整库同步	支持实时同步整库（或者多张表）的全量和增量数据到每张对应的目标表中。
表结构变更同步	在实时同步整库数据的同时，还支持将每张源表的表结构变更（加列等）实时同步到对应的目标表中。
多CDAS&CTAS语句	支持使用STATEMENT SET语法将多个CDAS和CTAS语句作为一个作业一起提交，并支持对Source节点的合并复用，降低对数据源的压力。

- 启动流程

  当执行CDAS语句时，阿里云Flink将会按照以下流程执行：

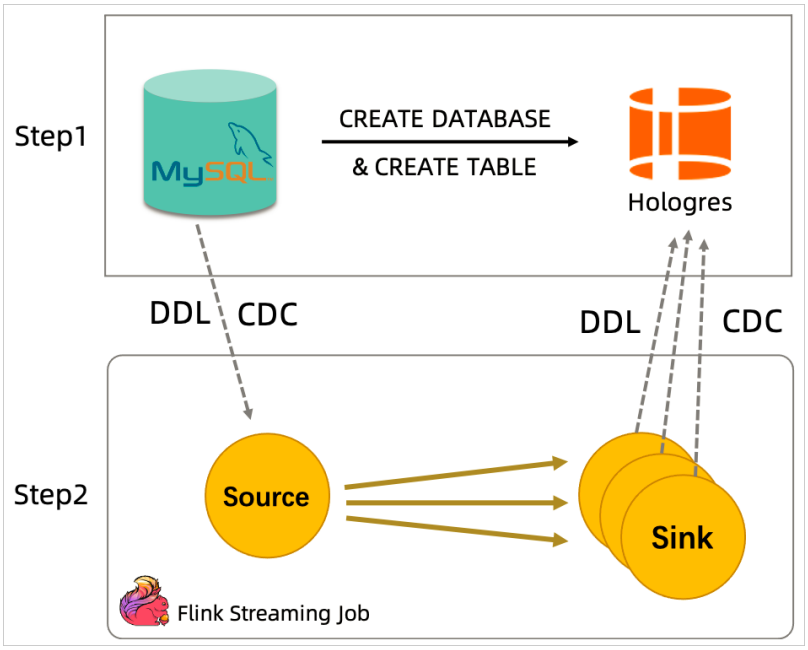
  i. 检查目标存储中是否存在目标库和目标表。

     如果不存在目标库，则通过目标端Catalog去目标存储中创建相应的目标库；如果存在，则跳过建库，并检查目标库是否存在该目标表。如果不存在，则在目标库中创建相应的目标表，该目标表具有和源库中表相同的表名和Schema。如果存在，则跳过建表。

  ii. 提交和启动相应的数据同步作业。

     将源库中的数据以及Schema变更同步到目标库下的表中。

  例如从MySQL到Hologres的CDAS数据同步流程如下图所示。



- 表结构变更同步策略

  因为CDAS是CTAS语法的一个语法糖，所以表结构变更能力与CTAS一致，详细的表结构变更同步策略请参见CREATE TABLE AS(CTAS)语句。

前提条件

执行CDAS语法前，需要在您的工作空间中已注册目标端的Catalog。详情请参见管理Hive Metastore、管理Hologres Catalog或管理MySQL Catalog。

使用限制

- 仅Flink计算引擎vvr-4.0.11-flink-1.13及以上版本支持CDAS语法。
- 目标端的Catalog仅支持Hologres Catalog。
- CDAS不支持作业调试，详情请参见作业调试。

基本语法

```
CREATE DATABASE IF NOT EXISTS <target_database>
[COMMENT database_comment]
[WITH (key1=val1, key2=val2, ...)]
AS DATABASE <source_database>
INCLUDING { ALL TABLES | TABLE 'table_name' }
[EXCLUDING TABLE 'table_name']
[/*+ OPTIONS(key1=val1, key2=val2, ... ) */]
<target_database>:
  [catalog_name.]db_name
<source_database>:
  [catalog_name.]db_name
```

CDAS语法复用了CREATE DATABASE语法的基本结构，其中的参数解释如下表所示。

参数	说明
target_database	数据同步的目标数据库名，可以指定具体的Catalog名称。
COMMENT	目标库的描述，默认使用source_database的描述。
WITH	目标库的参数，详情请参见对应的Catalog文档。例如管理Hologres Catalog。 **说明** key和value都需要为字符串类型，例如'sink.parallelism' = '4'。
source_database	数据同步的源库名称，可以指定具体的Catalog名称。
INCLUDING ALL TABLES	同步源库中的所有表。
INCLUDING TABLE	同步源库中指定的表。支持使用竖线（|）分隔指定多个表，也可以使用正则表达式指定符合某一规则的表。例如INCLUDING TABLE 'web.*'表示要同步源库中所有web开头的表。
EXCLUDING TABLE	用于指定不需要同步的表，支持使用竖线（|）分隔指定多个表，也可以使用正则表达式指定符合某一规则的表，例如INCLUDING ALL TABLES EXCLUDING TABLE 'web.*'表示同步源库中所有不是web开头的表。

参数	说明
OPTIONS	源表的参数，详情请参见每个源表Connector文档的WITH参数。Connector文档请参见数据源表。 说明 key和value都需要为字符串类型，例如'server-id' = '65500'。

说明 因为IF NOT EXISTS关键字为必填，所以如果目标库或目标表在目标存储中并不存在，则会先创建该目标库和目标表，否则跳过创建步骤。创建的目标表Schema会使用源表的Schema，包括主键以及物理字段的字段名和字段类型，不包括计算列、meta字段、Watermark。其中源表到目标表的字段类型会经过类型映射，详见各个Connector页面的类型映射表。

示例

- 示例一：整库同步

  通常，CDAS都会配合数据源的Catalog和目标的Catalog一起使用，例如MySQL Catalog和Hologres Catalog结合CDAS语法，完成 MySQL到Hologres的全量和增量数据同步。使用MySQL Catalog可以自动解析源表的Schema及相应的参数，而不用手动编写DDL。

  假设我们已在工作空间中注册了名为holo的Hologres Catalog和名为mysql的MySQL Catalog，MySQL中有一个tpcds的库。您可以使用以下语句将tpcds库下的24张表全部同步到Hologres中，包括未来的数据变更和表结构变更，无需提前在Hologres中创建表。

```
USE CATALOG holo;
CREATE IF NOT EXISTS DATABASE holo_tpcds  -- 在hologres中创建holo_tpcds库。
WITH ('sink.parallelism' = '4') -- 可选，指定目标库的参数，每个holo sink默认使用4并发。
AS DATABASE mysql.tpcds INCLUDING ALL TABLES  -- 同步mysql中tpcds库下所有表。
/*+ OPTIONS('server-id'='8001-8004') */ ; -- 可选，指定mysql-cdc源表的额外参数。
```

  说明 Hologres支持在创建目标Database时指定WITH参数，这些参数仅对当前作业生效，用于控制写入目标表时的行为，不会持久化到Hologres中。目前支持的参数类型及其具体作用详情请参见交互式分析Hologres结果表。

- 示例二：多CDAS&CTAS语句

  阿里云Flink也支持使用STATEMENT SET语法将多个CDAS和CTAS语句作为一个作业一起提交，并且阿里云Flink还能对Source进行优化，复用一个Source节点读取多业务表的数据，这对于MySQL CDC数据源场景尤为适用，因为这可以减少server-id的使用，从而减少对数据库的连接数和读取压力。

  假设MySQL实例中有tpcds、tpch、user_db01~user_db99（分库分表）多个库。您可以通过组合多条CDAS和CTAS语句，将MySQL实例下的所有库和表都同步到Hologres，只需一个Flink作业便能完成所有表的同步，只需一个Source便能读取所有表的数据，代码示例如下。

```
USE CATALOG holo;
BEGIN STATEMENT SET;
-- 同步user分库分表。
CREATE TABLE IF NOT EXISTS user
AS TABLE mysql.`user_db[0-9]+`.`user[0-9]+`
/*+ OPTIONS('server-id'='8001-8004') */;
-- 同步TPCDS库。
CREATE IF NOT EXISTS DATABASE holo_tpcds
AS DATABASE mysql.tpcds INCLUDING ALL TABLES
/*+ OPTIONS('server-id'='8001-8004') */ ;
-- 同步TPCH库。
CREATE IF NOT EXISTS DATABASE holo_tpch
AS DATABASE mysql.tpch INCLUDING ALL TABLES
/*+ OPTIONS('server-id'='8001-8004') */ ;
END;
```

9.2.5.4. AUTO OPTIMIZE语句

通过AUTO OPTIMIZE语句，将启动一个流式优化任务，自动地对外部数据湖的表进行优化。本文为您介绍AUTO OPTIMIZE语句的背景信息、前提条件、使用限制、注意事项、基本语法和参数配置。

背景信息

- 功能详情

  目前仅支持将Iceberg表的若干小文件重写为大文件，将小文件重写成大文件有利于减轻元数据的负载和提高查询效率。启动该优化任务后，该任务将定时检查指定的Iceberg表。如果表分区下存在多个小文件，则会将这些小文件重写成为大文件。当您的Iceberg表的读性能瓶颈是因为存在大量小文件时，建议启动该优化任务以提高Iceberg的读性能。

- 功能特性

功能	详情
优化单个表	对指定的表进行优化。
优化单个数据库	对指定的数据库下的所有表进行优化。

- 优化流程

  当执行AUTO OPTIMIZE语句时，阿里云Flink将会按照以下流程执行：

  i. 检查要优化的表或者数据库是否存在，如果不存在则会报错。

  ii. 持续监控对应的表，如果表满足对应的优化条件，则会对表进行优化。

前提条件

执行AUTO OPTIMIZE语句来优化表或者数据库前，需要保证在Flink全托管开发控制台上已经注册了对应的表或者数据库。注册方式详情请参见数据湖Iceberg结果表。

使用限制

- 仅Flink计算引擎vvr-4.0.12-flink-1.13及以上版本支持AUTO OPTIMIZE语句。
- 目前仅支持将Iceberg表的若干小文件重写为大文件。

- 仅支持优化包含了Iceberg表的数据库。如果数据库中除了Iceberg表，还包含了其他类型的表，则不支持优化该数据库。

  说明 您可以将这些需要优化的Iceberg表都注册到一个新的数据库中，然后优化该新的数据库。

注意事项

启动AUTO OPTIMIZE优化任务后，将会占用一定的网络带宽和Flink全托管的CU资源，但不会删除被重写的小文件，需要配合Iceberg的SnapShot过期机制来删除这些被重写的小文件。

基本语法

- 优化单个表

```
AUTO OPTIMIZE TABLE <target_table>
<target_table>:
  [catalog_name.][db_name.]table_name
```

- 优化单个数据库

```
AUTO OPTIMIZE DATABASE <target_database>
<target_database>:
  [catalog_name.]db_name
```

参数配置

- 配置方式

  您可以使用如下命令来设置相关参数：

```
ALTER TABLE <target_table> SET ('auto-optimize.rewrite.enable' = 'false');
<target_table>:
  [catalog_name.][db_name.]table_name
```

  说明 对于基于内存的Catalog下的Iceberg表，不支持直接在Flink开发控制台通过ALTER命令来修改表参数。但其他类型的Catalog下的表都可以修改表参数。

- 参数说明

类别	参数	说明	单位和默认值
表监控相关参数	auto-optimize.table-monitor.interval	每隔多长时间对表进行一次监控来判断是不是应该对表进行优化。 您可以根据您的期望，设置该参数值的大小。建议如下： ○ 如果您期望尽早对表进行优化，可以将该值设置得小一点。 ○ 如果您期望减少表优化的频率，可以将该值设置得大一点。	○ 单位是毫秒。 ○ 默认值是600000，即10分钟。

类别	参数	说明	单位和默认值
表优化相关参数	auto-optimize.rewrite.enable	是否开启文件重写的功能。参数取值如下： ○ true（默认值）：开启文件重写的功能。 ○ false：不开启文件重写的功能。 如果您需要优化单个数据库，但不希望优化数据库中的某些表，可以将这些表的该参数设置为false。	不涉及
	auto-optimize.rewrite.target-file-size-bytes	将文件重写至多大size的文件。	○ 单位是byte。 ○ 默认值为536870912，即512 MB。

9.2.6. DQL数据查询语句

本文为您介绍Flink提供的QUERY语句。

Flink V1.13 QUERY语句详情，请参见Query语句。

9.2.7. 窗口函数

9.2.7.1. 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

Flink SQL窗口函数支持基于无限大窗口的聚合（无需在SQL Query中，显式定义任何窗口）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实时计算。

Flink SQL支持的窗口聚合主要是两种：Window聚合和Over聚合。本文档主要为您介绍Window聚合。Window聚合支持Event Time和Processing Time两种时间属性定义窗口。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）。

> **说明** 滚动窗口、滑动窗口和会话窗口不能与last_value、first_value、TopN函数合用，会造成数据乱序被丢弃导致结果数据异常。

时间属性

Flink SQL支持Event Time和Processing Time两种时间属性，时间属性详情请参见时间属性。Flink可以基于这两种时间属性对数据进行窗口聚合。基于这两种时间属性开窗的区别如下：

- Event Time：您提供的事件时间，通常是数据的最原始的创建时间。

系统会根据数据的Event Time生成的Watermark来进行关窗。只有当Watermark大于关窗时间，才会触发窗口的结束，窗口结束才会输出结果。如果一直没有触发窗口结束的数据流入Flink，则该窗口就无法输出数据。单并发的Watermark是递增的，多并发或者多个源表的Watermark取最小值。

> **注意**
> - 如果源表中存在一条未来的乱序数据或者某个并发没有数据，则可能会无法触发窗口结束，从而导致结果数据异常。因此您需要根据数据乱序的程度设置合理的offset大小，并保证所有并发都有数据。如果某个源表并发或源表上游partition因没有数据导致窗口始终无法被触发，则可以在更多Flink配置中添加 `table.exec.source.idle-timeout: 10s` 来触发窗口结束。该参数含义详情请参见Configuration。
> - 数据经过GroupBy、双流JOIN或OVER窗口节点后，会导致Watermark属性丢失，无法再使用Event Time进行开窗。

- Processing Time：对事件进行处理的本地系统时间。

  Processing Time是Flink系统产生的，不在用户的原始数据中。因此需要您显示定义一个Processing Time列。

> **说明** 因为Processing Time容易受到事件到达Flink系统的速度及Flink内部处理数据顺序的影响，所以每次回溯数据的结果可能不一致。

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` 。

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，可以满足您的多维度开窗需求。

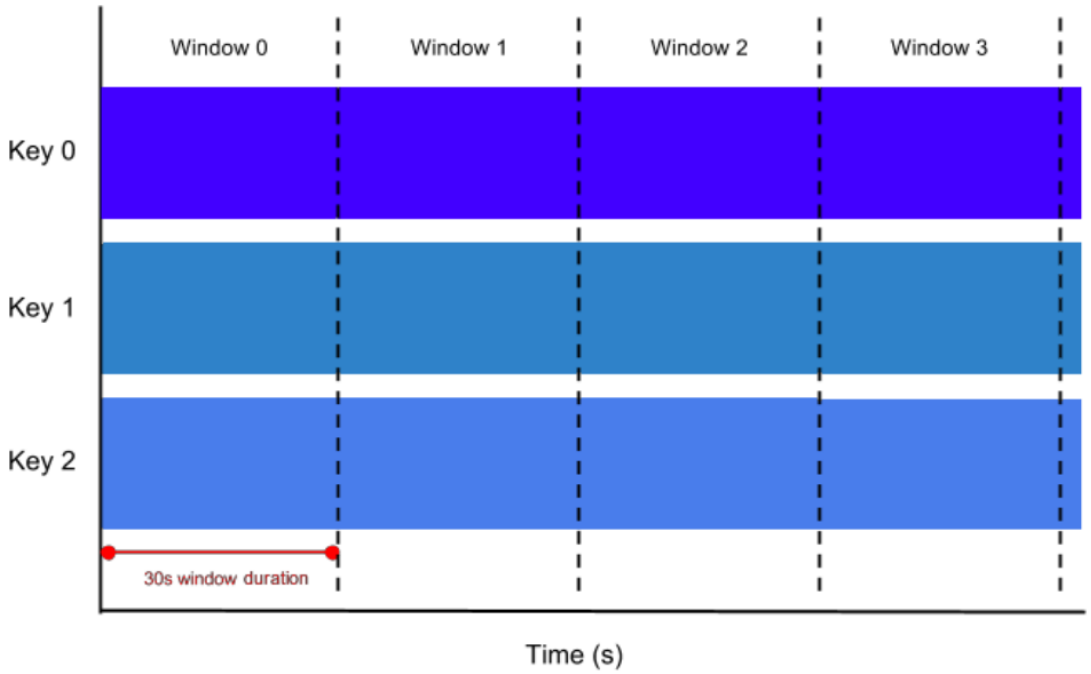
```
CREATE TEMPORARY TABLE user_clicks(
  username varchar,
  click_url varchar,
  eventtime varchar,
  ts AS TO_TIMESTAMP(eventtime),
  WATERMARK FOR ts AS ts - INTERVAL '2' SECOND  --为Rowtime定义Watermark。
) with (
  'connector'='sls',
  ...
);
CREATE TEMPORARY TABLE tumble_output(
  window_start TIMESTAMP,
  window_end TIMESTAMP,
  username VARCHAR,
  clicks BIGINT
) with (
  'connector'='datahub'       --目前SLS只支持输出VARCHAR类型的DDL，所以使用DataHub存储。
  ...
);
CREATE TEMPORARY VIEW one_minute_window_output AS
SELECT
  TUMBLE_ROWTIME(ts, INTERVAL '1' MINUTE) as rowtime,  --使用TUMBLE_ROWTIME作为二级Window的聚合时间。
  username,
  COUNT(click_url) as cnt
FROM user_clicks
GROUP BY TUMBLE(ts, INTERVAL '1' MINUTE),username;
BEGIN statement set;
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;
END;
```

9.2.7.2. 滚动窗口

本文为您介绍如何使用Flink滚动窗口函数。

定义

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为 `[0:00, 0:05)` 、 `[0:05, 0:10)` 、 `[0:10, 0:15)` 等窗口。下图展示了一个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>` 参数必须是时间流中的一个合法的时间属性字段，指定为Processing Time或Event Time，请参见概述，了解如何定义时间属性。

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）。例如 `[00:10,00:15)` 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）。例如 `[00:00, 00:15]` 窗口，返回 `00:15` 。

窗口标识函数	返回类型	描述
`TUMBLE_ROWTIME(time-attr, size-interval)`	TIMESTAMP(rowtime-attr)	返回窗口的结束时间（不包含边界）。例如 `(00:00, 00:15)` 窗口，返回 `00:14:59.999` 。返回值是一个rowtime attribute，即可以基于该字段进行时间属性的操作，例如，级联窗口只能用在基于Event Time的Window上，详情请参见级联窗口。
`TUMBLE_PROCTIME(time-attr, size-interval)`	TIMESTAMP(rowtime-attr)	返回窗口的结束时间（不包含边界）。例如 `(00:00, 00:15)` 窗口，返回 `00:14:59.999` 。返回值是一个Proctime Attribute，即可以基于该字段进行时间属性的操作。例如，级联窗口只能用在基于Processing Time的Window上，详情请参见级联窗口。

使用Event Time统计每个用户每分钟在指定网站的单击数示例

- 测试数据

username（VARCHAR）	click_url（VARCHAR）	eventtime（VARCHAR）
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 TEMPORARY TABLE user_clicks(
  username varchar,
  click_url varchar,
  eventtime varchar,
  ts AS TO_TIMESTAMP(eventtime),
  WATERMARK FOR ts AS ts - INTERVAL '2' SECOND  --为Rowtime定义Watermark。
) with (
  'connector'='sls',
  ...
);
CREATE TEMPORARY TABLE tumble_output(
  window_start TIMESTAMP,
  window_end TIMESTAMP,
  username VARCHAR,
  clicks BIGINT
) with (
  'connector'='datahub'          --目前SLS只支持输出VARCHAR类型的DDL，所以使用DataHub存储。
  ...
);
INSERT INTO tumble_output
SELECT
TUMBLE_START(ts, INTERVAL '1' MINUTE) as window_start,
TUMBLE_END(ts, INTERVAL '1' MINUTE) as window_end,
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

使用Processing Time统计每个用户每分钟在指定网站的单击数示例

- 测试数据

username （VARCHAR）	click_url（VARCHAR）
Jark	*http://taobao.com/xxx*
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*
Timo	*http://taobao.com/xxx*

- 测试语句

```
CREATE TEMPORARY TABLE window_test (
  username   VARCHAR,
  click_url  VARCHAR,
  ts as PROCTIME()
) WITH (
  'connector'='sls',
  ...
);
CREATE TEMPORARY TABLE tumble_output(
  window_start TIMESTAMP,
  window_end TIMESTAMP,
  username VARCHAR,
  clicks BIGINT
) with (
  'connector'='datahub'              --目前SLS只支持输出VARCHAR类型的DDL，所以使用DataHub存储
。
  ...
);
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

9.2.7.3. 滑动窗口

本文为您介绍如何使用Flink滑动窗口函数。

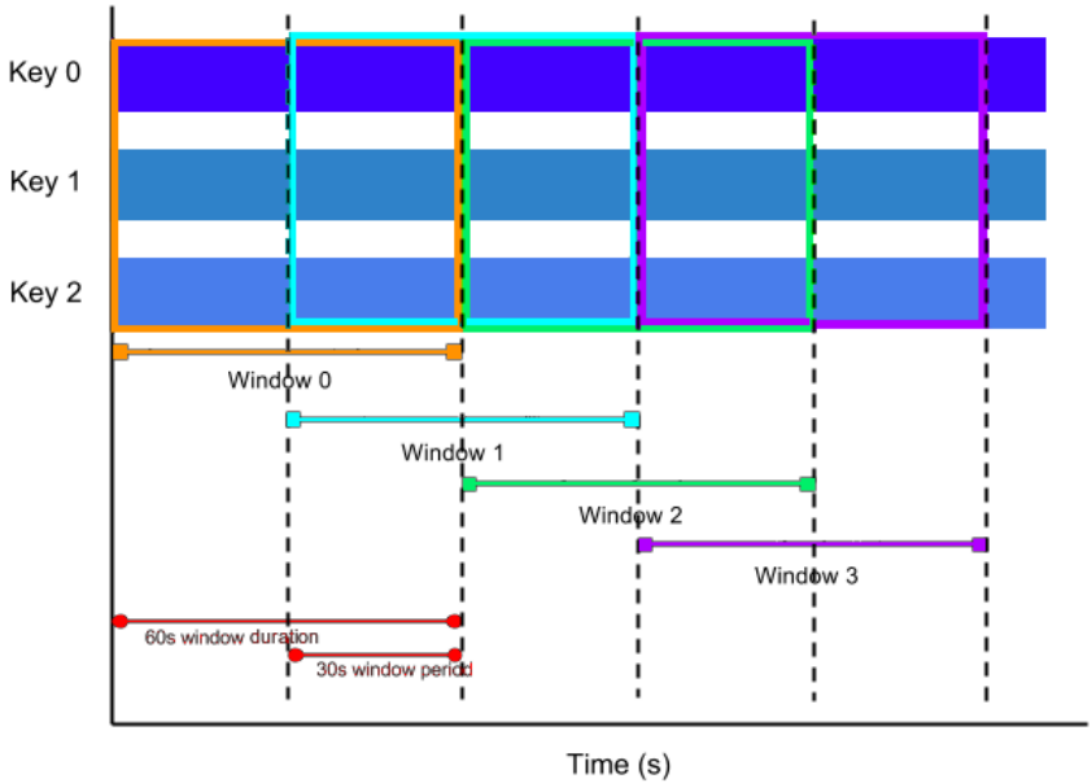
定义

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，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。



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。请参见概述，了解如何定义时间属性。

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）。例如 [00:10, 00:15] 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）。例如 [00:00, 00:15] 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）。例如 (00:00, 00:15) 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类型的操作，例如级联窗口，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）。例如 (00:00, 00:15) 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类型的操作，例如级联窗口，只能用在基于processing time的window上。

示例

统计每个用户过去1分钟的单击次数，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 TEMPORARY TABLE user_clicks (
  username VARCHAR,
  click_url VARCHAR,
  eventtime VARCHAR,
  ts AS TO_TIMESTAMP(eventtime),
  WATERMARK FOR ts AS ts - INTERVAL '2' SECOND  --为Rowtime定义Watermark。
) with (
  'connector'='sls',
  ...
);
CREATE TEMPORARY TABLE hop_output (
  window_start TIMESTAMP,
  window_end TIMESTAMP,
  username VARCHAR,
  clicks BIGINT
) with (
  'connector'='datahub'            --目前SLS只支持输出VARCHAR类型的DDL，所以使用DataHub存储。
  ...
);
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 10:01:00.0	Jark	3
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 10:02:00.0	2017-10-10 10:03:00.0	Timo	1

window_start（TIMESTAMP）	window_end（TIMESTAMP）	username （VARCHAR）	clicks （BIGINT）
`2017-10-10 10:02:30.0`	`2017-10-10 10:03:30.0`	Timo	1

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`

9.2.7.4. 会话窗口

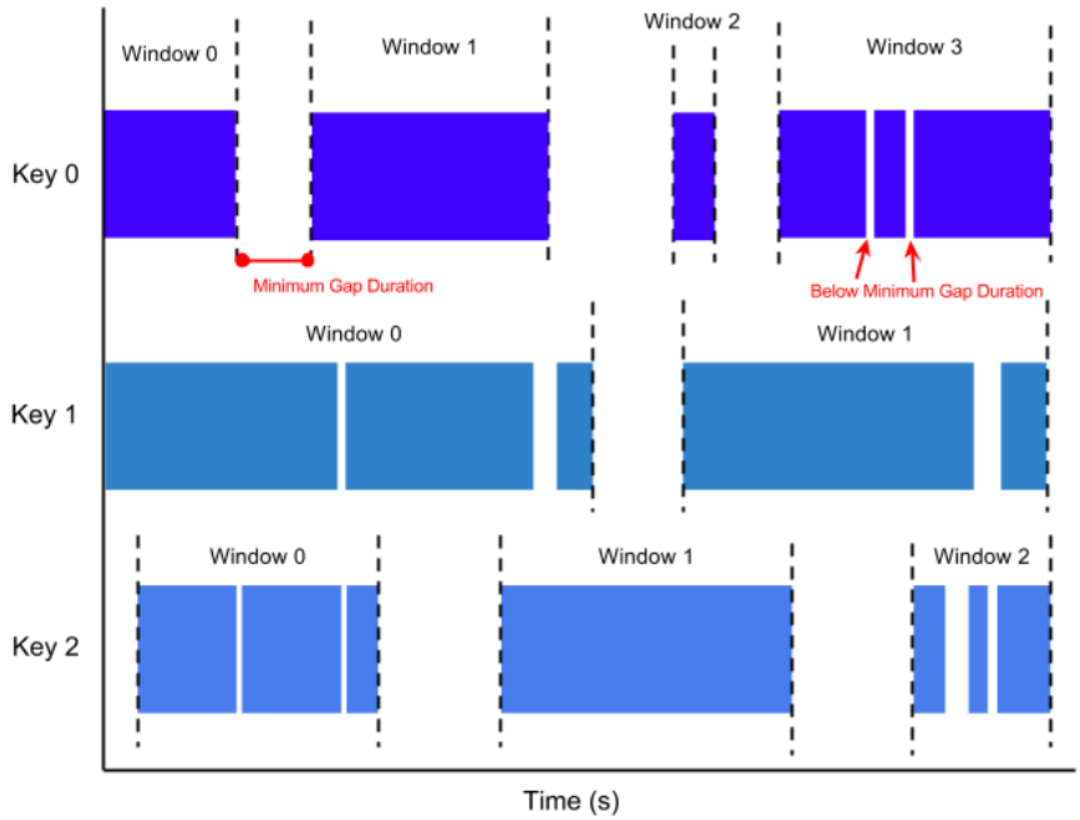
本文为您介绍如何使用Flink会话窗口函数。

定义

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，该窗口就会关闭。

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，一个表示鼠标单击活动的数据流可能具有长时间的空闲时间，并在两段空闲之间散布着高浓度的单击。如果数据在指定的间隔（Gap）之后到达，则会开始一个新的窗口。

会话窗口示例如下图，每个Key由于不同的数据分布，形成了不同的Window。



函数语法

SESSION函数用于在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>` 参数必须是数据流中的一个合法的时间属性字段，指定为Processing Time或Event Time，详情请参见概述，了解如何定义时间属性。

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）。例如 `[00:10,00:15]` 的窗口，返回 `00:10` ，即为此会话窗口内第一条记录的时间。

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）。例如 [00:00,00:15] 的窗口，返回 00:15 ，即为此会话窗口内最后一条记录的时间+ <gap-interval> 。
SESSION_ROWTIME (<time-attr>, <gap-interval>)	Timestamp（rowtime-attr）	返回窗口的结束时间（不包含边界）。例如 (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进行时间类型的操作，例如级联窗口。该参数只能用于基于Event Time的Window。
SESSION_PROCTIME (<time-attr>, <gap-interval>)	Timestamp（rowtime-attr）	返回窗口的结束时间（不包含边界）。例如 (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进行时间类型的操作，例如级联窗口。该参数只能用于基于Processing Time的Window。

示例

统计每个用户在每个活跃会话期间的单击次数，会话超时时长为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 TEMPORARY TABLE user_clicks(
  username varchar,
  click_url varchar,
  eventtime varchar,
  ts AS TO_TIMESTAMP(eventtime),
  WATERMARK FOR ts AS ts - INTERVAL '2' SECOND  --为Rowtime定义Watermark。
) with (
  'connector'='sls',
  ...
);
CREATE TEMPORARY TABLE session_output(
  window_start TIMESTAMP,
  window_end TIMESTAMP,
  username VARCHAR,
  clicks BIGINT
) with (
  'connector'='datahub'            --目前SLS只支持输出VARCHAR类型的DDL，所以使用DataHub存储。
  ...
);
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

9.2.7.5. OVER窗口

OVER窗口（OVER Window）是传统数据库的标准开窗，不同于Group By Window，OVER窗口中每1个元素都对应1个窗口。OVER窗口可以按照实际元素的行或实际的元素值（时间戳值）确定窗口，因此流数据元素可能分布在多个窗口中。

在应用OVER窗口的流式数据中，每1个元素都对应1个OVER窗口。每1个元素都触发1次数据计算，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1行。在实时计算的底层实现中，OVER窗口的数据进行全局统一管理（数据只存储1份），逻辑上为每1个元素维护1个OVER窗口，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;
```

- agg1(col1)：按照GROUP BY指定col1列对输入数据进行聚合计算。
- OVER (definition1)：OVER窗口定义。
- AS colName：别名。

> **说明**
> - agg1到aggN所对应的OVER definition1必须相同。
> - 外层SQL可以通过AS的别名查询数据。

类型

Flink SQL中对OVER窗口的定义遵循标准SQL的定义语法，传统OVER窗口没有对其进行更细粒度的窗口类型命名划分。按照计算行的定义方式，OVER Window可以分为以下两类：

- ROWS OVER Window：每1行元素都被视为新的计算行，即每1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

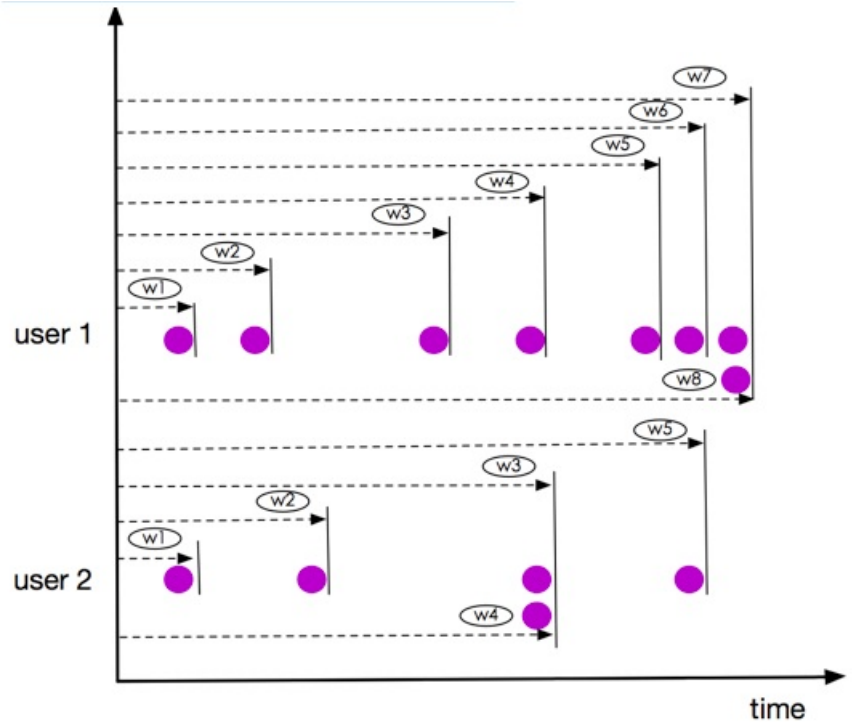
正交属性	说明	proctime	eventtime
ROWS OVER Window	按照实际元素的行确定窗口。	支持	支持
RANGE OVER Window	按照实际的元素值（时间戳值）确定窗口。	支持	支持

Rows OVER Window语义

- 窗口数据

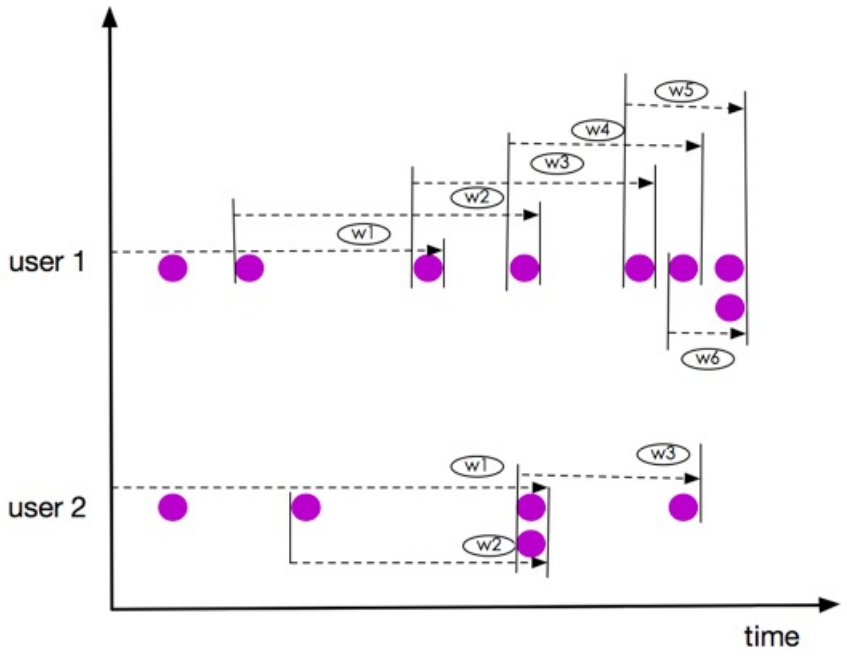
  ROWS OVER Window的每个元素都确定一个窗口。ROWS OVER Window分为Unbounded（无界流）和Bounded（有界流）两种情况。

Unbounded ROWS OVER Window数据示例如下图所示。



> **说明** 虽然上图所示窗口user1的w7、w8及user2的窗口w3、w4都是同一时刻到达，但它们仍然在不同的窗口，这一点与RANGE OVER Window不同。

Bounded ROWS OVER Window数据以3个元素（往前2个元素）的窗口为例，如下图所示。



> **说明** 虽然上图所示窗口user1的w5、w6及user2的窗口w1、w2都是同一时刻到达，但它们仍然在不同的窗口，这一点与RANGE OVER Window不同。

- 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：元素排序的时间字段。
  - rowCount：定义根据当前行开始向前追溯几行元素。

- 案例

  以Bounded ROWS OVER Window场景为例。假设，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。要求输出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 TEMPORARY TABLE tmall_item(
  itemID VARCHAR,
  itemType VARCHAR,
  eventtime varchar,
  onSellTime AS TO_TIMESTAMP(eventtime),
  price DOUBLE,
  WATERMARK FOR onSellTime AS onSellTime - INTERVAL '0' SECOND  --为Rowtime定义Watermark
。
) WITH (
  'connector' = '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

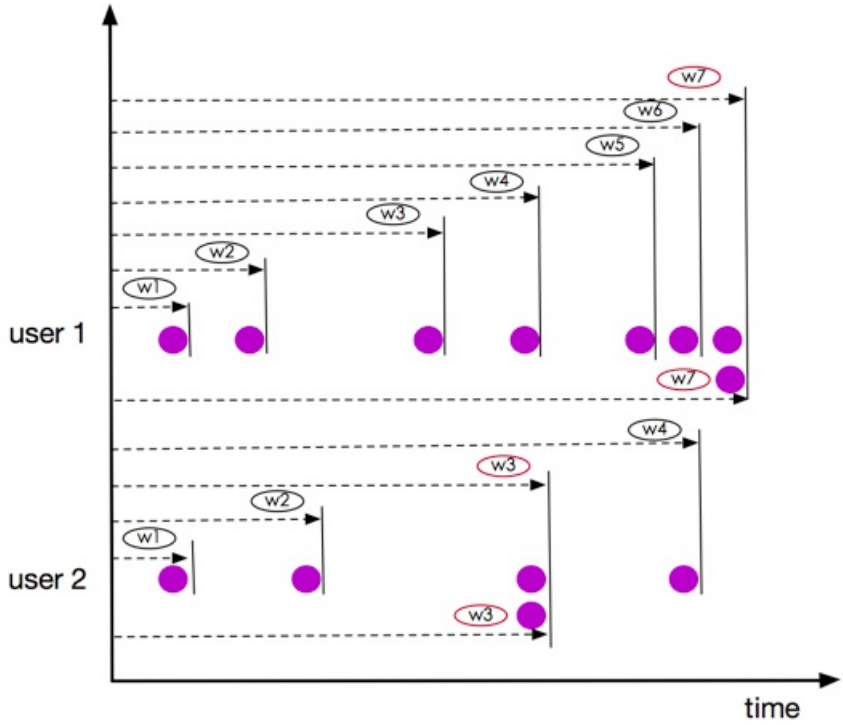
itemID	itemType	onSellTime	price	maxPrice
ITEM001	Electronic	2017-11-11 10:01:00	20	20
ITEM002	Electronic	2017-11-11 10:02:00	50	50
ITEM003	Electronic	2017-11-11 10:03:00	30	50
ITEM004	Electronic	2017-11-11 10:03:00	60	60
ITEM005	Electronic	2017-11-11 10:05:00	40	60
ITEM006	Electronic	2017-11-11 10:06:00	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

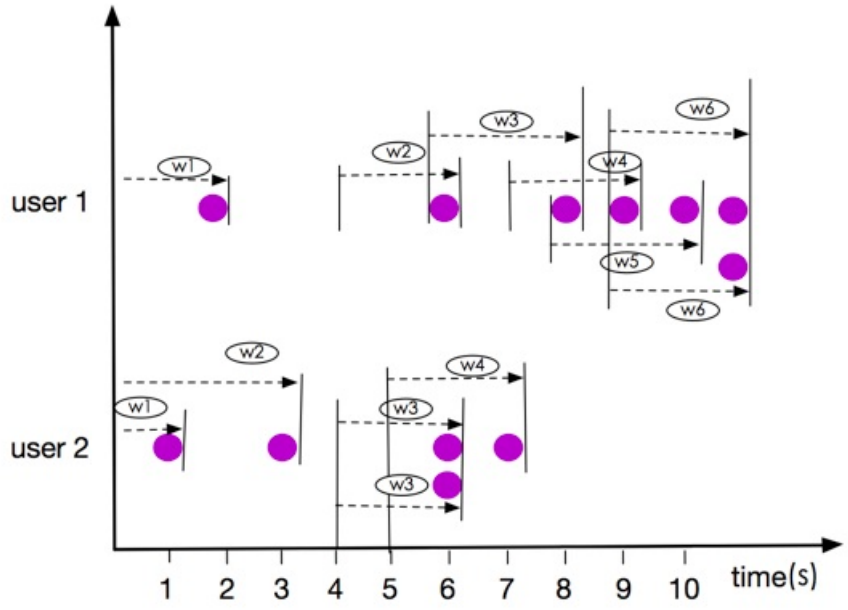
  RANGE OVER Window所有具有共同元素值（元素时间戳）的元素行确定一个窗口，RANGE OVER Window分为Unbounded和Bounded的两种情况。

  Unbounded RANGE OVER Window数据示例如下图所示。



> **说明** 上图所示窗口user1的w7、user2的窗口w3，两个元素同一时刻到达，属于相同的window，这一点与ROWS OVER Window不同。

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，属于相同的window，这一点与ROWS OVER Window不同。

- 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：元素排序的时间字段。
  - timeInterval：定义根据当前行开始向前追溯指定时间的元素行。

- 案例

  Bounded RANGE OVER Window场景示例：假设一张商品上架表，包含有商品ID、商品类型、商品上架时间、商品价格数据。需要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 TEMPORARY TABLE tmall_item(
  itemID VARCHAR,
  itemType VARCHAR,
  eventtime varchar,
  onSellTime AS TO_TIMESTAMP(eventtime),
  price DOUBLE,
  WATERMARK FOR onSellTime AS onSellTime - INTERVAL '0' SECOND  --为Rowtime定义Watermark
。
) WITH (
  'connector' = '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

9.2.8. 内置函数

9.2.8.1. 概述

本文为您介绍Flink提供的用于数据转换的内置函数。

- Flink V1.11：请参见Flink V1.11内置函数。
- Flink V1.12：请参见Flink V1.12内置函数。
- Flink V1.13：请参见Flink V1.13内置函数。

> **说明**
>
> - Flink全托管VVR 3.0.0版本在Flink V1.12的基础上增加了部分内置函数，详情请参见标量函数、表值函数、聚合函数和机器学习函数。
> - 如果内置函数中不存在您所需的函数，您可以通过自定义函数实现，自定义函数详情请参见自定义函数概述。

9.2.8.2. 标量函数

9.2.8.2.1. BITAND

本文为您介绍如何使用BIT AND函数进行按位与操作。

> **注意** 本文仅适用于VVP 2.4.0且VVR 3.0.0及以上版本。

语法

```
INT BITAND(INT number1,INT number2)
```

> **说明** 输入和输出类型均为整型，且类型一致。

入参

参数	数据类型
number1	INT
number2	INT

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

9.2.8.2.2. BITNOT

本文为您介绍如何使用BIT NOT函数进行按位取反。

> **注意** 本文仅适用于VVP 2.4.0且VVR 3.0.0及以上版本。

语法

```
INT BITNOT(INT number)
```

说明 输入和输出类型均为整型，且类型一致。

入参

参数	数据类型
number	INT

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)
-8

9.2.8.2.3. BITOR

本文为您介绍如何使用BITOR函数进行按位取或。

注意 本文仅适用于VVP 2.4.0且VVR 3.0.0及以上版本。

语法

```
INT BITOR(INT number1, INT number2)
```

入参

参数	数据类型
number1	INT
number2	INT

说明 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

9.2.8.2.4. BITXOR

本文为您介绍如何使用BITXOR函数进行按位取异或。

> **注意** 本文仅适用于VVP 2.4.0且VVR 3.0.0及以上版本。

语法

```
INT BITXOR(INT number1, INT number2)
```

入参

参数	数据类型
number1	INT
number2	INT

> **说明** 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

9.2.8.2.5. CONV

本文为您介绍如何使用CONV函数，对数字或字符类型的数值进行进制转换。

> **注意**
> - 本文仅适用于VVP 2.4.0且VVR 3.0.0及以上版本。
> - CONV()精度为64位。

语法

```
VARCHAR CONV(BIGINT number, INT FROM_BASE, INT TO_BASE)
or
VARCHAR CONV(VARCHAR number, INT FROM_BASE, INT TO_BASE)
```

入参

参数	数据类型	说明
number	BIGINT、VARCHAR	**说明** 当number为NULL或非法字符时，结果返回为NULL。
FROM_BASE	INT	取值范围[2, 36]。
TO_BASE	INT	可以为正数，取值范围[2, 36]；也可以为负数，取值范围[-36, -2]。

示例

- 测试数据

id(INT)	x(BIGINT)	y (VARCHAR)
1	12L	'12'
2	10L	'10'
3	0L	'test'

id(INT)	x(BIGINT)	y (VARCHAR)
4	NULL	NULL

- 测试语句

```
SELECT  id, conv(x, 10, 16) as var1, conv(y, 10, 2) as var2
FROM T1;
```

- 测试结果

id(INT)	var1(VARCHAR)	var2(VARCHAR)
1	C	1100
2	A	1010
3	0	NULL
4	NULL	NULL

9.2.8.2.6. DATEDIFF

本文为您介绍如何使用DATEDIFF函数，计算两个日期之间的天数差值。

> **注意** 本文仅适用于VVP 2.4.0且VVR 3.0.0及以上版本。

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、VARCHAR
enddate	TIMESTAMP、VARCHAR

> **说明**
> - VARCHAR日期格式：yyyy-MM-dd或yyyy-MM-dd HH:mm:ss。
> - 如果入参中任何一个为NULL或解析错误，则返回为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  DATEDIFF(datetime1, datetime2) as int1,
        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	30	-30	NULL	NULL	NULL	30

9.2.8.2.7. DATE_ADD

本文为您介绍如何使用DATE_ADD函数，返回指定日期增加目标天数后的日期。

> **注意** 本文仅适用于VVP 2.4.0且VVR 3.0.0及以上版本。

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、VARCHAR **说明** VARCHAR类型日期格式为yyyy-MM-dd或yyyy-MM-dd HH:mm:ss。
time	TIMESTAMP
days	INT

> **说明** 如果入参中任何一个为NULL或解析错误，则返回为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

9.2.8.2.8. DATE_FORMAT

本文为您介绍如何使用DATE_FORMAT函数，将字符串类型的日期从原格式转换至目标格式。

> **注意** 本文仅适用于VVP 2.4.0且VVR 3.0.0及以上版本。

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	说明
time	TIMESTAMP	源字符串。
date	VARCHAR	源字符串，默认格式为：yyyy-MM-dd HH:mm:ss。
from_format	VARCHAR	可选，原格式，默认格式为yyyy-MM-dd hh:mm:ss。
to_format	VARCHAR	目标格式。

> **说明** 如果入参中任何一个为NULL或解析错误，则返回为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

9.2.8.2.9. DATE_FORMAT_TZ

本文为您介绍如何使用DATE_FORMAT_TZ函数，将时间戳类型的数据根据指定的时区，转成指定格式的字符串。

> **注意** 本文仅适用于VVP 2.4.0且VVR 3.0.0及以上版本。

语法

```
VARCHAR DATE_FORMAT_TZ(TIMESTAMP timestamp, VARCHAR format, VARCHAR timezone)
VARCHAR DATE_FORMAT_TZ(TIMESTAMP timestamp, VARCHAR timezone)
```

入参

参数	数据类型	说明
timestamp	TIMESTAMP	时间类型。
format	VARCHAR	输出日期格式，例如yyyy-MM-dd HH:mm:ss。
timezone	VARCHAR	指定时区，例如Asia/Shanghai。

示例

- 测试数据

timestamp1(TIMESTAMP)	format1(VARCHAR)	timezone1(VARCHAR)
0	yyyy-MM-dd HH:mm:ss	Asia/Shanghai

- 测试语句

```
SELECT
   DATE_FORMAT_TZ(timestamp1, format1, timezone1) as var1,
   DATE_FORMAT_TZ(timestamp1, timezone1) as var2
FROM T1;
```

- 测试结果

var1(VARCHAR)	var2(VARCHAR)
1970-01-01 08:00:00	1970-01-01 08:00:00

9.2.8.2.10. DATE_SUB

本文为您介绍如何使用DATE_SUB函数，返回指定日期减去目标天数后的日期。

> **注意** 本文仅适用于VVP 2.4.0且VVR 3.0.0及以上版本。

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类型日期格式为yyyy-MM-dd或yyyy-MM-dd HH:mm:ss。
time	TIMESTAMP
days	INT

> **说明** 如果入参中任何一个为NULL或解析错误，则返回为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

9.2.8.2.11. FROM_UNIXTIME

本文为您介绍如何使用FROM_UNIXTIME函数，返回VARCHAR类型的日期值。

> **注意** 本文仅适用于VVP 2.4.0且VVR 3.0.0及以上版本。

语法

```
VARCHAR FROM_UNIXTIME(BIGINT unixtime, VARCHAR format)
```

入参

参数	数据类型	说明
unixtime	BIGINT	以秒为单位的时间戳。
format	VARCHAR	可选，日志格式，默认格式为yyyy-MM-dd HH:mm:ss。

> **说明** 如果入参中任何一个为NULL或解析错误，则返回为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

9.2.8.2.12. ISNAN

本文为您介绍如何使用ISNAN函数，判断指定的浮点数是否为NaN（Not a number），类似于Java中Double.isNaN()或Float.isNaN()函数。

> **注意** 本文仅适用于VVP 2.4.0且VVR 3.0.0及以上版本。

语法

```
BOOLEAN ISNAN(DOUBLE d)
BOOLEAN ISNAN(FLOAT f)
```

入参

参数	数据类型
d	DOUBLE
f	FLOAT

示例

- 测试数据T1

d	f
0.1	0.1
0	0

- 测试语句

```
SELECT
ISNAN(CAST(d AS DOUBLE)) as d1,
ISNAN(CAST(f AS FLOAT)) as d2
FROM T1;
```

- 测试结果

d1(BOOLEAN)	d2(BOOLEAN)
false	false
false	false

9.2.8.2.13. JSON_VALUE

本文为您介绍如何使用JSON_VALUE函数，从JSON字符串中提取指定路径的表达式。

> **注意** 本文仅适用于VVP 2.4.0且VVR 3.0.0及以上版本。

语法

```
VARCHAR JSON_VALUE(VARCHAR content, VARCHAR path)
```

入参

参数	数据类型	说明
content	VARCHAR	需要解析的JSON对象。
path	VARCHAR	指定路径表达式。自定义路径需要使用单引号（'），示例如下。 `JSON_VALUE(json,'$.passenger_name') AS ABC` 目前路径支持的符号及功能如下： • $：根对象。 • []：数组下标。 • *：数组通配符。 • .：取子元素。

> **说明** 如果入参中有不合法的JSON和NULL，则返回为NULL。

示例

- 测试数据 T1

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][*]

id(INT)	json(VARCHAR)	path1(VARCHAR)
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

9.2.8.2.14. KEYVALUE

本文为您介绍如何使用KEYVALUE函数，将目标字符串按照split1分隔符拆分成Key-Value（KV）对，然后按照split2分隔符将Key-Value对分开，返回Key对应的Value。

> **注意** 本文仅适用于VVP 2.4.0且VVR 3.0.0及以上版本。

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	待拆分的目标字符串。
split1	VARCHAR	按照split1分隔符，将目标字符串拆分成Key-Value（KV）对。

参数	数据类型	说明
split2	VARCHAR	按照split2分隔符将Key-Value对分开。
key_name	VARCHAR	Key的名称。

说明

- 如果split1或split2为NULL时，则返回NULL。
- 如果str或者key_name为NULL时，则返回NULL。
- 如果没有匹配的Key时，则返回NULL。
- 如果有多个key-value匹配，返回第一个匹配上的Key对应的Value。

示例

- 测试数据T1

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
NULL
NULL
NULL

9.2.8.2.15. MURMUR_HASH

本文为您介绍如何使用MURMUR_HASH函数，按Murmur 32哈希算法计算指定参数的32位哈希值。

注意 本文仅适用于VVP 2.4.0且VVR 3.0.0及以上版本。

语法

```
INT MURMUR_HASH(value)
```

入参

参数	数据类型
value	VARCHAR或数值类型。

示例1

- 测试数据T1

value
This is a test String.
NULL
test

- 测试语句

```
SELECT  MURMUR_HASH(value) as intt
FROM T1;
```

- 测试结果

intt(VARCHAR)
740959099
NULL
271330425

示例2

- 测试数据T2

str
4.5
NULL
4

- 测试语句

```
SELECT MURMUR_HASH(CAST(str AS FLOAT)) as intt
FROM T2;
```

- 测试结果

intt(VARCHAR)
-2130121231
NULL
1863595885

9.2.8.2.16. MURMUR_HASH_64

本文为您介绍如何使用MURMUR_HASH_64函数，按Murmur 128哈希算法计算指定参数的64位哈希值。

> **注意** 本文仅适用于VVP 2.4.0且VVR 3.0.0及以上版本。

语法

```
BIGINT MURMUR_HASH_64(value)
```

入参

参数	数据类型
value	VARCHAR或数值类型。

示例1

- 测试数据T1

value
This is a test String.
NULL
test

- 测试语句

```
SELECT  MURMUR_HASH_64(value) as big
FROM T1;
```

- 测试结果

big(VARCHAR)
2706826604463634835
NULL
5010266691562018262

示例2

- 测试数据T2

value
4.5
NULL
4

- 测试语句

```
SELECT MURMUR_HASH_64(CAST(value AS FLOAT)) as big
FROM T2;
```

- 测试结果

big(VARCHAR)
8033179276572534397
NULL
-6697622088025985219

9.2.8.2.17. PARSE_URL

本文为您介绍如何使用PARSE_URL函数，返回目标URL中指定部分解析后的值。

> **注意** 本文仅适用于VVP 2.4.0且VVR 3.0.0及以上版本。

语法

```
VARCHAR PARSE_URL(VARCHAR urlStr, VARCHAR partToExtract [, VARCHAR key])
```

入参

参数	数据类型	说明

参数	数据类型	说明
urlStr	VARCHAR	目标URL。 说明 如果urlStr参数值为NULL时，则返回值为NULL。
partToExtract	VARCHAR	指定链接中解析的部分。取值如下： • HOST • PATH • QUERY • REF • PROTOCOL • FILE • AUTHORITY • USERINFO
key	VARCHAR	可选，指定截取部分，具体的键。

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)

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

9.2.8.2.18. REGEXP

本文为您介绍如何使用REGEXP函数，对指定的字符串执行一个正则表达式搜索，并返回一个Boolean值表示是否找到指定的匹配模式。

> **注意** 本文仅适用于VVP 2.4.0且VVR 3.0.0及以上版本。

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。

> **说明** 如果str或pattern为空或为NULL，则返回false。

示例

- 测试数据T1

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

9.2.8.2.19. REGEXP_REPLACE

本文为您介绍如何使用REGEXP_REPLACE函数，用目标字符串替换指定字符串，并返回新的字符串。

> **注意** 本文仅适用于VVP 2.4.0且VVR 3.0.0及以上版本。

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	用于替换的目标字符串。

> **注意**
> - 如果入参为NULL或者正则不合法时，则返回NULL。
> - 系统会将pattern和replacement的字符串解析成正则表达式，正则表达式使用Java Pattern进行处理，因此pattern和replacement的字符串需遵从Java Regex语法。

示例

- 测试数据 T1

str1(VARCHAR)	pattern1(VARCHAR)	replace1(VARCHAR)
2014-03-13	-	''
''	-	''
2014-03-13	''	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
空
s2s0s1s4s-s0s3s-s1s3s
null
num-num

9.2.8.2.20. SPLIT_INDEX

本文为您介绍如何使用SPLIT_INDEX函数，将目标字符串按照指定分隔符分割成若干段，返回指定位置的字段信息。

> **注意** 本文仅适用于VVP 2.4.0且VVR 3.0.0及以上版本。

语法

```
VARCHAR SPLIT_INDEX(VARCHAR str, VARCHAR sep, INT index)
```

入参

参数	数据类型	说明
str	VARCHAR	目标字符串。
sep	VARCHAR	分隔符。

参数	数据类型	说明
index	INT	字段位置，index从0开始。 说明 如果获取不到字段，则返回NULL。

说明 如果入参任何一个为NULL，则返回NULL。

示例

- 测试数据T1

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

9.2.8.2.21. TO_DATE

本文为您介绍如何使用TO_DATE函数，将INT类型或者VARCHAR类型的日期转换成DATE类型。

注意 本文仅适用于VVP 2.4.0且VVR 3.0.0及以上版本。

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	说明
time	INT	表示从1970-1-1到所表示时间之间天数。
date	VARCHAR	默认格式为yyyy-MM-dd。
format	VARCHAR	日期格式。

示例

- 测试数据T1

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

9.2.8.2.22. TO_TIMESTAMP_TZ

本文为您介绍如何使用TO_TIMESTAMP_TZ函数，将VARCHAR类型的日期根据指定时区转换成TIMESTAMP类型。

> **注意** 本文仅适用于VVP 2.4.0且VVR 3.0.0及以上版本。

语法

```
TIMESTAMP TO_TIMESTAMP_TZ(VARCHAR date, VARCHAR timezone)
TIMESTAMP TO_TIMESTAMP_TZ(VARCHAR date, VARCHAR format, VARCHAR timezone)
```

入参

参数	数据类型	说明
timezone	VARCHAR	字符串表示的时区，例如Asia/Shanghai。
date	VARCHAR	默认格式为yyyy-MM-dd HH:mm:ss。
format	VARCHAR	日期格式。

示例

- 测试数据T1

timestamp1(VARCHAR)	format1(VARCHAR)	timezone1(VARCHAR)
1970-01-01 00:00:00	yyyy-MM-dd HH:mm:ss	Asia/Shanghai

- 测试语句

```
SELECT
     UNIX_TIMESTAMP(TO_TIMESTAMP_TZ(timestamp1, timezone1)) as var1,
     UNIX_TIMESTAMP(TO_TIMESTAMP_TZ(timestamp1, format1, timezone1)) as var2
FROM T1;
```

- 测试结果

var1(BIGINT)	var2(BIGINT)
-28800	-28800

9.2.8.2.23. UNIX_TIMESTAMP

本文为您介绍如何使用UNIX_TIMESTAMP函数，将入参转换为长整型的时间戳。

> **注意** 本文仅适用于VVP 2.4.0且VVR 3.0.0及以上版本。

语法

```
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	说明
timestamp	TIMESTAMP	无。
date	VARCHAR	默认日期格式为 `yyyy-MM-dd HH:mm:ss` 。
format	VARCHAR	默认日期格式为 `yyyy-MM-dd hh:mm:ss` 。

示例

- 测试数据 T1

date1(VARCHAR)	date2(VARCHAR)
2021-03-25 00:00:00	1970-01-01 00:00:00

- 测试语句

```
SELECT UNIX_TIMESTAMP(to_timestamp_tz(date1)) as big1,
       UNIX_TIMESTAMP(date2) as big2
FROM T1;
```

- 测试结果

big1（BIGINT）	big2（BIGINT）
1616601600	-28800

9.2.8.3. 表值函数

9.2.8.3.1. GENERATE_SERIES

本文为您介绍如何使用GENERATE_SERIES函数，按from、from+1、from+2 ... to-1的规则，生成一系列填充数据。

> **注意** 本文仅适用于VVP 2.4.0且VVR 3.0.0及以上版本。

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。

示例

- 测试数据 T1

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 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

9.2.8.3.2. JSON_TUPLE

本文为您介绍如何使用JSON_TUPLE函数，从JSON字符串中取出各路径字符串所表示的值。

> **注意** 本文仅适用于VVP 2.4.0且VVR 3.0.0及以上版本。

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 `$` 。

示例

- 测试数据 T1

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

9.2.8.3.3. MULTI_KEYVALUE

本文为您介绍如何使用MULTI_KEYVALUE函数，将目标字符串中拆分为Key-Value（KV）对，匹配split1和split2的KV对，并返回key_name1和key_name2对应的Value值。

> **注意** 本文仅适用于VVP 2.4.0且VVR 3.0.0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	目标字符串。
split1	VARCHAR	KV对的分隔符。 当split1为NULL时，表示按照空格作为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时，表示按照空格作为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。

示例

- 测试数据 T1

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	abc	:	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

9.2.8.3.4. STRING_SPLIT

本文为您介绍如何使用STRING_SPLIT函数，根据指定的分隔符将目标字符串拆分为子字符串，并返回子字符串列表。

> **注意** 本文仅适用于VVP 2.4.0且VVR 3.0.0及以上版本。

语法

```
string_split(string, separator)
```

入参

参数	数据类型	说明
string	VARCHAR	目标字符串。
separator	VARCHAR	指定的分隔符。 **说明** separator暂不支持多字符串形式，只支持单字符串形式。

> **说明**
> - 如果string为NULL，则返回一个空行。
> - 如果string包含两个或多个连续出现的分隔符时，则返回长度为零的空子字符串。
> - 如果string未包含指定分隔符，则返回目标字符串。

示例

- 测试数据 T1

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

9.2.8.4. 聚合函数

9.2.8.4.1. APPROX_COUNT_DISTINCT

本文为您可以介绍如何使用APPROX_COUNT_DISTINCT函数，提升作业性能。

> **注意** 本文仅适用于VVP 2.4.0且VVR 3.0.0及以上版本。

背景信息

Count Distinct优化时，Aggregate节点的State需要保存所有Distinct Key信息。当Distinct Key数目过大时，State的读写开销太大，因此Count Distinct优化存在性能瓶颈。但在很多场景，完全精确的统计并不那么必要。如果您希望牺牲部分精确度来换取性能上的提升，您可以使用新增的内置函数APPROX_COUNT_DISTINCT提升作业性能。APPROX_COUNT_DISTINCT支持MiniBatch或Local-Global等Aggregate上的优化，但是需要注意以下几点：

- 输入不含有撤回消息。
- Distinct Key数目需要足够大，例如UV。如果Distinct Key数目不大，APPROX_COUNT_DISTINCT性能相对精确计算提升不大。

语法

```
APPROX_COUNT_DISTINCT(col [, accuracy])
```

入参

参数	数据类型	说明
col	任意类型	字段名称
accuracy	FLOAT	可选，准确率，取值范围为(0.0, 1.0)，默认值为0.99，取值越高，准确率越高，state开销越大，性能越低。

示例

- 测试数据T1

a (VARCHAR)	b (BIGINT)
Hi	1

a (VARCHAR)	b (BIGINT)
Hi	2
Hi	3
Hi	4
Hi	5
Hi	6

- 测试代码

```
SELECT
  a,
  APPROX_COUNT_DISTINCT(b) as b,
  APPROX_COUNT_DISTINCT(b, 0.9) as c
FROM T1
GROUP BY a;
```

- 测试结果

a (VARCHAR)	b (BIGINT)	c (BIGINT)
Hi	6	6

9.2.8.4.2. FIRST_VALUE

本文为您介绍如何使用FIRST_VALUE函数，返回指定数据流的第1条非NULL数据。

> **注意** 本文仅适用于VVP 2.4.0且VVR 3.0.0及以上版本。

语法

```
FIRST_VALUE( T value )
FIRST_VALUE( T value, BIGINT order )
```

入参

参数	数据类型	说明
value	任意参数类型	指定数据流。 **说明** 如果输入数据全为NULL，则返回NULL。
order	BIGINT	根据order判定FIRST_VALUE所在行，取order值最小的记录作为FIRST_VALUE。

注意 所有输入参数需要为相同的数据类型。

示例

- 测试数据 T1

a(BIGINT)	b(INT)	c(VARCHAR)
1	1	Hello
2	2	Hello
3	3	Hello
4	4	Hello
5	5	Hello
6	6	Hello
7	7	NULL
8	7	Hello World
9	8	Hello World
10	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(INT)
Hello	1
Hello	1
Hello	1
Hello	1
Hello	1
Hello	1
NULL	7
Hello World	7

c(VARCHAR)	var1(INT)
Hello World	7
Hello World	7

9.2.8.4.3. LAST_VALUE

本文为您介绍如何使用LAST_VALUE函数，返回指定数据流的最后1条非NULL数据。

> **注意** 本文仅适用于VVP 2.4.0且VVR 3.0.0及以上版本。

语法

```
T LAST_VALUE(T value)
T LAST_VALUE(T value, BIGINT order)
```

入参

参数	数据类型	说明
value	任意参数类型	指定数据流。
order	BIGINT	根据order判定LAST_VALUE所在的行，取order值最大的记录作为LAST_VALUE。

> **注意** 所有输入参数需要为相同的数据类型。

示例

- 测试数据 T1

a(BIGINT)	b(INT)	c(VARCHAR)
1	1	Hello
2	2	Hello
3	3	Hello
4	4	Hello
5	5	Hello
6	6	Hello
7	7	NULL
8	7	Hello World

a(BIGINT)	b(INT)	c(VARCHAR)
9	8	Hello World
10	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
NULL	7
Hello World	7
Hello World	8
Hello World	20

9.2.8.5. 机器学习函数CLUSTER_SERVING

本文以输入格式为input_shape=(?,128)的Tensorflow SavedModel模型为例，为您介绍如何使用机器学习函数CLUSTER_SERVING进行在线推理。

使用限制

本文仅适用于VVP 2.4.2及以上版本。

背景信息

Analytics Zoo是开源的统一大数据分析和AI平台，支持分布式TensorFlow及PyTorch的训练和推理，通过OpenVINO工具套件和DL Boost指令集等，提升深度学习工作负载的性能。Analytics Zoo Cluster Serving是Analytics Zoo的分布式推理解决方案，可以部署在Apache Flink集群上进行分布式运算。通过使用Cluster Serving，您无需编写任何机器学习相关代码，即可实现端到端的分布式机器学习推理流程。Analytics Zoo详情请参见Analytics Zoo。

Cluster Serving将其部分功能集成在Flink的内置函数中，支持TensorFlow、PyTorch和OpenVINO的推理模型。您可以直接在Flink开发控制台中调用CLUSTER_SERVING函数，进行分布式机器学习推理。使用CLUSTER_SERVING函数时，您只需要提供模型文件和数据文件，即可完成机器学习端到端的应用。

语法

```
CLUSTER_SERVING(uri, `data`)
```

入参

参数	数据类型	说明
uri	String	数据ID。
data	String	数据。

示例

- 测试数据input.csv

uri(String)	data(String)
my-data	0.0 0.0 0.0

iii. 在作业**高级配置**的**更多Flink配置**和Preview Session集群**其他配置**均增加以下配置，并单击**保存**。

```
pipeline.global-job-parameters: '"modelPath:""[模型目录地址]"""'
```

其中，模型目录地址为您模型文件的OSS存放目录，例如*oss://***/tf_auto/*。

iv. 在作业运维界面，单击目标作业**操作**列下的**启动**。

- 测试结果

uri1	data1
0.03932024,0.017788976,-2.2538425E-4,0.021853039,0.030905273,0.052324444,0.053021688,0.08776946,0.0625467,0.12149326,0.048770625,0.04533424,0.09271586,0.023119276,0.0037812046,0.052707616,0.029312208,0.023519568,0.07501668,0.037939064,0.06363185,0.04994791,0.08504477,0.06940596,0.052632406,0.028651983,0.02205597,0.05861686,0.040175162,0.04682718,0.05283319,0.01967535,0.111778766,0.03182444,0.04509241,0.09942829,0.009491239,0.038903005,0.11488753,0.019503327,0.0346709,0.060727723,0.03718308,0.1295206,0.04292037,0.13272661,0.009957914,0.050370887,0.03016096,0.067269124,0.09981682,0.08606887,0.085193545,0.041775055,0.02956512,0.027142545,0.09836763,0.029330501,0.040107135,0.0753366,0.068517864,0.020111244,0.086271614,0.04449262,0.107297346,0.07932708,0.036529146,0.014892701,0.08245985,0.04272034,0.076421194,0.035283472,0.028554386,0.019374546,0.048012834,0.011391987,0.028220229,0.054405782,0.037807927,0.085761145,0.07974813,0.04150249,-0.017132144,0.08219319,0.019887058,0.034193676,0.06221051,0.07516215,0.0598385,0.09159884,0.038117558,0.05329266,0.021038251,0.072046585,0.05561088,0.017538168,0.025691401,0.08580602,0.02225845,0.117060736,0.060743354,0.11840888,0.065381594,0.08059649,0.045638822,0.06399781,0.011646094,0.043574452,0.123603456,0.045708194,0.05672055,0.08402837,0.118181005,0.09657015,0.08259893,0.0010145458,0.031954847,0.07388758,0.035380267,0.070080444,0.06764534,0.058731187,0.022044828,0.094359346,0.05559277,0.032255664,0.07360537,0.11992536	128

9.2.9. 自定义函数

9.2.9.1. Java

9.2.9.1.1. 概述

本文为您介绍Flink自定义函数开发时的注意事项、自定义函数的分类、定义和参数传递方法。

注意事项

为了避免JAR包依赖冲突，在开发自定义函数时您需要注意以下几点：

- 作业开发页面选择的Flink版本，请和Pom依赖中的Flink版本保持一致。
- Flink相关依赖，scope请使用provided，即在依赖中添加 `<scope>provided</scope>` 。
- 其他第三方依赖请采用Shade方式打包，Shade打包详情请参见Apache Maven Shade Plugin。

Flink依赖冲突问题，详情请参见如何解决Flink依赖冲突问题？

自定义函数分类

Flink支持以下3类自定义函数。

分类	描述
UDF（User Defined Scalar Function）	用户自定义标量值函数，将0个、1个或多个标量值映射到一个新的标量值。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。详情请参见自定义标量函数（UDF）。
UDAF（User Defined Aggregation Function）	自定义聚合函数，将多条记录聚合成1条记录。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。详情请参见自定义聚合函数（UDAF）。
UDTF（User Defined Table-valued Function）	自定义表值函数，将0个、1个或多个标量值作为输入参数（可以是变长参数）。与自定义的标量函数类似，但与标量函数不同。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。调用一次函数输出多行或多列数据。详情请参见自定义表值函数（UDTF）。

自定义函数参数传递

如果您的自定义函数中某些参数值后期需要修改，直接在UDF代码里修改会很麻烦。此时，您可以在Flink全托管控制台更多Flink配置项中配置该参数后，在UDF代码中使用该参数。后续您可以直接在Flink全托管控制台修改该参数值，从而达到快速修改UDF参数值的目的。

自定义函数中提供了可选的open(FunctionContext context)方法，FunctionContext具备参数传递功能，自定义配置项通过此对象来传递。自定义函数的参数传递操作步骤如下：

1. 在作业开发页面右侧**高级配置**面板**更多Flink配置**中添加参数及取值。代码示例如下。

   ```
   maxJobCreationAttempts: value
   ```

2. 在UDF代码中通过ConfigOptions.key()方式获取自定义函数参数。代码示例如下。

   ```
   String strValue = context.getJobParameter("key", "default_value");
   Integer intValue = Integer.parseInt(context.getJobParameter("key", "default_value"));
   ```

   如果函数参数是key-value格式的，只能使用FunctionContext#getJobParameter方法获取pipeline.global-job-parameters配置项的值。在pipeline.global-job-parameters中定义key-value时，key和value之间需要通过冒号（:）分隔，不同kv之间需要通过逗号（,）分隔。如果key或value中含有冒号或逗号，需要使用双引号（"）包围起来。如果在此情况下key或value中还含有双引号，则需要通过连写两个双引号（""）进行转义。代码示例如下。

   ```
   pipeline.global-job-parameters: 'json1:"{k1:""value1""}",json2:"k2"'
   ```

   该配置就定义了以下的map。

key	value
json1	{k1:""value1""}
json2	k2

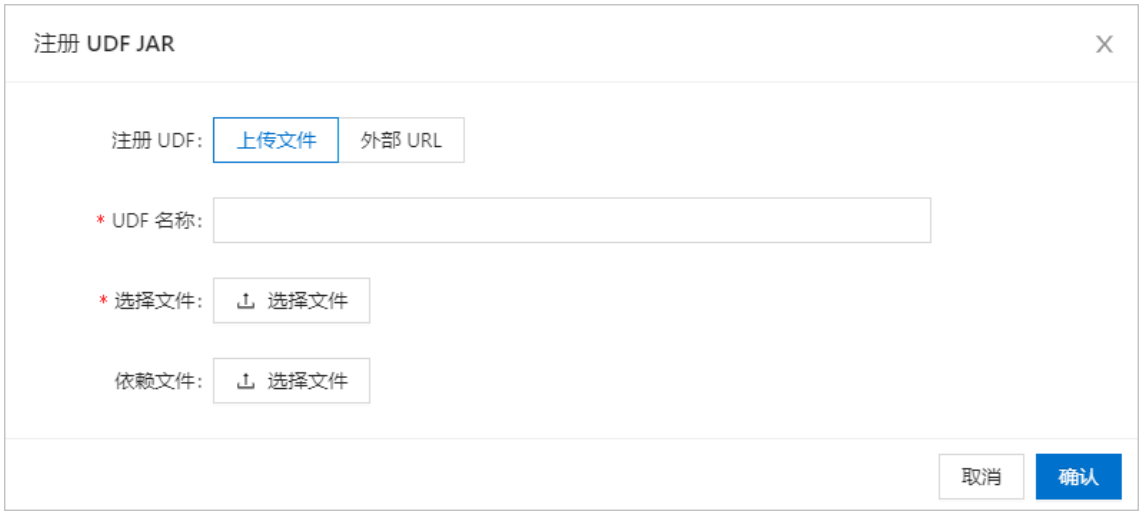
此时就能通过FunctionContext#getJobParameter获取上述map的各项内容。

```
context.getJobParameter("json1", null); // 获得字符串{k1:""value1""}
context.getJobParameter("json2", null); // 获得字符串k2
```

自定义函数注册方法

自定义函数分为全局自定义函数和作业级自定义函数，注册方法详情如下所示：

- 全局自定义函数
  i. 登录实时计算管理控制台。
  ii. 在**Flink全托管**页签，单击目标工作空间**操作**列下的**控制台**。
  iii. 在左侧导航栏上，单击**作业开发**。
  iv. 在页面左上角，单击 + 图标。
  v. 上传自定义函数JAR文件。



  您可以通过以下任何一种方式上传自定义函数JAR文件：
  - **上传文件**：单击**选择文件**项右侧的**选择文件**后，选择您的目标自定义函数JAR文件。如果有依赖文件，单击**依赖文件**项右侧的**选择文件**，选择您的目标自定义函数JAR所依赖的文件。
  - **外部URL**：输入外部URL地址。

  > **说明**
  > - 您的自定义函数JAR文件会被上传到您选择的OSS Bucket中的*sql-artifacts*目录下。此外，Flink开发控制台会解析自定义函数JAR文件中是否使用了Flink UDF、UDAF和UDTF接口的类，并自动提取类名，填充到Function Name字段中。
  > - 对于Java类型的UDF，其依赖可以打包到自定义函数JAR包中，也可以通过依赖文件项进行上传。
  > - 在自定义函数文件或者其依赖文件比较大时，推荐通过外部URL的方式进行上传。需要注意的是，如果外部URL是OSS Bucket地址，其依赖文件必须位于*sql-artifacts/namespaces/{namespace}*目录下。

  vi. 单击**确认**。

在SQL编辑器页面左侧**UDFs**列表，您可以看到所有注册成功的UDF。

- 作业级自定义函数

  因全局自定义函数（Catalog Function）在注册时，默认使用最新的Flink版本解析。如果您在作业中使用全局自定义函数，同时在高级配置选中了非最新的Flink版本，则有可能出现不兼容的问题。此时，您可以根据作业需要使用的引擎版本来实现UDF代码，并通过作业级自定义函数的方式解决此问题。作业级自定义函数的使用流程如下：

  i. 上传自定义函数JAR包。

     在左侧导航栏**资源上传**中，上传自定义函数JAR包。

  ii. 在作业中指定作业级自定义函数。

     在作业**附加依赖文件**中，指定自定义函数JAR包。

  iii. 注册自定义函数。

     通过 `CREATE TEMPORARY FUNCTION` 语法注册作业级自定义函数。

```
CREATE TEMPORARY FUNCTION MyScalarFunc AS 'com.test.MyScalarFunc';
```

VVR和Apache Flink的版本对应关系

VVR版本	Apache Flink版本
VVR 3.0.x	Flink 1.12
VVR 4.0x	Flink 1.13

9.2.9.1.2. 自定义标量函数（UDF）

本文为您介绍如何为Flink自定义标量函数（UDF）开发、注册和使用流程。

定义

自定义标量函数（UDF）将0个、1个或多个标量值映射到一个新的标量值。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。

UDF开发

> **说明** Flink为您提供了UDX示例，便于您快速开发业务。Flink UDX示例中包含UDF、UDAF和UDTF的实现，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。

1. 下载并解压ASI_UDX_Demo示例到本地。

   解压完成后，会生成ASI_UDX-main文件夹。其中：

   - pom.xml：项目级别的配置文件，主要描述了项目的Maven坐标、依赖关系，开发者需要遵循的规则、缺陷管理系统，组织和Licenses，以及其他所有的项目相关因素。
   - \ASI_UDX-main\src\main\java\ASI_UDF\ASI_UDF.java：自定义标量函数（UDF）示例的Java代码。

2. 在IntellIJ IDEA中，单击**file > open**，打开刚才解压缩完成的ASI_UDX-main。
3. 双击打开*\ASI_UDX-main\src\main\java\ASI_UDF*后，根据您的业务，配置ASI_UDF.java。

   该示例中，ASI_UDF.java已配置了获取每条数据中从begin~end位的字符的代码。

```
package ASI_UDF;
import org.apache.flink.table.functions.ScalarFunction;
public class ASI_UDF extends ScalarFunction {
    public String eval(String s, Integer begin, Integer end) {
        return s.substring(begin, end);
    }
}
```

4. 双击打开*\ASI_UDX-main*后，配置pom.xml。

   该示例中，pom.xml文件已配置了Flink 1.11版依赖的主要JAR包信息。如果您的业务：

   - 不依赖其他JAR包：不用配置pom.xml文件，继续下一步。
   - 依赖其他JAR包：在pom.xml文件中添加您所需依赖的JAR包信息。

   Flink 1.11版依赖的主要JAR包如下。

```
<dependencies>
        <dependency>
            <groupId>org.apache.flink</groupId>
            <artifactId>flink-streaming-java_2.12</artifactId>
            <version>1.11.0</version>
            <!--<scope>provided</scope>-->
        </dependency>
        <dependency>
            <groupId>org.apache.flink</groupId>
            <artifactId>flink-table</artifactId>
            <version>1.11.0</version>
            <type>pom</type>
            <!--<scope>provided</scope>-->
        </dependency>
        <dependency>
            <groupId>org.apache.flink</groupId>
            <artifactId>flink-core</artifactId>
            <version>1.11.0</version>
        </dependency>
        <dependency>
            <groupId>org.apache.flink</groupId>
            <artifactId>flink-table-common</artifactId>
            <version>1.11.0</version>
        </dependency>
    </dependencies>
```

5. 在下载文件中pom.xml所在的目录执行如下命令打包文件。

```
mvn package -Dcheckstyle.skip
```

   *\ASI_UDX-main\target*目录下会出现ASI_UDX-1.0-SNAPSHOT.jar的JAR包，即代表完成了UDF开发工作。

UDF注册

UDF注册过程，请参见管理自定义函数（UDF）。

UDF使用

在注册UDF完成后，您就可以使用UDF，详细的操作步骤如下。

1. Flink SQL作业开发。详情请参见作业开发。

   获取ASI_UDF_Source表中a字段中每行字符串中第2~4位的字符，代码示例如下。

```
CREATE TEMPORARY TABLE ASI_UDF_Source (
  a VARCHAR,
  b INT,
  c INT
) WITH (
  'connector' = 'datagen'
);
CREATE TEMPORARY TABLE ASI_UDF_Sink (
  a VARCHAR
) WITH (
  'connector' = 'blackhole'
);
INSERT INTO ASI_UDF_Sink
SELECT ASI_UDF(a,2,4)
FROM ASI_UDF_Source;
```

2. 在**作业运维**页面，单击目标作业名称**操作**列的**启动**。

   启动成功后，ASI_UDF_Sink表每行会被插入ASI_UDF_Source表中a字段每行字符串的第2~4位字符。

9.2.9.1.3. 自定义聚合函数（UDAF）

本文为您介绍如何为Flink自定义聚合函数（UDAF）开发、注册和使用流程。

定义

自定义聚合函数（UDAF），将多条记录聚合成1条记录。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。

UDAF开发

> **说明** Flink为您提供了UDX示例，便于您快速开发业务。Flink UDX示例中包含UDF、UDAF和UDTF的实现，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。

1. 下载并解压ASI_UDX_Demo示例到本地。

   解压完成后，会生成ASI_UDX-main文件夹。其中：

   - pom.xml：项目级别的配置文件，主要描述了项目的Maven坐标、依赖关系、开发者需要遵循的规则、缺陷管理系统，组织和Licenses，以及其他所有的项目相关因素。
   - \ASI_UDX-main\src\main\java\ASI_UDAF\ASI_UDAF.java：自定义聚合函数（UDAF）示例的Java代码。

2. 在IntellIJ IDEA中，选择**file > open**，打开刚才解压缩完成的ASI_UDX-main。
3. 双击打开*\ASI_UDX-main*后，配置pom.xml。

   该示例中，pom.xml文件已配置了Flink 1.12版开发自定义函数需要的最小化依赖信息。如果您的业务：

   - 没有其他依赖：不用配置pom.xml文件，继续下一步。
   - 有其他依赖：在pom.xml文件中添加您所需的依赖信息。

Flink 1.12版最小化依赖如下。

```
<dependencies>
    <dependency>
        <groupId>org.apache.flink</groupId>
        <artifactId>flink-table-common</artifactId>
        <version>1.12.7</version>
        <scope>provided</scope>
    </dependency>
</dependencies>
```

说明 建议填对应Flink版本的最新版本，详情请参见VVR和Apache Flink的版本对应关系。

4. 双击打开*\ASI_UDX-main\src\main\java\ASI_UDAF*后，根据您的业务，配置ASI_UDAF.java。

   该示例中，ASI_UDAF.java实现了累积求和的代码，详情如下。

```
package ASI_UDAF;
import org.apache.flink.table.functions.AggregateFunction;
import java.util.Iterator;
public class ASI_UDAF {
    public static class AccSum {
        public long sum;
    }
    public static class MySum extends AggregateFunction<Long, AccSum> {
        @Override
        public Long getValue(AccSum acSum) {
            return acSum.sum;
        }
        @Override
        public AccSum createAccumulator() {
            AccSum acCount = new AccSum();
            acCount.sum = 0;
            return acCount;
        }
        public void accumulate(AccSum acc, long num) {
            acc.sum += num;
        }
        /**
         * Support retract a msg generated by upstream operator.
         */
        public void retract(AccSum acc, long num) {
            acc.sum -= num;
        }
        /**
         * Support local-global two stage aggregate optimization.
         */
        public void merge(AccSum acc, Iterable<AccSum> it) {
            Iterator<AccSum> iter = it.iterator();
            while (iter.hasNext()) {
                AccSum accSum = iter.next();
                if (null != accSum) {
                    acc.sum += accSum.sum;
                }
            }
        }
    }
}
```

该UDAF的是一个简单累加和的操作。例如，同一个分组键（GROUP BY字段）的3条输入数据分别为1、2、3，输出结果有以下两种情况：

- 在未启用MiniBatch优化时，即默认配置，则输出的结果为1、3、6。
- 在开启了MiniBatch优化时，因为输出的数据条数取决于设置的MiniBatch参数和输入数据的分布情况，所以能确定的是最后输出一条结果为6，但输出的中间结果条数不确定。

> **说明** MiniBatch优化详情，请参见高性能Flink SQL优化技巧。

5. 在下载文件中pom.xml所在目录执行如下打包命令。

```
mvn package -Dcheckstyle.skip
```

*\ASI_UDX-main\target*目录下会出现ASI_UDX-1.0-SNAPSHOT.jar的JAR包，即代表完成了UDAF开发工作。

UDAF使用

您可以通过以下两种方式在SQL作业中使用自定义UDAF：

- 方式一：先注册UDAF，再在作业中直接使用已注册的UDAF。

  通过该方式进行函数注册的优点为便于后续开发进行代码复用。UDAF注册过程，请参见管理自定义函数（UDF）。如果注册完的函数名称为ASI_UDAF$MySum，则在作业中直接使用的代码示例如下。

```
CREATE TEMPORARY TABLE ASI_UDAF_Source (
  a BIGINT NOT NULL
) WITH (
  'connector' = 'datagen'
);
CREATE TEMPORARY TABLE ASI_UDAF_Sink (
  sum  BIGINT
) WITH (
  'connector' = 'print'
);
INSERT INTO ASI_UDAF_Sink
SELECT `ASI_UDAF$MySum`(a)
FROM ASI_UDAF_Source;
```

- 方式二：先将自定义函数JAR包上传至Flink**作业开发**的**高级配置**页签的**附加依赖文件**选项，再在作业的SQL语句中添加创建临时函数的语句，并使用该函数。

  上传自定义函数JAR包到附加依赖文件中后，只能在本作业中使用该自定义函数，其他作业中不可使用。如果创建的临时函数名称为mysum，则在作业中使用该函数的代码示例如下。

```
CREATE TEMPORARY TABLE ASI_UDAF_Source (
  a   BIGINT
) WITH (
  'connector' = 'datagen'
);
CREATE TEMPORARY TABLE ASI_UDAF_Sink (
  sum  BIGINT
) WITH (
  'connector' = 'print'
);
CREATE TEMPORARY FUNCTION `mysum` AS 'ASI_UDAF.ASI_UDAF$MySum'; --创建临时函数mysum。
INSERT INTO ASI_UDAF_Sink
SELECT `mysum`(a)
FROM ASI_UDAF_Source;
```

说明 SQL作业开发完成后，需要在**作业运维**页面，单击目标作业名称**操作**列的**启动**。启动成功后，ASI_UDAF_Sink表中就会插入ASI_UDAF_Source表中a字段数据的累加和。

9.2.9.1.4. 自定义表值函数（UDTF）

本文为您介绍如何为Flink自定义表值函数（UDTF）开发、注册和使用流程。

定义

自定义表值函数（UDTF），自定义表值函数，将0个、1个或多个标量值作为输入参数（可以是变长参数）。与自定义的标量函数类似，但与标量函数不同。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。调用一次函数输出多行或多列数据。

UDTF开发

> **说明** Flink为您提供了UDX示例，便于您快速开发业务。Flink UDX示例中包含UDF、UDAF和UDTF的实现，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。

1. 下载并解压ASI_UDX_Demo示例到本地。

   解压完成后，会生成ASI_UDX-main文件夹。其中：

   - pom.xml：项目级别的配置文件，主要描述了项目的Maven坐标、依赖关系，开发者需要遵循的规则、缺陷管理系统，组织和Licenses，以及其他所有的项目相关因素。
   - \ASI_UDX-main\src\main\java\ASI_UDF\ASI_UDTF.java：自定义表值函数（UDTF）示例的Java代码。

2. 在IntellIJ IDEA中，单击**file > open**，打开刚才解压缩完成的ASI_UDX-main。
3. 双击打开*\ASI_UDX-main\src\main\java\ASI_UDTF*后，根据您的业务，修改ASI_UDTF.java文件内容。

   该示例中，ASI_UDTF.java已配置了将一行字符串按照竖线（|）分割成多列字符串的代码。

```
package ASI_UDTF;
import org.apache.flink.api.java.tuple.Tuple2;
import org.apache.flink.table.functions.TableFunction;
public class ASI_UDTF extends TableFunction<Tuple2<String,String>> {
    public void eval(String str){
        String[] split = str.split("\\|");
        String name = split[0];
        String place = split[1];
        Tuple2<String,String> tuple2 = Tuple2.of(name,place);
        collect(tuple2);
    }
}
```

4. 双击打开*\ASI_UDX-main*后，配置pom.xml。

   该示例中，pom.xml文件已配置了Flink 1.11版依赖的主要JAR包信息。如果您的业务：

   - 不依赖其他JAR包：不用配置pom.xml文件，继续下一步。
   - 依赖其他JAR包：在pom.xml文件中添加您所需依赖的JAR包信息。

   Flink 1.11版依赖的主要JAR包如下。

```
<dependencies>
        <dependency>
            <groupId>org.apache.flink</groupId>
            <artifactId>flink-streaming-java_2.12</artifactId>
            <version>1.11.0</version>
            <!--<scope>provided</scope>-->
        </dependency>
        <dependency>
            <groupId>org.apache.flink</groupId>
            <artifactId>flink-table</artifactId>
            <version>1.11.0</version>
            <type>pom</type>
            <!--<scope>provided</scope>-->
        </dependency>
        <dependency>
            <groupId>org.apache.flink</groupId>
            <artifactId>flink-core</artifactId>
            <version>1.11.0</version>
        </dependency>
        <dependency>
            <groupId>org.apache.flink</groupId>
            <artifactId>flink-table-common</artifactId>
            <version>1.11.0</version>
        </dependency>
    </dependencies>
```

5. 在下载文件中pom.xml所在的目录执行如下命令打包文件。

```
mvn package -Dcheckstyle.skip
```

*\ASI_UDX-main\target*目录下会出现ASI_UDX-1.0-SNAPSHOT.jar的JAR包，即代表完成了UDTF开发工作。

UDTF注册

UDTF注册过程，请参见管理自定义函数（UDF）。

UDTF使用

在注册UDTF完成后，您就可以使用UDTF，详细的操作步骤如下。

1. Flink SQL作业开发。详情请参见作业开发。

   ASI_UDTF_Source表中message字段每行字符串按照竖线（|）分割成多列，代码示例如下。

```
CREATE TEMPORARY TABLE ASI_UDTF_Source (
  `message`  VARCHAR
) WITH (
  'connector'='datagen'
);
CREATE TEMPORARY TABLE ASI_UDTF_Sink (
  name  VARCHAR,
  place  VARCHAR
) WITH (
  'connector' = 'blackhole'
);
INSERT INTO ASI_UDTF_Sink
SELECT name,place
FROM ASI_UDTF_Source,lateral table(ASI_UDTF(`message`)) as T(name,place);
```

2. 在**作业运维**页面，单击目标作业名称**操作**列的**启动**。

   启动成功后，ASI_UDTF_Sink表会被插入ASI_UDTF_Source表中message字段按照竖线（|）分割成多列的字符。

9.2.9.2. Python

9.2.9.2.1. 概述

Flink全托管支持在SQL作业中使用Python自定义函数，本文为您介绍Flink自定义函数UDX的分类、定义和依赖，以及如何进行调试和调优。

UDX分类

Flink支持以下3类自定义函数。

UDX分类	描述
UDF（User Defined Scalar Function）	用户自定义标量值函数，将0个、1个或多个标量值映射到一个新的标量值。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。详情请参见自定义标量函数（UDF）。
UDAF（User Defined Aggregation Function）	自定义聚合函数，将多条记录聚合成1条记录。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。详情请参见自定义聚合函数（UDAF）。
UDTF（User Defined Table-valued Function）	自定义表值函数，将0个、1个或多个标量值作为输入参数（可以是变长参数）。与自定义的标量函数类似，但与标量函数不同。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。调用一次函数输出多行或多列数据。详情请参见自定义表值函数（UDTF）。

依赖

Flink全托管集群已预装了Pandas、NumPy和PyArrow等常用的Python包，您可以在作业开发页面，了解Flink全托管集群中已安装的第三方Python包列表。

此外，您也可以在Python自定义函数中使用其它类型的第三方Python包。需要注意的是，如果使用了非预装的第三方Python包，在注册Python UDF时，需要将其作为依赖文件上传，详情请参见管理自定义函数（UDF）和使用Python依赖。

调试

您可以在Python自定义函数的代码实现中，通过Logging的方式，输出日志信息，方便问题定位，示例如下。

```
@udf(result_type=DataTypes.BIGINT())
def add(i, j):
  logging.info("hello world")
  return i + j
```

日志输出后，您可以在TaskManager的日志文件中查看日志。

调优

- 使用Pandas库

  除了普通Python自定义函数之外，Flink全托管也支持您使用Pandas自定义函数。对于Pandas自定义函数，输入数据的类型是Pandas中定义的数据结构，例如pandas.Series和pandas.DataFrame等，您可以在Pandas自定义函数中使用Pandas和Numpy等高性能的Python库，开发出高性能的Python自定义函数，详情请参见Vectorized User-defined Functions。

- 配置以下参数

  Python自定义函数的性能在很大程度取决于Python自定义函数自身的实现，如果遇到性能问题，您需要尽可能优化Python自定义函数的实现。除此之外，Python自定义函数的性能也受以下参数取值的影响。

参数	说明
python.fn-execution.bundle.size	Python UDF的计算是异步的，在执行过程中，Java算子将数据异步发送给Python进程进行处理。Java算子在将数据发送给Python进程之前，会先将数据缓存起来，到达一定阈值之后，再发送给Python进程。python.fn-execution.bundle.size参数可用来控制可缓存的数据最大条数。 默认值为100000，单位是条数。
python.fn-execution.bundle.time	用来控制数据的最大缓存时间。当缓存的数据条数到达python.fn-execution.bundle.size定义的阈值或缓存时间到达python.fn-execution.bundle.time定义的阈值时，会触发缓存数据的计算。 默认值为1000，单位是毫秒。
python.fn-execution.arrow.batch.size	使用Pandas UDF时，一个arrow batch可容纳的数据最大条数，默认值为10000。 **说明** python.fn-execution.arrow.batch.size参数值不能大于python.fn-execution.bundle.size参数值。

> **说明** 以上3个参数并不是配置的越大越好，当这些参数取值配置过大时，可能会导致Checkpoint时，需要处理过多的数据，从而导致Checkpoint时间过长，甚至会导致Checkpoint失败。以上参数的更多详情，请参见Configuration。

作业级自定义函数

因全局自定义函数（Catalog Function）在注册时，默认使用最新的Flink版本解析。如果您在作业中使用全局自定义函数，同时在高级配置选中了非最新的Flink版本，则有可能出现不兼容的问题。此时，您可以根据作业需要使用的引擎版本来实现UDF代码，并通过作业级自定义函数的方式解决此问题。作业级自定义函数的使用流程如下：

1. 上传自定义函数Python包。

   在左侧导航栏**资源上传**中，上传自定义函数Python包。

2. 在作业中指定作业级自定义函数。

   在作业**附加依赖文件**中，指定自定义函数Python包。

3. 注册自定义函数。

   通过 `CREATE TEMPORARY FUNCTION` 语法注册作业级自定义函数。

   ```
   CREATE TEMPORARY FUNCTION MyScalarFunc AS 'com.test.MyScalarFunc';
   ```

9.2.9.2.2. 自定义标量函数（UDF）

本文为您介绍Python自定义标量函数（UDF）的开发、注册和使用流程。

定义

自定义标量函数（UDF）将0个、1个或多个标量值映射到一个新的标量值。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。

使用限制

由于Flink全托管产品受部署环境和网络环境等因素的影响，所以开发Python自定义函数时，需要注意以下限制：

- 仅支持开源Flink V1.12及以上版本。
- Flink全托管集群上已预装了Python 3.7.9，因此需要您在Python 3.7.9版本开发代码。
- Flink全托管运行环境使用的是JDK 1.8，如果Python作业中依赖第三方JAR包，请确保JAR包兼容JDK 1.8。
- 仅支持开源Scala V2.11版本，如果Python作业中依赖第三方JAR包，请确保使用Scala V2.11对应的JAR包依赖。

UDF开发

> **说明** Flink为您提供了Python UDX示例，便于您快速开发UDX。Flink Python UDX示例中包含了Python UDF、Python UDAF和Python UDTF的实现。本文以Windows操作系统为例，为您介绍如何进行UDF开发。

1. 下载并解压python_demo-master示例到本地。
2. 在PyCharm中，单击**file > open**，打开刚才解压缩完成的python_demo-master。

3. 双击打开*\python_demo-master\udx\udfs.py*后，根据您的业务，修改udfs.py。

   该示例中，sub_string定义了获取每条数据中从begin~end位的字符的代码。

```
from pyflink.table import DataTypes
from pyflink.table.udf import udf
@udf(result_type=DataTypes.STRING())
def sub_string(s: str, begin: int, end: int):
    return s[begin:end]
```

4. 在下载文件中udx所在的目录（即*\python_demo-master*目录）下执行如下命令打包文件。

```
zip -r python_demo.zip udx
```

   *\python_demo-master*目录下会出现python_demo.zip的ZIP包，即代表完成了Python UDF的开发工作。

UDF注册

UDF注册过程，请参见注册UDF。

UDF使用

在完成注册UDF后，您就可以使用UDF，详细的操作步骤如下。

1. Flink SQL作业开发。详情请参见作业开发。

   获取ASI_UDF_Source表中a字段中每行字符串中第2~4位的字符，代码示例如下。

```
CREATE TEMPORARY TABLE ASI_UDF_Source (
  a VARCHAR,
  b INT,
  c INT
) WITH (
  'connector' = 'datagen'
);
CREATE TEMPORARY TABLE ASI_UDF_Sink (
  a VARCHAR
) WITH (
  'connector' = 'blackhole'
);
INSERT INTO ASI_UDF_Sink
SELECT ASI_UDF(a,2,4)
FROM ASI_UDF_Source;
```

2. 在**作业运维**页面，单击目标作业名称**操作**列的**启动**。

   启动成功后，ASI_UDF_Sink表每行会被插入ASI_UDF_Source表中a字段每行字符串的第2~4位字符。

9.2.9.2.3. 自定义聚合函数（UDAF）

本文为您介绍Python自定义聚合函数（UDAF）开发、注册和使用流程。

定义

自定义聚合函数（UDAF），将多条记录聚合成1条记录。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。

使用限制

由于Flink全托管产品受部署环境和网络环境等因素的影响，所以开发Python自定义函数时，需要注意以下限制：

- 仅支持开源Flink V1.12及以上版本。
- Flink全托管集群上已预装了Python 3.7.9，因此需要您在Python 3.7.9版本开发代码。
- Flink全托管运行环境使用的是JDK 1.8，如果Python作业中依赖第三方JAR包，请确保JAR包兼容JDK 1.8。
- 仅支持开源Scala V2.11版本，如果Python作业中依赖第三方JAR包，请确保使用Scala V2.11对应的JAR包依赖。

UDAF开发

> **说明** Flink为您提供了Python UDX示例，便于您快速开发UDX。Flink Python UDX示例中包含了Python UDF、Python UDAF和Python UDTF的实现。本文以Windows操作系统为例，为您介绍如何进行UDAF开发。

1. 下载并解压python_demo-master示例到本地。
2. 在PyCharm中，单击**file > open**，打开刚才解压缩完成的python_demo-master。
3. 双击打开*\python_demo-master\udx\udafs.py*后，根据您的业务，配置udafs.py。

   该示例中，weighted_avg定义了当前数据和历史数据求含权重的均值的代码。

```
from pyflink.common import Row
from pyflink.table import AggregateFunction, DataTypes
from pyflink.table.udf import udaf
class WeightedAvg(AggregateFunction):
    def create_accumulator(self):
        # Row(sum, count)
        return Row(0, 0)
    def get_value(self, accumulator: Row) -> float:
        if accumulator[1] == 0:
            return 0
        else:
            return accumulator[0] / accumulator[1]
    def accumulate(self, accumulator: Row, value, weight):
        accumulator[0] += value * weight
        accumulator[1] += weight
    def retract(self, accumulator: Row, value, weight):
        accumulator[0] -= value * weight
        accumulator[1] -= weight
weighted_avg = udaf(f=WeightedAvg(),
                    result_type=DataTypes.DOUBLE(),
                    accumulator_type=DataTypes.ROW([
                        DataTypes.FIELD("f0", DataTypes.BIGINT()),
                        DataTypes.FIELD("f1", DataTypes.BIGINT())]))
```

4. 在下载文件中udx所在的目录（即*\python_demo-master*目录）下执行如下命令打包文件。

```
zip -r python_demo.zip udx
```

   *\python_demo-master*目录下会出现python_demo.zip的ZIP包，即代表完成了UDAF开发工作。

UDAF注册

UDAF注册过程，请参见注册UDF。

UDAF使用

在完成注册UDAF后，您就可以使用UDAF，详细的操作步骤如下。

1. Flink SQL作业开发。详情请参见作业开发。

   获取ASI_UDAF_Source表中a字段以b字段为权重的值，代码示例如下。

```
CREATE TEMPORARY TABLE ASI_UDAF_Source (
  a   BIGINT,
  b   BIGINT
) WITH (
  'connector' = 'datagen'
);
CREATE TEMPORARY TABLE ASI_UDAF_Sink (
  avg_value  DOUBLE
) WITH (
  'connector' = 'blackhole'
);
INSERT INTO ASI_UDAF_Sink
SELECT weighted_avg(a, b)
FROM ASI_UDAF_Source;
```

2. 在**作业运维**页面，单击目标作业名称**操作**列的**启动**。

   启动成功后，ASI_UDAF_Sink表每行会被插入ASI_UDAF_Source表中以b字段为权重的a字段当前数据和历史数据的均值。

9.2.9.2.4. 自定义表值函数（UDTF）

本文为您介绍Python自定义表值函数（UDTF）的开发、注册和使用流程。

定义

自定义表值函数（UDTF），将0个、1个或多个标量值作为输入参数（可以是变长参数）。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。调用一次函数输出多行或多列数据。与自定义的标量函数类似，但与标量函数不同。

使用限制

由于Flink全托管产品受部署环境和网络环境等因素的影响，所以开发Python自定义函数时，需要注意以下限制：

- 仅支持开源Flink V1.12及以上版本。
- Flink全托管集群上已预装了Python 3.7.9，因此需要您在Python 3.7.9版本开发代码。
- Flink全托管运行环境使用的是JDK 1.8，如果Python作业中依赖第三方JAR包，请确保JAR包兼容JDK 1.8。
- 仅支持开源Scala V2.11版本，如果Python作业中依赖第三方JAR包，请确保使用Scala V2.11对应的JAR包依赖。

UDTF开发

> **说明** Flink为您提供了Python UDX示例，便于您快速开发UDX。Flink Python UDX示例中包含了Python UDF、Python UDAF和Python UDTF的实现。本文以Windows操作系统为例，为您介绍如何进行UDTF开发。

1. 下载并解压python_demo-master示例到本地。
2. 在PyCharm中，单击**file > open**，打开刚才解压缩完成的python_demo-master。
3. 双击打开*\python_demo-master\udx\udtfs.py*后，根据您的业务，修改udtfs.py文件内容。

   该示例中，split定义了将一行字符串按照竖线（|）分割成多列字符串的代码。

```
from pyflink.table import DataTypes
from pyflink.table.udf import udtf
@udtf(result_types=[DataTypes.STRING(), DataTypes.STRING()])
def split(s: str):
    splits = s.split("|")
    yield splits[0], splits[1]
```

4. 在下载文件中udx所在的目录（即*\python_demo*目录）下执行如下命令打包文件。

```
zip -r python_demo.zip udx
```

   *\python_demo*目录下会出现python_demo.zip的ZIP包，即代表完成了UDTF开发工作。

UDTF注册

UDTF注册过程，请参见注册UDF。

UDTF使用

在完成UDTF注册后，您就可以使用UDTF，详细的操作步骤如下。

1. Flink SQL作业开发。详情请参见作业开发。

   ASI_UDTF_Source表中每行字符串的message字段与字符串aa按照竖线（|）连接之后，按照竖线（|）分割成多列，代码示例如下。

```
CREATE TEMPORARY TABLE ASI_UDTF_Source (
  `message`  VARCHAR
) WITH (
  'connector'='datagen'
);
CREATE TEMPORARY TABLE ASI_UDTF_Sink (
  name  VARCHAR,
  place  VARCHAR
) WITH (
  'connector' = 'blackhole'
);
INSERT INTO ASI_UDTF_Sink
SELECT name,place
FROM ASI_UDTF_Source,lateral table(split(concat_ws('|', `message`, 'aa'))) as T(name,pl
ace);
```

2. 在**作业运维**页面，单击目标作业名称**操作**列的**启动**。

启动成功后，ASI_UDTF_Sink表会被插入两列数据，这两列数据是ASI_UDTF_Source表中message和aa字段按照竖线（|）连接后，再按照竖线（|）分割成多列的字符。

9.3. Flink State兼容性参考

9.3.1. 概述

Flink作业是通过状态数据（包括Checkpoint和Savepoint）保存作业中间的计算结果。Flink全托管提供状态兼容性检查功能，本文为您介绍系统根据您选择的状态数据给出的兼容性判断结果详情。

当前Flink SQL已经可以帮助您支持各种流计算场景。为了满足开发迭代需求或业务发展需求，SQL作业也需要不断变更，否则修改SQL作业后，并使用原来的状态数据重启作业，会出现状态不兼容的问题。

从vvr-4.0.11-flink-1.13版本开始，Flink全托管提供状态兼容性检查和状态数据迁移功能，可以最大可能的复用原来的状态数据，帮助您快速升级SQL作业。作业修改上线后，如果您选择基于状态启动作业时，则系统会根据您选择的状态数据，给出兼容性判断，结果有以下几种：

- 完全兼容

  当前作业和状态数据完全兼容。即基于状态数据运行的结果和完全基于历史数据运行的结果一致。为了达到完全兼容，可能需要进行状态数据迁移，详情请参见状态迁移。

- 部分兼容

当前作业和状态数据部分兼容。即兼容的部分列基于状态数据运行的结果和完全基于历史数据运行的结果一致，其余列在状态中没有对应的数据，和完全基于历史数据运行的结果不一致。为了达到部分兼容，可能需要进行状态数据迁移，详情请参见状态迁移。

- 不兼容

  当前作业和状态数据不兼容。如果基于该状态数据启动，大概率会出现启动失败或者运行结果不符合预期的问题，请谨慎选择。

- 无法判断

当前没法给出兼容性判断。如果基于该状态数据启动，可能会出现启动失败或者运行结果不符合预期的问题，请谨慎选择。

9.3.2. SQL修改及兼容性

9.3.2.1. 概述

本文分场景为您介绍SQL修改所造成的兼容性影响。

场景	描述	相关文档
修改Query	别名的变化不影响状态的兼容性。	Alias
	为您介绍Group Aggregate修改的可兼容性和不可兼容性详情。	Group Aggregate
	为您介绍Over Aggregate修改的可兼容性和不可兼容性详情。	Over Aggregate
	为您介绍Group Window Aggregate修改的可兼容性和不可兼容性详情。	Group Window Aggregate
	为您介绍Window TVF修改的可兼容性和不可兼容性详情。	Window TVF
	为您介绍Deduplication修改的可兼容性和不可兼容性详情。	Deduplication
	为您介绍Top-N修改的可兼容性和不可兼容性详情。	Top-N
	为您介绍Window Top-N修改的可兼容性和不可兼容性详情。	Window Top-N
	为您介绍Reglar Join、Interval Join、Temporal Join和Lookup Join修改的可兼容性和不可兼容性详情。	Join

场景	描述	相关文档
	为您介绍Pattern Recognition修改的可兼容性和不可兼容性详情。	Pattern Recognition
	为您介绍Where修改的可兼容性和不可兼容性详情。	Where
	为您介绍Union All修改的可兼容性和不可兼容性详情。	Union All
	为您介绍Temporal Sort修改的可兼容性和不可兼容性详情。	Temporal Sort
	为您介绍Limit修改的可兼容性和不可兼容性详情。	Limit
修改Source	为您介绍Source修改所造成的可兼容性和不可兼容性影响。	修改Source
修改Sink	为您介绍Sink修改所造成的可兼容性和不可兼容性影响。	修改Sink
其他限制	介绍修改SQL除Query、Source和Sink之外的其他限制。	其他限制

上述语法示例中将用以下DDL作为源表。

```
CREATE TABLE MyTable (
  a int,
  b bigint,
  c varchar,
  d int,
  ts timestamp(3),
  proctime as proctime(),
  watermark for ts AS ts - interval '1' second
);
```

9.3.2.2. 修改Query

9.3.2.2.1. Alias

别名的变化不影响状态的兼容性。

> **说明** 后续所有Query的兼容性判断不再列举别名的变化。

别名的变化代码示例如下。

```
-- 原始SQL
select a, sum(b), max(c) from MyTable group by a;
-- 添加别名：a -> a1
-- 属于完全兼容修改。
select a1, sum(b), max(c) from (select a as a1, b, c from MyTable) group by a1;
```

9.3.2.2.2. Group Aggregate

本文为您介绍Group Aggregate修改的可兼容性和不可兼容性详情。

可兼容的修改

- 新增、删除、修改非Distinct的统计指标（Aggregate Function）。
  - 对于新增统计指标，属于部分兼容，从当前作业启动时开始累计。
  - 对于删除统计指标，属于完全兼容修改。删除的统计指标对应的状态数据会被丢弃。
  - 对于既有新增又有删除统计指标，则属于部分兼容修改。新增的统计指标从当前作业启动时开始累计，删除的统计指标对应的状态数据会被丢弃。
  - 对于修改统计指标，被视为删除和新增两个操作，属于部分兼容。新增的统计指标从当前作业启动时开始累计，删除的统计指标对应的状态数据会被丢弃。

> **说明** 对于未进行任何变更的统计指标，复用状态数据后计算的结果和基于历史数据运行的结果是一致的。

```
-- 原始SQL。
select a, sum(b), max(c) from MyTable group by a;
-- 新增统计指标：count(c)，属于部分兼容修改。
-- sum(b)、max(c) 的计算结果不受影响，count(c) 的值在作业启动时从 0 开始累计
select a, sum(b), max(c), count(c) from MyTable group by a;
-- 删除统计指标：sum(b)，属于完全兼容修改。
-- max(c)的计算结果不受影响。
select a, max(c) from MyTable group by a;
-- 修改统计指标：max(c) -> min(c)，属于部分兼容修改。
-- sum(b) 的计算结果不受影响。max(c) 被认为删除，其对应的状态数据会被丢弃。
-- min(c) 被认为是新增指标，其值在作业启动时开始计算。
select a, sum(b), min(c) from MyTable group by a;
```

- 调整非Distinct的统计指标位置，该修改属于完全兼容修改。

```
-- 原始SQL。
select a, sum(b), max(c) from MyTable group by a;
-- 调整统计指标位置：sum(b)、max(c)，属于完全兼容修改。
-- sum(b)、max(c)的计算结果不受影响。
select a, max(c), sum(b) from MyTable group by a;
```

- 当非Distinct的统计指标涉及的字段有计算逻辑变化时，该统计指标被认为做了修改操作，该修改属于部分兼容修改。

```
-- 原始SQL。
select a, sum(b), max(c) from MyTable group by a;
-- 修改统计指标：max(c) -> max(substring(c, 1, 5))，属于部分兼容修改。
-- sum(b)的计算结果不受影响。max(c)被认为删除，其对应的状态数据会被丢弃。
-- max(substring(c, 1, 5))被认为新增指标，其值在作业启动时开始计算。
select a, sum(b), max(c) from (
  select a, b, substring(c, 1, 5) as c from MyTable
) group by a;
```

不兼容的修改

- 新增、删除、修改统计维度（group key）或者统计维度涉及字段的计算逻辑发生变化，该修改属于不兼容修改。

```
-- 原始SQL。
select a, sum(b), max(c) from MyTable group by a;
-- 新增统计维度：d，当前修改属于不兼容修改。
select a, sum(b), max(c) from MyTable group by a, d;
-- 删除统计维度：a，当前修改属于不兼容修改。
select sum(b), max(c) from MyTable;
-- 修改统计维度：a -> d，当前修改属于不兼容修改。
select d, sum(b), min(c) from MyTable group by d;
-- 修改了统计维度：a -> a + 1，当前修改属于不兼容修改。
select a, sum(b), max(c) from (
  select a + 1 as a, b, c from MyTable
) group by a;
```

- 新增、删除、修改Distinct统计指标（Distinct Aggregate Function）或者Distinct统计指标涉及字段的计算逻辑发生变化，该修改属于不兼容修改。

```
-- 原始SQL。
select a, sum(b), max(c), sum(distinct b), count(distinct c) from MyTable group by a;
-- 新增distinct统计指标：count(distinct b)，当前修改属于不兼容修改。
select a, sum(b), max(c), sum(distinct b), count(distinct b), count(distinct c) from MyTa
ble group by a;
-- 删除distinct统计指标：sum(distinct b)，当前修改属于不兼容修改。
select a, sum(b), max(c), count(distinct c) from MyTable group by a;
-- 修改distinct统计指标：sum(distinct b) -> avg(distinct b)，当前修改属于不兼容修改。
select a, sum(b), max(c), avg(distinct b), count(distinct c) from MyTable group by a;
-- 修改distinct统计指标：count(distinct c) -> count(distinct substring(c, 1, 5))，当前修改属
于不兼容修改。
select a, sum(b), max(c), sum(distinct b), count(distinct c) from (
  select a, b, substring(c, 1, 5) as c from MyTable
) group by a;
```

- 级联聚合时，新增统计指标，该修改属于不兼容修改。（级联聚合会发生消息撤回，新增统计指标结果不可预期。）

```
-- 原始SQL。
select a/2, avg(b), min(c) from (
   select a, sum(b), max(c) from MyTable group by a
) group by a/2;
-- 新增统计指标：count(c)，当前修改属于不兼容修改。
select a/2, avg(b), min(c), count(c) from (
   select a, sum(b), max(c) from MyTable group by a
) group by a/2;
```

- 删除了所有统计指标，该修改属于不兼容修改。（所有统计指标的状态数据都被丢弃，没有状态复用。）

```
-- 原始SQL。
select a, sum(b), max(c) from MyTable group by a;
-- 删除所有统计指标：sum(b), max(c)，当前修改属于不兼容修改。
select a from MyTable group by a;
```

9.3.2.2.3. Over Aggregate

本文为您介绍Over Aggregate修改的可兼容性和不可兼容性详情。

可兼容的修改

- 新增、删除、修改非Distinct的统计指标（Aggregate Function）。
  - 对于新增统计指标，属于部分兼容，从当前作业启动时开始累计。
  - 对于删除统计指标，属于完全兼容修改。删除的统计指标对应的状态数据会被丢弃。
  - 对于既有新增又有删除统计指标，则属于部分兼容修改。新增的统计指标从当前作业启动时开始累计，删除的统计指标对应的状态数据会被丢弃。
  - 对于修改统计指标，被视为删除和新增两个操作，属于部分兼容。新增的统计指标从当前作业启动时开始累计，删除的统计指标对应的状态数据会被丢弃。

> **说明**
>
> - 对于未进行任何变更的统计指标，复用状态数据后计算的结果和基于历史数据运行的结果是一致的。
> - Over Aggregate除了输出聚合指标，还会输出原始输入数据。因此输入的Schema发生变化时，状态不兼容。

```
-- 原始SQL。
select a, b, c,
  sum(b) over (partition by a order by ts),
  max(c) over (partition by a order by ts)
from MyTable;
-- 新增统计指标：count(c)，属于部分兼容修改。
-- sum(b)、max(c) 的计算结果不受影响，count(c)的值在作业启动时从0开始累计。
select a, b, c,
  sum(b) over (partition by a order by ts),
  max(c) over (partition by a order by ts),
  count(c) over (partition by a order by ts)
from MyTable;
-- 删除统计指标：sum(b)，属于完全兼容修改。
-- max(c) 的计算结果不受影响。
select a, b, c,
  max(c) over (partition by a order by ts)
from MyTable;
-- 修改统计指标：max(c) -> min(c)，属于部分兼容修改。
-- sum(b)的计算结果不受影响。max(c)被认为删除，其对应的状态数据会被丢弃。
-- min(c)被认为是新增指标，其值在作业启动时开始计算，min(c) 对应的状态数据会被丢弃。
select a, b, c,
  sum(b) over (partition by a order by ts),
  min(c) over (partition by a order by ts)
from MyTable;
```

- 调整非Distinct的统计指标位置，该修改属于完全兼容修改。

```
-- 原始SQL。
select a, b, c,
  sum(b) over (partition by a order by ts),
  max(c) over (partition by a order by ts)
from MyTable
-- 调整统计指标位置：sum(b)、max(c)，属于完全兼容修改。
-- sum(b)、max(c) 的计算结果不受影响。
select a, b, c,
  max(c) over (partition by a order by ts),
  sum(b) over (partition by a order by ts)
from MyTable;
```

不兼容的修改

- Over Aggregate输入的Schema发生变化，该修改属于不兼容修改。

```
-- 原始SQL。
select a, b, c,
  sum(b) over (partition by a order by ts),
  max(c) over (partition by a order by ts)
from MyTable;
-- 新增输入字段d，当前修改属于不兼容修改。
select a, b, c, d,
  max(c) over (partition by a order by ts),
  sum(b) over (partition by a order by ts)
from MyTable;
-- 修改输入字段c，当前修改属于不兼容修改。
select a, b, c,
  max(c) over (partition by a order by ts),
  sum(b) over (partition by a order by ts)
from (
  select a, b, substring(c, 1, 5) as c from MyTable
);
```

- 修改Over窗口相关属性（Partition By、Order By、Bound Definition），该修改属于不兼容修改。

```
-- 原始SQL。
select a, b, c,
  max(c) over (partition by a order by ts asc rows between unbounded preceding and curren
t row)
from MyTable;
-- 修改partition key：a -> b，当前修改属于属于不兼容修改。
select a, b, c,
  max(c) over (partition by b order by ts asc rows between unbounded preceding and curren
t row)
from MyTable;
-- 修改order by：ts asc -> ts desc，当前修改属于不兼容修改。
select a, b, c,
  max(c) over (partition by a order by ts desc rows between unbounded preceding and curre
nt row)
from MyTable;
-- 修改bound definition：unbounded preceding -> 2 preceding，当前修改属于不兼容修改。
select a, b, c,
  max(c) over (partition by a order by ts asc rows between 2 preceding and current row)
from MyTable;
```

- 新增、删除、修改Distinct统计指标（Distinct Aggregate Function），该修改属于不兼容修改。

```
-- 原始SQL。
select a, b, c,
  max(c) over (partition by a order by ts)
from MyTable;
-- 新增Distinct统计指标count(distinct b)，当前修改属于不兼容修改。
select a, b, c,
  max(c) over (partition by b order by ts),
  count(distinct b) over (partition by b order by ts)
from MyTable;
```

9.3.2.2.4. Group Window Aggregate

本文为您介绍Group Window Aggregate修改的可兼容性和不可兼容性详情。

可兼容的修改

- 新增、删除、修改非Distinct的统计指标（Aggregate Function）。
  - 对于新增统计指标，属于部分兼容，从当前作业启动时开始累计。
  - 对于删除统计指标，属于完全兼容修改。删除的统计指标对应的状态数据会被丢弃。
  - 对于既有新增又有删除统计指标，则属于部分兼容修改。新增的统计指标从当前作业启动时开始累计，删除的统计指标对应的状态数据会被丢弃。
  - 对于修改统计指标，被视为删除和新增两个操作，属于部分兼容。新增的统计指标从当前作业启动时开始累计，删除的统计指标对应的状态数据会被丢弃。

> **说明** 对于未进行任何变更的统计指标，复用状态数据后计算的结果和基于历史数据运行的结果是一致的。

```
-- 原始SQL。
select a, sum(b), max(c)
from MyTable
group by a, tumble(ts, interval '1' minute);
-- 新增统计指标：count(c)，属于部分兼容修改。
-- sum(b)、max(c)的计算结果不受影响，count(c)的值在作业启动时从0开始累计。
select a, sum(b), max(c), count(c)
from MyTable
group by a, tumble(ts, interval '1' minute);
-- 删除统计指标：sum(b)，属于完全兼容修改。
-- max(c) 的计算结果不受影响
select a, max(c)
from MyTable
group by a, tumble(ts, interval '1' minute);
-- 修改统计指标：max(c) -> min(c)，属于部分兼容修改。
-- sum(b)的计算结果不受影响。max(c)被认为删除，其对应的状态数据会被丢弃。
-- min(c)被认为是新增指标，其值在作业启动时开始计算。
select a, sum(b), min(c)
from MyTable
group by a, tumble(ts, interval '1' minute);
```

- 调整非Distinct的统计指标位置，该修改属于完全兼容修改。

```
-- 原始SQL。
select a, sum(b), max(c)
from MyTable
group by a, tumble(ts, interval '1' minute);
-- 调整统计指标位置：sum(b)、max(c)，属于完全兼容修改。
-- sum(b)、max(c) 的计算结果不受影响。
select a, max(c), sum(b)
from MyTable
group by a, tumble(ts, interval '1' minute);
```

- 当非Distinct的统计指标涉及的字段有计算逻辑变化时，该统计指标被认为进行了修改操作，该修改属于部分兼容修改。

```
-- 原始SQL。
select a, sum(b), max(c)
from MyTable
group by a, tumble(ts, interval '1' minute);
-- 修改统计指标：max(c) -> max(substring(c, 1, 5))，属于部分兼容修改。
-- sum(b)的计算结果不受影响。max(c)被认为删除，其对应的状态数据会被丢弃。
-- max(substring(c, 1, 5))被认为新增指标，其值在作业启动时开始计算。
select a, sum(b), max(c)
from (select a, b, substring(c, 1, 5) as c from MyTable)
group by a, tumble(ts, interval '1' minute);
```

- 新增、删除window属性字段，该修改属于完全兼容修改。

```
-- 原始SQL。
select a,
  sum(b),
  max(c),
  tumble_start(ts, interval '1' minute) as window_start
from MyTable
  group by a, tumble(ts, interval '1' minute);
-- 新增window end属性，该修改属于完全兼容修改。
select a,
  sum(b),
  max(c),
  tumble_start(ts, interval '1' minute) as window_start,
  tumble_end(ts, interval '1' minute) as window_end
from MyTable
  group by a, tumble(ts, interval '1' minute);
-- 删除window start属性，该修改属于完全兼容修改。
select a,
  sum(b),
  max(c)
from MyTable
  group by a, tumble(ts, interval '1' minute);
```

不兼容的修改

- 修改window相关属性（window的类型，window的大小，时间相关属性），该修改属于不兼容修改。

```
-- 原始SQL。
select a,
  sum(b),
  max(c),
  tumble_start(ts, interval '1' minute) as window_start
from MyTable
  group by a, tumble(ts, interval '1' minute);
-- 修改window类型：tumble -> hop，当前修改属于不兼容修改。
select a,
  sum(b),
  max(c),
  hop_start(ts, interval '1' minute, interval '2' minute) as window_start
from MyTable
  group by a, hop(ts, interval '1' minute, interval '2' minute);
-- 修改window 大小：'1' minute -> '2' minute，当前修改属于不兼容修改。
select a,
  sum(b),
  max(c),
  tumble_start(ts, interval '2' minute) as window_start
from MyTable
  group by a, tumble(ts, interval '2' minute);
-- 修改时间相关属性：ts -> proctime，当前修改属于不兼容修改。
select a,
  sum(b),
  max(c),
  tumble_start(ts, interval '1' minute) as window_start
from MyTable
  group by a, tumble(proctime, interval '1' minute);
```

- 新增、删除、修改统计维度（group key）或者统计维度涉及字段的计算逻辑发生变化，该修改属于不兼容修改。

```
-- 原始SQL。
select a,
  sum(b),
  max(c),
  tumble_start(ts, interval '1' minute) as window_start
from MyTable
  group by a, tumble(ts, interval '1' minute);
-- 新增统计维度：d，当前修改属于不兼容修改。
select a,
  sum(b),
  max(c),
  tumble_start(ts, interval '1' minute) as window_start
from MyTable
  group by a, d, tumble(ts, interval '1' minute);
-- 其他示例可参考Group Aggregate修改group key示例。
```

- 新增、删除、修改Distinct的统计指标（distinct aggregate function）或者Distinct统计指标涉及字段的计算逻辑发生变化，该修改属于不兼容修改。

```
-- 原始SQL。
select a,
  sum(b),
  count(distinct b),
  max(c),
  count(distinct c),
  tumble_start(ts, interval '1' minute) as window_start
from MyTable
  group by a, tumble(ts, interval '1' minute);
-- 新增Distinct统计指标：count(distinct b)，当前修改属于不兼容修改。
select a,
  sum(b),
  max(c),
  count(distinct c),
  tumble_start(ts, interval '1' minute) as window_start
from MyTable
  group by a, tumble(ts, interval '1' minute);
-- 其他示例可参考Group Aggregate修改distinct统计指标示例。
```

- 删除了所有统计指标，该修改属于不兼容修改。（所有统计指标的状态数据都被丢弃，没有状态复用。）

```
-- 原始SQL。
select a,
  sum(b),
  count(distinct b),
  max(c),
  count(distinct c),
  tumble_start(ts, interval '1' minute) as window_start
from MyTable
  group by a, tumble(ts, interval '1' minute);
-- 删除所有统计指标：sum(b), max(c)，当前修改属于不兼容修改。
select a,
  tumble_start(ts, interval '1' minute) as window_start
from MyTable
  group by a, tumble(ts, interval '1' minute);
```

- 当添加、删除、修改了earlyFire或lateFire时，该修改属于不兼容修改。

9.3.2.2.5. Window TVF

本文为您介绍Window TVF修改的可兼容性和不可兼容性详情。

可兼容的修改

- 新增、删除、修改非Distinct的统计指标（Aggregate Function）。
  - 对于新增统计指标，属于部分兼容，从当前作业启动时开始累计。
  - 对于删除统计指标，属于完全兼容修改。删除的统计指标对应的状态数据会被丢弃。
  - 对于既有新增又有删除统计指标，则属于部分兼容修改。新增的统计指标从当前作业启动时开始累计，删除的统计指标对应的状态数据会被丢弃。
  - 对于修改统计指标，被视为删除和新增两个操作，属于部分兼容。新增的统计指标从当前作业启动时开始累计，删除的统计指标对应的状态数据会被丢弃。

> **说明** 对于未进行任何变更的统计指标，复用状态数据后计算的结果和基于历史数据运行的结果是一致的。

```
-- 原始SQL。
select a, sum(b), max(c)
from table (tumble(table MyTable, descriptor(ts), interval '1' minute))
group by a, window_start, window_end;
-- 新增统计指标：count(c)，属于部分兼容修改。
-- sum(b)、max(c)的计算结果不受影响，count(c)的值在作业启动时从0开始累计。
select a, sum(b), max(c), count(c)
from table (tumble(table MyTable, descriptor(ts), interval '1' minute))
group by a, window_start, window_end;
-- 删除统计指标：sum(b)，属于完全兼容修改。
-- max(c)的计算结果不受影响。
select a, max(c)
from table (tumble(table MyTable, descriptor(ts), interval '1' minute))
group by a, window_start, window_end;
-- 修改统计指标：max(c) -> min(c)，属于部分兼容修改。
-- sum(b)的计算结果不受影响。max(c)被认为删除，其对应的状态数据会被丢弃。
-- min(c)被认为是新增指标，其值在作业启动时开始计算。
select a, sum(b), min(c)
from table (tumble(table MyTable, descriptor(ts), interval '1' minute))
group by a, window_start, window_end;
```

- 调整非Distinct的统计指标位置，该修改属于完全兼容修改。

```
-- 原始SQL。
select a, sum(b), max(c)
from table (tumble(table MyTable, descriptor(ts), interval '1' minute))
group by a, window_start, window_end;
-- 调整统计指标位置：sum(b)、max(c)，属于完全兼容修改。
-- sum(b)、max(c) 的计算结果不受影响。
select a, max(c), sum(b)
from table (tumble(table MyTable, descriptor(ts), interval '1' minute))
group by a, window_start, window_end;
```

- 当非Distinct的统计指标涉及的字段有计算逻辑变化时，该统计指标被认为做了修改操作，该修改属于部分兼容修改。

```
-- 原始SQL。
select a, sum(b), max(c)
from table (tumble(table MyTable, descriptor(ts), interval '1' minute))
group by a, window_start, window_end;
-- 修改统计指标：max(c) -> max(substring(c, 1, 5))，属于部分兼容修改。
-- sum(b)的计算结果不受影响。max(c)被认为删除，其对应的状态数据会被丢弃。
-- max(substring(c, 1, 5)) 被认为新增指标，其值在作业启动时开始计算。
create temporary view MyView select a, b, substring(c, 1, 5) as c, ts from MyTable;
select a, sum(b), max(c)
from table (tumble(table MyView, descriptor(ts), interval '1' minute))
group by a, window_start, window_end;
```

- 新增、删除window属性字段，该修改属于完全兼容修改。

```
-- 原始SQL。
select a,
  sum(b),
  max(c),
  window_start
from table (tumble(table MyTable, descriptor(ts), interval '1' minute))
  group by a, window_start, window_end;
-- 新增window end属性，该修改属于完全兼容修改。
select a,
  sum(b),
  max(c),
  window_start,
  window_end
from table (tumble(table MyTable, descriptor(ts), interval '1' minute))
  group by a, window_start, window_end;
-- 删除window start属性，该修改属于完全兼容修改。
select a,
  sum(b),
  max(c)
from table (tumble(table MyTable, descriptor(ts), interval '1' minute))
  group by a, window_start, window_end;
```

不兼容的修改

- 修改window相关属性（window的类型，window的大小，时间相关属性），该修改属于不兼容修改。

```
--原始SQL。
select a,
  sum(b),
  max(c),
  window_start
from table (tumble(table MyTable, descriptor(ts), interval '1' minute))
  group by a, window_start, window_end;
-- 修改window类型：tumble -> hop，该修改属于不兼容修改。
select a,
  sum(b),
  max(c),
  hop_start(ts, interval '1' minute, interval '2' minute) as window_start
from MyTable
  group by a, hop(ts, interval '1' minute, interval '2' minute);
-- 修改window大小：'1' minute -> '2' minute，该修改属于不兼容修改。
select a,
  sum(b),
  max(c),
  tumble_start(ts, interval '2' minute) as window_start
from MyTable
  group by a, tumble(ts, interval '2' minute);
-- 修改时间相关属性：ts -> proctime，该修改属于不兼容修改。
select a,
  sum(b),
  max(c),
  tumble_start(ts, interval '1' minute) as window_start
from MyTable
  group by a, tumble(proctime, interval '1' minute);
```

- 新增、删除、修改统计维度（group key）或者统计维度涉及字段的计算逻辑发生变化，该修改属于不兼容修改。

```
-- 原始SQL。
select a,
  sum(b),
  max(c),
  tumble_start(ts, interval '1' minute) as window_start
from MyTable
  group by a, tumble(ts, interval '1' minute);
-- 新增统计维度：d，该修改属于不兼容修改。
select a,
  sum(b),
  max(c),
  tumble_start(ts, interval '1' minute) as window_start
from MyTable
  group by a, d, tumble(ts, interval '1' minute);
-- 其他示例可参考Group Aggregate修改group key示例。
```

- 新增、删除、修改Distinct统计指标（distinct aggregate function）或者Distinct统计指标涉及字段的计算逻辑发生变化，该修改属于不兼容修改。

```
-- 原始SQL。
select a,
  sum(b),
  count(distinct b),
  max(c),
  count(distinct c),
  tumble_start(ts, interval '1' minute) as window_start
from MyTable
  group by a, tumble(ts, interval '1' minute);
-- 新增Distinct统计指标：count(distinct b)，该修改属于不兼容修改。
select a,
  sum(b),
  max(c),
  count(distinct c),
  tumble_start(ts, interval '1' minute) as window_start
from MyTable
  group by a, tumble(ts, interval '1' minute);
-- 其他示例可参考Group Aggregate修改Distinct统计指标示例。
```

- 删除了所有统计指标，该修改属于不兼容修改。（所有统计指标的状态数据都被丢弃，没有状态复用。）

```
-- 原始SQL。
select a,
  sum(b),
  count(distinct b),
  max(c),
  count(distinct c),
  tumble_start(ts, interval '1' minute) as window_start
from MyTable
  group by a, tumble(ts, interval '1' minute);
-- 删除所有统计指标：sum(b), max(c)，该修改属于不兼容修改。
select a,
  tumble_start(ts, interval '1' minute) as window_start
from MyTable
  group by a, tumble(ts, interval '1' minute);
```

9.3.2.2.6. Deduplication

本文为您介绍Deduplication修改的可兼容性和不可兼容性详情。

可兼容的修改

当基于proctime做order by且按升序排序时（order by protime asc），可以修改除partition by key之前的字段，该修改属于完全兼容修改。

```
-- 原始SQL。
select a, b, c from (
  select *,
   row_number() over (partition by a order by proctime asc) as rk
   from MyTable)
where rk = 1;
-- 新增输入字段：d，该修改属于完全兼容修改。
select a, b, c, d from (
  select *,
   row_number() over (partition by a order by proctime asc) as rk
   from MyTable)
where rk = 1;
-- 删除输入字段：b，该修改属于完全兼容修改。
select a, c from (
  select *,
   row_number() over (partition by a order by proctime asc) as rk
   from MyTable)
where rk = 1;
-- 修改输入字段：c -> substring(c, 1, 5)，该修改属于完全兼容修改。
select a, b, c from (
  select *,
   row_number() over (partition by a order by proctime asc) as rk
   from (select a, b, substring(c, 1, 5) as c from MyTable))
where rk = 1;
```

不兼容的修改

- 新增、删除、修改partition by key或者partition by key涉及字段的计算逻辑发生变化，该修改属于不兼容修改。

```
-- 原始SQL。
select a, b, c from (
  select *,
    row_number() over (partition by a order by proctime) as rk
   from MyTable)
where rk = 1;
-- 新增partition by字段：d，该修改属于不兼容修改。
select a, b, c from (
  select *,
   row_number() over (partition by a, d order by proctime) as rk
   from MyTable)
where rk = 1;
-- 删除partition by字段：a，该修改属于不兼容修改。
select a, b, c from (
  select *,
   row_number() over (order by proctime) as rk
   from MyTable)
where rk = 1;
-- 修改partition by字段：a -> a + 1，该修改属于不兼容修改。
select a, b, c from (
  select *,
   row_number() over (partition by a order by proctime) as rk
   from (select a + 1 as a, b, c from MyTable))
where rk = 1;
```

- 修改order by相关属性（排序字段和方向），该修改属于不兼容修改。

```
-- 原始SQL。
select a, b, c from (
  select *,
   row_number() over (partition by a order by proctime asc) as rk
   from MyTable)
where rk = 1;
-- 修改order key：protime -> ts，当前修改属于不兼容修改。
select a, b, c from (
  select *,
   row_number() over (partition by a order by ts asc) as rk
   from MyTable)
where rk = 1;
-- 修改order：asc -> desc，当前修改属于不兼容修改。
select a, b, c from (
  select *,
   row_number() over (partition by a order by proctime desc) as rk
   from MyTable)
where rk = 1;
```

- 当基于rowtime做order by或基于proctime进行order by且按降序排序时（order by protime desc）时，修改了输入字段，则该修改属于不兼容修改。

```
-- 原始SQL。
select a, b, c from (
  select *,
   row_number() over (partition by a order by ts) as rk
   from MyTable)
where rk = 1;
-- 新增输入字段：d，当前修改属于不兼容修改。
select a, b, c, d from (
  select *,
   row_number() over (partition by a order by ts) as rk
   from MyTable)
where rk = 1;
-- 删除输入字段：c，当前修改属于不兼容修改。
select a, b from (
  select *,
   row_number() over (partition by a order by ts) as rk
   from MyTable)
where rk = 1;
-- 输入字段：c -> substring(c, 1, 5)，当前修改属于不兼容修改。
select a, b, c from (
  select *,
   row_number() over (partition by a order by ts) as rk
 from (select a, b, substring(c, 1, 5) as c from MyTable))
where rk = 1;
```

9.3.2.2.7. Top-N

本文为您介绍Top-N修改的可兼容性和不可兼容性详情。

可兼容的修改

是否输出rank number的值，不影响兼容性，该修改属于完全兼容修改。

```
-- 原始SQL。
select a, b, c from (
  select *,
   row_number() over (partition by a order by c) as rk
   from MyTable)
where rk < 3;
-- 输出rank number字段：rk，该修改属于完全兼容修改。
select a, b, c, rk from (
  select *,
   row_number() over (partition by a order by c) as rk
   from MyTable)
where rk < 3;
```

不兼容的修改

- 新增、删除、修改partition by key或者partition by key涉及字段的计算逻辑发生变化，该修改属于不兼容修改。

```
-- 原始SQL。
select a, b, c from (
  select *,
   row_number() over (partition by a order by c) as rk
   from MyTable)
where rk < 3;
-- 新增partition by字段：d，该修改属于不兼容修改。
select a, b, c from (
  select *,
   row_number() over (partition by a, d order by c) as rk
   from MyTable)
where rk < 3;
-- 删除partition by字段：a，该修改属于不兼容修改。
select a, b, c from (
  select *,
   row_number() over (order by c) as rk
   from MyTable)
where rk < 3;
-- 修改partition by字段：a -> a + 1，该修改属于不兼容修改。
select a, b, c from (
  select *,
   row_number() over (partition by a order by c) as rk
   from (select a + 1 as a, b, c from MyTable))
where rk < 3;
```

- 修改order by相关属性（排序字段和方向），该修改属于不兼容修改。

```
-- 原始SQL。
select a, b, c from (
  select *,
   row_number() over (partition by a order by c) as rk
   from MyTable)
where rk < 3;
-- 修改order key：c -> b，当前修改属于不兼容修改。
select a, b, c from (
  select *,
   row_number() over (partition by a order by b) as rk
   from MyTable)
where rk < 3;
-- 修改order key：c -> substring(c, 1, 5)，当前修改属于不兼容修改。
select a, b, c from (
  select *,
   row_number() over (partition by a order by c) as rk
   from (select a, b, substring(c, 1, 5) as c from MyTable))
where rk < 3;
-- 修改order：asc -> desc，当前修改属于不兼容修改。
select a, b, c from (
  select *,
   row_number() over (partition by a order by c desc) as rk
   from MyTable)
where rk < 3;
```

- 修改rank range的值（Top-N中N的值），则该修改属于不兼容修改。

```
-- 原始SQL。
select a, b, c from (
  select *,
   row_number() over (partition by a order by c) as rk
   from MyTable)
where rk < 3;
-- 修改rank range：3 -> 5，当前修改属于不兼容修改。
select a, b, c from (
  select *,
   row_number() over (partition by a order by c) as rk
   from MyTable)
where rk < 5;
```

- 修改了输入字段，则该修改属于不兼容修改。

```
-- 原始SQL。
select a, b, c from (
  select *,
   row_number() over (partition by a order by c) as rk
   from MyTable)
where rk < 3;
-- 新增输入字段：d，当前修改属于不兼容修改。
select a, b, c, d from (
  select *,
   row_number() over (partition by a order by c) as rk
   from MyTable)
where rk < 3;
-- 删除输入字段：b，当前修改属于不兼容修改。
select a, c from (
  select *,
   row_number() over (partition by a order by c) as rk
   from MyTable)
where rk < 3;
-- 修改输入字段：b -> b + 1，当前修改属于不兼容修改。
select a, b, c from (
  select *,
   row_number() over (partition by a order by c) as rk
   from (select a, b + 1 as b, c from MyTable))
where rk < 3;
```

9.3.2.2.8. Window Top-N

Window TOP-N需要同时遵循Window TVF和Top-N两者的修改要求，支持的兼容性修改较少。本文为您介绍Window Top-N修改的可兼容性和不可兼容性详情。

可兼容的修改

- 新增、删除window属性字段，该修改属于完全兼容修改。

```
-- 原始SQL。
select a, b, c, window_start from (
  select *,
      row_number() over (partition by b, window_start, window_end order by c) as rk
  from (
    select a,
      sum(b) as b,
      max(c) as c,
        window_start,
        window_end
    from table (tumble(table MyTable, descriptor(ts), interval '1' minute))
      group by a, window_start, window_end)
  ) where rk < 3;
-- 新增window end，该修改属于完全兼容修改。
select a, b, c, window_start, window_end from (
  select *,
      row_number() over (partition by b, window_start, window_end order by c) as rk
  from (
    select a,
      sum(b) as b,
      max(c) as c,
        window_start,
        window_end
    from table (tumble(table MyTable, descriptor(ts), interval '1' minute))
      group by a, window_start, window_end)
  ) where rk < 3;
```

- 是否输出rank number的值，不影响兼容性，该修改属于完全兼容修改。

```
-- 原始SQL。
select a, b, c, window_start from (
  select *,
      row_number() over (partition by b, window_start, window_end order by c) as rk
  from (
    select a,
      sum(b) as b,
      max(c) as c,
        window_start,
        window_end
    from table (tumble(table MyTable, descriptor(ts), interval '1' minute))
      group by a, window_start, window_end)
  ) where rk < 3;
-- 输入rank number值：rk，该修改属于完全兼容修改。
select a, b, c, window_start, rk from (
  select *,
      row_number() over (partition by b, window_start, window_end order by c) as rk
  from (
    select a,
      sum(b) as b,
      max(c) as c,
        window_start,
        window_end
    from table (tumble(table MyTable, descriptor(ts), interval '1' minute))
      group by a, window_start, window_end)
  ) where rk < 3;
```

不兼容的修改

- 修改window相关属性（window 的类型，window 的大小，时间相关属性），该修改属于不兼容修改。

  示例可参见Window TVF修改window相关属性示例，详情请参见不兼容的修改。

- 新增、删除、修改统计维度（group key）或者统计维度涉及字段的计算逻辑发生变化，该修改属于不兼容修改。

  示例可参见Window TVF修改group key示例，详情请参见不兼容的修改。

- 新增、删除、修改统计指标，该修改属于不兼容修改。TOP-N的输入发生变化，该修改属于不兼容修改。

```
-- 原始SQL。
select a, b, c, window_start from (
  select *,
      row_number() over (partition by b, window_start, window_end order by c) as rk
  from (
    select a,
      sum(b) as b,
      max(c) as c,
        window_start,
        window_end
    from table (tumble(table MyTable, descriptor(ts), interval '1' minute))
      group by a, window_start, window_end)
  ) where rk < 3;
-- 新增统计指标：min(d) as d，属于不兼容修改。
-- 该修改会导致TOP-N的输入发生变化。
select a, b, c, d, window_start from (
  select *,
      row_number() over (partition by b, window_start, window_end order by c) as rk
  from (
    select a,
      sum(b) as b,
      max(c) as c,
      min(d) as d,
        window_start,
        window_end
    from table (tumble(table MyTable, descriptor(ts), interval '1' minute))
      group by a, window_start, window_end)
  ) where rk < 3;
```

- 新增、删除、修改partition by key或者partition by key涉及字段的计算逻辑发生变化，该修改属于不兼容修改。

  示例可参见Top N修改partition by key示例，详情请参见不兼容的修改。

- 修改order by相关属性（排序字段和方向），该修改属于不兼容修改。

  示例可参见Top N修改ordery by key示例，详情请参见不兼容的修改。

- 修改rank range的值（Top N中N 的值），则该修改属于不兼容修改。

  示例可参见Top N修改rank range示例，详情请参见不兼容的修改。

9.3.2.2.9. Join

本文为您介绍Regular Join、Interval Join、Temporal Join和Lookup Join修改的可兼容性和不可兼容性详情。

- Regular Join

  当前不支持Regular Join相关算子的状态迁移，因此任何Join相关的修改，状态都不兼容。修改包括Join类型、Join condition和Join输入的Schema等。

- Interval Join

  当前不支持Interval Join相关算子的状态迁移，因此任何Join相关的修改，状态都不兼容。修改包括Join类型、Join condition、interval的大小和Join输入的Schema等。

- Temporal Join

当前不支持temporal Join相关算子的状态迁移，因此任何Join相关的修改，状态都不兼容。修改包括Join类型、Join condition和Join输入的Schema等。

- Lookup Join

  Lookup Join的计算不涉及到状态，因此它的修改不会影响自身算子的状态兼容，但它的修改会影响其输出算子的兼容性。

9.3.2.2.10. Pattern Recognition

本文为您介绍Pattern Recognition修改的可兼容性和不可兼容性详情。

当前不支持CEP相关算子的状态迁移，因此任何pattern recongnition相关的修改，状态都不兼容。Pattern Recognition相关修改包括pattern、measures、after、partition by和order by等的定义。

9.3.2.2.11. Where

本文为您介绍Where修改的可兼容性和不可兼容性详情。

Where的计算不涉及到状态，因此它的修改不会影响自身算子的状态兼容，但它的修改会影响其输出算子的兼容性。

如果需要通过修改Where条件来修正作业数据（例如过滤脏数据），则可以在**作业开发**页面**高级配置**页签的**更多Flink配置**中添加如下配置。添加以下配置后，如果您修改了Where条件后，则系统会忽略该修改对下游算子状态的兼容性判断的影响。

```
table.optimizer.state-compatibility.ignore-filter=true
```

> **说明** 该参数默认值为false。

```
-- 原始SQL。
select a, sum(b), max(c) from MyTable group by a;
-- 添加Where并且设置table.optimizer.state-compatibility.ignore-filter=false，该修改属于不兼容修改。
select a, sum(b), max(c) from (select * from MyTable where a > 10) group by a;
-- 添加Where并且设置table.optimizer.state-compatibility.ignore-filter=true，该修改属于完全兼容修改。
select a, sum(b), max(c) from (select * from MyTable where a > 10) group by a;
```

9.3.2.2.12. Union All

本文为您介绍Union All修改的可兼容性和不可兼容性详情。

可兼容的修改

调整Union All的输入Query的顺序，对下游有状态算子属于完全兼容修改。

```
create table MyTable2 (
  a int,
  b bigint,
  c varchar,
  d int
);
-- 原始SQL。
select a, sum(b), max(c) from (
  select a, b, c from MyTable union all select a, b, c from MyTable2
) group by a;
--调整Union All的输入Query的顺序，该修改属于完全兼容修改。
select a, sum(b), max(c) from (
  select a, b, c from MyTable2 union all select a, b, c from MyTable
) group by a;
```

不兼容的修改

添加或删除Union All的输入Query，对下游有状态算子属于不兼容性修改。

```
create table MyTable2 (
  a int,
  b bigint,
  c varchar,
  d int
);
-- 原始SQL。
select a, sum(b), max(c) from (
  select a, b, c from MyTable union all select a, b, c from MyTable2
) group by a;
create table MyTable3 (
  a int,
  b bigint,
  c varchar,
  d int
);
-- 添加Union All的输入Query，该修改属于不兼容修改。
select a, sum(b), max(c) from (
  select a, b, c from MyTable
  union all
  select a, b, c from MyTable2
  union all
  select a, b, c from MyTable3
) group by a;
```

9.3.2.2.13. Temporal Sort

本文为您介绍Temporal Sort修改的可兼容性和不可兼容性详情。

可兼容的修改

当仅基于proctime进行order by，且按升序排序时（order by protime asc），可变更select字段，该修改属于完全兼容修改。

```
-- 原始SQL。
select a, b, c from MyTable order by proctime asc;
-- 新增输入字段：d，该修改属于完全兼容修改。
select a, b, c, d from MyTable order by proctime asc;
```

不兼容的修改

- 当基于rowtime进行order by时，不支持变更select字段，该修改属于不兼容修改。

```
-- 原始SQL。
select a, b, c from MyTable order by ts asc;
-- 新增输入字段：d，该修改属于不兼容修改。
select a, b, c, d from MyTable order by ts asc;
```

- 修改order by相关属性（排序字段和方向），该修改属于不兼容修改。

```
-- 原始SQL。
select a, b, c from MyTable order by ts asc;
-- 新增order by字段：a，该修改属于不兼容修改。
select a, b, c from MyTable order by ts asc, a desc;
```

9.3.2.2.14. Limit

本文为您介绍Limit修改的可兼容性和不可兼容性详情。

当前不支持limit相关算子的状态迁移，因此任何limit相关的修改，状态都不兼容。Limit相关修改包括limit number、offset number和limit输入的schema等。

9.3.2.3. 修改Source

本文为您介绍Source修改的可兼容性和不可兼容性详情。

背景信息

当前Create Table DDL的Schema部分的修改，需要根据使用该表的Query的变化来判断整体的兼容性。With属性中除了Connector的类型，其他的属性暂不进行兼容性检查。

可兼容的修改

字段的修改，不影响下游算子的兼容性，该修改属于完全或部分兼容修改。

```
-- 原始SQL。
create table MyTable (
  a int,
  b bigint,
  c varchar
) with (
  'connector'='datagen'
);
select a, sum(b), max(c) from MyTable group by a;
-- 新增字段：d，该修改属于完全兼容修改。
-- Query中未使用到字段d。
create table MyTable (
  a int,
  b bigint,
  c varchar,
  d int
) with (
  'connector'='datagen'
);
select a, sum(b), max(c) from MyTable group by a;
-- 修改字段：b -> b as d + 1，该修改属于部分兼容修改。
-- max(c) 的计算结果不受影响。b列发生变化，因此原始的sum(b)被认为删除。
-- 其对应的状态数据会被丢弃，新的sum(b)被认为新增指标，其值在作业启动时开始计算。
create table MyTable (
  a int,
  d bigint,
  c varchar,
  b as d + 1
) with (
  'connector'='datagen'
);
select a, sum(b), max(c) from MyTable group by a;
```

不兼容的修改

- 修改Connector类型，该修改属于不兼容修改。

```
-- 原始SQL。
create table MyTable (
  a int,
  b bigint,
  c varchar
) with (
  'connector'='datagen'
);
select a, sum(b), max(c) from MyTable group by a;
-- Connnector类型变化：datagen -> kafka，该修改属于不兼容修改。
create table MyTable (
  a int,
  b bigint,
  c varchar
) with (
  'connector'='kafka',
  ...
);
select a, sum(b), max(c) from MyTable group by a;
```

- 修改表名，该修改属于不兼容修改。

```
-- 原始SQL。
create table MyTable (
  a int,
  b bigint,
  c varchar
) with (
  'connector'='datagen'
);
select a, sum(b), max(c) from MyTable group by a;
-- 表名类型变化：MyTable -> MyTable2，该修改属于不兼容修改。
create table MyTable2 (
  a int,
  b bigint,
  c varchar
) with (
  'connector'='datagen'
);
select a, sum(b), max(c) from MyTable2 group by a;
```

- 字段的修改，且影响下游算子的兼容性，该修改属于不兼容修改。

```
-- 原始SQL。
create table MyTable1 (
  a int,
  b bigint
) with (
  'connector'='datagen'
);
create table MyTable2 (
  c int,
  d bigint
) with (
  'connector'='datagen'
);
select * from MyTable1 join MyTable2 on c = d;
-- MyTable2中新增e字段，该修改属于不兼容修改。
-- join要求其输入的字段不能修改。
create table MyTable1 (
  a int,
  b bigint
) with (
  'connector'='datagen'
);
create table MyTable2 (
  c int,
  d bigint,
  e varchar
) with (
  'connector'='datagen'
);
-- 原始SQL。
select * from MyTable1 join MyTable2 on c = d;
```

9.3.2.4. 修改Sink

本文为您介绍Sink修改的可兼容性和不可兼容性详情。

可兼容的修改

- 删除多路Sink的某路Sink，该修改属于完全兼容修改。

```
-- 原始SQL。
CREATE TABLE MyTable (
  a int,
  b bigint,
  c varchar
);
CREATE TABLE MySink1 (
  a int,
  b bigint,
  c varchar
);
CREATE TABLE MySink2 (
  a int,
  b bigint,
  c varchar
);
INSERT INTO MySink1 SELECT a, sum(b), max(c) FROM MyTable GROUP BY a;
INSERT INTO MySink2 SELECT a, b, c FROM MyTable WHERE a > 10;
-- 删除MySink1对应的Query，该修改属于完全兼容修改。
-- 该Query中的group aggregate对应的状态会被丢弃。
INSERT INTO MySink2 SELECT a, b, c FROM MyTable WHERE a > 10;
-- 删除MySink2对应的Query，该修改属于完全兼容修改。
INSERT INTO MySink1 SELECT a, sum(b), max(c) FROM MyTable GROUP BY a;
```

- 新增Sink且不带有状态的Query，该修改属于完全兼容修改。

```
-- 原始SQL。
CREATE TABLE MyTable (
  a int,
  b bigint,
  c varchar
);
CREATE TABLE MySink1 (
  a int,
  b bigint,
  c varchar
);
INSERT INTO MySink1 SELECT a, sum(b), max(c) FROM MyTable GROUP BY a;
-- 新增无状态的Query，该修改属于完全兼容修改。
CREATE TABLE MySink2 (
  a int,
  b bigint,
  c varchar
);
INSERT INTO MySink1 SELECT a, sum(b), max(c) FROM MyTable GROUP BY a;
INSERT INTO MySink2 SELECT a, b, c FROM MyTable WHERE a > 10;
```

- 默认情况下，Sink被认为是无状态的算子（大部分的Sink Connector没有状态）。因此修改Sink表名、Connector类型、WITH属性都是兼容修改。

```
-- 原始SQL。
CREATE TABLE MyTable (
  a int,
  b bigint,
  c varchar
);
CREATE TABLE MySink (
  a int,
  b bigint,
  c varchar
) WITH (
  'connector' = 'print'
);
INSERT INTO MySink SELECT a, sum(b), max(c) FROM MyTable GROUP BY a;
-- 修改表名、Connector属性等，该修改属于完全兼容修改。
CREATE TABLE MySink2 (
  a int,
  b bigint,
  c varchar
) WITH (
  'connector' = 'kafka',
  ...
);
INSERT INTO MySink2 SELECT a, sum(b), max(c) FROM MyTable GROUP BY a;
```

不兼容的修改

- 新增Sink且带有状态的Query，该修改属于不兼容修改。

```
-- 原始SQL。
CREATE TABLE MyTable (
  a int,
  b bigint,
  c varchar
);
CREATE TABLE MySink1 (
  a int,
  b bigint,
  c varchar
);
INSERT INTO MySink1 SELECT a, sum(b), max(c) FROM MyTable GROUP BY a;
-- 新增有状态的Query，该修改属于不兼容修改。
CREATE TABLE MySink2 (
  b bigint,
  a int,
  c varchar
);
INSERT INTO MySink1 SELECT a, sum(b), max(c) FROM MyTable GROUP BY a;
INSERT INTO MySink2 SELECT b, sum(a), max(c) FROM MyTable GROUP BY b;
```

- 如果需要将Sink作为有状态算子时，例如设置table.optimizer.state-compatibility.ignore-sink=false，修改Sink表名、Connector类型都是不兼容修改。

```
-- 原始SQL。
CREATE TABLE MyTable (
  a int,
  b bigint,
  c varchar
);
CREATE TABLE MySink (
  a int,
  b bigint,
  c varchar
) WITH (
  'connector' = 'print'
);
INSERT INTO MySink SELECT a, sum(b), max(c) FROM MyTable GROUP BY a;
-- 修改表名，设置table.optimizer.state-compatibility.ignore-sink=false，该修改属于不兼容修改。
CREATE TABLE MySink2 (
  a int,
  b bigint,
  c varchar
) WITH (
  'connector' = 'print'
);
INSERT INTO MySink2 SELECT a, sum(b), max(c) FROM MyTable GROUP BY a;
-- 修改Connector属性，设置table.optimizer.state-compatibility.ignore-sink=false，该修改属于不兼容修改。
create table MySink (
  a int,
  b bigint,
  c varchar
) WITH (
  'connector' = 'kafka',
  ...
);
INSERT INTO MySink SELECT a, sum(b), max(c) FROM MyTable GROUP BY a;
```

9.3.2.5. 其他限制

本文为您介绍修改SQL除Query、Source和Sink之外的其他限制。

- 当前作业的Flink版本和产生Checkpoint或Savepoint对应作业的Flink版本要求相同，不同的Flink版本当前不支持状态兼容性检查。
- 当前作业的依赖和产生Checkpoint或Savepoint对应作业的依赖要求能兼容。当前系统无法识别自定义Connector、自定义函数依赖的状态的兼容性变化，需要您自己保证。
- 当前只支持单个的修改，包括只修改会影响状态计算的Where条件、只修改统计指标（Aggregate Function）、只修改Sink。

```
-- 原始SQL。
create table MyTable (
  a int,
  b bigint,
  c varchar
);
create table MySink (
  a int,
  b bigint,
  c varchar
) with (
  'connector' = 'print'
);
insert into MySink select a, sum(b), max(c) from MyTable group by a;
-- 修改Sink表：MySink -> MySink2，和修改统计函数：max(c) -> min(c)，该修改属于无法判断的修改。
create table MySink2 (
  a int,
  b bigint,
  c varchar
) with (
  'connector' = 'print'
);
insert into MySink2 select a, sum(b), min(c) from MyTable group by a;
-- 添加Where条件：a > 10，和修改统计函数：max(c) -> min(c)，该修改属于无法判断的修改。
insert into MySink
select a, sum(b), min(c) from (
  select * from MyTable where a > 10
) group by a;
```

- 不支持新增全新状态的Query，当前修改属于不兼容修改。

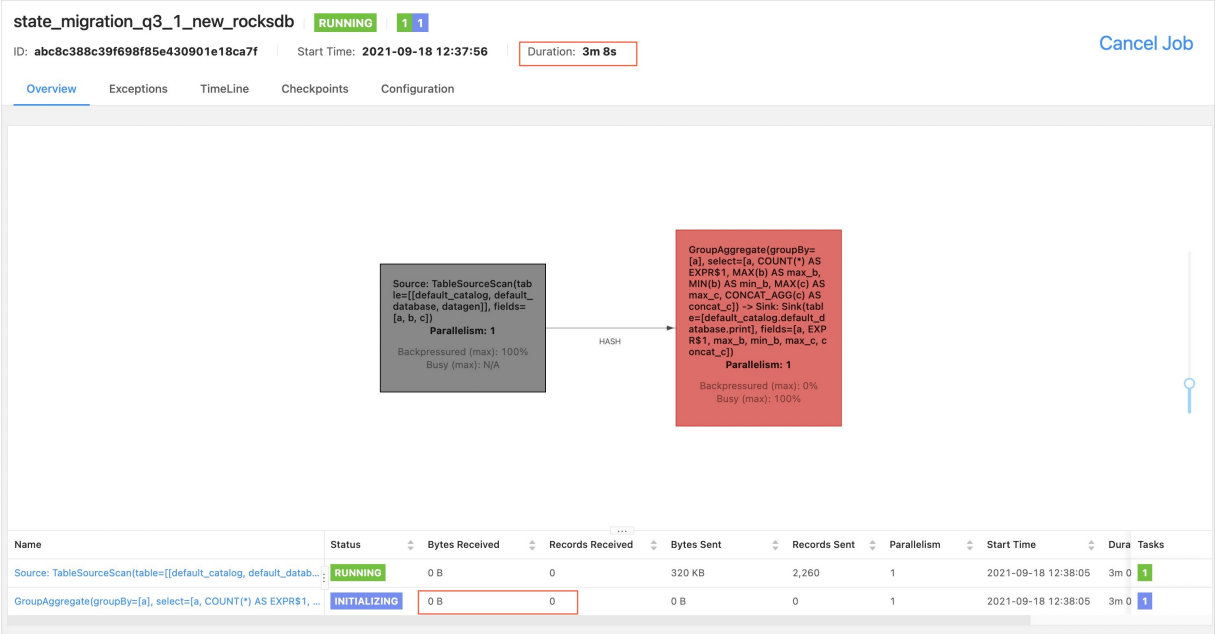
```
-- 原始SQL。
create table MyTable (
  a int,
  b bigint,
  c varchar
);
create table MySink (
  a int,
  b bigint,
  c varchar
) with (
  'connector' = 'print'
);
insert into MySink select a, b, c from MyTable;
-- 新增group aggregate query，当前修改属于不兼容修改。
insert into MySink select a, sum(b), min(c) from MyTable group by a;
```

9.3.3. 状态迁移

为了让您选择的状态数据能适配新作业，则需要对状态数据进行迁移。Flink全托管支持RocksDB和Gemini两种StateBackend，本文为您介绍RocksDB和Gemini的迁移效率和作业表现方面的区别。

RocksDB

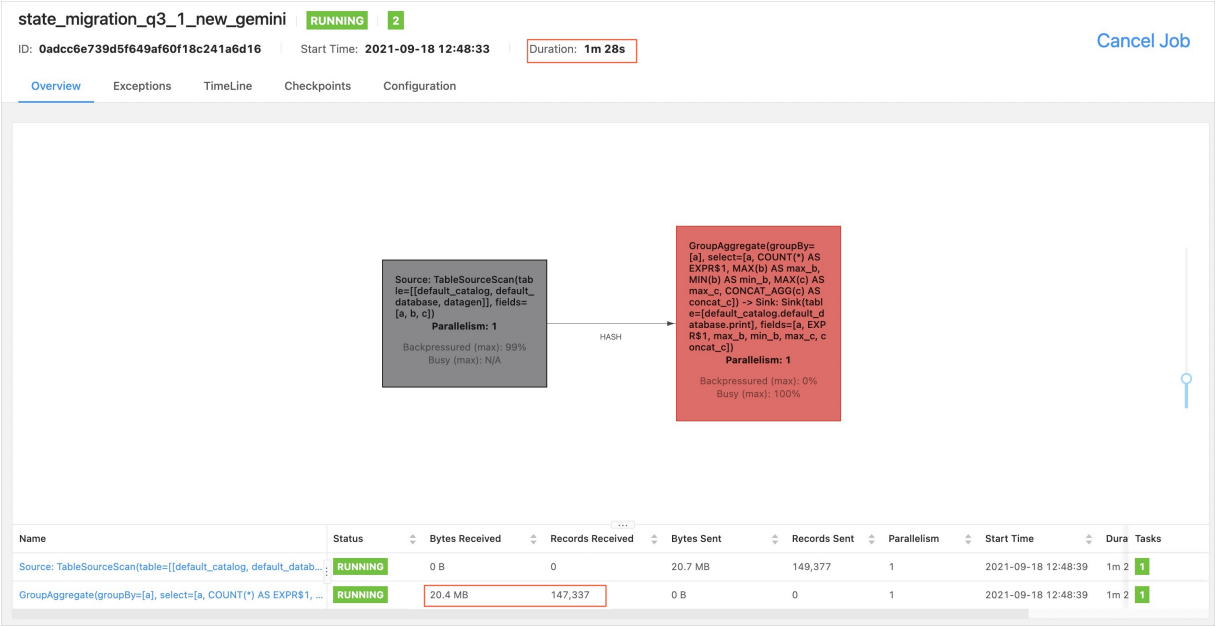
RocksDB是在作业启动时，会对所有的状态数据进行一次全量的迁移。此时整个作业已经启动，处于RUNNING状态，而需要发生数据迁移的算子处于INITIALIZING状态，并且不会消费和处理任何数据。等该算子所有数据都迁移完成，算子的状态会变为RUNNING状态，表示所有状态迁移完成，开始正常消费数据。



> **说明** 因为修改了Aggregate Function，所以在作业启动时，RocksDB会进行全量状态迁移。上图中的GroupAggregate算子处于INITIALIZING状态，此时该算子不能处理数据。

Gemini

与RocksDB在作业启动时进行状态全量迁移的行为不同，Gemini是在作业运行时按需迁移，即只有访问到状态中某条数据时，这条数据才会发生迁移。整个作业启动后，作业处于RUNNING状态，需要发生状态迁移的算子会很快从INITIALIZING状态变为RUNNING状态，并开始正常消费数据。随着状态迁移的进行，TPS会逐渐恢复到正常值，表示所有状态迁移完成。相比RocksDB，Gemini的整体迁移时间更短。



说明 在Gemini和RocksDB对应完全相同的SQL和状态数据时，Gemini很快从INITIALIZING状态（加载状态数据）变为RUNNING状态，并开始处理数据。

9.4. Flink Datastream参考

9.4.1. MaxCompute DataStream Connector

如果您通过DataStream的方式读写MaxCompute数据，则需要使用MaxCompute DataStream Connector连接Flink全托管。本文为您介绍如何在Flink全托管控制台上使用MaxCompute DataStream Connector来读写MaxCompute数据。

背景信息

MaxCompute DataStream Connector基于MaxCompute SQL Connector代码开发，连接配置与对应SQL配置完全相同，SQL配置详情请参见全量MaxCompute源表WITH参数、增量MaxCompute源表WITH参数和MaxCompute结果表WITH参数。

Maven中央库中已经放置了VVR Connector，以供您在作业开发时直接使用。您可以通过以下任何一种方式来使用DataStream Connector：

- （推荐）直接将Connector作为项目依赖打进作业JAR包
- 上传Connector JAR包到Flink全托管开发控制台后，填写配置信息

（推荐）直接将Connector作为项目依赖打进作业JAR包

1. 在Maven项目的pom.xml文件中添加以下配置以引用SNAPSHOT仓库。

```
<repositories>
  <repository>
    <id>oss.sonatype.org-snapshot</id>
    <name>OSS Sonatype Snapshot Repository</name>
    <url>http://oss.sonatype.org/content/repositories/snapshots</url>
    <releases>
      <enabled>false</enabled>
    </releases>
    <snapshots>
      <enabled>true</enabled>
    </snapshots>
  </repository>
  <repository>
    <id>apache.snapshots</id>
    <name>Apache Development Snapshot Repository</name>
    <url>https://repository.apache.org/content/repositories/snapshots/</url>
    <releases>
      <enabled>false</enabled>
    </releases>
    <snapshots>
      <enabled>true</enabled>
    </snapshots>
  </repository>
</repositories>
```

2. 检查您的settings.xml配置文件中是否存在 `<mirrorOf>*</mirrorOf>` 配置。

   如果存在 `<mirrorOf>*</mirrorOf>` 配置，则需要将此配置改为 `<mirrorOf>*,!oss.sonatype.org-snapshot,!apache.snapshots</mirrorOf>` 。

   修改的目的是为了避免SNAPSHOT仓库被覆盖，因为mirrorOf中只使用星号（*）会导致第一步中配置的两个repository被覆盖。

3. 在作业的Maven POM文件中添加您需要的Connector作为项目依赖。
   - MaxCompute源表和结果表依赖

     ```
     <dependency>
         <groupId>com.alibaba.ververica</groupId>
         <artifactId>ververica-connector-odps</artifactId>
         <version>${connector.version}</version>
     </dependency>
     ```

   - MaxCompute增量源表依赖

     ```
     <dependency>
         <groupId>com.alibaba.ververica</groupId>
         <artifactId>ververica-connector-continuous-odps</artifactId>
         <version>${connector.version}</version>
     </dependency>
     ```

   每个Connector版本对应的Connector类型可能不同，建议您使用最新版本。Connector版本、VVR/Flink版本和Connector类型的对应关系请参见Connector列表。完整的依赖信息请参见MaxCompute示例代码中的pom.xml文件。

注意

- 您需要在SNAPSHOT仓库（*oss.sonatype.org*）查找带SNAPSHOT的Connector版本，在Maven中央库（*search.maven.org*）上会查找不到。
- 在使用多个Connector时，请注意META-INF目录需要Merge，即在pom.xml文件中添加如下代码。

```
<transformers>
    <!-- The service transformer is needed to merge META-INF/services files -
->
    <transformer implementation="org.apache.maven.plugins.shade.resource.Serv
icesResourceTransformer"/>
    <transformer implementation="org.apache.maven.plugins.shade.resource.Apac
heNoticeResourceTransformer">
        <projectName>Apache Flink</projectName>
        <encoding>UTF-8</encoding>
    </transformer>
</transformers>
```

4. 修改MaxCompute连接配置和Schema信息。

类别	说明
MaxCompute连接配置	替换成您MaxCompute的链接配置信息。与对应SQL配置完全相同，详情请参见： ○ 全量MaxCompute源表WITH参数 ○ 增量MaxCompute源表WITH参数 ○ MaxCompute结果表WITH参数
Schema信息	表的模式信息。

- MaxCompute源表

  VVR提供了SourceFunction的实现类ODPSStreamSource来读取MaxCompute表数据。以下为构建MaxCompute Source读取分区数据的示例。

```
//MaxCompute连接配置。
Configuration conf = new Configuration();
conf.setString("endpoint", "yourEndpoint");
conf.setString("tunnelEndpoint","YourTunnelEndpoint");
conf.setString("project", "yourProjectName");
conf.setString("tablename", "yourTableName");
conf.setString("accessid", "yourAccessKeyID");
conf.setString("accesskey", "yourAccessKeySecret");
conf.setString("partition", "ds=yourPartitionName");
//Schema信息。
TableSchema schema = org.apache.flink.table.api.TableSchema.builder()
        .field("a", DataTypes.STRING())
        .field("b", DataTypes.STRING())
        .build();
StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment()
;
//构建MaxCompute Source Function。
ODPSStreamSource odpsSource =
    new OdpsSourceBuilder(schema, conf).buildSourceFunction();
DataStreamSource<RowData> source = env.addSource(odpsSource);
source.addSink(new PrintSinkFunction<>());
env.execute("odps source");
```

说明

- 构建类OdpsSourceBuilder不在VVR Connector依赖中，是MaxCompute示例代码中提供的以方便构建ODPSStreamSource。
- MaxCompute源表是以RowData格式（GenericRowData）读取数据的。

- MaxCompute增量源表

  VVR提供了SourceFunction的实现类ContinuousODPSStreamSource来增量读取MaxCompute表数据。以下为构建MaxCompute Source增量读取分区数据的示例。

```
//MaxCompute连接配置。
Configuration conf = new Configuration();
conf.setString("endpoint", "yourEndpoint");
conf.setString("tunnelEndpoint","YourTunnelEndpoint");
conf.setString("project", "yourProjectName");
conf.setString("tablename", "yourTableName");
conf.setString("accessid", "yourAccessKeyID");
conf.setString("accesskey", "yourAccessKeySecret");
conf.setString("startPartition", "ds=yourPartitionName");
//Schema信息。
TableSchema schema = org.apache.flink.table.api.TableSchema.builder()
        .field("a", DataTypes.STRING())
        .field("b", DataTypes.STRING())
        .build();
StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment()
;
//构建Continuous MaxCompute Source Function。
ContinuousODPSStreamSource continuousOdpsSource =
        new OdpsSourceBuilder(schema, conf).buildContinuousOdpsSource();
DataStreamSource<RowData> source = env.addSource(continuousOdpsSource);
source.addSink(new PrintSinkFunction<>());
env.execute("continuous odps source");
```

说明

- 构建类OdpsSourceBuilder不在VVR Connector依赖中，是示例代码中提供的以方便构建ODPSStreamSource。
- MaxCompute增量源表是以RowData格式（GenericRowData）读取数据的。

- MaxCompute结果表

  VVR提供了OutputFormat的实现类OdpsOutputFormat，基于MaxCompute TableTunnel实现对表或分区进行数据上传。以下为构建OdpsOutputFormat上传数据到特定分区的代码示例。

```
//MaxCompute连接配置。
DescriptorProperties properties = new DescriptorProperties();
properties.putString("endpoint", "yourEndpoint");
properties.putString("tunnelEndpoint","YourTunnelEndpoint");
properties.putString("project", "yourProjectName");
properties.putString("tablename", "yourTableName");
properties.putString("accessid", "yourAccessKeyID");
properties.putString("accesskey", "yourAccessKeySecret");
properties.putString("partition", "ds=yourPartitionName");
//Schema信息。
TableSchema schema = TableSchema.builder()
        .field("a", DataTypes.STRING())
        .field("b", DataTypes.STRING())
        .build();
StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment()
;
//MaxCompute Sink在Checkpoint时提交已上传的数据，来尽量实现exactly-once。
env.enableCheckpointing(10, CheckpointingMode.EXACTLY_ONCE);
// 构建MaxCompute Sink Function。
TupleOutputFormatSinkFunction<Row> odpsSink = new TupleOutputFormatSinkFunction<>(
        new OdpsOutputFormat(schema, properties));
env.fromCollection(
        Arrays.asList(Row.of("aa", "111"), Row.of("bb", "222"))
).map(new MapFunction<Row, Tuple2<Boolean, Row>>() {
    @Override
    public Tuple2<Boolean, Row> map(Row row) throws Exception {
        return Tuple2.of(true, row);
    }
}).addSink(odpsSink);
env.execute("odps sink");
```

说明

- MaxCompute Sink在Checkpoint时以及任务结束时才会提交已经上传的数据（数据在提交后才可见）。如果您要保证数据实时可见，则需要开启Checkpoint，更详细内容参见实现原理。
- MaxCompute Sink接收的数据格式是Tuple2<Boolean, Row>，因此在写入前需要通过MapFunction将数据转化为该格式。

上传Connector JAR包到Flink全托管开发控制台后，填写配置信息

1. 登录实时计算管理控制台。
2. 在**Flink全托管**页签，单击目标工作空间**操作**列下的**控制台**。
3. 在左侧导航栏，单击**资源上传**。
4. 单击**上传资源**，选择您要上传的目标Connector的JAR包。

   您可以上传您自己开发的Connector，也可以上传Flink全托管产品提供的Connector。Flink全托管产品提供的Connector官方JAR包的下载地址，请参见Connector列表。
5. 在目标作业开发页面**附加依赖文件**项，选择目标Connector的JAR包。

9.4.2. DataHub DataStream Connector

如果您通过DataStream的方式读写DataHub数据，则需要使用DataHub DataStream Connector连接Flink全托管。本文为您介绍如何在Flink全托管控制台上使用DataHub DataStream Connector来读写DataHub数据。

背景信息

DataHub DataStream Connector基于DataHub SQL Connector代码开发，连接配置与对应SQL配置完全相同，SQL配置详情请参见DataHub源表WITH参数和DataHub结果表WITH参数。

Maven中央库中已经放置了VVR Connector，以供您在作业开发时直接使用。您可以通过以下任何一种方式来使用DataStream Connector：

- （推荐）直接将Connector作为项目依赖打进作业JAR包
- 上传Connector JAR包到Flink全托管开发控制台后，填写配置信息

（推荐）直接将Connector作为项目依赖打进作业JAR包

1. 在Maven项目的pom.xml文件中添加以下配置以引用SNAPSHOT仓库。

```
<repositories>
  <repository>
    <id>oss.sonatype.org-snapshot</id>
    <name>OSS Sonatype Snapshot Repository</name>
    <url>http://oss.sonatype.org/content/repositories/snapshots</url>
    <releases>
      <enabled>false</enabled>
    </releases>
    <snapshots>
      <enabled>true</enabled>
    </snapshots>
  </repository>
  <repository>
    <id>apache.snapshots</id>
    <name>Apache Development Snapshot Repository</name>
    <url>https://repository.apache.org/content/repositories/snapshots/</url>
    <releases>
      <enabled>false</enabled>
    </releases>
    <snapshots>
      <enabled>true</enabled>
    </snapshots>
  </repository>
</repositories>
```

2. 检查您的settings.xml配置文件中是否存在 `<mirrorOf>*</mirrorOf>` 配置。

   如果存在 `<mirrorOf>*</mirrorOf>` 配置，则需要将此配置改为 `<mirrorOf>*,!oss.sonatype.org-snapshot,!apache.snapshots</mirrorOf>` 。

   修改的目的是为了避免SNAPSHOT仓库被覆盖，因为mirrorOf中只使用星号（*）会导致第一步中配置的两个repository被覆盖。

3. 在作业的Maven POM文件中添加您需要的Connector作为项目依赖。

DataHub源表和结果表相同。

```
<dependency>
    <groupId>com.alibaba.ververica</groupId>
    <artifactId>ververica-connector-datahub</artifactId>
    <version>${connector.version}</version>
</dependency>
```

每个Connector版本对应的Connector类型可能不同，建议您使用最新版本。Connector版本、VVR/Flink版本和Connector类型的对应关系请参见Connector列表。完整的依赖信息请参见DataHub示例代码中的pom.xml文件。

> **注意**
>
> - 您需要在SNAPSHOT仓库（*oss.sonatype.org*）查找带SNAPSHOT的Connector版本，在Maven中央库（*search.maven.org*）上会查找不到。
> - 在使用多个Connector时，请注意META-INF目录需要Merge，即在pom.xml文件中添加如下代码。
>
> ```
> <transformers>
>  <!-- The service transformer is needed to merge META-INF/services files -->
>  <transformer implementation="org.apache.maven.plugins.shade.resource.ServicesResourceTransformer"/>
>  <transformer implementation="org.apache.maven.plugins.shade.resource.ApacheNoticeResourceTransformer">
>  <projectName>Apache Flink</projectName>
>  <encoding>UTF-8</encoding>
>  </transformer>
> </transformers>
> ```

4. 修改DataHub连接配置。

   DataHub的连接配置信息，与对应SQL配置完全相同，SQL配置详情请参见：

   - DataHub源表WITH参数
   - DataHub结果表WITH参数

   DataHub源表和结果表的示例代码如下：

   - DataHub源表

     VVR提供了SourceFunction的实现类DatahubSourceFunction来读取DataHub表数据。以下为读取DataHub表数据的示例。

```
StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment()
;
env.setParallelism(1);
--DataHub连接配置。
DatahubSourceFunction datahubSource =
    new DatahubSourceFunction(
    <yourEndPoint>,
    <yourProjectName>,
    <yourTopicName>,
    <yourSubId>,
    <yourAccessId>,
    <yourAccessKey>,
    <yourStartTime>,
    <yourEndTime>
    );
datahubSource.setRequestTimeout(30 * 1000);
datahubSource.enableExitAfterReadFinished();
env.addSource(datahubSource)
    .map((MapFunction<RecordEntry, Tuple2<String, Long>>) this::getStringLongTuple2)
    .print();
env.execute();
private Tuple2<String, Long> getStringLongTuple2(RecordEntry recordEntry) {
    Tuple2<String, Long> tuple2 = new Tuple2<>();
    TupleRecordData recordData = (TupleRecordData) (recordEntry.getRecordData());
    tuple2.f0 = (String) recordData.getField(0);
    tuple2.f1 = (Long) recordData.getField(1);
    return tuple2;
}
```

- DataHub结果表

  VVR提供了OutputFormatSinkFunction的实现类DatahubSinkFunction将数据写入DataHub。以下为将数据写入DataHub的示例。

```
StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment()
;
--DataHub连接配置。
env.generateSequence(0, 100)
    .map((MapFunction<Long, RecordEntry>) aLong -> getRecordEntry(aLong, "default:"))
    .addSink(
    new DatahubSinkFunction<>(
       <yourEndPoint>,
       <yourProjectName>,
       <yourTopicName>,
       <yourSubId>,
       <yourAccessId>,
       <yourAccessKey>
       );
env.execute();
private RecordEntry getRecordEntry(Long message, String s) {
    RecordSchema recordSchema = new RecordSchema();
    recordSchema.addField(new Field("f1", FieldType.STRING));
    recordSchema.addField(new Field("f2", FieldType.BIGINT));
    recordSchema.addField(new Field("f3", FieldType.DOUBLE));
    recordSchema.addField(new Field("f4", FieldType.BOOLEAN));
    recordSchema.addField(new Field("f5", FieldType.TIMESTAMP));
    recordSchema.addField(new Field("f6", FieldType.DECIMAL));
    RecordEntry recordEntry = new RecordEntry();
    TupleRecordData recordData = new TupleRecordData(recordSchema);
    recordData.setField(0, s + message);
    recordData.setField(1, message);
    recordEntry.setRecordData(recordData);
    return recordEntry;
}
```

上传Connector JAR包到Flink全托管开发控制台后，填写配置信息

1. 登录实时计算管理控制台。
2. 在**Flink全托管**页签，单击目标工作空间**操作**列下的**控制台**。
3. 在左侧导航栏，单击**资源上传**。
4. 单击**上传资源**，选择您要上传的目标Connector的JAR包。

   您可以上传您自己开发的Connector，也可以上传Flink全托管产品提供的Connector。Flink全托管产品提供的Connector官方JAR包的下载地址，请参见Connector列表。
5. 在目标作业开发页面右侧的**高级配置**面板中的**附加依赖文件**项，选择目标Connector的JAR包。

9.4.3. Kafka DataStream Connector

如果您通过DataStream的方式以精确一次（Exactly-once）的语义在Kafka Topic中读写数据，则需要使用Kafka DataStream Connector连接Flink全托管。本文为您介绍如何在Flink全托管控制台上使用Kafka DataStream Connector来读写Kafka数据。

背景信息

Maven中央库中已经放置了VVR Connector，以供您在作业开发时直接使用。您可以通过以下任何一种方式来使用DataStream Connector：

- （推荐）直接将Connector作为项目依赖打进作业JAR包
- 上传Connector JAR包到Flink全托管开发控制台后，填写配置信息

（推荐）直接将Connector作为项目依赖打进作业JAR包

1. 在Maven项目的pom.xml文件中添加以下配置以引用SNAPSHOT仓库。

```
<repositories>
  <repository>
    <id>oss.sonatype.org-snapshot</id>
    <name>OSS Sonatype Snapshot Repository</name>
    <url>http://oss.sonatype.org/content/repositories/snapshots</url>
    <releases>
      <enabled>false</enabled>
    </releases>
    <snapshots>
      <enabled>true</enabled>
    </snapshots>
  </repository>
  <repository>
    <id>apache.snapshots</id>
    <name>Apache Development Snapshot Repository</name>
    <url>https://repository.apache.org/content/repositories/snapshots/</url>
    <releases>
      <enabled>false</enabled>
    </releases>
    <snapshots>
      <enabled>true</enabled>
    </snapshots>
  </repository>
</repositories>
```

2. 检查您的settings.xml配置文件中是否存在 `<mirrorOf>*</mirrorOf>` 配置。

   如果存在 `<mirrorOf>*</mirrorOf>` 配置，则需要将此配置改为 `<mirrorOf>*,!oss.sonatype.org-snapshot,!apache.snapshots</mirrorOf>` 。

   修改的目的是为了避免SNAPSHOT仓库被覆盖，因为mirrorOf中只使用星号（*）会导致第一步中配置的两个repository被覆盖。

3. 在作业的Maven POM文件中添加您需要的Connector作为项目依赖。

```
<dependency>
    <groupId>com.alibaba.ververica</groupId>
    <artifactId>ververica-connector-kafka</artifactId>
    <version>${connector.version}</version>
</dependency>
```

   每个Connector版本对应的Connector类型可能不同，建议您使用最新版本。Connector版本、VVR/Flink版本和Connector类型的对应关系请参见Connector列表。完整的依赖信息请参见Kafka示例代码中的pom.xml文件。

注意

- 您需要在SNAPSHOT仓库（*oss.sonatype.org*）查找带SNAPSHOT的Connector版本，在Maven中央库（*search.maven.org*）上会查找不到。
- 在使用多个Connector时，请注意META-INF目录需要Merge，即在pom.xml文件中添加如下代码。

```
<transformers>
    <!-- The service transformer is needed to merge META-INF/services files -
->
    <transformer implementation="org.apache.maven.plugins.shade.resource.Serv
icesResourceTransformer"/>
    <transformer implementation="org.apache.maven.plugins.shade.resource.Apac
heNoticeResourceTransformer">
        <projectName>Apache Flink</projectName>
        <encoding>UTF-8</encoding>
    </transformer>
</transformers>
```

4. 构建Kafka Source或者Kafka Producer。
   - Kafka Source

     Kafka Source提供了构建类来创建Kafka Source的实例。我们将通过以下示例代码为您介绍如何构建Kafka Source来消费input-topic最早位点的数据，消费组名称为my-group，并将Kafka消息体反序列化为字符串。

```
KafkaSource<String> source = KafkaSource.<String>builder()
    .setBootstrapServers(brokers)
    .setTopics("input-topic")
    .setGroupId("my-group")
    .setStartingOffsets(OffsetsInitializer.earliest())
    .setValueOnlyDeserializer(new SimpleStringSchema())
    .build();
env.fromSource(source, WatermarkStrategy.noWatermarks(), "Kafka Source");
```

在构建KafkaSource时，必须指定以下参数。

参数	说明
BootstrapServers	Kafka Broker地址，通过setBootstrapServers(String)方法配置。
GroupId	消费者组ID，通过setGroupId(String)方法配置。

参数	说明

Topics或Partition	订阅的Topic或Partition名称。Kafka Source提供了以下三种Topic或Partition的订阅方式：
	■ Topic列表，订阅Topic列表中所有Partition。
	`KafkaSource.builder().setTopics("topic-a","topic-b")`
	■ 正则表达式匹配，订阅与正则表达式所匹配的Topic下的所有Partition。
	`KafkaSource.builder().setTopicPattern("topic.*")`
	■ Partition列表，订阅指定的Partition。
	`final HashSet<TopicPartition> partitionSet = new HashSet<> (Arrays.asList( new TopicPartition("topic-a", 0), // Partition 0 of topic "topic-a" new TopicPartition("topic-b", 5))); // Partition 5 of topic "topic-b" KafkaSource.builder().setPartitions(partitionSet)`

Deserializer	解析Kafka消息的反序列化器。 反序列化器通过setDeserializer(KafkaRecordDeserializationSchema)来指定，其中KafkaRecordDeserializationSchema定义了如何解析Kafka的ConsumerRecord。如果只解析Kafka消息中的消息体（Value）的数据，则您可以通过以下任何一种方式实现： ■ 使用Flink提供的KafkaSource构建类中的setValueOnlyDeserializer(DeserializationSchema)方法，其中DeserializationSchema定义了如何解析Kafka消息体中的二进制数据。 ■ 使用Kafka提供的解析器，包括多种实现类。例如，您可以使用StringDeserializer来将Kafka消息体解析成字符串。 `import org.apache.kafka.common.serialization.StringDeserializer; KafkaSource.<String>builder() .setDeserializer(KafkaRecordDeserializationSchema.valueOnly (StringDeserializer.class));` **说明** 如果需要完整地解析ConsumerRecord，则需要您自己实现KafkaRecordDeserializationSchema接口。

在使用Kafka DataStream Connector时，您还需要了解以下Kafka属性：

- 起始消费位点

  Kafka Source能够通过位点初始化器（OffsetsInitializer）来指定从不同的偏移量开始消费。内置的位点初始化器包括以下内容。

位点初始化器	代码设置
从最早位点开始消费。	`KafkaSource.builder().setStartingOffsets(OffsetsInitializer.earliest())`
从最末尾位点开始消费。	`KafkaSource.builder().setStartingOffsets(OffsetsInitializer.latest())`
从时间戳大于等于指定时间的数据开始消费。	`KafkaSource.builder().setStartingOffsets(OffsetsInitializer.timestamp(1592323200L))`
从消费组提交的位点开始消费，如果提交位点不存在，使用最早位点。	`KafkaSource.builder().setStartingOffsets(OffsetsInitializer.committedOffsets(OffsetResetStrategy.EARLIEST))`
从消费组提交的位点开始消费，不指定位点重置策略。	`KafkaSource.builder().setStartingOffsets(OffsetsInitializer.committedOffsets())`

> **说明**
> - 如果以上内置的初始化器不能满足需求，您也可以自己实现自定义的位点初始化器。
> - 如果您未指定位点初始化器，则默认使用OffsetsInitializer.earliest()，即最早位点。

- 流模式和批模式

  Kafka Source支持流式和批式两种运行模式。默认情况下，Kafka Source设置为以流模式运行，因此作业永远不会停止，直到Flink作业失败或被取消。如果要配置Kafka Source在批模式下运行，您可以使用setBounded(OffsetsInitializer)指定停止偏移量，当所有分区都达到其停止偏移量时，Kafka Source会退出运行。

> **说明** 通常，流模式下Kafka Source没有停止偏移量。为了方便对代码进行调试，流模式下您也可以使用 setUnbounded(OffsetsInitializer) 指定停止偏移量。请留意指定流模式和批模式停止偏移量的方法名（setUnbounded 和 setBounded）是不同的。

- 动态分区检查

  为了在不重启Flink作业的情况下，处理Topic扩容或新建Topic等场景，您可以在提供的Topic或Partition订阅模式下，启用动态分区检查功能。

> **说明** 默认不开启动态分区检查功能。如果您需要开启此功能，则需要将partition.discovery.interval.ms设置为非负值，即可启用动态分区检查功能。代码示例如下。
>
> ```
> KafkaSource.builder()
>  .setProperty("partition.discovery.interval.ms", "10000") // 每10秒检查一次新分区。
> ```

- 事件时间和水印

  Kafka Source默认使用Kafka消息中的时间戳作为事件时间。您可以自定义水印策略（Watermark Strategy）以从消息中提取事件时间，并向下游发送水印。

```
env.fromSource(kafkaSource, new CustomWatermarkStrategy(), "Kafka Source With Custo
m Watermark Strategy")
```

  如果您需要了解自定义水印策略（Watermark Strategy），请参见Generating Watermarks。

- 消费位点提交

  Kafka Source在Checkpoint完成时，提交当前的消费位点，以保证Flink的Checkpoint状态和Kafka Broker上的提交位点一致。如果未开启Checkpoint，Kafka Source依赖于Kafka Consumer内部的位点定时自动提交逻辑，自动提交功能由enable.auto.commit和 auto.commit.interval.ms两个Kafka Consumer配置项进行配置。

  > **说明** Kafka Source不依赖于Broker上提交的位点来恢复失败的作业。提交位点只是为了上报Kafka Consumer和消费组的消费进度，以在Broker端进行监控。

- 其他属性

  除了上述属性之外，您还可以使用setProperties(Properties) 和setProperty(String, String) 为Kafka Source和Kafka Consumer设置任意属性。KafkaSource通常有以下配置项。

配置项	说明
client.id.prefix	指定用于Kafka Consumer的客户端ID前缀。
partition.discovery.interval.ms	定义Kafka Source检查新分区的时间间隔。 **说明** partition.discovery.interval.ms会在批模式下被覆盖为-1。
register.consumer.metrics	指定是否在Flink中注册Kafka Consumer的指标。

配置项	说明
其他Kafka Consumer配置	Kafka Consumer的配置详情，请参见Apache Kafka。 **注意** Kafka Connector会强制覆盖部分您手动配置的参数项，覆盖详情如下： ■ key.deserializer始终被覆盖为ByteArrayDeserializer。 ■ value.deserializer始终被覆盖为ByteArrayDeserializer。 ■ auto.offset.reset.strategy被覆盖为OffsetsInitializer#getAutoOffsetResetStrategy()。

以下示例为您展示如何配置Kafka Consumer，以使用PLAIN作为SASL机制并提供JAAS配置。

```
KafkaSource.builder()
    .setProperty("sasl.mechanism", "PLAIN")
    .setProperty("sasl.jaas.config", "org.apache.kafka.common.security.plain.PlainL
oginModule required username=\"username\" password=\"password\";")
```

- 监控

  Kafka Source在Flink中注册指标，用于监控和诊断。

  - 指标范围

    Kafka source reader的所有指标都注册在KafkaSourceReader指标组下，KafkaSourceReader是operator指标组的子组。与特定主题分区相关的指标注册在KafkaSourceReader.topic.<topic_name>.partition.<partition_id>指标组中。

    例如Topic "my-topic"、分区1的当前消费位点（currentOffset）注册在<some_parent_groups>.operator.KafkaSourceReader.topic.my-topic.partition.1.currentOffset。成功提交位点的次数（commitsSucceeded）注册在<some_parent_groups>.operator.KafkaSourceReader.commitsSucceeded。

  - 指标列表

指标名称	描述	范围
currentOffset	当前消费位点	TopicPartition
committedOffset	当前提交位点	TopicPartition
commitsSucceeded	成功提交的次数	KafkaSourceReader
commitsFailed	失败的提交次数	KafkaSourceReader

  - Kafka Consumer指标

    Kafka Consumer的指标注册在指标组KafkaSourceReader.KafkaConsumer。例如Kafka Consumer指标records-consumed-total注册在<some_parent_groups>.operator.KafkaSourceReader.KafkaConsumer.records-consumed-total。

    您可以使用配置项register.consumer.metrics配置是否注册Kafka消费者的指标。默认此选项设置为true。对于Kafka Consumer的指标，您可以参见Apache Kafka。

- Kafka Producer

  Flink Kafka Producer可以实现将流数据写入一个或多个Kafka Topic。

```
DataStream<String> stream = ...
Properties properties = new Properties();
properties.setProperty("bootstrap.servers", "localhost:9092");
FlinkKafkaProducer<String> myProducer = new FlinkKafkaProducer<>(
        "my-topic",                  // target topic
        new SimpleStringSchema(),    // serialization schema
        properties,                  // producer config
        FlinkKafkaProducer.Semantic.EXACTLY_ONCE); // fault-tolerance
stream.addSink(myProducer);
```

  您需要配置以下参数。

参数	说明
Topic	数据写入的默认Topic名称。

参数	说明
数据序列化	用于将数据序列化的SerializationSchema或KafkaSerializationSchema。 Flink Kafka Producer需要知道如何将Java或Scala对象转换为二进制数据，KafkaSerializationSchema允许用户指定如何进行数据的序列化。ProducerRecord<byte[], byte[]> serialize(T element, Long timestamp)方法会在每条数据流入的时候调用，来生成ProducerRecord写入Kafka。 用户可以对每条数据如何写入Kafka进行细粒度地控制。通过ProducerRecord可以进行以下操作： ■ 设置写入的Topic名称。 ■ 定义消息键（Key）。 ■ 指定数据写入的Partition。
Kafka客户端属性	bootstrap.servers必填，以逗号分隔的Kafka Broker列表。
容错语义	启用Flink的Checkpoint后，Flink Kafka Producer可以保证精确一次的语义。除了启用Flink的Checkpoint外，您还可以通过Semantic参数来指定不同的容错语义，Semantic参数详情如下： ■ Semantic.NONE：Flink不做任何保证，数据可能会丢失或重复。 ■ Semantic.AT_LEAST_ONCE（默认设置）：保证不会丢失任何数据，但可能会重复。 ■ Semantic.EXACTLY_ONCE：使用Kafka事务提供精确一次的语义保证。 说明 使用EXACTLY_ONCE语义时，需要注意的事项请参见EXACTLY_ONCE语义注意事项。

上传Connector JAR包到Flink全托管开发控制台后，填写配置信息

1. 登录实时计算管理控制台。
2. 在**Flink全托管**页签，单击目标工作空间**操作**列下的**控制台**。
3. 在左侧导航栏，单击**资源上传**。
4. 单击**上传资源**，选择您要上传的目标Connector的JAR包。

   您可以上传您自己开发的Connector，也可以上传Flink全托管产品提供的Connector。Flink全托管产品提供的Connector官方JAR包的下载地址，请参见Connector列表。
5. 在目标作业开发页面**附加依赖文件**项，选择目标Connector的JAR包。

EXACTLY_ONCE语义注意事项

- 当使用事务写入Kafka时，请为所有消费Kafka数据的应用配置isolation.level参数。该参数取值如下：
  - read_committed：只读取已提交的数据。
  - read_uncommitted（默认值）：可以读取未提交的数据。
- Semantic.EXACTLY_ONCE模式依赖于在从某个Checkpoint恢复后，且在该Checkpoint开始之前所提交的事务。如果Flink作业崩溃与完成重启之间的时间大于Kafka的事务超时时间，则会有数据丢失，因为Kafka会自动中止超过超时时间的事务。因此，请根据您的预期停机时间适当地配置您的事务超时。

Kafka Broker默认的transaction.max.timeout.ms设置为15分钟，Producer设置的事务超时不能超过Broker指定的时间。Flink Kafka Producer默认会将Kafka Producer配置中的transaction.timeout.ms属性设置为1小时，因此在使用Semantic.EXACTLY_ONCE模式前，需要增加Broker端的transaction.max.timeout.ms值。

- Semantic.EXACTLY_ONCE模式为在Flink Kafka Producer实例中使用一个固定大小的Kafka Producer池。每个Checkpoint会使用池中的一个Kafka Producer。如果并发的Checkpoint数量超过Producer池的大小，Flink Kafka Producer会抛出异常并使整个作业失败。请相应地配置Producer池大小和最大并发的Checkpoint数量。
- Semantic.EXACTLY_ONCE会尽可能清除阻止Consumer从Kafka topic中读取数据的残留事务。但如果Flink作业在第一个 Checkpoint之前就出现故障，则在重启该作业后并不会保留重启前Producer池的信息。因此，在第一个Checkpoint完成之前缩减Flink作业的并行度是不安全的，即使要缩减并行度，也不能小于FlinkKafkaProducer.SAFE_SCALE_DOWN_FACTOR。
- 对于在read_committed模式下运行的Kafka Consumer，任何未结束（既未中止也未完成）的事务都将阻塞对该Kafka Topic的所有读取。如果您按照以下步骤进行了操作：
  i. 用户开启事务1并通过该事务写入了一些数据。
  ii. 用户开启事务2并通过该事务写入了更多的数据。
  iii. 用户提交事务2。

  即使来自事务2的数据已经提交，在事务1提交或中止之前，事务2的数据对消费者是不可见的。因此：
  - 在Flink作业正常工作期间，您可以预期写入Kafka topic的数据会有延迟，约为Checkpoint的平均间隔。
  - 在Flink作业失败的情况下，该作业正在写入的Topic将会阻塞Consumer的读取，直到作业重新启动或事务超时。

9.4.4. MySQL CDC DataStream Connector

MySQL CDC DataStream Connector支持从全量读取到增量读取的无缝切换，保证数据不重不丢。本文为您介绍如何在Flink全托管控制台上使用MySQL CDC DataStream Connector来读取MySQL数据。

背景信息

类别	详情
功能介绍	MySQL CDC Connector是一个Flink的Source Connector，会先读取数据库的历史全量数据，并平滑切换到Binlog读取上，保证无论是否发生异常，数据不重不丢，实现Exactly Once语义。MySQL CDC Connector支持并发地读取全量数据，通过增量快照算法实现了全程无锁和断点续传。
实现原理	Source在启动时会扫描全表，将表按照主键分成多个chunk，并使用增量快照算法逐个读取每个chunk的数据。作业会周期性执行Checkpoint，记录下已经完成的chunk，以保证在发生Failover时，只需要继续读取未完成的chunk。当chunk全部读取完后，会从之前获取的Binlog位点读取增量的变更记录。Flink作业会继续周期性执行Checkpoint，记录下Binlog位点，当作业发生Failover，便会从之前记录的Binlog位点继续处理，从而实现Exactly Once语义。更详细的增量快照算法，请参见MySQL CDC Connector。

类别	详情
核心特性	• 支持读取全量和增量Binlog数据，方便您一站式初始化全量数据和获取变化数据。 • 支持并发读取全量数据，扩展性能水平。 • 支持从全量读取到增量读取的无缝切换，自动缩容，节省计算资源。 • 支持无锁读取全量数据，不影响在线业务。 • 全量阶段读取支持断点续传，提升数据容错能力。

前提条件

- MySQL和VVP的网络连通。
- MySQL服务器配置如下：
  - MySQL版本为5.7或者8.0.X。
  - 已开启了Binlog。
  - Binlog格式已设置为ROW。
  - binlog_row_image已设置为FULL。
- 已在MySQL配置文件中配置了交互超时或等待超时参数。
- 已创建MySQL用户，并授予了SELECT 、 SHOW DATABASES 、REPLICATION SLAVE和REPLICATION CLIENT权限。

> **说明** 以上配置需要在RDS MySQL、PolarDB MySQL或者自建MySQL上操作，详情请参见配置MySQL。

使用限制

MySQL CDC Connector支持读取的MySQL版本为5.7和8.0.X。

注意事项

每个用于读取Binlog的MySQL数据库客户端都应该具有唯一的ID，称为server-id。MySQL服务器将使用此ID来维护网络连接和Binlog位点。因此，如果不同的作业共享相同的server-id，则可能导致从错误的Binlog位点读取数据，造成数据缺失、重复和MySQL服务器的CPU陡增，影响线上业务稳定性等问题。因此建议您在每个DataStream作业的每个MySqlSource数据源，都需要显式配置上不同的server-id。配置不同的Server ID代码示例如下。

```
MySqlSource.builder().serverId("56000")
```

> **说明** 如果开启增量快照读取数据，则server-id配置项需要配置成与作业并发匹配的Server ID范围。例如作业并发度为4，则需要配置为serverId("56001-56004")。

操作流程

Maven中央库中已经内置了VVR Connector，以供您在作业开发时直接使用。您可以通过以下步骤来使用MySQL CDC DataStream Connector：

1. 步骤一：准备DataStream作业开发环境
2. 步骤二：开发DataStream作业

3. 步骤三：打包并提交DataStream作业

步骤一：准备DataStream作业开发环境

推荐您直接将Connector作为项目依赖打进作业JAR包中。您可以按照如下步骤操作：

1. 在Maven项目的pom.xml文件中，添加以下配置以引用SNAPSHOT仓库。

```
<repositories>
  <repository>
    <id>oss.sonatype.org-snapshot</id>
    <name>OSS Sonatype Snapshot Repository</name>
    <url>http://oss.sonatype.org/content/repositories/snapshots</url>
    <releases>
      <enabled>false</enabled>
    </releases>
    <snapshots>
      <enabled>true</enabled>
    </snapshots>
  </repository>
  <repository>
    <id>apache.snapshots</id>
    <name>Apache Development Snapshot Repository</name>
    <url>https://repository.apache.org/content/repositories/snapshots/</url>
    <releases>
      <enabled>false</enabled>
    </releases>
    <snapshots>
      <enabled>true</enabled>
    </snapshots>
  </repository>
</repositories>
```

2. 检查您的settings.xml配置文件中是否存在 `<mirrorOf>*</mirrorOf>` 配置。

   如果存在 `<mirrorOf>*</mirrorOf>` 配置，则需要将此配置改为 `<mirrorOf>*,!oss.sonatype.org-snapshot,!apache.snapshots</mirrorOf>` 。

   修改的目的是为了避免SNAPSHOT仓库被覆盖，因为mirrorOf中只使用星号（*）会导致第一步中配置的两个repository被覆盖。

3. 在作业的Maven项目的pom.xml文件中，添加您需要的Connector作为项目依赖。

```
<dependency>
    <groupId>com.ververica</groupId>
    <artifactId>flink-connector-mysql-cdc</artifactId>
    <version>2.1.0</version>
</dependency>
```

   每个Connector版本对应的Connector类型可能不同，建议您使用最新版本。Connector版本、VVR/Flink版本和Connector类型的对应关系请参见Connector列表。

注意

- 您需要在SNAPSHOT仓库（*oss.sonatype.org*）查找带SNAPSHOT的Connector版本，在Maven中央库（*search.maven.org*）上会查找不到。
- 在使用多个Connector时，请注意META-INF目录需要Merge，即在pom.xml文件中添加如下代码。

```
<transformers>
    <!-- The service transformer is needed to merge META-INF/services files -
->
    <transformer implementation="org.apache.maven.plugins.shade.resource.Serv
icesResourceTransformer"/>
    <transformer implementation="org.apache.maven.plugins.shade.resource.Apac
heNoticeResourceTransformer">
        <projectName>Apache Flink</projectName>
        <encoding>UTF-8</encoding>
    </transformer>
</transformers>
```

步骤二：开发DataStream作业

创建DataStream API程序并使用MySqlSource。代码示例如下。

```
import org.apache.flink.api.common.eventtime.WatermarkStrategy;
import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
import com.ververica.cdc.debezium.JsonDebeziumDeserializationSchema;
import com.ververica.cdc.connectors.mysql.source.MySqlSource;
public class MySqlSourceExample {
  public static void main(String[] args) throws Exception {
    MySqlSource<String> mySqlSource = MySqlSource.<String>builder()
        .hostname("yourHostname")
        .port(yourPort)
        .databaseList("yourDatabaseName") // set captured database
        .tableList("yourDatabaseName.yourTableName") // set captured table
        .username("yourUsername")
        .password("yourPassword")
        .deserializer(new JsonDebeziumDeserializationSchema()) // converts SourceRecord to
JSON String
        .build();
    StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
    // enable checkpoint
    env.enableCheckpointing(3000);
    env
      .fromSource(mySqlSource, WatermarkStrategy.noWatermarks(), "MySQL Source")
      // set 4 parallel source tasks
      .setParallelism(4)
      .print().setParallelism(1); // use parallelism 1 for sink to keep message ordering
    env.execute("Print MySQL Snapshot + Binlog");
  }
}
```

在构建MySqlSource时，必须指定以下参数。

参数	说明
hostname	MySQL数据库的IP地址或者Hostname。
port	MySQL数据库服务的端口号。
databaseList	MySQL数据库名称。 说明 数据库名称支持正则表达式以读取多个数据库的数据。
username	MySQL数据库服务的用户名。
password	MySQL数据库服务的密码。
deserializer	反序列化器，将SourceRecord类型记录反序列化到指定类型。参数取值如下： • RowDataDebeziumDeserializeSchema：将SourceRecord转成Flink Table或SQL内部数据结构RowData。 • JsonDebeziumDeserializationSchema：将SourceRecord转成JSON格式的String。

步骤三：打包并提交DataStream作业

使用Maven工具打包工程项目，并将生成的JAR包上传和提交到Flink全托管平台上，详细请参见作业提交。

9.5. GeminiStateBackend参考

9.5.1. GeminiStateBackend介绍

本文为您整体介绍GeminiStateBackend，以及其与RocksDBStateBackend的性能对比情况。

整体介绍

有状态计算是流处理中非常复杂和有挑战的场景。流处理的数据访问具有以下特点：

- 存在大量的随机访问且很少有范围查询。
- 数据流量和热点会更加频繁地动态变化，使得即便是同一算子的不同并发，也会有不同的数据访问模式。

GeminiStateBackend是针对这些特点研发的一款面向流计算场景的KV存储引擎，并作为实时计算Flink版产品的默认状态存储后端（StateBackend），大规模应用在阿里巴巴集团和阿里云客户生产实践当中。GeminiStateBackend的核心设计亮点如下：

- 全新的架构和数据结构设计，全面的性能提升。

  GeminiStateBackend的整体架构是在LSM数据结构的基础上设计的，包含了随数据规模和访问特点变化而自适应、数据冷热分层，可以在Anti-caching和Caching架构间灵活切换的三种能力。此外，还具备随机查询友好的哈希存储结构。Nexmark性能对比的结果显示，GeminiStateBackend相比RocksDBStateBackend的性能有了较大的提升。

- 支持存储计算分离，彻底摆脱状态数据的本地盘存储限制。

在本地盘空间限制的环境下，State较大的作业经常会遇到本地磁盘空间不足的问题。基于RocksDBStateBackend的作业通常需要通过扩并发等增加资源的手段来解决该问题。GeminiStateBackend具有存储计算分离的技术，使得状态存储可以不依赖于本地盘，可以避免因本地状态数据过大而引发作业故障的问题。有关存储计算分离相关配置，请参见存算分离配置。

- 支持KV分离，大幅提升双流或多流Join作业的效能。

  双流或多流Join作为流计算中最有挑战的场景之一，也是状态存储会遇到瓶颈的一个典型场景。GeminiStateBackend基于KV分离技术，极大提升了双流或多流Join作业的效能。经过阿里巴巴集团双十一核心业务验证，开启KV分离后计算资源利用率平均可以提升50%。典型优势场景下可以提升100%～200%。有关KV分离相关配置，请参见KV分离配置。

- 自适应参数调优，告别手动调参烦恼。

  在流计算任务中，不同算子往往具有不同的状态访问模式。状态存储通常需要不同的参数组合才可以达到最佳的性能，而这些参数往往很多而且涉及底层细节，手动调参会给您带来较高的学习和理解成本。GeminiStateBackend通过自适应参数调优技术，在作业运行时根据当前数据访问模式和流量自动调参，以达到各种场景下的最佳性能。经过阿里巴巴集团双十一核心业务验证，该技术可以避免95%以上的人工调参，同时提升10%～40%的单核吞吐能力。有关自适应调参相关配置，请参见自适应调参配置。

Nexmark性能对比

我们使用Nexmark中State瓶颈的用例和相同的硬件资源，测试并对比了RocksDBStateBackend和GeminiStateBackend的性能。结果显示GeminiStateBackend对作业整体效能（单核吞吐能力）的优化效果十分显著，具体数据如下表所示。

CaseName	Gemini TPS/Core	RocksDB TPS/Core	Gemini VS RocksDB提升
q4	83.63 K/s	53.26 K/s	57.02%
q5	84.52 K/s	57.86 K/s	46.08%
q8	468.96 K/s	361.37 K/s	29.77%
q9	59.42 K/s	26.56 K/s	123.72%
q11	93.08 K/s	48.82 K/s	90.66%
q18	150.93 K/s	87.37 K/s	72.75%
q19	143.46 K/s	58.5 K/s	145.23%
q20	75.69 K/s	22.44 K/s	237.30%

9.5.2. GeminiStateBackend配置

本文为您介绍在使用GeminiStateBackend时可能会用到的一些调优参数。

背景信息

在大部分场景下，基于自适应调参功能，GeminiStateBackend会自动调整参数而不需要您手动配置。在遇到特定场景时，您可以开启一些配置来进行针对性的调优。例如：

- 在希望协调内存资源和性能时，请使用基础配置，详情请参见基本配置。
- 在遇到本地盘空间不足的问题时，请使用存算分离配置，详情请参见存算分离配置。

- 在遇到Join算子有性能瓶颈时，请使用KV分离配置，详情请参见KV分离配置。

基本配置

参数	说明	数据类型	默认值	备注	支持的版本
state.backend.gemini.memory.managed	GeminiStateBackend是否自动根据Managed Memory来分配内存。	Boolean	true	参数取值如下： • true（默认）：系统将自动根据Managed Memory和task slot数计算每个Backend的内存。 • false：每个Backend的内存大小将设置为state.backend.gemini.total.writebuffer.size和state.backend.gemini.offheap.size参数值的和。 您可以通过将该参数取值设置为false，并单独配置WriteBuffer和Offheap的大小来协调内存资源和性能。 说明 推荐您保持参数默认值，不用修改。	VVR 4.0及以上版本
state.backend.gemini.total.writebuffer.size	WriteBuffer占用的内存总大小。	String	128 MB	在state.backend.gemini.memory.managed参数值为false时生效，否则WriteBuffer占用的内存总大小将由Managed Memory自动计算。 配置该参数值时，需要明确增加单位后缀，单位为B、KB、MB和GB。 说明 • 单位大小写不敏感。 • 数字和单位之间支持有空格。	

参数	说明	数据类型	默认值	备注	支持的版本
state.backend.gemini.offheap.size	GeminiStateBackend使用的堆外内存的大小。 说明 Gemini使用的这部分堆外内存不包括WriteBuffer使用的部分。	String	(none)	在state.backend.gemini.memory.managed参数值为false时生效，否则GeminiStateBackend使用的堆外内存的大小将由Managed Memory自动计算。 配置该参数值时，需要明确增加单位后缀，单位为B、KB、MB和GB。 说明 • 默认不配置该参数。 • 单位大小写不敏感。 • 数字和单位之间支持有空格。	

说明 Apache Flink中Checkpoint和StateBackend的基础配置同样适用于GeminiStateBackend，详情请参见Checkpoints and State Backends。

存算分离配置

参数	说明	数据类型	默认值	备注	支持的版本

参数	说明	数据类型	默认值	备注	支持的版本
state.backend.gemini.file.cache.type	存算分离的模式。	String	INFINITE	参数取值如下： • INFINITE（默认）：不打开存算分离，State数据只存在本地。 • LIMITED：State数据首先存储到本地，在超过本地磁盘限制时，存储State数据到DFS中。 当遇到因状态数据过大造成本地盘空间不足的问题时，可以将该参数取值设置为LIMITED，同时根据本地磁盘限制情况，设置state.backend.gemini.file.cache.preserved-space参数。 说明 ○ 参数取值大小写敏感。 ○ 开启存算分离会在一定程度上影响作业性能，因此需要权衡空间和性能两个因素。推荐您保持参数默认值，不用修改。	VVR 4.0.11及以上版本
state.backend.gemini.file.cache.preserved-space	单个TaskManager上剩余的State数据可用磁盘空间。	String	2 GB	当实际可用空间小于该值时，GeminiStateBackend将把State数据存储到DFS中，以摆脱本地存储规格的限制。 配置该参数值时，需要明确增加单位后缀，单位为B、KB、MB和GB。 说明 • 单位大小写不敏感。 • 数字和单位之间支持有空格。	

> **说明** 在使用OSS作为分布式文件系统时，可能会发生意料外的磁盘空间占用情况，这是由于OSS Client SDK会将文件先完整写到本地磁盘，再进行上传。而Flink在执行Savepoint时，单个StateBackend仅会产生一个文件，这就导致了创建Savepoint时，本地会创建一个很大体积的未压缩文件，会导致意料外的磁盘空间占用。目前在这种情况下存算分离并不能起到作用，需要使用扩并发等手段减少单节点的State大小。

KV分离配置

参数	说明	数据类型	默认值	备注	支持的版本
state.backend.gemini.kv.separate.mode	KV分离的模式。	String	DISABLE	参数取值如下： • DISABLE（默认值）：禁用KV分离。 • GLOBAL_ENABLE：启用KV分离。 当带Join算子的作业产生反压或需要进行性能调优时，在发现Join成功率比较低（可以通过观察Join节点的Records Received和Records Sent的比例，大致估算Join成功率）而Value又比较大时，可以将该参数取值设置为GLOBAL_ENABLE，进而取得更好的性能。 **说明** ○ 参数取值大小写敏感。 ○ 由于设置该参数对不满足上述条件的作业会带来一些额外开销，因此推荐您保持参数默认值，不用修改。	VVR 4.0.12及以上版本
state.backend.gemini.kv.separate.value.size.threshold	在启用KV分离后，触发KV分离的Value大小阈值。	Integer	200	在达到该阈值后的Record会将Key和Value分离存储。建议取值范围为150～1000，可以根据Join成功率调整该参数，Join成功率越高，该参数可以设置得越大。 单位为字节（Bytes）。	

自适应调参配置

参数	说明	数据类型	默认值	备注	支持的版本
state.backend.gemini.auto-tune.mode	自适应调参的模式。	String	ACTIVE	参数取值如下： • DISABLED：禁用该功能。 • MONITORING：GeminiStateBackend会持续监控运行情况，并在日志中给出调参建议，但不自动进行参数调整。 • ACTIVE（默认值）：GeminiStateBackend将持续监控运行情况，并自动进行参数调整。但只针对未在flink-conf.yaml中配置的Gemini相关的部分参数。 • FORCEFUL：GeminiStateBackend将持续监控运行情况，并自动进行参数调整，同时会覆盖在flink-conf.yaml中配置的Gemini相关的部分参数。 **说明** • 参数取值大小写不敏感。 • 推荐您保持参数默认值，不用修改。	VVR 4.0.12及以上版本
state.backend.gemini.auto-tune.burst.start.x	自适应调参下采用性能优先模式的时间段。	String	(none)	参数名称中的x为任意数字，start.x与end.x对应，可以借此来设置多个时间段。参数值的格式为yyyy-MM-dd HH:mm:ss。 当您更关注TPS而非综合效能时，可以配置这两个参数，GeminiStateBackend将在该时间段采用TPS优先的策略，以耗费更多的资源（CPU或MEM）为代价，达到更高的TPS。 **说明** • 默认不配置该参数。 • 输入的end时间需要晚于start时间。 • 设置该参数后会导致更多的资源消耗，因此推荐您保持参数默认值，不用修改。	
state.backend.gemini.auto-tune.burst.end.x					

10.常见问题

10.1. 基本概念

本文为您介绍实时计算Flink版基本概念的常见问题。

- 什么是实时计算？
- 实时计算与批量计算相比存在哪些差异？
- 阿里云实时计算Flink版有哪些使用限制？
- 什么是流数据？
- 阿里云实时计算Flink版的数据存储有哪些类型？

什么是实时计算？

数据的业务价值会随着时间的流失而迅速降低，因此在数据发生后必须尽快对其进行计算和处理。目前，对于信息的高时效性和可操作性要求越来越高，这就要求软件系统能够在更短的时间内处理更多的数据。而传统的大数据处理模型将在线事务处理和离线分析从时序上完全分开，对于数据加工均遵循传统的日清日毕模式，即以小时甚至以天为计算周期对当前数据进行累计并处理。显然，传统的大数据处理方式无法满足数据实时计算的需求。数据处理时延造成的影响对要求苛刻的业务场景会非常明显，例如实时大数据分析、风控预警、实时预测和金融交易等领域。

实时计算可以有效地缩短全链路数据流时延、实时化计算业务逻辑、平摊计算成本，最终有效满足实时处理大数据的业务需求。

实时计算具备三大特点：

- 实时（Realtime）且无界（Unbounded）的数据流

  实时计算面对的数据是实时且流式的，这些数据按照时间发生顺序被实时计算订阅和消费。例如网站的访问单击日志流，只要网站不关闭，其单击日志流将不停产生并进入实时计算系统。

- 持续（Continous）且高效的计算

  实时计算是一种事件触发的计算模式，触发源为无界流式数据。一旦有新的流数据进入实时计算系统，它就立刻发起并进行一次计算任务，因此整个过程是持续进行的。

- 流式（Streaming）且实时的数据集成

  被流数据触发的计算结果，可以被直接写入目的数据存储。例如将计算后的报表数据直接写入阿里云关系型数据库RDS（Relational Database Service）进行报表展示。因此流数据的计算结果可以同流式数据一样，持续被写入目的数据存储。

实时计算与批量计算相比存在哪些差异？

下面从用户和产品层面来理解两类计算方式的区别。

- 批量计算

批量计算是一种批量、高时延、主动发起的计算。目前绝大部分传统数据计算和数据分析服务均基于批量数据处理模型，即使用ETL系统或者OLTP系统进行构造数据存储，在线的数据服务（包括Ad-Hoc查询、DashBoard等）通过构造SQL语言访问上述数据存储并取得分析结果。这套数据处理的方法论伴随着关系型数据库在工业界的演进而被广泛采用。传统的批量数据处理模型如下图所示。

传统的批量数据处理流程如下：

i. 装载数据

对于批量计算，用户需要预先将数据加载到计算系统，您可以使用ETL系统或者OLTP系统装载原始数据。系统将根据自己的存储和计算情况，对于装载的数据进行一系列查询优化、分析和计算。

ii. 提交请求

系统主动发起一个计算作业（例如MaxCompute的SQL作业，或Hive的SQL作业）并向上述数据系统进行请求。此时计算系统开始调度（启动）计算节点进行大量数据计算，该过程的计算量可能非常大，耗时长达数分钟乃至于数小时。由于数据累计处理不及时，上述计算过程中可能就会存在一些历史数据，导致数据不新鲜。

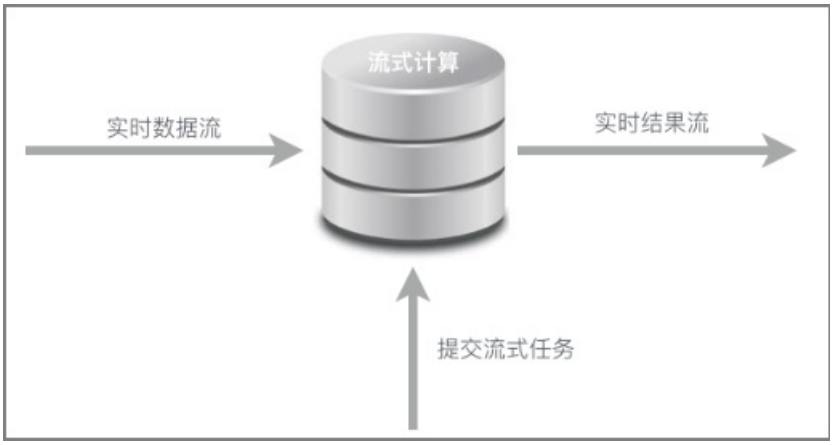
> **说明** 您可以根据业务需要，随时调整SQL语句后，再次提交作业。您甚至可以使用AdHoc查询做到即时修改和即时查询。

iii. 返回结果

计算作业完成后将数据以结果集形式返回给用户，由于保存在数据计算系统中的计算结果数据量巨大，需要用户再次集成数据到其他系统。一旦数据结果巨大，整体的数据集成过程就会漫长，耗时可能长达数分钟乃至于数小时。

- 实时计算

实时计算是一种持续、低时延、事件触发的计算作业。相对于批量计算，流式计算整体上还属于比较新颖的计算概念。由于当前实时计算的计算模型较为简单，所以在大部分大数据计算场景下，实时计算可以看做是批量计算的增值服务，实时计算更强调计算数据流和低时延。实时计算数据处理模型如下。



i. 实时数据流

使用实时数据集成工具，将实时变化的数据传输到流式数据存储（例如消息队列、DataHub）。此时数据的传输实时化，将长时间累积的大量数据平摊到每个时间点，不停地小批量实时传输，因此数据集成的时延得以保证。

源源不断的数据被写入流数据存储，不需要预先加载的过程。同时，流计算对于流式数据不提供存储服务，数据持续流动，在计算完成后就被立刻丢弃。

ii. 提交流式任务

批量计算要等待数据集成全部就绪后才能启动计算作业，而流式计算作业是一种常驻计算服务。实时计算作业启动后，一旦有小批量数据进入流式数据存储，流计算会立刻计算并得出结果。同时，阿里云流计算还使用了增量计算模型，将大批量数据分批进行增量计算，进一步减少单次运算规模并有效降低整体运算时延。

从用户角度，对于流式作业，必须预先定义计算逻辑，并提交到流式计算系统中。在整个运行期间，流计算作业逻辑不可更改。用户通过停止当前作业运行后再次提交作业，此时之前已经计算完成的数据是无法重新再次被计算。

iii. 实时结果流

不同于批量计算，结果数据需等待数据计算结果完成后，批量将数据传输到在线系统。流式计算作业在每次小批量数据计算后，无需等待整体的数据计算结果，会立刻将数据结果投递到在线/批量系统，实现计算结果的实时化展现。

使用实时计算的顺序如下：

i. 提交实时计算作业。
ii. 等待流式数据触发实时计算作业。
iii. 计算结果持续不断对外写出。

计算模型差别对比，详情内容如下表所示。

对比指标	批量计算	实时计算
数据集成方式	预先加载数据。	实时加载数据到实时计算。
使用方式	业务逻辑可以修改，数据可重新计算。	业务逻辑一旦修改，之前的数据不可重新计算（流数据易逝性）。
数据范围	对加载的所有或大部分数据进行查询或处理。	对滚动时间窗口内的数据或仅对最近的数据记录进行查询或处理。
数据大小	大批量数据。	单条记录或几条记录的微批量数据。
性能	几分钟至几小时的延迟。	大约几秒或几毫秒的延迟。
分析	复杂分析。	简单的响应函数、聚合和滚动指标。

阿里云实时计算Flink版有哪些使用限制？

请您评估以下限制对您业务的影响，并做相应的调整。如有疑问，请您提交工单进行咨询。

- 阿里云实时计算Flink版购买指导，请参见开通流程。
- 阿里云实时计算Flink版WebConsole仅支持Chrome浏览器访问。
- 阿里云实时计算Flink版包年包月支持华北2（北京）、华东2（上海）、华东1（杭州）、华南1（深圳）、

新加坡和中国（香港）地域，您可以按区域购买。

- 阿里云实时计算Flink版按量付费支持华南1（深圳）、华北2（北京）、华东2（上海）和华东1（杭州）地域，您可以按区域购买。
- 阿里云实时计算Flink版在简单的流式压测处理场景下（例如过滤、清洗等），一个实时计算CU的处理能力为1000条/秒。在复杂的流式压测处理场景下（例如复杂UDF计算、聚合操作等），一个实时计算CU的处理能力为500条/秒。

> **说明** 请根据您的业务情况，评估所需的CU数量。

- 阿里云实时计算Flink版对整个下属的Task、Task版本、IDE打开Task页面均有不同限制，包括：
  - 一个阿里云账号至多可以创建20个项目空间。
  - 单个项目下最多允许创建100个文件夹，深度最大不超过5层。
  - 单个项目下最多允许上传50个UDF或JAR。
  - 单个文件夹内最大文件数为500，包含目录。

什么是流数据？

所有大数据的产生均可以看作是一系列离散事件，这些离散事件是一条条事件流或数据流。相对于离线数据，流数据的规模普遍较小。流数据是由数据源持续产生的数据，示例如下：

- 使用移动或Web应用程序生成的日志文件。
- 网购数据。
- 游戏内玩家活动信息。
- 社交网站信息。
- 金融交易大厅或地理空间服务数据中心内所连接设备或仪器的遥测数据。
- 地理空间服务信息。
- 设备或仪器的遥测数据。

阿里云实时计算Flink版的数据存储有哪些类型？

阿里云实时计算Flink版本身不带有业务存储，所有的数据均是来自于外部存储系统持有的数据。阿里云实时计算Flink版支持以下几类数据存储类型：

- 流式的数据输入：流式数据的输入会触发和推动实时计算Flink版进行数据处理。每个实时计算Flink版作业必须至少声明一个流式数据输入源。
- 静态数据输入：静态数据输入也被称为维表，对于每条流式数据，可以关联一个外部静态数据源进行查询，为实时计算Flink版提供数据关联查询。
- 结果表输出：实时计算Flink版将计算的结果数据写出到目的数据表，为下游数据继续消费提供各类读写接口。

10.2. 计费常见问题

本文介绍Flink全托管实例计费方面的常见问题。

- Flink全托管实例是否支持包年包月和按量计费两种计费模式相互转换？
- Flink全托管续费后，该续费默认包含哪些项？
- Flink全托管产品如何续费？
- 当不再需要使用集群时，如何申请退款？

- 集群创建失败，会扣费吗？
- 按量付费的集群创建成功后，但实际没有运行作业，会收费吗？
- SLB欠费会影响Flink全托管集群的使用吗？

Flink全托管实例是否支持包年包月和按量计费两种计费模式相互转换？

不支持。您可以释放当前实例，重新购买对应计费模式的实例。

Flink全托管续费后，该续费默认包含哪些项？

Flink全托管的费用由以下两部分组成。

计费项	资源说明
管控资源	创建实例时，系统为每个集群部署一个开发控制台，每个开发控制台及其必要组件需要大概2 CU的管控资源。
计算资源	实际可用于计算的资源，根据您包年包月预购的资源量或按量付费实际的使用量来收费，计量单位为CU。

Flink全托管开通流程中的费用仅包含Flink全托管，在Flink全托管开通和使用过程中还涉及其他云产品的费用，详情请参见下表。

相关产品	计费说明
VPC	收费详情请参见产品计费。
OSS	收费详情请参见计费概述。
SLB	在您开通Flink全托管集群后，会自动开通SLB服务，单击创建Flink全托管集群后就开始收费，详情请参见按量计费。 **注意** 负载均衡SLB可以让您以公网的形式访问Flink全托管开发控制台，该SLB一旦创建，请不要用于其他情况，并且不要删除或者修改配置，否则可能导致Flink全托管集群不可用。
ARMS	在您开通Flink全托管集群后，会自动开通ARMS服务，为您提供专家版Prometheus监控能力。ARMS收费详情请参见收费规则。 目前提供的标准的Metric模版中，除Operator相关Metric信息收费外，其他Metric均不收费。Operator相关指标主要存在于Apache Flink官方文档中IO和Connectors相关章节。详情请参见Flink 1.11、Flink 1.12和Flink 1.13。 **注意** 如果将ARMS关闭，则在Flink控制台无法通过数据曲线排查定位问题，也无法配置监控告警。

Flink全托管产品如何续费？

产品支持手动续费和自动续费，具体操作请参见续费。

当不再需要使用集群时，如何申请退款？

Flink全托管集群的退费规则及相关说明如下所示，详情请参见退款说明。

计费方式	退款规则及说明
包年包月	包年包月的集群在购买时根据购买时长预先支付了费用，如果因业务需求变更不再使用集群，可以通过提交工单退订该资源，申请退款。 退款类型、退款规则及退款流程的详细信息，请参见退款规则及退款流程。
按量计费	按量付费的集群是在使用后才支付费用，因此不涉及退款。 如果因业务需求变更不再使用集群，可以随时释放集群。集群释放后，将不再进行计费。

包年包月的Flink全托管集群在购买时根据购买时长预先支付了费用，如果因业务需求变更不再使用集群，可以通过提交工单退订该资源，申请退款。

集群创建失败，会扣费吗？

对于创建失败的集群，系统不会扣费。关于扣费问题，您可以参考消费明细查看自己的消费记录，明确扣费项。详情请参见查看消费明细。

按量付费的集群创建成功后，但实际没有运行作业，会收费吗？

会收费。无论您是否运行作业，只要您开通了Flink全托管按量付费模式，系统就会按照计费规则收费。在不运行作业的情况下，系统也会对管控资源（固定2 CU）计费。详情请参见计费公式。

SLB欠费会影响Flink全托管集群的使用吗？

会影响。负载均衡SLB可以让您以公网的形式访问Flink全托管实例，如果您的SLB欠费7天没有续费，SLB将会被释放，释放后就不能通过公网访问，需要您重新购买Flink全托管实例。详情请参见SLB欠费说明。

10.3. 操作指导

本文为您介绍实时计算Flink版的操作指导方面的常见问题，包括控制台操作、网络连通性和JAB包等问题。

- 控制台操作
  - 如何设置白名单？
  - 如何在OSS控制台上传JAR包？
  - 如何设置Checkpoint和State相关参数？
  - 如何查找引发告警的作业？
  - 如何查看工作空间ID等信息？
  - 如何查看当前作业的Flink版本？
  - 对于系统自动开通的Prometheus监控服务，是否可以直接关闭？
- 网络连通性
  - Flink全托管集群如何访问公网？
  - 如何访问跨VPC里的存储资源？
- JAR包问题

- 如何解决Flink依赖冲突问题？
- 如何解析Flink作业所依赖服务的域名？

如何设置白名单？

通常，Flink全托管上下游存储设备默认拒绝外部设备的访问。因此，您需要在目标存储设备白名单中添加Flink全托管虚拟交换机的网段。操作步骤详情如下：

1. 登录实时计算控制台。
2. 在目标**工作空间**右侧**操作**列，选择**更多 > 工作空间详情**。
3. 在**工作空间详情**对话框，查看**Flink全托管**虚拟交换机的**网段**信息。

4. 在您的目标存储设备白名单中，添加**Flink全托管**虚拟交换机的**网段**。

   例如，您要为云数据库RDS MySQL版数据库设置白名单，您可以参见设置IP白名单。

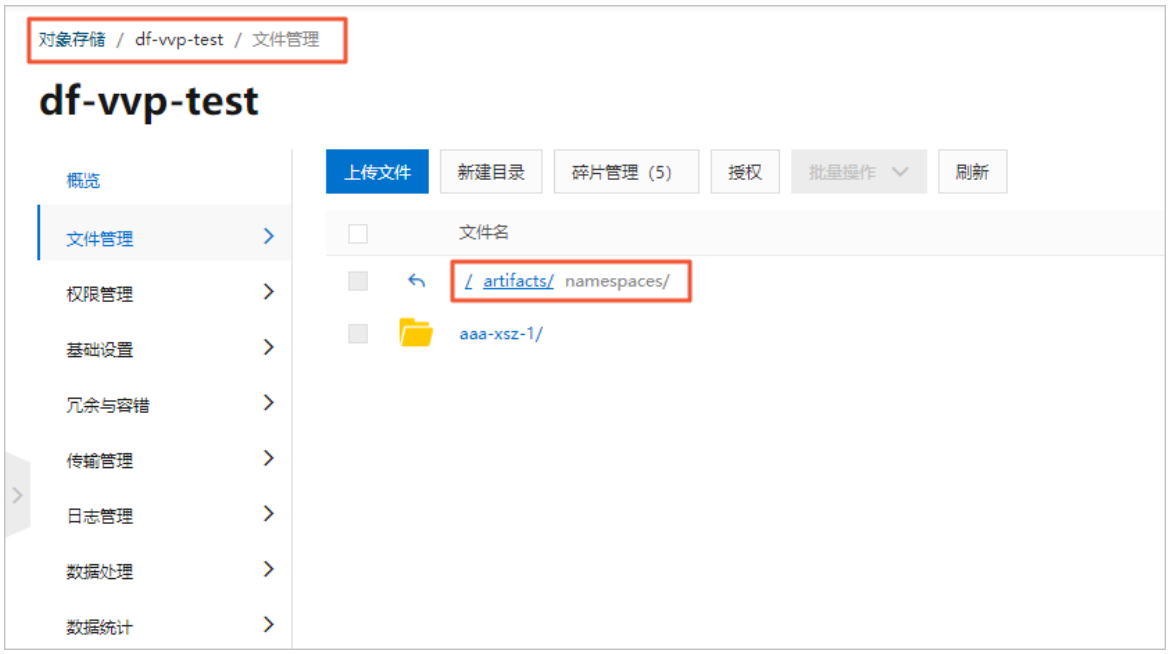
> **说明**
>
> - 如果您后续新增了虚拟交换机，也需要将新增虚拟交换机的网段添加至目标存储设备的白名单中。
> - 如果您的虚拟交换机和上下游存储不在同一可用区，添加虚拟交换机网段至白名单后，网络可以连通。

如何在OSS控制台上传JAR包？

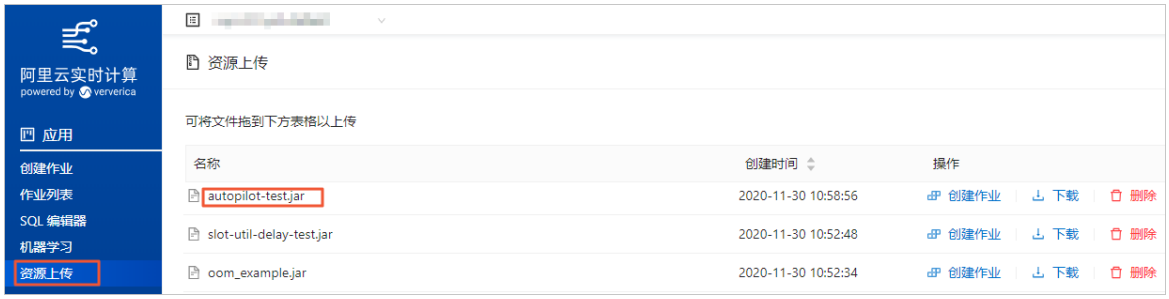
1. 在Flink全托管控制台上查看当前集群的OSS Bucket。

OSS Bucket信息如下图所示。

2. 登录OSS管理控制台，上传JAR包至目标Bucket的*/artifats/namespaces*目录下。



3. 在Flink全托管控制台左侧导航栏，单击**资源上传**，查看通过OSS控制台上传的JAR包。



如何设置Checkpoint和State相关参数？

在目标作业详情页面右上角，单击**编辑**后，在页面右侧**高级配置**面板的**更多Flink配置**中，添加代码后保存生效，代码示例如下。

```
state.backend: rocksdb
state.backend.incremental: true
table.exec.state.ttl: 129600000
```

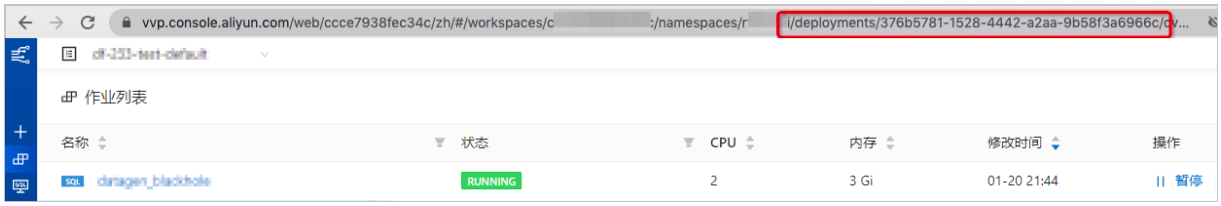
Flink全托管在开源Flink的基础上，还可以设置如下表所示参数。

参数	说明	备注
state.backend	流状态存储系统。	取值如下： • GeminiStateBackend（默认值）：阿里自研的流状态存储系统。 • RocksDB：开源Flink流状态存储系统。
table.exec.state.ttl	SQL作业的State TTL（Time To Live）。	单位为毫秒，例如State TTL如果设置为129600000，则表示过期时间为1.5天。默认值为空，代表State不过期。
state.backend.gemini.savepoint.external-sort.local-storage.enabled	Savepoint过程中产生的临时数据是否储存在本地磁盘。	取值如下： • false（默认值）：临时数据储存在DFS上。 • true：临时数据储存在本地磁盘，从而减少与DFS的交互，加快Savepoint速度。 如果您的本地磁盘空间足够存放临时数据，即本地磁盘空间使用率在30%以下，建议设置该参数为true。 说明 仅VVR 4.0及以上版本支持该参数。

对于DataStream和Python作业，请参见State Time-To-Live (TTL)。对于Flink中的Checkpoint和State更多参数详情，请参见Checkpoints and State Backends。

如何查找引发告警的作业？

告警事件中包含JobID和Deployment ID信息，但是由于作业Failover后导致JobID发生变化，因此您需要通过Deployment ID查找是哪个作业报错。目前，Flink全托管控制台上暂无Deployment ID信息，您需要在作业的URL中获取Deployment ID信息。



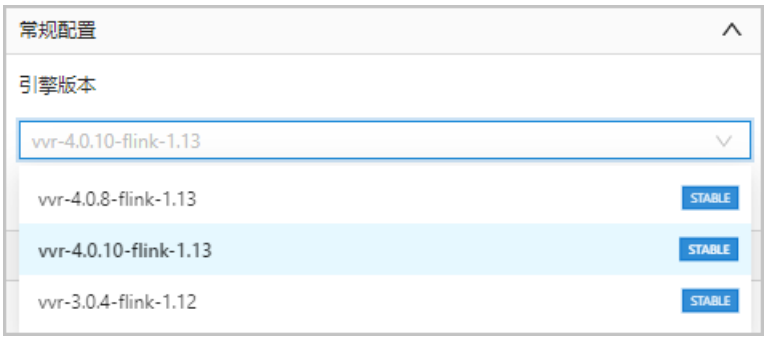
如何查看工作空间ID等信息？

在管理控制台目标工作空间名称右侧，选择**其他 > 工作空间详情**。

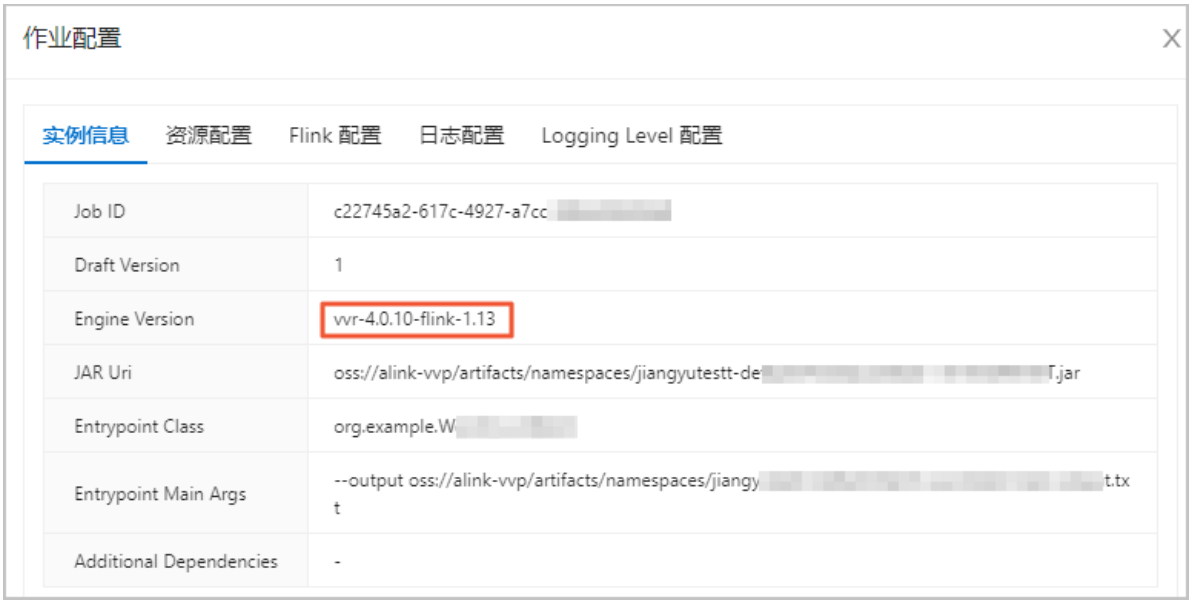
如何查看当前作业的Flink版本？

您可以通过以下方式查看：

- 在作业开发页面右侧，单击**高级配置**，选择**常规配置 > 引擎版本**，即可查看。



- 在作业运维页面，单击目标作业名称，在**作业总览**页签下，单击当前实例右侧的**详情**，即可在**实例信息**中看到版本信息。



对于系统自动开通的Prometheus监控服务，是否可以直接关闭？

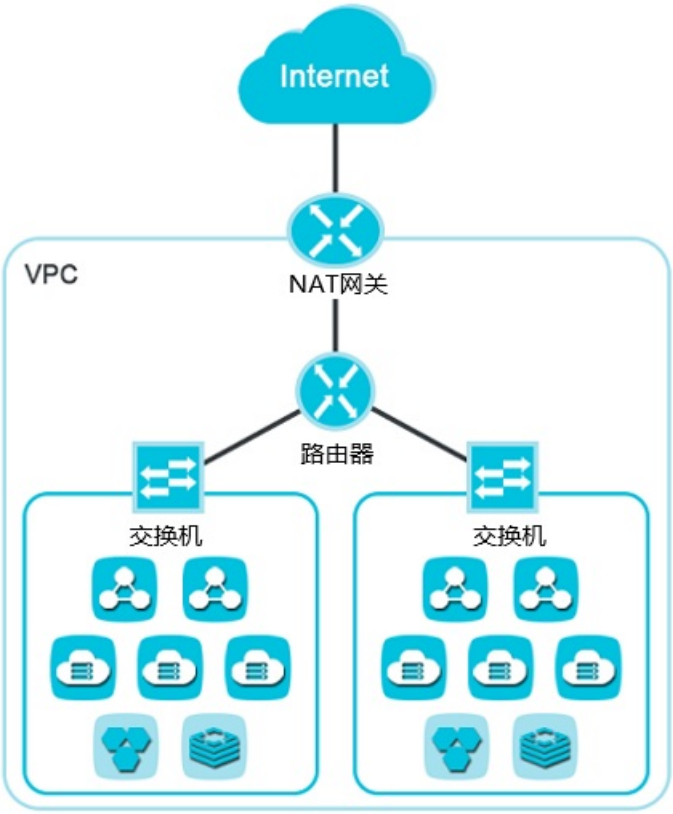
不支持直接关闭。但是您可以卸载Prometheus监控插件，详情请参见卸载Prometheus监控插件。如果卸载还有问题，请提交工单处理。

> **注意** 该监控插件一旦被卸载，VVP将不再显示数据曲线。如果作业出现异常现象，将无法定位初始异常时间，并且无法上报监控告警，请谨慎操作。

Flink全托管集群如何访问公网？

- 背景说明

Flink全托管集群默认不具备访问公网的能力，但阿里云提供的NAT网关可以实现VPC网络与公网网络互通，以满足部分Flink全托管集群用户通过UDX或Datastream代码访问公网的需求。



- 解决方案

  通过在VPC中创建NAT网关，并创建SNAT条目，将Flink全托管集群所在的交换机绑定至弹性公网IP（EIP），即可通过EIP访问公网。具体配置方法请参见：

  i. 创建NAT网关，详情请参见创建NAT网关。
  ii. 创建SNAT条目，详情请参见创建SNAT条目。
  iii. 将Flink全托管集群所在的交换机绑定至弹性公网IP（EIP），详情请参见绑定EIP。

如何访问跨VPC里的存储资源？

您可以通过以下几种方式跨VPC访问存储资源：

- 提交工单，产品名称选择**VPC**，要求通过**高速通道**或其它产品建立网络连接，但是此种方式需要付费。
- 使用VPN网关建立VPC到VPC的VPN连接，详情请参见建立VPC到VPC的连接。
- 退掉和Flink全托管不同VPC的存储服务后，重新购买一个与Flink全托管相同VPC的存储服务。
- 释放Flink全托管服务后，重新购买一个和存储服务相同VPC的Flink全托管服务。
- 开通Flink全托管的公网访问能力，通过公网访问存储服务。Flink全托管集群默认不具备访问公网的能力，如有需求，详情请参见Flink全托管集群如何访问公网？。

> **说明** 因为在延迟性方面，公网不如内网，如果您对性能有要求，建议不要使用此方式。

如何解决Flink依赖冲突问题？

- 问题现象

- 有明显报错，且引发错误的为Flink或Hadoop相关类。

```
java.lang.AbstractMethodError
java.lang.ClassNotFoundException
java.lang.IllegalAccessError
java.lang.IllegalAccessException
java.lang.InstantiationError
java.lang.InstantiationException
java.lang.InvocationTargetException
java.lang.NoClassDefFoundError
java.lang.NoSuchFieldError
java.lang.NoSuchFieldException
java.lang.NoSuchMethodError
java.lang.NoSuchMethodException
```

- 无明显报错，但会引起一些不符合预期的现象，例如：
  - 日志不输出或log4j配置不生效。

    该类问题通常是由于依赖中携带了log4j相关配置导致的。需要排查作业JAR包中是否引入了log4j配置的依赖，可以通过在dependency中配置exclusions的方式去掉log4j配置。

    > **说明** 如果必须要使用不同版本的log4j，需要使用maven-shade-plugin将log4j相关的class relocation掉。

  - RPC调用异常。

    Flink的Akka RPC调用出现依赖冲突可能导致的异常，默认不会显示在日志中，需要开启Debug日志进行确认。

    例如，Debug日志中出现 `Cannot allocate the requested resources. Trying to allocate ResourceProfile{xxx}` ，但是JM日志在 `Registering TaskManager with ResourceID xxx` 后，没有下文，直到资源请求超时报错 `NoResourceAvailableException` 。此外TM持续报错 `Cannot allocate the requested resources. Trying to allocate ResourceProfile{xxx}` 。

    原因：开启Debug日志后，发现RPC调用报错 `InvocationTargetException` ，该报错导致TM Slot分配到一半失败出现状态不一致，RM持续尝试分配Slot失败无法恢复。

- 问题原因
  - 作业JAR包中包含了不必要的依赖（例如基本配置、Flink、Hadoop和log4j依赖），造成依赖冲突从而引发各种问题。
  - 作业需要的Connector对应的依赖未被打入JAR包中。
- 排查方法
  - 查看作业pom.xml文件，判断是否存在不必要的依赖。
  - 通过 `jar tf foo.jar` 命令查看作业JAR包内容，判断是否存在引发依赖冲突的内容。
  - 通过 `mvn dependency:tree` 命令查看作业的依赖关系，判断是否存在冲突的依赖。
- 解决方案

- 基本配置建议将scope全部设置为provided，即不打入作业JAR包。
  - DataStream Java

```
<dependency>
  <groupId>org.apache.flink</groupId>
  <artifactId>flink-streaming-java_2.11</artifactId>
  <version>${flink.version}</version>
  <scope>provided</scope>
</dependency>
```

  - DataStream Scala

```
<dependency>
  <groupId>org.apache.flink</groupId>
  <artifactId>flink-streaming-scala_2.11</artifactId>
  <version>${flink.version}</version>
  <scope>provided</scope>
</dependency>
```

  - DataSet Java

```
<dependency>
  <groupId>org.apache.flink</groupId>
  <artifactId>flink-java</artifactId>
  <version>${flink.version}</version>
  <scope>provided</scope>
</dependency>
```

  - DataSet Scala

```
<dependency>
  <groupId>org.apache.flink</groupId>
  <artifactId>flink-scala_2.11</artifactId>
  <version>${flink.version}</version>
  <scope>provided</scope>
</dependency>
```

- 添加作业需要的Connector对应的依赖，并将scope全部设置为compile（默认的scope是compile），即打入作业JAR包中，以Kafka Connector为例，代码如下。

```
<dependency>
    <groupId>org.apache.flink</groupId>
    <artifactId>flink-connector-kafka_2.11</artifactId>
    <version>${flink.version}</version>
</dependency>
```

- 其他Flink、Hadoop和log4j依赖不建议添加。但是：
  - 如果作业本身存在基本配置或Connector相关的直接依赖，建议将scope设置为provided，示例如下。

```
<dependency>
    <groupId>org.apache.hadoop</groupId>
    <artifactId>hadoop-common</artifactId>
    <scope>provided</scope>
</dependency>
```

  - 如果作业存在基本配置或Connector相关的间接依赖，建议通过exclusion将依赖去掉，示例如下。

```
<dependency>
    <groupId>foo</groupId>
      <artifactId>bar</artifactId>
      <exclusions>
        <exclusion>
        <groupId>org.apache.hadoop</groupId>
        <artifactId>hadoop-common</artifactId>
       </exclusion>
    </exclusions>
</dependency>
```

如何解析Flink作业所依赖服务的域名？

如果您自建的Flink作业中所依赖的服务填写的是域名，则迁移上云至Flink全托管服务时，可能会报无法解析域名的问题。此时，您可以按照以下方案解析Flink作业所依赖服务的域名：

- 您已有自建的DNS，并且Flink VPC能够连通该自建DNS服务，且该自建DNS能够正常解析域名。

  此时，您可以基于Flink全托管作业模板进行域名解析。假如您的自建DNS IP为192.168.0.1，操作步骤如下：

  i. 登录实时计算管理控制台。
  ii. 在**Flink全托管**页签，单击目标工作空间**操作**列下的**控制台**。
  iii. 在左侧导航栏，选择**系统管理 > 作业模板**。
  iv. 在**Flink配置**文本框，添加如下代码。

```
env.java.opts: >-
  -Dsun.net.spi.nameservice.provider.1=default
  -Dsun.net.spi.nameservice.provider.2=dns,sun
  -Dsun.net.spi.nameservice.nameservers=192.168.0.1
```

  > **说明** 如果您的自建DNS有多个IP，建议IP之间使用英文逗号（,）分隔。

  v. 在Flink全托管控制台新建作业并运行。

  如果还报UnknownHostException的错误，则意味着无法解析域名，请您提交工单进行咨询。

- 您没有自建的DNS或者Flink VPC无法与自建的DNS连通。

  此时，您需要基于阿里云云解析PrivateZone进行域名解析。假如您的Flink VPC为vpc-flinkxxxxxxx，Flink作业需要访问的服务的域名分别为aaa.test.com 127.0.0.1、bbb.test.com 127.0.0.2和ccc.test.com 127.0.0.3，操作步骤如下：

i. 开通云解析PrivateZone。详情请参见开通云解析PrivateZone

ii. 添加Zone，以Flink作业需要访问服务的公共后缀作为Zone名称。详情请参见添加Zone。

iii. 关联Flink全托管的VPC。详情请参见关联/解关联VPC

iv. 添加解析记录至Zone。详情请参见添加PrivateZone解析记录。

v. 在Flink全托管控制台新建作业并运行或者停止后再启动历史作业 。

如果还报unknowhost错误，则意味着无法解析域名，请您提交工单进行咨询。

10.4. 分场景排错指引

本文为您介绍实时计算Flink版分场景排错指引方面的常见问题，包括权限、运维和数据异常等问题。

- 权限

  不小心删除了角色或者变更了授权策略，导致Flink全托管服务不可用怎么办？

- 运维
  - JobManager没有运行起来，如何快速定位问题？
  - Python作业，如果Checkpoint慢怎么办？
  - 报错：Invalid versionName string

- 数据异常
  - 如何定位Flink无法读取源数据的问题？
  - 如何定位Flink无法将数据写入到结果表的问题？
  - 如何定位数据丢失的问题？

不小心删除了角色或者变更了授权策略，导致Flink全托管服务不可用怎么办？

- 方式一：创建AliyunStreamAsiDefaultRole角色后，添加AliyunStreamAsiDefaultRolePolicy0、AliyunStreamAsiDefaultRolePolicy1和FlinkServerlessPolicy自定义授权策略，实现重新授权。操作步骤详情请参见手动授权（方式一）。
- 方式二：先删除资源编排服务ROS的资源栈、RAM角色和RAM权限策略后，登录实时计算管理控制台进行自动化重新授权。操作步骤详情请参见手动授权（方式二）。

JobManager没有运行起来，如何快速定位问题？

JobManager没有运行起来即无法进入**Flink UI**页面。此时，您可以通过以下操作进行问题定位：

1. 在**作业运维**页面，单击目标作业名称。
2. 单击**运行事件**页签。

3. 通过快捷键搜索error，获取异常信息。
   - Windows系统：Ctrl+F
   - Mac系统：Command+F

Python作业，如果Checkpoint慢怎么办？

- 问题原因

  Python算子内部有一定的缓存，在进行Checkpoint时，需要将缓存中的数据全部处理完。因此，如果Python UDF的性能较差，则会导致Checkpoint时间变长，从而影响作业执行。

- 解决方案

  在作业开发页面，单击右侧的**高级配置**后，在**更多Flink配置**项中，设置以下参数，将缓存调小。

```
python.fn-execution.bundle.size：默认值为100000，单位是条数。
python.fn-execution.bundle.time：默认值为1000，单位是毫秒。
```

  参数的详细信息请参见Flink Python配置。

报错：Invalid versionName string

- 报错详情

  作业启动时报错Invalid versionName string。

> **说明** SQL作业不受影响。Flink引擎版本<= 1.10.4或Flink引擎版本<= 1.11.3的JAR或PYTHON作业会受影响。

- 报错原因

  创建Session模式的JAR或PYTHON作业时，没有设置Flink引擎版本。

- 解决方案

在作业开发页面，设置Flink引擎版本后，重新上线启动作业。

对于SDK方式创建的Session作业，建议您升级依赖的SDK common包到1.0.21版本，并在创建Artifact时，设置Flink引擎版本。详细信息如下所示：

- common包

```
<dependency>
    <groupId>com.aliyun</groupId>
    <artifactId>ververica-common</artifactId>
    <version>1.0.21</version>
</dependency>
```

- Artifact设置

```
com.ververica.common.model.deployment.Artifact.SqlScriptArtifact#setVersionName
com.ververica.common.model.deployment.Artifact.JarArtifact#setVersionName
```

> **说明** versionName取值须和Session集群上的Flink引擎版本保持一致。

如何定位Flink无法读取源数据的问题？

当Flink无法读取源数据时，建议从以下几个方面进行排查并处理：

- 检查上游存储和Flink全托管之间网络是否连通。

  Flink全托管仅支持访问相同地域、相同VPC下的存储。如果您有访问跨VPC存储资源或者通过公网访问Flink全托管的特殊需求，请查看以下文档：

  - 如果您需要跨VPC访问存储资源，则可以通过5种方式解决，详情请参见如何访问跨VPC里的存储资源？
  - 如果您需要通过公网访问Flink全托管，则可以使用阿里云提供的NAT网关实现VPC网络和公网的连通。详情请参见Flink全托管集群如何访问公网？

- 检查上游存储中是否已配置了白名单。

  上游存储中需要配的产品有Kafka和ES。您可以按照以下步骤配置白名单：

  i. 获取Flink全托管虚拟交换机的网段。

     获取方法请参见设置白名单。

  ii. 在上游存储中配置Flink全托管白名单。

     上游存储中配置白名单的方法，请参见对应DDL文档的前提条件中的文档链接，例如Kafka源表前提条件。

- 检查DDL中定义的字段类型、字段顺序和字段大小写是否和物理表一致。

  上游存储支持的字段类型和Flink全托管支持的字段类型可能不完全一致，但存在一定的映射关系。您需要按照DDL定义的字段类型映射关系一对一匹配，详情请参见对应DDL文档类型映射文档，例如日志服务SLS源表类型映射。

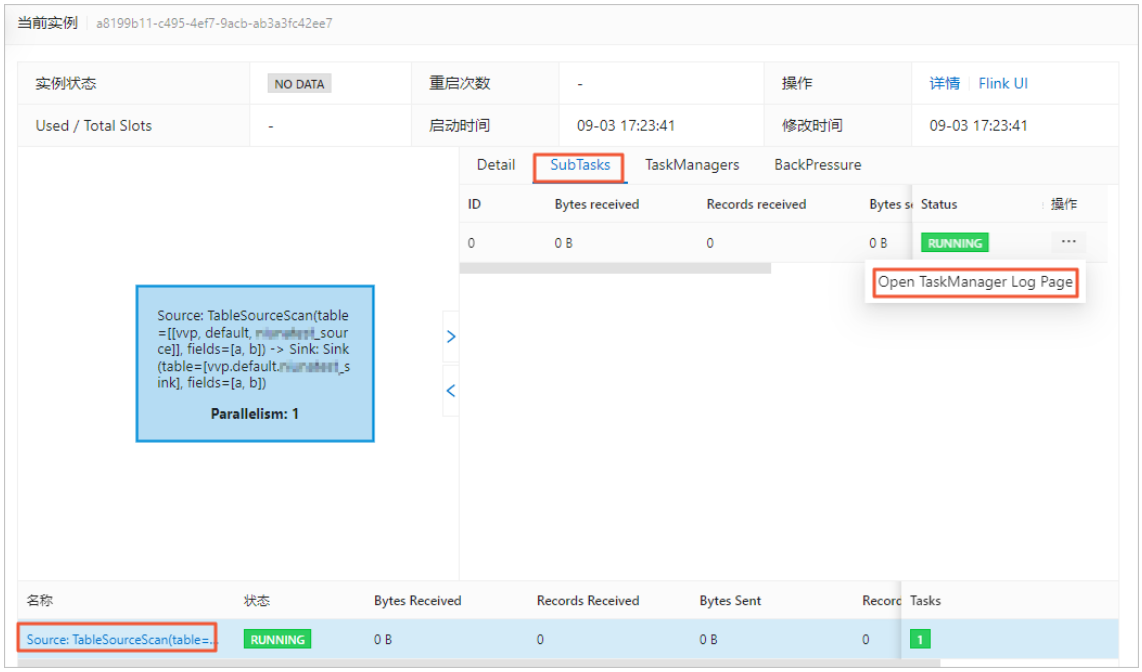
- 查看源表Taskmanager.log日志中是否有异常信息。

  如果有异常报错，请先按照报错提示处理问题。查看源表Taskmanager.log日志的操作如下：

  i. 在作业运维页面，单击目标作业名称。

  ii. 在**作业总览**页签，单击Source节点。

  iii. 在**SubTasks**页签**操作**列，单击**Open TaskManager Log Page**。



iv. 在logs页签，查看日志信息。

在当前页面查找最后一个Caused by信息，即第一个Failover中的Cause by信息，往往是导致作业异常的根因，根据该根因的提示信息，可以快速定位作业异常的原因。

> **说明** 如果以上方面您排查后，仍然解决不了问题，请提交工单咨询。

如何定位Flink无法将数据写入到结果表的问题？

当Flink无法将数据写入到结果表时，建议从以下几个方面进行排查并处理：

- 确认下游存储和Flink全托管之间网络是否连通。

  Flink全托管仅支持访问相同地域、相同VPC下的存储。如果您有访问跨VPC存储资源或者通过公网访问Flink全托管的特殊需求，请查看以下文档：

  - 如果您需要跨VPC访问存储资源，则可以通过5种方式解决，详情请参见如何访问跨VPC里的存储资源？
  - 如果您需要通过公网访问Flink全托管，则可以使用阿里云提供的NAT网关实现VPC网络和公网的连通。详情请参见Flink全托管集群如何访问公网？

- 确认下游存储中是否已配置了白名单。

  下游存储中需要配置白名单的产品包括RDS MYSQL、Kafka、ES、云原生数据仓库AnalyticDB MySQL版3.0、Hbase、Redis和Clickhouse。您可以按照以下步骤配置白名单：

  i. 获取Flink全托管虚拟交换机的网段。

  获取方法请参见设置白名单。

  ii. 在下游存储中配置Flink全托管白名单。

  下游存储中配置白名单的方法，请参见对应DDL文档的前提条件中的文档链接，例如RDS Mysql结果表前提条件。

- 确认DDL中定义的字段类型、字段顺序和字段大小写是否和物理表一致。

下游存储支持的字段类型和Flink全托管支持的字段类型可能不完全一致，但存在一定的映射关系。您需要按照DDL定义的字段类型映射关系一对一匹配，详情请参见对应DDL文档类型映射文档，例如日志服务SLS结果表类型映射。

- 确认数据是否被中间节点过滤了，例如WHERE、JOIN和窗口等。

  具体请查看Vertex拓扑图上每个计算节点数据输入和输出情况。例如WHERE节点输入为5，输出为0，则代表被WHERE节点过滤了，因此下游存储中无数据写入。

- 确认下游存储中设置的输出条件相关参数的默认值是否合适。

  如果您的数据源的数据量较小，但结果表DDL定制中设置的输出条件的默认值较大，会导致一直达不到输出条件，而无法下发数据至下游存储。此时，您需要将输出条件相关参数的默认值改小。常见的下游存储中的输出条件参数情况如下表所示。

输出条件	参数	涉及的下游存储
一次批量写入的条数。	batchSize	○ DataHub ○ Tablestore ○ MongoDB ○ Phoenix5 ○ RDS MYSQL ○ 云原生数据仓库AnalyticDB MySQL版3.0 ○ ClickHouse ○ InfluxDB
每次批量写入数据的最大数据条数。	batchCount	DataHub
Odps tunnel writer缓冲区Flush间隔。	flushIntervalMs	Maxcompute
写入HBase前，内存中缓存的数据量（字节）大小。	sink.buffer-flush.max-size	Hbase
写入HBase前，内存中缓存的数据条数。	sink.buffer-flush.max-rows	Hbase
将缓存数据周期性写入到HBase的间隔，可以控制写入HBase的延迟。	sink.buffer-flush.interval	Hbase
Hologres Sink节点数据攒批的最大值。	jdbcWriteBatchSize	Hologres

- 确认窗口是否因为乱序而导致数据无法输出。

  假如，Flink全托管一开始就流入一条2100年的未来数据，它的Watermark为2100年，系统会默认2100年前的数据已被处理完，只会处理比2100年大的数据。而后续流入的2021年的正常数据因为Watermark小于2100年而被丢弃。直到出现大于2100年的数据流入Flink全托管，则会触发窗口关闭而输出数据，否则就会导致结果表一直没有数据输出。

您可以通过Print Sink或者Log4j的方式确认数据源中是否存在乱序的数据，详情请参见print结果表和配置历史作业实例日志输出。找到乱序数据后，您可以过滤或者采取延迟触发窗口计算的方式处理乱序的数据。

- 确认是否因为个别并发没有数据而导致数据无法输出。

  如果作业为多并发，但个别并发没有数据流入Flink全托管，则它的Watermark就为1970年0点0分，而多个并发的Watermark取最小值，因此就永远没有满足窗口结束的Watermark，就不能触发窗口结束而输出数据。

  此时，您需要检查您上游的Vertex拓扑图的Subtask每个并发是不是都有数据流入。如果有个别并发无数据，建议调整作业并发数小于等于源表Shard数，从而保证所有并发都有数据。

- 确认Kafka的某个分区是否无数据而导致数据无法输出。

  如果Kafka某个分区没有数据，则会影响Watermark的产生，从而导致Kafka源表数据基于Event Time的窗口后，不能输出数据。解决方案请参见为什么Kafka源表数据基于Event Time的窗口后，不能输出数据？

> **说明** 如果以上方面您排查后，仍然解决不了问题，请提交工单咨询。

如何定位数据丢失的问题？

数据经过JOIN、WHERE或窗口等节点时，数据量减少是正常现象，这是因条件限制被过滤或JOIN不上。但如果您的数据丢失异常，建议从以下几个方面进行排查并处理：

- 确认维表Cache缓存策略是否有问题。

  如果维表DDL中Cache缓存策略设置的有问题，则会导致维表的数据没有被拉取到，从而导致数据丢失。此时建议检查并修改作业Cache策略。作业Cache策略详情请参见各维表的Cache策略，例如HBase维表Cache参数。

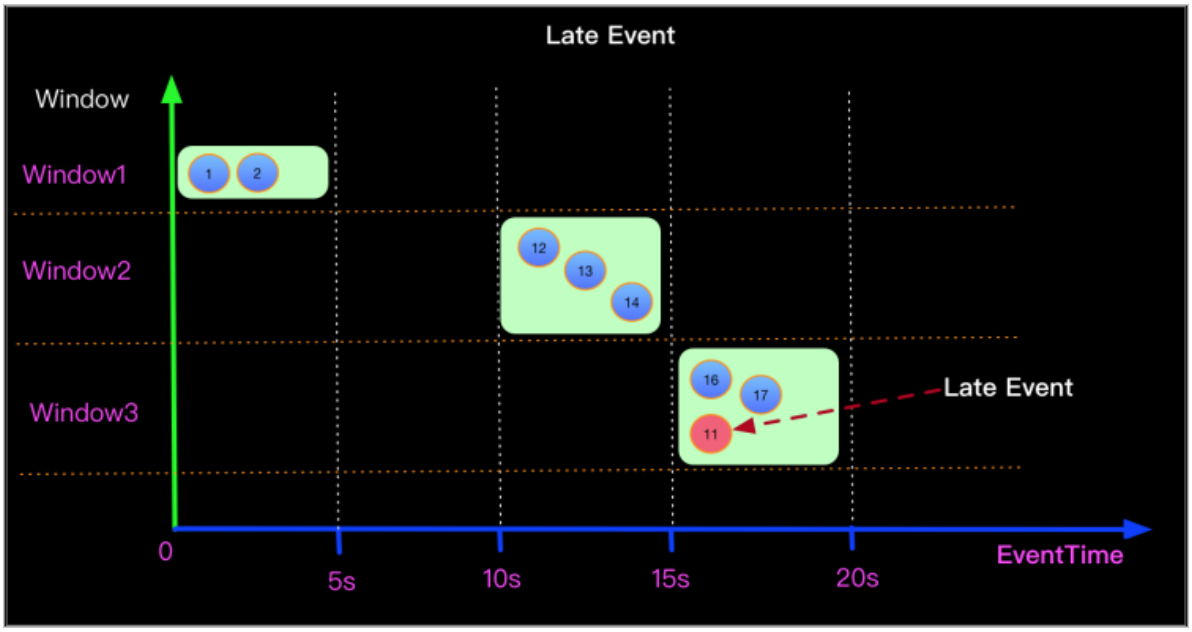
- 确认函数使用方法是否不正确。

  如果您在作业中使用了to_timestamp_tz、date_format等函数，而函数的使用方法不对，导致数据转化出问题，数据被丢失。

  此时，您可以通过Print Sink或者Log4j的方式，单独将使用的函数的信息打印到日志中，确认函数的使用方法是否正确。详情请参见print结果表或配置历史作业实例日志输出。

- 确认数据是否乱序。

如果作业中存在乱序的数据，这些乱序的数据的Watermark不在新窗口的开窗和关窗时间范围内，导致这些数据被丢弃。例如下图中11秒的数据在16秒进入15~20秒的窗口，而它的Watermark为11，会被系统认为是迟到数据，从而导致被丢弃。



通常丢失的数据都是一个窗口的，您可以通过Print Sink或者Log4j的方式确认数据源中是否存在乱序的数据。详情请参见print结果表或配置历史作业实例日志输出。

找到乱序数据后，可以根据乱序的程度，合理的设置Watermark，采取延迟触发窗口计算的方式处理乱序的数据。例如该示例中，可以定义Watermark生成策略为Watermark = Event time -5s，从而让乱序的数据可以被正确的处理。建议以整天整时整分开窗口求聚合，否则数据乱序严重，增加offset后还是会有数据丢失问题。

> **说明** 如果以上方面您排查后，仍然解决不了问题，请提交工单咨询。

10.5. 上下游存储

本文为您介绍实时计算Flink版上下游存储方面的常见问题。

- Kafka源表
  - Flink如何获取JSON数据？
  - Flink和Kafka网络连通，但Flink无法消费或者写入数据？
  - 为什么Kafka源表数据基于Event Time的窗口后，不能输出数据？
  - Flink中的Commit Offset有什么作用？
  - 为什么Flink和Kafka之间的网络是连通的，但是依然会有timeout expired while fetching topic metadata的报错？
- DataHub源表
  - 分裂或者缩容DataHub Topic后导致Flink作业失败，如何恢复？
  - 可以删除正在消费的DataHub Topic吗？
- 全量和增量MaxCompute源表
  - endPoint和tunnelEndpoint是指什么？如果配置错误会产生什么结果？

- 启动作业时出现Akka超时报错，但是全量MaxCompute源表和增量MaxCompute获取Metadata速率正常，应该如何处理？
- 全量MaxCompute和增量MaxCompute是如何读取MaxCompute数据的？
- 引用MaxCompute作为数据源，在作业启动后，向已有的分区或者表里追加数据，这些新数据是否能被全量MaxCompute或增量MaxCompute源表读取？
- 全量MaxCompute和增量MaxCompute源表作业是否支持暂停作业后修改并发数，再恢复作业？
- 作业启动位点设置了2019-10-11 00:00:00， 为什么启动位点前的分区也会被全量MaxCompute源表读取？
- 增量MaxCompute源表监听到新分区时，如果该分区还有数据没有写完，如何处理？
- 报错：ErrorMessage=Authorization Failed [4019], You have NO privilege'ODPS:***'

- MySQL CDC源表
  - 不支持定义Watermark，那如何进行窗口聚合？
  - 如何跳过Snapshot阶段，只从变更数据开始读取？
  - 如何读取一个分库分表的MySQL数据库？
  - 全表读取阶段效率慢、存在反压，应该如何解决？
  - 如何判断CDC作业是否已完成全量数据同步？
  - 报错：com.github.shyiko.mysql.binlog.network.ServerException
  - 报错：The primary key is necessary when enable 'Key: 'scan.incremental.snapshot.enabled'
- RDS MySQL结果表
  - Flink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？
  - 使用RDS表中的唯一索引进行GROUP BY时需要注意什么？
  - 为什么MySQL物理表（包含RDS MySQL和ADB）的INT UNSIGNED字段类型，在Flink SQL中要被声明为其他类型？
  - 报错：Incorrect string value: '\xF0\x9F\x98\x80\xF0\x9F...' for column 'test' at row 1
- ClickHouse结果表

  ClickHouse结果表是否支持回撤更新数据？
- Print结果表

  如何在控制台查看print数据结果？
- Tablestore维表

  维表进行JOIN时，如果查询不到数据，应该如何处理？
- MaxCompute维表

  max_pt()和max_pt_with_done()的区别是什么？

Flink如何获取JSON数据？

- 如果您需要获取普通JSON数据，方法详情请参见JSON Format。
- 如果您需要获取嵌套的JSON数据，则源表DDL中使用ROW格式定义JSON Object，结果表DDL中定义好要获取的JSON数据对应的Key，在DML语句中设置好Key获取的方式，就可以获取到对应的嵌套Key的Value值。代码示例如下：

- 测试数据

```
{
    "a":"abc",
    "b":1,
    "c":{
        "e":["1","2","3","4"],
        "f":{"m":"567"}
    }
}
```

- 源表DDL定义

```
CREATE TEMPORARY TABLE `kafka_table` (
  `a` VARCHAR,
   b int,
  `c` ROW<e ARRAY<VARCHAR>,f ROW<m VARCHAR>>  --c是一个JSON Object，对应Flink里面是ROW；e是json list，对应ARRAY。
) WITH (
  'connector' = 'kafka',
  'topic' = 'xxx',
  'properties.bootstrap.servers' = 'xxx',
  'properties.group.id' = 'xxx',
  'format' = 'json',
  'scan.startup.mode' = 'xxx'
);
```

- 结果表DDL定义

```
CREATE TEMPORARY TABLE `sink` (
 `a` VARCHAR,
  b INT,
  e VARCHAR,
  `m` varchar
) WITH (
  'connector' = 'print',
  'logger' = 'true'
);
```

- DML语句

```
INSERT INTO `sink`
  SELECT
  `a`,
  b,
  c.e[1], --Flink从1开始遍历数组，本示例为获取数组中的元素1。如果获取整个数组，则去掉[1]。
  c.f.m
FROM `kafka_table`;
```

- 测试结果

```
Metrics   Logs   Thread Dump   Log List
409 2021-04-08 10:13:11,214 INFO  org.apache.flink.kafka.shaded.org.apache.kafka.common.utils.AppInfo
410 2021-04-08 10:13:11,215 INFO  org.apache.flink.kafka.shaded.org.apache.kafka.common.utils.AppInfoParser [] - Kafka commitId: cb8625948210849f
411 2021-04-08 10:13:11,215 INFO  org.apache.flink.kafka.shaded.org.apache.kafka.common.utils.AppInfoParser [] - Kafka startTimeMs: 1617847991214
412 2021-04-08 10:13:11,270 INFO  org.apache.flink.kafka.shaded.org.apache.kafka.clients.consumer.KafkaConsumer [] - [Consumer clientId=consumer-lb_test-2,
    groupId=lb_test] Subscribed to partition(s): lb_test-0, lb_test-1, lb_test-2, lb_test-3, lb_test-4, lb_test-5
413 2021-04-08 10:13:11,280 INFO  org.apache.flink.kafka.shaded.org.apache.kafka.clients.consumer.KafkaConsumer [] - [Consumer clientId=consumer-lb_test-2,
    groupId=lb_test] Seeking to offset 1 for partition lb_test-0
414 2021-04-08 10:13:11,285 INFO  org.apache.flink.kafka.shaded.org.apache.kafka.clients.consumer.KafkaConsumer [] - [Consumer clientId=consumer-lb_test-2,
    groupId=lb_test] Seeking to offset 0 for partition lb_test-1
415 2021-04-08 10:13:11,285 INFO  org.apache.flink.kafka.shaded.org.apache.kafka.clients.consumer.KafkaConsumer [] - [Consumer clientId=consumer-lb_test-2,
    groupId=lb_test] Seeking to offset 0 for partition lb_test-2
416 2021-04-08 10:13:11,285 INFO  org.apache.flink.kafka.shaded.org.apache.kafka.clients.consumer.KafkaConsumer [] - [Consumer clientId=consumer-lb_test-2,
    groupId=lb_test] Seeking to offset 0 for partition lb_test-3
417 2021-04-08 10:13:11,290 INFO  org.apache.flink.kafka.shaded.org.apache.kafka.clients.consumer.KafkaConsumer [] - [Consumer clientId=consumer-lb_test-2,
    groupId=lb_test] Seeking to offset 0 for partition lb_test-4
418 2021-04-08 10:13:11,291 INFO  org.apache.flink.kafka.shaded.org.apache.kafka.clients.consumer.KafkaConsumer [] - [Consumer clientId=consumer-lb_test-2,
    groupId=lb_test] Seeking to offset 0 for partition lb_test-5
419 2021-04-08 10:13:11,302 INFO  org.apache.flink.kafka.shaded.org.apache.kafka.clients.Metadata [] - [Consumer clientId=consumer-lb_test-2, groupId=lb_test]
    Cluster ID: -1flJPwnTvuGFSuyCtU1hw
420 2021-04-08 10:15:31,597 INFO  org.apache.flink.api.common.functions.util.PrintSinkOutputWriter [] - +I(abc,1,1,567)
```

Flink和Kafka网络连通，但Flink无法消费或者写入数据？

- 问题原因

  如果Flink与Kafka之间存在代理或端口映射等转发机制，则Kafka客户端拉取Kafka服务端的网络地址为Kafka服务器本身的地址而非代理的地址。此时，虽然Flink与Kafka之间网络连通，但是Flink无法消费或者写入数据。

  Flink和Kafka客户端（Flink Kafka Connector）之间建立连接分为两个步骤：

  i. Kafka客户端拉取Kafka服务端（Kafka Broker）元信息，包括Kafka服务端所有Broker的网络地址。

  ii. Flink使用Kafka客户端拉取下来的Kafka服务端网络地址来消费或者写入数据。

- 排查方法

  通过以下步骤来确认Flink与Kafka之间是否存在代理或端口映射等转发机制：

  i. 使用ZooKeeper命令行工具（zkCli.sh或zookeeper-shell.sh）登录您Kafka使用的ZooKeeper集群。

  ii. 根据您的集群实际情况执行正确的命令，来获取您的Kafka Broker元信息。

  通常可以使用 `get /brokers/ids/0` 命令来获取Kafka Broker元信息。Kafka的连接地址位于 `endpoints` 字段中。

```
# bin/zookeeper-shell.sh localhost:2181
Connecting to localhost:2181
Welcome to ZooKeeper!
JLine support is disabled

WATCHER::

WatchedEvent state:SyncConnected type:None path:null
get /brokers/ids/0
{"listener_security_protocol_map":{"PLAINTEXT":"PLAINTEXT"},"endpoints":["PLAINTEXT://116.62.[redacted]:9092"] "jmx_port":-1,"host":"116.62.[redacted]","timestamp":"1614840078030","port":9092,"version":4}
```

  iii. 使用 `ping` 或 `telnet` 等命令来测试Endpoint中显示的地址与Flink的连通性。

  如果无法连通，则代表Flink与Kafka之间存在代理或端口映射等转发机制。

- 解决方案
  - 不使用代理或端口映射等转发机制，直接打通Flink与Kafka之间的网络，使Flink能够直接连通Kafka元信息中显示的Endpoint。
  - 联系Kafka运维人员，将转发地址作为Kafka Broker端的advertised.listeners，以使Kafka客户端拉取的Kafka服务端元信息包含转发地址。

> **说明** 仅Kafka 0.10.2.0及以上版本支持将代理地址添加到Kafka Broker的Listener中。

如果您想了解更多的关于该问题的原理和解释，请参见KIP-103：区分内部与外部网络流量和Kafka网络连接问题详解。

为什么Kafka源表数据基于Event Time的窗口后，不能输出数据？

- 问题详情

  Kafka作为源表，基于Event Time的窗口后，不能输出数据。

- 问题原因

  Kafka某个分区没有数据，会影响Watermark的产生，从而导致Kafka源表数据基于Event Time的窗口后，不能输出数据。

- 解决方案

  i. 确保所有分区都存在数据。

  ii. 开启源数据空闲监测功能。在目标作业详情页面右上角，单击**编辑**后，在页面右侧**高级配置**面板的**更多Flink配置**中，添加如下代码后保存生效。

  ```
  table.exec.source.idle-timeout: 5
  ```

  table.exec.source.idle-timeout参数详情，请参见Configuration。

Flink中的Commit Offset有什么作用？

Flink在每次Checkpoint成功时，才会向Kafka提交当前读取Offset。如果未开启Checkpoint，或者Checkpoint设置的间隔过大，在Kafka端可能会查询不到当前读取的Offset。

为什么Flink和Kafka之间的网络是连通的，但是依然会有timeout expired while fetching topic metadata的报错？

Flink和Kafka之间的网络连通并不意味着能读取数据，只有Kafka Broker在bootstrap过程中返回的集群metadata中描述的Endpoint， 才可以连通Flink和Kafka，并读取到Kafka的数据，详情请参见Flink-cannot-connect-to-Kafka。检查办法为：

1. 使用zkCli.sh或者zookeeper-shell.sh工具登录Kafka使用的Zookeeper。
2. 执行 `ls /brokers/ids` 命令列出所有的Kafka Broker ID。
3. 使用 `get /brokers/ids/{your_broker_id}` 命令查看Broker metadata信息。

   Endpoint信息显示在listener_security_protocol_map中。
4. 确认Flink是否可以连通该Endpoint。

   如果该Endpoint中使用了域名，请为Flink配置对应的域名解析服务。

分裂或者缩容DataHub Topic后导致Flink作业失败，如何恢复？

如果分裂或者缩容了Flink正在读取的某个Topic，则会导致任务持续出错，无法自行恢复。该情况下需要重新启动（先停止再启动）来使任务恢复正常。

可以删除正在消费的DataHub Topic吗？

不支持删除或重建正在消费的DataHub Topic。

endPoint和tunnelEndpoint是指什么？如果配置错误会产生什么结果？

endPoint和tunnelEndpoint参数说明参见Endpoint。VPC环境中这两个参数如果配置错误可能会导致任务异常，异常情况详情如下：

- 如果endPoint配置错误，则任务上线停滞在91%的进度。
- 如果tunnelEndpoint配置错误，则任务运行失败。

启动作业时出现Akka超时报错，但是全量MaxCompute源表和增量MaxCompute获取Metadata速率正常，应该如何处理？

请合并小文件，具体步骤请参见文档小文件优化及作业诊断常见问题。

全量MaxCompute和增量MaxCompute是如何读取MaxCompute数据的？

全量MaxCompute和增量MaxCompute是通过Tunnel读取MaxCompute数据的，读取速度受MaxCompute Tunnel带宽限制。

引用MaxCompute作为数据源，在作业启动后，向已有的分区或者表里追加数据，这些新数据是否能被全量MaxCompute或增量MaxCompute源表读取？

启动Flink作业后，如果正在被Source读取或已经被Source读取完成的表或分区有新的数据追加，则这部分数据不会被读取，而且可能导致作业Failover。

全量MaxCompute和增量MaxCompute源表均使用 `ODPS DOWNLOAD SESSION` 读取表数据或者分区数据。新建 `DOWNLOAD SESSION` ，服务端会创建一个Index文件，相当于创建 `DOWNLOAD SESSION` 一瞬间数据的映射，后续的数据读取以这个映射为基础。因此在新建 `DOWNLOAD SESSION` 后，追加到MaxCompute表或者分区里的数据，正常流程下是不会被读取的。但如果MaxCompute源表中写入新数据，则会出现两种异常：

- 产生报错：如果Tunnel在读取数据的过程中写入新数据，则会产生报错 `ErrorCode=TableModified,ErrorMessage=The specified table has been modified since the download initiated.` 。
- 数据正确性无法保证：如果在Tunnel已经关闭后写入新数据，则这些数据不会被读取。但当作业发生Failover或者暂停后恢复作业，已经读过的数据可能会被重读，新写入的数据可能被读不全。

全量MaxCompute和增量MaxCompute源表作业是否支持暂停作业后修改并发数，再恢复作业？

MaxCompute源表暂时不支持暂停作业后修改并发数，再恢复作业。系统根据并发数，计算每个并发读取哪些分区的哪些数据。此外，每个并发会把消费情况记录至State，以便暂停恢复后或者Failover后，系统可以从上次读取的位置继续读取数据。

如果暂停作业后，修改了全量MaxCompute和增量MaxCompute源表的并发数后再恢复作业，会对作业产生无法预估的影响。因为已经读取的数据可能会被再次读取，没有读的数据可能会被遗漏。

作业启动位点设置了 `2019-10-11 00:00:00` ，为什么启动位点前的分区也会被全量MaxCompute源表读取？

设置启动位点只对消息队列（例如DataHub）类型的数据源有效，对MaxCompute源表无效。Flink作业启动后数据读取的范围如下：

- 分区表：读取当前所有分区。
- 非分区表：读取当前存在的数据。

增量MaxCompute源表监听到新分区时，如果该分区还有数据没有写完，如何处理？

没有机制可以标志一个分区的数据是否已经完整。目前只要监听到新分区，就会读入。用增量MaxCompute源表读取一个MaxCompute分区表T，分区列是ds，推荐的写入方法为：不创建分区，先执行Insert overwrite table T partition (ds='20191010') ...语句，作业结束且成功后，分区和数据一起可见。

> **注意** 不允许的写入方法为：先创建好分区，例如ds=20191010，再往分区里写数据。增量MaxCompute源表监听到新分区ds=20191010，立刻读入，如果此时该分区还有数据没有写完，就会漏读数据。

报错：ErrorMessage=Authorization Failed [4019], You have NO privilege'ODPS:***'

- 报错详情

  作业运行过程中会在Failover页面或TaskManager.log页面报错，报错信息如下。

```
ErrorMessage=Authorization Failed [4019], You have NO privilege'ODPS:***'
```

- 报错原因

  MaxCompute DDL定义中填写的用户身份信息无法访问MaxCompute。

- 解决方案

  通过阿里云账号、RAM用户账号或RAM角色认证用户身份，详情请参见用户认证。如果您有其他疑问，请提交工单咨询，产品名称选择为MaxCompute。

不支持定义Watermark，那如何进行窗口聚合？

如果您需要在MySQL CDC源表上进行窗口聚合，可以考虑采用非窗口聚合的方式，即将时间字段转换成窗口值，然后根据窗口值进行GROUP BY聚合。例如，统计每个店铺每分钟的订单数和销量，代码示例如下。

```
SELECT shop_id, DATE_FORMAT(order_ts, 'yyyy-MM-dd HH:mm'), COUNT(*), SUM(price) FROM order_
mysql_cdc GROUP BY shop_id, DATE_FORMAT(order_ts, 'yyyy-MM-dd HH:mm')
```

如何跳过Snapshot阶段，只从变更数据开始读取？

可以通过WITH参数debezium.snapshot.mode来控制，您可以设置为：

- never：在启动时，不读取数据库的快照，而是直接从变更的最开始位置读取。但需要注意MySQL的变更旧数据可能会被自动清理，因此不能保证变更数据中包含了全量的数据，读取的数据不完整。
- schema_only：如果你不需要保证数据的一致性，只关心作业启动后数据库的新增变更，则可以设置为schema_only，仅快照schema，不快照数据，从变更的最新数据开始读取。

如何读取一个分库分表的MySQL数据库？

如果MySQL是一个分库分表的数据库，分成了user_00、user_02和user_99等多个表，且所有表的schema一致。则可以通过table-name选项，指定一个正则表达式来匹配读取多张表，例如设置table-name为user_.*，监控所有user_前缀的表。database-name选项也支持该功能，但需要所有的表schema一致。

全表读取阶段效率慢、存在反压，应该如何解决？

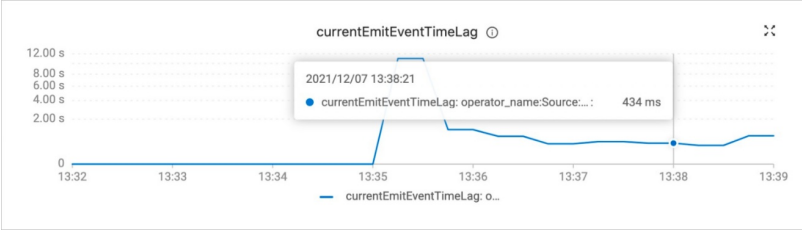
可能是下游节点处理太慢导致反压了。因此您需要先排查下游节点是否存在反压。如果存在，则需要先解决下游节点的反压问题。您可以通过以下方式处理：

- 增加并发数。

- 开启minibatch等聚合优化参数（下游聚合节点）。

如何判断CDC作业是否已完成全量数据同步？

您可以通过数据曲线页面currentEmitEventTimeLag指标来判断作业是否完成了全量数据同步。currentEmitEventTimeLag指标记录的是 Source发送一条记录到下游节点的时间点和该记录在数据库中生成时间点的差值，用于衡量数据从数据库中产生到离开Source节点的延迟。



currentEmitEventTimeLag指标取值含义如下：

- 当该指标为0时，则代表作业还在全量数据同步阶段。
- 当该指标大于0时，则代表作业完成了全量数据同步，进入了Binlog读取阶段。

报错：com.github.shyiko.mysql.binlog.network.ServerException

- 报错详情

```
Caused by: com.github.shyiko.mysql.binlog.network.ServerException: The slave is connectin
g using CHANGE MASTER TO MASTER_AUTO_POSITION = 1, but the master has purged binary logs
containing GTIDs that the slave requires.
```

- 报错原因

  MySQL CDC在启动全量读取之前记录Binlog位点，等全量读取结束后再从Binlog位点读取增量数据。该报错一般是因为全量读取耗时太长，超过了MySQL Binlog的淘汰周期，导致MySQL Binlog位点的数据已经被MySQL清理掉了。

- 解决方案

  查看MySQL Binlog的清理规则，例如时间、存储空间和文件个数等，建议保留Binlog一天以上，RDS Binlog详情请参见删除本地日志（Binlog）。

> **说明**：VVR 4.0.8及以上版本，MySQL CDC支持并发读取全量数据，可以提高全量数据的读取速度，针对该问题会起到缓解作用。

报错：The primary key is necessary when enable 'Key: 'scan.incremental.snapshot.enabled'

- 报错详情

  MySQL CDC源表在Flink 1.13版本语法检查时报错详情如下。

```
Caused by: org.apache.flink.table.api.ValidationException: The primary key is necessary w
hen enable 'Key: 'scan.incremental.snapshot.enabled' , default: true (fallback keys: [])'
to 'true'
    at com.alibaba.ververica.cdc.connectors.mysql.table.MySqlTableSourceFactory.validateP
rimaryKeyIfEnableParallel(MySqlTableSourceFactory.java:186)
    at com.alibaba.ververica.cdc.connectors.mysql.table.MySqlTableSourceFactory.createDyn
amicTableSource(MySqlTableSourceFactory.java:85)
    at org.apache.flink.table.factories.FactoryUtil.createTableSource(FactoryUtil.java:13
4)
    ... 30 more
```

- 报错原因

  在MySQL CDC DDL WITH参数中，您未设置主键（Primary Key）信息。因为Flink 1.13版本，新增支持按PK分片，进行多并发读取数据的功能。

  > **说明** Flink 1.12及以下版本，MySQL CDC源表仅支持单并发读取数据。

- 解决方案
  - 如果在Flink 1.13版本，您需要多并发读取MySQL数据，则在DDL中添加PK信息。
  - 如果在Flink 1.13版本，您不需要多并发读取MySQL数据，则在DDL中添加 `scan.incremental.snapshot.enabled` 参数，且把该参数值设置为false，无需设置PK信息。

Flink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？

如果在DDL中定义了主键，会采用 `INSERT INTO tablename(field1,field2, field3, ...) VALUES(value1, value2, value3, ...) ON DUPLICATE KEY UPDATE field1=value1,field2=value2, field3=value3, ...;` 的方式更新记录，即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。

使用RDS表中的唯一索引进行GROUP BY时需要注意什么？

- 需要在作业中的GROUP BY中声明该唯一索引。
- RDS中只有一个自增主键，Flink作业中不能声明为PRIMARY KEY。

为什么MySQL物理表（包含RDS MySQL和ADB）的INT UNSIGNED字段类型，在Flink SQL中要被声明为其他类型？

因为MySQL的JDBC Driver在获取数据时，由于精度问题，会采用不同的数据类型进行承接。具体说来，对于MySQL的INT UNSIGNED类型会使用LONG类型来承接数据，而对于MySQL的BIGINT UNSIGNED类型会使用BIGINTEGER类型来承接数据。

报错：Incorrect string value: '\xF0\x9F\x98\x80\xF0\x9F...' for column 'test' at row 1

- 报错详情

```
Caused by: java.sql.BatchUpdateException: Incorrect string value: '\xF0\x9F\x98\x80\xF0\x
9F...' for column 'test' at row 1
at sun.reflect.GeneratedConstructorAccessor59.newInstance(Unknown Source)
at sun.reflect.DelegatingConstructorAccessorImpl.newInstance(DelegatingConstructorAccesso
rImpl.java:45)
at java.lang.reflect.Constructor.newInstance(Constructor.java:423)
at com.mysql.cj.util.Util.handleNewInstance(Util.java:192)
at com.mysql.cj.util.Util.getInstance(Util.java:167)
at com.mysql.cj.util.Util.getInstance(Util.java:174)
at com.mysql.cj.jdbc.exceptions.SQLError.createBatchUpdateException(SQLError.java:224)
at com.mysql.cj.jdbc.ClientPreparedStatement.executeBatchedInserts(ClientPreparedStatemen
t.java:755)
at com.mysql.cj.jdbc.ClientPreparedStatement.executeBatchInternal(ClientPreparedStatement
.java:426)
at com.mysql.cj.jdbc.StatementImpl.executeBatch(StatementImpl.java:796)
at com.alibaba.druid.pool.DruidPooledPreparedStatement.executeBatch(DruidPooledPreparedSt
atement.java:565)
at com.alibaba.ververica.connectors.rds.sink.RdsOutputFormat.executeSql(RdsOutputFormat.j
ava:488)
... 15 more
```

- 报错原因

  数据中存在Emoji表情，导致数据库编码不能被正常解析。

- 解决方案

  在使用JDBC连接MySQL数据库时，URL地址后面需要加上UTF-8，例如 `jdbc:mysql://<内网地址>/<databaseName>?characterEncoding=UTF-8` 。详情请参见Using Character Sets and Unicode。

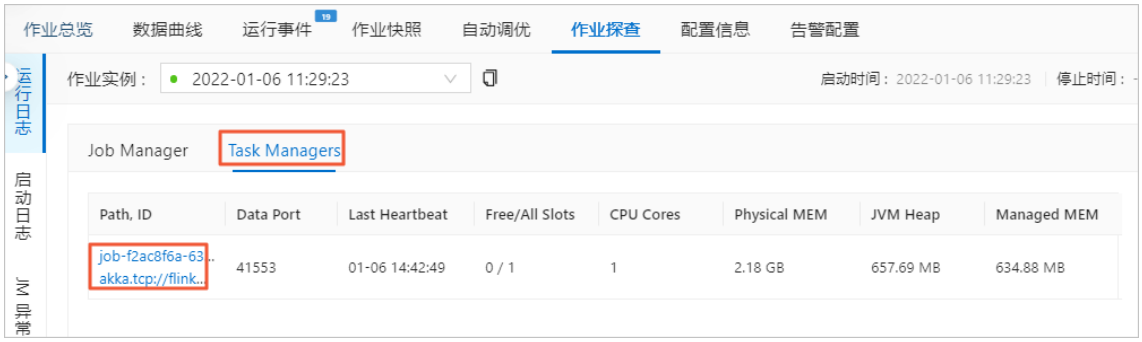
ClickHouse结果表是否支持回撤更新数据？

当Flink结果表的DDL上指定了Primary Key时，则支持回撤更新数据。因为ClickHouse是一个用于联机分析（OLAP）的列式数据库管理系统，对UPDATE和DELETE的支持不够完善。如果在Flink DDL上指定了Primary Key，则尝试使用ALTER TABLE UPDATE和DELETE来更新和删除数据，因此会造成性能显著下降。

如何在控制台查看print数据结果？

查看print数据结果的步骤，有以下两种方式：

- 在Flink开发控制台查看：
  i. 在Flink全托管开发控制台左侧导航栏，选择**应用 > 作业运维**。
  ii. 单击目标作业名称。
  iii. 单击**作业探查**。
  iv. 在左侧**运行日志**页签，选择正在运行的作业实例。
  v. 在**Task Managers**页签，单击**Path, ID**。



vi. 单击**日志**，查看print数据结果。

- 跳转到Flink UI界面查看：
  i. 在在Flink全托管开发控制台左侧导航栏，选择**应用 > 作业运维**。
  ii. 单击目标作业名称。
  iii. 在目标作业**作业总览**页签，单击**Flink UI**。

iv. 单击**Task Managers**。
v. 单击**Path, ID**。
vi. 在**logs**页签，查看print数据结果。

维表进行JOIN时，如果查询不到数据，应该如何处理？

检查DDL语句和物理表中的Schema类型和名称是否一致。

max_pt()和max_pt_with_done()的区别是什么？

max_pt()选取的是所有分区中字典序最大的分区。max_pt_with_done()选取的是所有分区中字典序最大，且伴随有 `.done` 分区的分区。如果分区列表示例如下：

- ds=20190101
- ds=20190101.done
- ds=20190102
- ds=20190102.done
- ds=20190103

则 `max_pt()` 和 `max_pt_with_done()` 的区别如下：

- `` `partition`='max_pt_with_done()' `` 匹配的分区是 `ds=20190102` 。
- `` `partition`='max_pt()' `` ，匹配的分区是 `ds=20190103` 。

10.6. SQL常见问题

本文为您介绍实时计算Flink版的SQL常见问题，包括作业开发报错和作业运维报错。

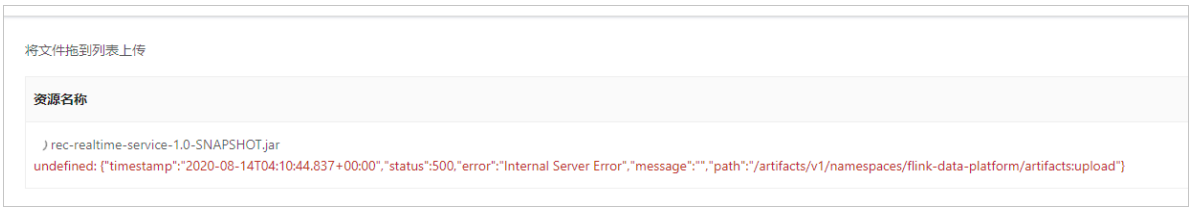
- 作业开发报错

- 报错：undefined
- 报错：Object '****' not found
- 报错：Only a single 'INSERT INTO' is supported

- 作业运维报错
  - 报错：exceeded quota: resourcequota
  - 报错：Exceeded checkpoint tolerable failure threshold
  - 报错：Flink version null is not configured for sql
  - 报错：java.util.concurrent.CompletionException: java.lang.IllegalArgumentException
  - 报错：The difference between the request time and the current time is too large
  - INFO：org.apache.flink.fs.osshadoop.shaded.com.aliyun.oss

报错：undefined

- 报错详情



- 报错原因

  您的JAR包较大。

- 解决方案

  您可以在OSS管理控制台上传JAR包，详情请参见如何在OSS控制台上传JAR包？

报错：Object '****' not found

- 报错详情

单击**运行**后，验证报错详情如下。

```
1  --****************************************************************--
2  -- Author:
3  -- Created Time:   2021-04-19 15:12:03
4  -- Description:    Write your description here
5  --****************************************************************--
6  CREATE  TABLE datagen_source (`a` BIGINT, b int, `c` VARCHAR)
7    WITH ('connector' = 'datagen');
8
9  CREATE  TABLE print_sink (`c` VARCHAR, VAR1 int)
10   WITH ('connector' = 'print','logger' = 'true');
11
12 insert into print_sink
13 SELECT `c`,8 from datagen_source;
14
```

```
问题: test_
SQL validation failed. From line 13, column 19 to line 13, column 32: Object 'datagen_source' not found.
    StackTrace:
    org.apache.flink.table.api.ValidationException: SQL validation failed. From line 13, column 19 to line 13, column 32: Object
'datagen_source' not found
    at org.apache.flink.table.planner.calcite.FlinkPlannerImpl.org$apache$flink$table$planner$calcite$FlinkPlannerImpl$$validate
(FlinkPlannerImpl.scala:153)
    at org.apache.flink.table.planner.calcite.FlinkPlannerImpl.validate(FlinkPlannerImpl.scala:112)
    at org.apache.flink.table.planner.operations.SqlToOperationConverter.convert(SqlToOperationConverter.java:189)
```

- 报错原因

  在DDL和DML同在一个文本中提交运行时，DDL没有声明为CREATE TEMPORARY TABLE。

- 解决方案

  在DDL和DML同在一个文本中提交运行时，DDL需要声明为CREATE TEMPORARY TABLE，而不是声明为CREATE TABLE。

报错：Only a single 'INSERT INTO' is supported

- 报错详情

单击**运行**后，验证报错详情如下。

```
1  --********************************************************************--
2  -- Author:
3  -- Created Time:   2021-04-19 15:12:03
4  -- Description:    Write your description here
5  --********************************************************************--
6  CREATE TEMPORARY TABLE datagen_source (`a` BIGINT, b int, `c` VARCHAR)
7    WITH ('connector' = 'datagen');
8
9  CREATE TEMPORARY TABLE print_sink (`c` VARCHAR, i1 int)
10   WITH ('connector' = 'print','logger' = 'true');
11 CREATE TEMPORARY TABLE print_sink2 (`c` VARCHAR, i2 int)
12   WITH ('connector' = 'print','logger' = 'true');
13
14 insert into print_sink
15 SELECT `c`,8 from datagen_source;
16
17 insert into print_sink
18 SELECT `c`,8 from datagen_source;
19
```

```
问题: test_
Only a single 'INSERT INTO' is supported..
    StackTrace:
    org.apache.flink.table.api.TableException: Only a single 'INSERT INTO' is supported.
    at com.ververica.platform.sql.environment.VvpSqlParser.lambda$parse$0(VvpSqlParser.java:65)
    at java.base/java.lang.Iterable.forEach(Iterable.java:75)
    at com.ververica.platform.sql.environment.VvpSqlParser.parse(VvpSqlParser.java:62)
    at com.ververica.platform.sql.service.SqlScriptValidationService$SqlScriptValidator.get(SqlScriptValidationService.java:136)
    at com.ververica.platform.sql.service.SqlScriptValidationService$SqlScriptValidator.get(SqlScriptValidationService.java:121)
```

- 报错原因

  多个DML语句没有写在关键语句 `BEGIN STATEMENT SET;` 和 `END;` 之间。

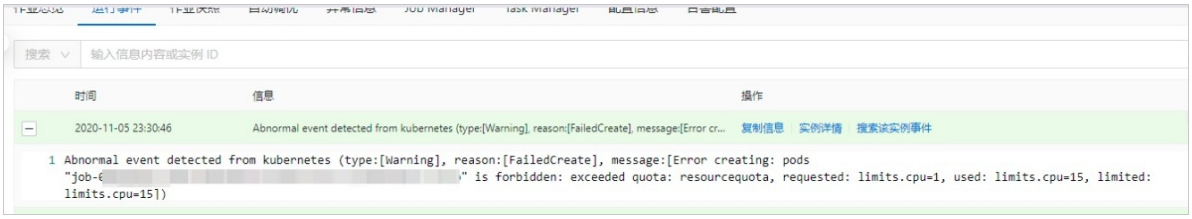
- 解决方案

  将多个DML语句写在 `BEGIN STATEMENT SET;` 和 `END;` 之间。详情请参见INSERT INTO语句。

报错：exceeded quota: resourcequota

- 报错详情

  作业启动过程中报错。



- 报错原因

  当前项目空间资源不足导致作业启动失败。

- 解决方案

  您需要对项目资源进行资源变配，详情请参见创建与管理项目空间。

报错：Exceeded checkpoint tolerable failure threshold

- 报错详情

作业运行过程中报错。

```
2021-05-06 04:27:24
org.apache.flink.util.FlinkRuntimeException: Exceeded checkpoint tolerable failure threshold.
    at org.apache.flink.runtime.checkpoint.CheckpointFailureManager.handleJobLevelCheckpointException(CheckpointFailureManager.java:66)
    at org.apache.flink.runtime.checkpoint.CheckpointCoordinator.abortPendingCheckpoint(CheckpointCoordinator.java:1772)
    at org.apache.flink.runtime.checkpoint.CheckpointCoordinator.abortPendingCheckpoint(CheckpointCoordinator.java:1749)
    at org.apache.flink.runtime.checkpoint.CheckpointCoordinator.access$600(CheckpointCoordinator.java:94)
    at org.apache.flink.runtime.checkpoint.CheckpointCoordinator$CheckpointCanceller.run(CheckpointCoordinator.java:1882)
    at java.util.concurrent.Executors$RunnableAdapter.call(Executors.java:511)
    at java.util.concurrent.FutureTask.run(FutureTask.java:266)
    at java.util.concurrent.ScheduledThreadPoolExecutor$ScheduledFutureTask.access$201(ScheduledThreadPoolExecutor.java:186)
    at java.util.concurrent.ScheduledThreadPoolExecutor$ScheduledFutureTask.run(ScheduledThreadPoolExecutor.java:299)
    at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1147)
    at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:622)
    at java.lang.Thread.run(Thread.java:834)
```

- 报错原因

  未设置任务允许Checkpoint失败的次数，系统默认Checkpoint失败一次就触发一次Failover。

- 解决方案

  i. 在作业开发页面右侧，单击**高级配置**。

  ii. 在**更多Flink配置**文本框，输入如下参数。

  ```
  execution.checkpointing.tolerable-failed-checkpoints: num
  ```

  您需要设置num值来调整任务允许Checkpoint失败的次数。num需要为0或正整数。如果num为0时，则表示不允许存在任何Checkpoint异常或者失败。

报错：Flink version null is not configured for sql

- 报错详情

  ```
  StatusRuntimeException: INTERNAL: Flink version null is not configured for sql.
  ```

- 报错原因

  系统升级至VVR 4.0.8，导致作业的Flink计算引擎版本信息没有了。

- 解决方案

  在作业开发页面右侧高级配置面版中，配置正确的Flink计算引擎版本。

> **说明** 如果您需要使用SQL Preview功能，则还需要检查Session集群页面的引擎版本是否选择正确。

报错：java.util.concurrent.CompletionException: java.lang.IllegalArgumentException

- 报错详情

```
 Caused by: java.util.concurrent.CompletionException: java.lang.IllegalArgumentException:
The bucket name "fakeBucket" is invalid. A bucket name must: 1) be comprised of lower-cas
e characters, numbers or dash(-); 2) start with lower case or numbers; 3) be between 3-63
characters long.
```

- 报错原因

  您在作业**高级配置**面板的**更多Flink配置**中，配置了state.backend参数，但未配置对应的state.checkpoints.dir参数。

- 解决方案

  配置了state.backend参数，必须配置对应的state.checkpoints.dir参数。

> **说明** 您也可以直接删除state.backend参数。

报错：The difference between the request time and the current time is too large

- 报错详情

```
Caused by: org.apache.flink.fs.osshadoop.shaded.com.aliyun.oss.OSSException: The differen
ce between the request time and the current time is too large.
[ErrorCode]: RequestTimeTooSkewed
[RequestId]: 61691C47AF60CA3639812A28
[HostId]: oss-bucket-name.oss-cn-shanghai-internal.aliyuncs.com
[ResponseError]:
<?xml version="1.0" encoding="UTF-8"?>
<Error>
  <Code>RequestTimeTooSkewed</Code>
  <Message>The difference between the request time and the current time is too large.</Me
ssage>
  <RequestId>61691C47AF60C******12A28</RequestId>
  <HostId>******.oss-cn-shanghai-internal.aliyuncs.com</HostId>
  <MaxAllowedSkewMilliseconds>900000</MaxAllowedSkewMilliseconds>
  <RequestTime>2021-10-15T05:57:31.000Z</RequestTime>
  <ServerTime>2021-10-15T06:14:31.000Z</ServerTime>
</Error>
    at org.apache.flink.fs.osshadoop.shaded.com.aliyun.oss.common.utils.ExceptionFactory.
createOSSException(ExceptionFactory.java:100) ~[?:?]
    at org.apache.flink.fs.osshadoop.shaded.com.aliyun.oss.internal.OSSErrorResponseHandl
er.handle(OSSErrorResponseHandler.java:70) ~[?:?]
    at org.apache.flink.fs.osshadoop.shaded.com.aliyun.oss.common.comm.ServiceClient.hand
leResponse(ServiceClient.java:257) ~[?:?]
    at org.apache.flink.fs.osshadoop.shaded.com.aliyun.oss.common.comm.ServiceClient.send
RequestImpl(ServiceClient.java:140) ~[?:?]
    at org.apache.flink.fs.osshadoop.shaded.com.aliyun.oss.common.comm.ServiceClient.send
Request(ServiceClient.java:70) ~[?:?]
    at org.apache.flink.fs.osshadoop.shaded.com.aliyun.oss.internal.OSSOperation.send(OSS
Operation.java:83) ~[?:?]
    at org.apache.flink.fs.osshadoop.shaded.com.aliyun.oss.internal.OSSOperation.doOperat
ion(OSSOperation.java:145) ~[?:?]
    at org.apache.flink.fs.osshadoop.shaded.com.aliyun.oss.internal.OSSOperation.doOperat
ion(OSSOperation.java:102) ~[?:?]
    at org.apache.flink.fs.osshadoop.shaded.com.aliyun.oss.internal.OSSBucketOperation.li
stObjects(OSSBucketOperation.java:411) ~[?:?]
    at org.apache.flink.fs.osshadoop.shaded.com.aliyun.oss.OSSClient.listObjects(OSSClien
t.java:443) ~[?:?]
    at org.apache.flink.fs.shaded.hadoop3.org.apache.hadoop.fs.aliyun.oss.AliyunOSSFileSy
stemStore.listObjects(AliyunOSSFileSystemStore.java:506) ~[?:?]
    at org.apache.flink.fs.shaded.hadoop3.org.apache.hadoop.fs.aliyun.oss.AliyunOSSFileSy
stem.getFileStatus(AliyunOSSFileSystem.java:264) ~[?:?]
    at org.apache.flink.fs.shaded.hadoop3.org.apache.hadoop.fs.aliyun.oss.AliyunOSSFileSy
stem.validatePath(AliyunOSSFileSystem.java:548) ~[?:?]
    at org.apache.flink.fs.shaded.hadoop3.org.apache.hadoop.fs.aliyun.oss.AliyunOSSFileSy
stem.mkdirs(AliyunOSSFileSystem.java:532) ~[?:?]
    at org.apache.flink.fs.shaded.hadoop3.org.apache.hadoop.fs.FileSystem.mkdirs(FileSyst
em.java:2326) ~[?:?]
    at org.apache.flink.fs.osshadoop.common.HadoopFileSystem.mkdirs(HadoopFileSystem.java
:196) ~[?:?]
```

- 报错原因

  您的OSS Bucket开通了版本控制功能，但未配置生命周期规则来定期清理删除标记和不必要的历史版本。过多的删除标记会导致List慢，作业启动失败的问题。详情请参见注意事项。

- 解决方案

  在OSS上配置生命周期规则来定期清理删除标记和不必要的历史版本，详情请参见使用生命周期管理文件版本。

INFO：org.apache.flink.fs.osshadoop.shaded.com.aliyun.oss

- 报错详情

```
2020-08-09 10:18:06,010 INFO  org.apache.flink.runtime.jobmaster.JobMaster                  [] - Configuring application-defined state backend with job/cluster config
2020-08-09 10:18:06,249 INFO  org.apache.flink.fs.osshadoop.shaded.com.aliyun.oss           [] - [Server]Unable to execute HTTP request: Not Found
[ErrorCode]: NoSuchKey
[RequestId]: 5F2F5CDE79BCB53539AF510C
[HostId]: null
2020-08-09 10:18:06,262 INFO  org.apache.flink.fs.osshadoop.shaded.com.aliyun.oss           [] - [Server]Unable to execute HTTP request: Not Found
[ErrorCode]: NoSuchKey
[RequestId]: 5F2F5CDE79BCB53539CF510C
[HostId]: null
2020-08-09 10:18:06,349 INFO  org.apache.flink.fs.osshadoop.shaded.com.aliyun.oss           [] - [Server]Unable to execute HTTP request: Not Found
[ErrorCode]: NoSuchKey
[RequestId]: 5F2F5CDE79BCB53539914520C
[HostId]: null
```

- 报错原因

  OSS每次创建新目录时，会先检查是否存在该目录，如果不存在，就会报这个INFO信息，但该INFO信息不影响Flink作业运行。

- 解决方案

  在日志模板中添加 `<Logger level="ERROR" name="org.apache.flink.fs.osshadoop.shaded.com.aliyun.oss"/>` 。详情请参见配置历史作业实例日志输出。

10.7. State常见问题

本文为您介绍实时计算Flink版State相关的常见问题。

- VVR 3.x.x和VVR 4.x.x中GeminiStateBackend有什么区别？
- 报错：No space left on device
- 报错：java.lang.IllegalArgumentException: Illegal Capacity: -1
- 报错：java.lang.NegativeArraySizeException

VVR 3.x.x和VVR 4.x.x中GeminiStateBackend有什么区别？

相比VVR 3.x.x中的GeminiStateBackend，4.x.x对GeminiStateBackend实现了全面的升级，性能得到了进一步的提升。VVR 3.x.x中的GeminiStateBackend只做缺陷修复而不增加新功能和改进。建议将VVR升级至4.x.x版本，以获得更好的功能和性能支持。

报错：No space left on device

- 报错详情

  在作业运行过程中，出现类似以下异常。

```
java.io.IOException: No space left on device
  at java.io.FileOutputStream.writeBytes(Native Method) ~[?:1.8.0_102]
  at java.io.FileOutputStream.write(FileOutputStream.java:326) ~[?:1.8.0_102]
  at java.io.BufferedOutputStream.flushBuffer(BufferedOutputStream.java:82) ~[?:1.8.0_102
]
  at java.io.BufferedOutputStream.flush(BufferedOutputStream.java:140) ~[?:1.8.0_102]
  at java.io.FilterOutputStream.close(FilterOutputStream.java:158) ~[?:1.8.0_102]
  at org.apache.flink.fs.shaded.hadoop3.org.apache.hadoop.fs.aliyun.oss.AliyunOSSOutputSt
ream.close(AliyunOSSOutputStream.java:82) ~[?:?]
  at org.apache.flink.fs.shaded.hadoop3.org.apache.hadoop.fs.FSDataOutputStream$PositionC
ache.close(FSDataOutputStream.java:72) ~[?:?]
  at org.apache.flink.fs.shaded.hadoop3.org.apache.hadoop.fs.FSDataOutputStream.close(FSD
ataOutputStream.java:101) ~[?:?]
  at org.apache.flink.fs.osshadoop.common.HadoopDataOutputStream.close(HadoopDataOutputSt
ream.java:52) ~[?:?]
  at com.alibaba.flink.statebackend.FlinkDataOutputStreamWapper.close(FlinkDataOutputStre
amWapper.java:31) ~[flink-statebackend-gemini-2.1.23-vvr-3.0-SNAPSHOT.jar:2.1.23-vvr-3.0-
SNAPSHOT]
  at com.alibaba.gemini.common.io.GeminiFileOutputViewImpl.close(GeminiFileOutputViewImpl
.java:188) ~[flink-statebackend-gemini-2.1.23-vvr-3.0-SNAPSHOT.jar:2.1.23-vvr-3.0-SNAPSHO
T]
  at com.alibaba.gemini.engine.filecache.InfiniteFileCache.lambda$flushBatchPages$1(Infin
iteFileCache.java:635) ~[flink-statebackend-gemini-2.1.23-vvr-3.0-SNAPSHOT.jar:2.1.23-vvr
-3.0-SNAPSHOT]
  at com.alibaba.gemini.engine.handler.GeminiEventExecutor.lambda$execute$1(GeminiEventEx
ecutor.java:137) ~[flink-statebackend-gemini-2.1.23-vvr-3.0-SNAPSHOT.jar:2.1.23-vvr-3.0-S
NAPSHOT]
  at com.alibaba.gemini.engine.handler.GeminiEventExecutor.doEventInQueue(GeminiEventExec
utor.java:86) [flink-statebackend-gemini-2.1.23-vvr-3.0-SNAPSHOT.jar:2.1.23-vvr-3.0-SNAPS
HOT]
  at com.alibaba.gemini.engine.handler.GeminiEventExecutor.run(GeminiEventExecutor.java:7
1) [flink-statebackend-gemini-2.1.23-vvr-3.0-SNAPSHOT.jar:2.1.23-vvr-3.0-SNAPSHOT]
  at org.apache.flink.shaded.netty4.io.netty.util.concurrent.SingleThreadEventExecutor$4.
run(SingleThreadEventExecutor.java:989) [flink-dist_2.11-1.12-vvr-3.0.4-SNAPSHOT.jar:1.12
-vvr-3.0.4-SNAPSHOT]
  at org.apache.flink.shaded.netty4.io.netty.util.internal.ThreadExecutorMap$2.run(Thread
ExecutorMap.java:74) [flink-dist_2.11-1.12-vvr-3.0.4-SNAPSHOT.jar:1.12-vvr-3.0.4-SNAPSHOT
]
  at java.lang.Thread.run(Thread.java:834) [?:1.8.0_102]
```

- 报错原因

  本地磁盘空间不足。目前单Pod磁盘大小限制为20 GB，在该限制下，造成本地磁盘空间不足的原因通常有以下几点：

  - 状态数据堆积。
  - 计算节点的非状态数据（例如日志）堆积。
  - 异常导致的旧状态数据堆积。

- 解决方案

  您可以通过快照里的State Size判断状态数据是不是过大。如果状态数据相对较小，请您提交工单进行咨询。如果确定因为状态数据过大造成该报错，则可以采用以下解决方案：

- VVR 4.x.x版本优化方案

  您可以采用以下任何一种方案：

  - 启用存算分离功能。

    即配置state.backend.gemini.file.cache.type和state.backend.gemini.file.cache.preserved-space参数。详情请参见存算分离配置

  - 增加并发度。

    如果原来并发度是1，只能用一个pod，磁盘总空间是20 GB。如果并发加到4，作业就可以使用4个Pod，磁盘总空间相当于是80 GB了。

  - 基于TTL进行磁盘清理。

    State层面配置TTL，这个是State负责数据清理的，数据进到State之后，经过TTL设置的时长就无效了。

- VVR 3.x.x版本优化方案

  您可以采用以下任何一种方案：

  - 对State进行压缩。

    VVR 3.0.x版本配置state.backend.gemini.page.flush.local.compression: Lz4参数，本地State会有压缩，能降低本地磁盘空间，但一定程度上会降低作业性能。

  - 启用存算分离功能。

    VVR 3.0.3及以上版本配置state.backend.gemini.file.cache.type: LIMITED。本地盘会有一个18 GB的State限制，超过18 GB的数据会被存储到远程DFS，下次读取该文件内容时，会从DFS读取，相当于把本地盘当作一个本地文件缓存。

报错：java.lang.IllegalArgumentException: Illegal Capacity: -1

- 报错详情

  在作业使用Map State的遍历操作时，在作业运行过程中，有小概率会触发到以下异常。

```
java.lang.IllegalArgumentException: Illegal Capacity: -1
  at java.util.ArrayList.<init>(ArrayList.java:156)
  at com.alibaba.gemini.engine.pagestore.PageStoreImpl$3.<init>(PageStoreImpl.java:1113)
  at com.alibaba.gemini.engine.pagestore.PageStoreImpl.prefixIterator(PageStoreImpl.java:
1094)
  at com.alibaba.gemini.engine.pagestore.PageStoreImpl.prefixIterator(PageStoreImpl.java:
112)
  at com.alibaba.gemini.engine.table.BinaryKMapTable.internalEntries(BinaryKMapTable.java
:83)
  at com.alibaba.gemini.engine.table.AbstractBinaryKMapTable.iterator(AbstractBinaryKMapT
able.java:282)
  at com.alibaba.flink.statebackend.gemini.keyed.AbstractGeminiKeyedMapStateImpl.doIterat
or(AbstractGeminiKeyedMapStateImpl.java:496)
  at com.alibaba.flink.statebackend.gemini.keyed.AbstractGeminiKeyedMapStateImpl.iterator
WithMetrics(AbstractGeminiKeyedMapStateImpl.java:501)
  at com.alibaba.flink.statebackend.gemini.keyed.AbstractGeminiKeyedMapStateImpl.iterator
(AbstractGeminiKeyedMapStateImpl.java:489)
  at com.alibaba.flink.statebackend.gemini.context.ContextMapState.entries(ContextMapStat
e.java:97)
  at org.apache.flink.runtime.state.ttl.TtlMapState.entries(TtlMapState.java:107)
  at org.apache.flink.runtime.state.ttl.TtlMapState.entries(TtlMapState.java:102)
```

```
  at org.apache.flink.runtime.state.UserFacingMapState.entries(UserFacingMapState.java:77
)
  at org.apache.flink.table.runtime.operators.join.stream.state.OuterJoinRecordStateViews
$InputSideHasNoUniqueKey$1.<init>(OuterJoinRecordStateViews.java:279)
  at org.apache.flink.table.runtime.operators.join.stream.state.OuterJoinRecordStateViews
$InputSideHasNoUniqueKey.getRecordsAndNumOfAssociations(OuterJoinRecordStateViews.java:27
6)
  at org.apache.flink.table.runtime.operators.join.stream.AbstractStreamingJoinOperator$A
ssociatedRecords.of(AbstractStreamingJoinOperator.java:229)
  at org.apache.flink.table.runtime.operators.join.stream.StreamingJoinOperator.processEl
ement(StreamingJoinOperator.java:216)
  at org.apache.flink.table.runtime.operators.join.stream.StreamingJoinOperator.processRi
ght(StreamingJoinOperator.java:134)
  at org.apache.flink.table.runtime.operators.join.stream.AbstractStreamingJoinOperator.p
rocessElement2(AbstractStreamingJoinOperator.java:136)
  at org.apache.flink.streaming.runtime.io.StreamTwoInputProcessorFactory.processRecord2(
StreamTwoInputProcessorFactory.java:221)
  at org.apache.flink.streaming.runtime.io.StreamTwoInputProcessorFactory.lambda$create$1
(StreamTwoInputProcessorFactory.java:190)
  at org.apache.flink.streaming.runtime.io.StreamTwoInputProcessorFactory$StreamTaskNetwo
rkOutput.emitRecord(StreamTwoInputProcessorFactory.java:291)
  at org.apache.flink.streaming.runtime.io.AbstractStreamTaskNetworkInput.processElement(
AbstractStreamTaskNetworkInput.java:134)
  at org.apache.flink.streaming.runtime.io.AbstractStreamTaskNetworkInput.emitNext(Abstra
ctStreamTaskNetworkInput.java:105)
  at org.apache.flink.streaming.runtime.io.StreamOneInputProcessor.processInput(StreamOne
InputProcessor.java:66)
  at org.apache.flink.streaming.runtime.io.StreamTwoInputProcessor.processInput(StreamTwo
InputProcessor.java:98)
  at org.apache.flink.streaming.runtime.tasks.StreamTask.processInput(StreamTask.java:423
)
  at org.apache.flink.streaming.runtime.tasks.mailbox.MailboxProcessor.runMailboxLoop(Mai
lboxProcessor.java:204)
  at org.apache.flink.streaming.runtime.tasks.StreamTask.runMailboxLoop(StreamTask.java:6
84)
  at org.apache.flink.streaming.runtime.tasks.StreamTask.executeInvoke(StreamTask.java:63
9)
  at org.apache.flink.streaming.runtime.tasks.StreamTask.runWithCleanUpOnFail(StreamTask.
java:650)
  at org.apache.flink.streaming.runtime.tasks.StreamTask.invoke(StreamTask.java:623)
  at org.apache.flink.runtime.taskmanager.Task.doRun(Task.java:779)
  at org.apache.flink.runtime.taskmanager.Task.run(Task.java:566)
  at java.lang.Thread.run(Thread.java:834)
```

- 报错原因

  产品已知缺陷，仅在VVR 4.0.10版本出现。

- 解决方案

  将VVR版本升级到4.0.11及以上。

报错：java.lang.NegativeArraySizeException

- 报错详情

当作业使用了List State，在作业运行过程中，会出现以下异常。

```
Caused by: java.lang.NegativeArraySizeException
  at com.alibaba.gemini.engine.rm.GUnPooledByteBuffer.newTempBuffer(GUnPooledByteBuffer.j
ava:270)
  at com.alibaba.gemini.engine.page.bmap.BinaryValue.merge(BinaryValue.java:85)
  at com.alibaba.gemini.engine.page.bmap.BinaryValue.merge(BinaryValue.java:75)
  at com.alibaba.gemini.engine.pagestore.PageStoreImpl.internalGet(PageStoreImpl.java:428
)
  at com.alibaba.gemini.engine.pagestore.PageStoreImpl.get(PageStoreImpl.java:271)
  at com.alibaba.gemini.engine.pagestore.PageStoreImpl.get(PageStoreImpl.java:112)
  at com.alibaba.gemini.engine.table.BinaryKListTable.get(BinaryKListTable.java:118)
  at com.alibaba.gemini.engine.table.BinaryKListTable.get(BinaryKListTable.java:57)
  at com.alibaba.flink.statebackend.gemini.subkeyed.GeminiSubKeyedListStateImpl.getOrDefa
ult(GeminiSubKeyedListStateImpl.java:97)
  at com.alibaba.flink.statebackend.gemini.subkeyed.GeminiSubKeyedListStateImpl.get(Gemin
iSubKeyedListStateImpl.java:88)
  at com.alibaba.flink.statebackend.gemini.subkeyed.GeminiSubKeyedListStateImpl.get(Gemin
iSubKeyedListStateImpl.java:47)
  at com.alibaba.flink.statebackend.gemini.context.ContextSubKeyedListState.get(ContextSu
bKeyedListState.java:60)
  at com.alibaba.flink.statebackend.gemini.context.ContextSubKeyedListState.get(ContextSu
bKeyedListState.java:44)
  at org.apache.flink.streaming.runtime.operators.windowing.WindowOperator.onProcessingTi
me(WindowOperator.java:533)
  at org.apache.flink.streaming.api.operators.InternalTimerServiceImpl.onProcessingTime(I
nternalTimerServiceImpl.java:289)
  at org.apache.flink.streaming.runtime.tasks.StreamTask.invokeProcessingTimeCallback(Str
eamTask.java:1435)
```

- 报错原因

  List State中单个key对应的State数据过大，即超过了2 GB。State数据过大产生的过程如下：

  i. 在作业正常运行时，List State中单个Key下Append的Value会通过Merge进行组合（例如在包含Window的List State中），State数据不断累积。

  ii. 在State数据累积到一定程度时，一开始会触发OOM。而在作业从故障中恢复之后，List State的Merge过程会进一步导致StateBackend申请的临时Byte数组的大小超过可用的限制，从而抛出该异常。

  > **说明** RocksDBStateBackend也会遇到类似的问题并触发ArrayIndexOutOfBoundsException或者Segmentation fault。详情请参见The EmbeddedRocksDBStateBackend。

- 解决方案
  - 如果是Window算子导致的State数据过大，则建议减小窗口大小。
  - 如果是作业逻辑不合理，则建议调整作业逻辑，例如将Key进行拆分。

11.视频专区

11.1. 什么是阿里云实时计算Flink版

11.2. Flink全托管开通流程

11.3. Flink全托管产品使用

11.4. MySQL实时同步至Hologres

12.相关协议

12.1. Flink全托管服务等级协议（SLA）

Flink全托管服务等级协议（SLA）的详情，请参见Flink全托管服务等级协议。